High-Performance Parallel Graph Coloring with Strong Guarantees on Work, Depth, and Quality

Maciej Besta[1], Armon Carigiet[1], Kacper Janda[2], Zur Vonarburg-Shmaria[1], Lukas Gianinazzi[1], Torsten Hoefler[1]
[1]*Department of Computer Science, ETH Zurich*
[2]*Department of Computer Science, Electronics and Telecommunications, AGH-UST*

***Abstract*—We develop the first parallel graph coloring heuristics with strong theoretical guarantees on *work* and *depth* and *coloring quality*. The key idea is to design a *relaxation* of the *vertex degeneracy order*, a well-known graph theory concept, and to color vertices in the order dictated by this relaxation. This introduces a tunable amount of parallelism into the degeneracy ordering that is otherwise hard to parallelize. This simple idea enables significant benefits in several key aspects of graph coloring. For example, one of our algorithms ensures polylogarithmic depth and a bound on the number of used colors that is superior to all other parallelizable schemes, while maintaining work-efficiency. In addition to provable guarantees, the developed algorithms have competitive run-times for several real-world graphs, while almost always providing superior coloring quality. Our degeneracy ordering relaxation is of separate interest for algorithms outside the context of coloring.**

Extended paper version: https://arxiv.org/abs/2008.11321

I. Introduction

Graph coloring, more specifically *vertex coloring*, is a well studied problem in computer science, with many practical applications in domains such as sparse linear algebra computations [1]–[3] or conflicting task scheduling [4]–[6]. A *vertex coloring* of a graph G is an assignment of colors to vertices, such that no two neighboring vertices share the same color. A k-coloring is a vertex coloring of G which uses k distinct colors. The minimal amount of colors k for which a k-coloring can be found for G is referred to as the *chromatic number* $\chi(G)$. An *optimal coloring*, also sometimes referred to as *the coloring problem* or a χ-coloring, is the problem of coloring G with $\chi(G)$ colors. Unfortunately, finding such and optimal coloring was shown to be NP-complete [7].

Nonetheless, colorings with a reasonably low number of colors can in practice be computed quite efficiently in the sequential setting using *heuristics*. One of the most important is the *Greedy* heuristic [8], which sequentially colors vertices by choosing, for each selected vertex v, the smallest color not already taken by v's neighbors. This gives a *guarantee* for a coloring of G with at most $\Delta + 1$ colors, where Δ is the maximum degree in G. To further improve the coloring quality (i.e., #colors used), Greedy is in practice often used with a certain *vertex ordering heuristic*, which decides the *order in which Greedy colors the vertices*. Example heuristics include: *first-fit* (FF) [8] which uses the natural (default) order of the vertices in G, *largest-degree-first* (LF) [8] which orders vertices according to their degrees, *random* (R) [9] which chooses vertices in a uniformly random manner, *incidence-degree* (ID) [1] which always picks vertices with the largest number of uncolored neighbors first, *saturation-degree* (SD) [10], where a vertex whose neighbors use the largest number of distinct colors is chosen first, and *smallest-degree-last* (SL) [11] that removes lowest degree vertices, recursively colors the resulting graph, and then colors the removed vertices. All these ordering heuristics, combined with Greedy, have the inherent problem of *no parallelism*.

Jones and Plassmann combined this line of work with earlier parallel schemes for deriving maximum independent sets [12], [13] and obtained a *parallel* graph coloring algorithm (JP) that colors a vertex v once all of v's neighbors that come later in the provided ordering have been colored. They showed that JP, combined with a *random vertex ordering* (JP-R), runs in expected depth $O(\log n / \log\log n)$ and $O(n + m)$ work for constant-degree graphs (n and m are #vertices and #edges in G, respectively). Recently, Hasenplaugh et al. [14] extended JP with the *largest-log-degree first* (LLF) and *smallest-log-degree-last* (SLL) orderings with better bounds on depth; these orderings approximate the LF and SL orderings, respectively. There is also another (earlier) work [15] that – similarly to JP-SLL – approximates SL with the *"ASL" ordering*. The resulting coloring combines JP with ASL, we denote it as JP-ASL [15]. However, it offers *no bounds* for work or depth.

Overall, *there is no parallel algorithm with strong theoretical guarantees on work and depth and quality*. Whilst having a reasonable theoretical run-time, JP-R may offer colorings of poor quality [14], [16]. On the other hand, JP-LF and JP-SL, which provide a better coloring quality, run in $\Omega(n)$ or $\Omega(\Delta^2)$ for some graphs [14]. This was addressed by the recent JP-LLF and JP-SLL algorithms [14] that produce colorings of similarly good quality to their counterparts JP-LF and JP-SL, and run in an expected depth that is within a logarithmic factor of JP-R. However, *no guaranteed upper bounds on the coloring quality (#colors), better than the trivial $\Delta+1$ bound from Greedy, exist for JP-LLF, JP-SLL, or JP-ASL.*

To alleviate these issues, we present ***the first*** *graph coloring algorithms with provably good bounds on* ***work*** *and* ***depth*** *and* ***quality****, simultaneously ensuring high performance and competitive quality in practice*. The key idea is to use a novel vertex ordering, the *provably approximate degeneracy ordering* (**ADG**, contribution **#1**) when selecting which vertex is the next to be colored. The exact degeneracy ordering is – intuitively – an ordering obtained by iteratively removing vertices of smallest degrees. Using the degeneracy ordering with

SC20, November 9-19, 2020, Is Everywhere We Are
978-1-7281-9998-6/20/$31.00 ©2020 IEEE

JP leads to the best possible Greedy coloring quality [11]. Still, computing the exact degeneracy ordering is hard to parallelize: for some graphs, it leads to $\Omega(n)$ coloring run-time [14]. To tackle this, we (provably) relax the strict degeneracy order by assigning the same rank (in the ADG ordering) to a *batch* of vertices that – intuitively – have *similarly small degrees*. This approach also results in *provably higher parallelization* because each batch of vertices can be processed in parallel.

This simple idea, when applied to graph coloring, gives surprisingly rich outcome. We use it to develop three novel graph coloring algorithms that enhance two relevant lines of research. We first combine ADG with JP, obtaining **JP-ADG** (contribution **#2**), a coloring algorithm that is *parallelizable*: vertices with the same ADG rank are colored in parallel. It has the expected worst-case depth of $O(\log^2 n + \log\Delta(d\log n + \log d\log^2 n/\log\log n))$. Here, d is the *degeneracy of a graph* G: an upper bound on the minimal degree of every induced subgraph of G (detailed in § II-B) [11]. JP-ADG is also *work-efficient* ($O(n+m)$ work) and has *good coloring quality*: it uses at most $2(1+\varepsilon)d+1$ colors, for any $\varepsilon > 0$. Moreover, we also combine ADG with another important line of graph coloring algorithms that are *not* based on JP but instead use *speculation* [15], [17]–[29]. Here, vertices are colored independently ("speculative coloring"). Potential *coloring conflicts* (adjacent vertices assigned the same colors) are resolved by repeating coloring attempts. Combining ADG with this design gives **DEC-ADG** (contribution **#3**), *the first scheme based on speculative coloring with provable strong guarantees on all key aspects of parallel graph coloring*: work $O(n+m)$, depth $O(\log d\log^2 n)$, and quality $(2+\varepsilon)d$. Finally, we combine key design ideas in DEC-ADG with an existing recent ITR algorithm [23] also based on speculative coloring. We derive an algorithm called **DEC-ADG-ITR** that improves coloring quality of ITR *both in theory and practice*.

We conduct the most extensive theoretical analysis of graph coloring algorithms so far, considering 20 parallel graph coloring routines *with provable guarantees* (contribution **#5**). All our algorithms offer substantially better bounds than past work. We also perform a broad empirical evaluation. For comparison, we use *the most competitive* variants of *all classes* of considered coloring algorithms, from four existing repositories (ColPack [30], [31], Zoltan [18], [32]–[35], original code by Hasenplaugh et al. (HP) [14], and Graph Based Benchmark Suite (GBBS) with Ligra [36]–[38]), giving in total more than 25 evaluated, analyzed, and compared specific baselines. Our algorithms (1) are competitive in run-times for several real-world graphs, while (2) offering superior coloring quality for almost all datasets. *Our algorithms offer the best coloring quality at the smallest required runtime overhead.*

In a brief summary, we offer the following:

- The first parallel algorithm for deriving the (approximate) graph degeneracy ordering (ADG).
- The first parallel graph coloring algorithm (JP-ADG), in a line of heuristics based on Jones and Plassman's scheme, with strong bounds on work, depth, and coloring quality.
- The first parallel graph coloring algorithm (DEC-ADG), in a line of heuristics based on speculative coloring, with strong bounds on work, depth, and coloring quality
- A use case of how ADG can seamlessly enhance an existing state-of-the-art graph coloring scheme (DEC-ADG-ITR).
- The most extensive (so far) theoretical analysis of parallel graph coloring algorithms, showing advantages of our algorithms over state-of-the-art in several dimensions.
- Superior coloring quality offered by our algorithms over tuned modern schemes for many real-world graphs.

We note that degeneracy ordering is used beyond graph coloring [39]–[42]; thus, our *ADG scheme is of separate interest*.

II. Fundamental Concepts

We start with background; Table I lists key symbols. Vertex coloring was already described in Section I.

A. Graph Model and Representation

We model a graph G as a tuple (V, E); V is a set of vertices and $E \subseteq V \times V$ is a set of edges; $|V| = n$ and $|E| = m$. We focus on graph coloring problems where edge directions are not relevant. Thus, G is undirected. The maximum, minimum, and average degree of a given graph G are Δ, δ, and $\widehat{\delta}$, respectively. The neighbors and the degree of a given vertex v are $N(v)$ and $deg(v)$, respectively. $G[U] = (U, E[U])$ denotes an induced subgraph of G: a graph where $U \subseteq V$ and $E[U] = \{(v,u) \mid v \in U \wedge u \in U\}$, i.e., $E[U]$ contains edges with both endpoints in U. $N_U(v)$ and $deg_U(v)$ are the neighborhood and the degree of $v \in V$ in $G[U]$. The vertices are identified by integer IDs that define a total order: $V = \{1, \ldots, n\}$. We store G using CSR, the standard graph representation that consists of n sorted arrays with neighbors of each vertex ($2m$ words) and offsets to each array (n words).

G	A graph $G = (V, E)$; V and E are sets of vertices and edges.
$G[U]$	$G[U] = (U, E[U])$ is a subgraph of G induced on $U \subseteq V$.
n, m	Numbers of vertices and edges in G; $\|V\| = n, \|E\| = m$.
$\Delta, \delta, \widehat{\delta}$	Maximum degree, minimum degree, and average degree of G.
d	The degeneracy of G.
$deg(v), N(v)$	The degree and the neighborhood of a vertex $v \in V$.
$deg_U(v)$	The degree of v in a subgraph induced by the vertex set $U \subseteq V$.
$N_U(v)$	The neighborhood of v in a subgraph induced by $U \subseteq V$.
$\rho_X(v)$	A priority function $V \to \mathbb{R}$ associated with vertex ordering X.
P	The number of processors (in a given PRAM machine).

TABLE I: Selected symbols used in the paper. When we use a symbol in the context of a specific loop iteration ℓ, we add ℓ in brackets or as subscript (e.g., $\widehat{\delta}_\ell$ is $\widehat{\delta}$ in iteration ℓ).

B. Degeneracy and Related Concepts

A graph G is *s-degenerate* [43] if, in each of its induced subgraphs, there is a vertex with a degree of at most s. The *degeneracy* d of G [44]–[47] is the *smallest* s, such that G is still s-degenerate. The *degeneracy ordering* of G [11] is an ordering, where each vertex v has at most d neighbors that are ordered higher than v. Then, a *k-approximate degeneracy ordering* differs from the exact one in that v has at most $k \cdot d$ neighbors ranked higher in this order. A *partial k-approximate degeneracy ordering* is a similar ordering, where multiple vertices can be ranked equally, and we have that each vertex has at most $k \cdot d$ neighbors with *equal or higher* rank. A *partial k-approximate degeneracy ordering* can be trivially extended into a *k-approximate degeneracy ordering* by imposing an

(arbitrary) order on vertices ranked equally. Both *degeneracy* and a *degeneracy ordering* of G can be computed in linear time by sequentially removing vertices of smallest degree [11].

C. Models for Algorithm Analysis

As a **compute model**, we use the *DAG model of dynamic multithreading* [48], [49]. In this model, a specific computation (resulting from running some parallel program) is modeled as a *directed acyclic graph* (DAG). Each *node* in a DAG corresponds to a constant time operation. *In-edges* of a node model the data used for the operation. As operations run in constant time, there are $O(1)$ in-edges per node. The *out-edges* of a node correspond to the computed output. A node can be executed as soon as all predecessors finish executing.

Following related work [14], [36], we assume that a parallel computation (modeled as a DAG) runs on the *ideal parallel computer* (**machine model**). Each instruction executes in unit time and there is support for concurrent reads, writes, and read-modify-write atomics (any number of such instructions finish in $O(1)$ time). We develop algorithms based on these assumptions *but we also provide algorithms that use weaker assumptions* (algorithms that only rely on *concurrent reads*).

We use the **work-depth (W–D) analysis** for bounding runtimes of parallel algorithms in the DAG model. The *work* of an algorithm is the total number of nodes and the *depth* is defined as the longest directed path in the DAG [50], [51].

Our Analyses vs. PRAM In our W-D analysis, two used machine model variants (1) only need concurrent reads and (2) may also need concurrent writes. These variants are analogous to those of the well-known PRAM model [51]–[54]: CREW and CRCW, respectively. Thus, when describing a W–D algorithm that *only relies on concurrent reads*, we use a term "the CREW setting". Similarly, for a W–D algorithm that *needs concurrent writes*, we use a term "the CRCW setting".

The well-known Brent's result states that any deterministic algorithm with work W and depth D can be executed on P processors in time T such that $\max\{W/P, D\} \leq T \leq W/P + D$ [55]. Thus, all our results are applicable to a PRAM setting.

D. Compute Primitives

We use a $Reduce$ operation. It takes as input a set $S = \{s_1, ..., s_n\}$ implemented as an array (or a bitmap). It uses a function $f : S \to \mathbb{N}$ called the *operator*; $f(s)$ must be defined for any $s \in S$. $Reduce$ calculates the sum of elements in S with respect to f: $f(s_1) + ... + f(s_n)$. This takes $O(\log n)$ depth and $O(n)$ work in the CREW setting [56], [57], where n is the array size. We use $Reduce$ to implement $Count(S)$, which computes the size $|S|$. For this, the associated operator f is defined as $f(s) = 1$ if $s \in S$, and $f(s) = 0$ otherwise. We also assume a *DecrementAndFetch* (DAF) to be available; it atomically decrements its operand and returns a new value [14]. We use DAF to implement *Join* to synchronize processors (Join decrements its operand, returns the new value, and releases a processor under a specified condition).

III. Parallel Approximate Degeneracy Ordering

We first describe **ADG**, a parallel algorithm for computing a partial approximate degeneracy ordering. ADG outputs vertex *priorities* ρ_{ADG}, which are then used by our coloring algorithms (Section IV). Specifically, these priorities produce an order in which to color the vertices (ties are broken randomly).

ADG is shown in Algorithm 1. ADG is similar to SL [11], which iteratively removes vertices of the smallest degree to construct the exact degeneracy ordering. *The key difference and our core idea is to repeatedly remove* ***in parallel*** *all vertices with degrees smaller than* $(1+\varepsilon)\widehat{\delta}$. The parameter $\varepsilon \geq 0$ controls the approximation accuracy. We multiply $1+\varepsilon$ by the average degree $\widehat{\delta}$ as it enables good bounds on quality and run-time, as we show in Lemma 1 and 3. Compared to SL (which has depth $O(n)$), ADG has depth $O(\log^2 n)$ and obtains a partial $2(1+\varepsilon)$-approximate degeneracy ordering.

```
1  /* Input: A graph G(V, E).
2   * Output: A priority (ordering) function ρ : V → ℝ. */
3  D = [ deg(v_1) deg(v_2) ... deg(v_n) ] //An array with vertex degrees
4  ℓ = 1; U = V //U is the induced subgraph used in each iteration ℓ
5  while U ≠ ∅ do:
6    |U| = Count(U); //Derive |U| using a primitive Count
7    cnt = Reduce(U); //Derive the sum of degrees in U: Σ_{v∈U} D[v]
8    δ̂ = cnt/|U| //Derive the average degree for vertices in U
9    //R contains vertices assigned priority in a given iteration:
10   R = {u ∈ U | D[u] ≤ (1 + ε)δ̂ }
11   UPDATE(U, R, D) //Update D to reflect removing R from U
12   U = U \ R //Remove selected low-degree vertices (that are in R)
13   for all v ∈ R do in parallel: //Set the priority of vertices
14     ρ_ADG(v) = ℓ //The priority is the current iteration number ℓ
15   ℓ = ℓ + 1
16 //Update D to reflect removing vertices in R from a set U:
17 UPDATE(U, R, D):
18 for all v ∈ R do in parallel:
19   for all u ∈ N_U(v) do in parallel:
20     DecrementAndFetch(D[u])
```

Algorithm 1: **ADG**, our algorithm for computing the $2(1+\varepsilon)$-approximate degeneracy ordering; it runs in the CRCW setting.

In ADG, we maintain a set $U \subseteq V$ of active vertices that starts as V (Line 4). In each step (Lines 6–15), we use ε and $\widehat{\delta}$ to select vertices with small enough degrees (Line 10); The average degree $\widehat{\delta}$ is computed in Lines 6–8. The selected vertices form a set R and receive a priority ρ_{ADG} equal to the step counter ℓ. We then remove them from the set U (Line 12) and update the degrees D accordingly (Line 11). We continue until the set U is empty.

Design Details We implement D as an array and use n-bit dense bitmaps for U and R. This enables updating vertex degrees in $O(1)$ and resolving $v \in U$ and $v \in R$ in $O(1)$ time. Constructing R in each step can be implemented in $O(1)$ depth and $O(|U|)$ work. The operation $U = U \backslash R$ takes $O(1)$ depth and $O(|R|)$ work by overwriting the bitmap U. To calculate the average degree on Line 8, we derive $|U|$ and sum all degrees of vertices in U. The former is done with a $Count$ over U. The latter uses $Reduce$ with the associated operator $f(v) = D[v]$. As both $Reduce$ operations run in $O(\log n)$ depth and $O(|U|)$ work, the same holds for the average degree calculation.

Depth First, note that each line in the `while` loop runs in $O(\log n)$ depth, as discussed above. We will now prove that the `while` loop iterates $O(\log n)$ many times, giving the total ADG depth of $O(\log^2 n)$. The key notion is that, in each

iteration, we remove *a constant fraction of vertices* due to the way that we construct R (based on the *average degree* $\widehat{\delta}$).

Lemma 1. *For a constant $\varepsilon > 0$, ADG does $O(\log n)$ iterations and has $O(\log^2 n)$ depth in the CRCW setting.*

Proof. At each step ℓ of the algorithm we can have at most $\frac{n}{1+\varepsilon}$ vertices with a degree larger than $(1+\varepsilon)\widehat{\delta}_\ell$. This can be seen from the fact, that the sum of degrees in the current subgraph can be at most n times the average degree $\widehat{\delta}_\ell$. For vertices with a degree exactly $(1+\varepsilon)\widehat{\delta}_\ell$ we get $\frac{n}{1+\varepsilon}\cdot(1+\varepsilon)\widehat{\delta}_\ell = n\widehat{\delta}_\ell$, which would result in a contradiction if we had more than $\frac{n}{1+\varepsilon}$ vertices with larger degree. Thus, if we remove all vertices with degree $\leq (1+\varepsilon)\widehat{\delta}_\ell$, *we remove a constant fraction of vertices in each iteration* (at least $\frac{\varepsilon}{1+\varepsilon}n$ vertices), which implies that ADG performs $O(\log n)$ iterations in the worst case, immediately giving the $O(\log^2 n)$ depth. □

Work The proof of work is similar and included in the extended report due to space constraints; it also uses the fact that a constant fraction of vertices is removed in each iteration.

Approximation ratio We now prove that the approximation ratio of ADG on the degeneracy order is $2(1+\varepsilon)$. First, we give a small lemma used throughout the analysis.

Lemma 2. *Every induced subgraph of a graph G with degeneracy d, has an average degree of at most $2d$.*

Proof. By the definition of a d-degenerate graph, in every induced subgraph $G[U]$, there is a vertex v with $deg_U(v) \leq d$. If we remove v from $G[U]$, at most d edges are removed. Thus, if we iteratively remove such vertices from $G[U]$, until only one vertex is left, we remove at most $d\cdot(|U|-1)$ edges. We conclude that $\widehat{\delta}(G[U]) = \frac{1}{|U|}\sum_{v\in U} deg_U(v) \leq 2d$. □

Lemma 3. *ADG computes a partial $2(1+\varepsilon)$-approximate degeneracy ordering of G.*

Proof. By the definition of R (Line 10), all vertices removed in step ℓ have a degree of at most $(1+\varepsilon)\widehat{\delta}_\ell$, where $\widehat{\delta}_\ell$ is the average degree of vertices in subgraph U in step ℓ. From Lemma 2, we know that $\widehat{\delta}_\ell \leq 2d$. Thus, each vertex has a degree of at most $2(1+\varepsilon)d$ in the subgraph $G[U]$ (in the current step). Hence, each vertex has at most $2(1+\varepsilon)d$ neighbors that are ranked equal or higher. The result follows by the definition of a partial $2(1+\varepsilon)$-approximate degeneracy order. □

A. Comparison to Other Vertex Orderings

We analyze orderings in Table II. While SLL and ASL heuristically approximate SL, which computes a degeneracy ordering, they do not offer guaranteed approximation factors. *Only ADG comes with provable bounds on the accuracy of the degeneracy order while being (provably) parallelizable*.

Ordering heuristic	Time / Depth	Work	F.?	B., Approx.?
FF (first-fit) [8]	$O(1)$	$O(1)$	n/a	n/a
R (random) [9], [14]	$O(1)$	$O(n)$	👍	n/a
ID (incidence-degree) [1]	$O(n+m)$	$O(n+m)$	n/a	n/a
SD (saturation-degree) [10], [14]	$O(n+m)$	$O(n+m)$	n/a	n/a
LF (largest-degree-first) [14]	$O(1)$	$O(n)$	👍	n/a
LLF (largest-log-degree-first) [14]	$O(1)$	$O(n)$	👍	n/a
SLL (smallest-log-degree-last) [14]	$O(\log\Delta\log n)$	$O(n+m)$	👍	👎
SL (smallest-degree-last) [11], [14]	$O(n)$	$O(m)$	👍	👍 exact
ASL (approximate-SL) [15]	$O(n)$	$O(m)$	👍	👎
ADG [approx. degeneracy]	$O\left(\log^2 n\right)$	$O(n+m)$	👍	👍 $2(1+\varepsilon)$

TABLE II: **Ordering heuristics related to the degeneracy ordering**. **"F. (Free)?":** Is the scheme free from concurrent writes? **"B. (Bounds)?":** Are there **provable** bounds and approximation ratio on degeneracy ordering? "👍": support, "👎": no support. Notation is explained in Table I and in Section II.

IV. Parallel Graph Coloring

We now use our approximate degeneracy ordering to develop new parallel graph coloring algorithms. We directly enhance the recent line of works based on *scheduling colors*, i.e., assigning colors to vertices without generating coloring conflicts (§ IV-A). In two other algorithms, we *allow conflicts* but we also *provably* resolve them *fast* (§ IV-B, § IV-C).

A. Graph Coloring by Color Scheduling (JP-ADG)

We directly enhance recent works of Hasenplaugh et al. [14] by combining their Jones-Plassmann (JP) version of coloring with our ADG, obtaining **JP-ADG**. For this, we first overview JP and definitions used in JP. The JP algorithm uses the notion of a computation DAG $G_\rho(V, E_\rho)$, which is a directed version of the input graph G. Specifically, the DAG G_ρ is used by JP to *schedule the coloring of the vertices*: The position of a vertex v in the DAG G_ρ determines the moment the vertex v is colored. The DAG G_ρ contains the edges of G directed from the higher priority to lower priority vertices according to the priority function ρ, i.e. , $E_\rho = \{(u,v) \in E \mid \rho(u) > \rho(v)\}$.

JP is described in Algorithm 2. First, JP uses ρ to compute a DAG $G_\rho(V, E_\rho)$, where edges always go from vertices with higher ρ to ones with lower ρ (Lines 6–9). Vertices can then be safely assigned a color if all neighbors of higher ρ (predecessors in the DAG) have been colored. JP does this by calling JPColor with the set of vertices without predecessors (Lines 13–15). JPColor then colors v by calling GetColor, which chooses the smallest color not already taken by v's predecessors. Then, JPColor checks if any of v's successors can be colored, and if yes, it calls again JPColor on them.

Now, in JP-ADG, we first call ADG to derive ρ_{ADG}. Then, we run JP using ρ_{ADG}. More precisely, we use $\rho = \langle \rho_{\text{ADG}}, \rho_{\text{R}} \rangle$ where ρ_R randomly breaks ties of vertices that were removed in the same iteration in Algorithm 1, and thus have the same rank in ρ_{ADG}. The obtained JP-ADG algorithm is similar to past work based on JP in that it follows the same "skeleton" in which coloring of vertices is guided by the pre-computed order, on our case ρ_{ADG}. However, as we prove later in this section, ***using ADG is key*** *to our novel bounds on depth, work, and coloring quality*. Intuitively, ADG gives an ordering of vertices in which each vertex has a *bounded number of predecessors* (by definition of s-degenerate graphs and graph degeneracy d). We use this to bound coloring quality and sizes of subgraphs in G_ρ. The latter enables bounding the maximum path in G_ρ, which in turn gives depth and work bounds.

We first prove a general property of JP-ADG, which we will use to derive bounds on coloring quality, depth, and work.

Lemma 4. *JP, using a priority function ρ that defines a k-approximate degeneracy ordering, colors a graph G with at most $kd+1$ colors, for $\varepsilon > 0$.*

```
1 /* Input: A graph G(V, E), a priority function ρ.
2  * Output: An array C, it assigns a color to each vertex. */
3
4 //Part 1: compute the DAG G_ρ based on ρ
5 C = [0 0 ... 0] //Initialize colors
6 for all v ∈ V do in parallel:
7   //Predecessors and successors of each vertex v in G_ρ:
8   pred[v] = {u ∈ N(v) | ρ(u) > ρ(v)}
9   succ[v] = {u ∈ N(v) | ρ(u) < ρ(v)}
10  //Number of uncolored predecessors of each vertex v in G_ρ:
11  count[v] = |pred[v]|
12
13 //Part 2: color vertices using G_ρ
14 for all v ∈ V do in parallel:
15  //Start by coloring all vertices without predecessors:
16  if pred[v] == ∅: JPColor(v)
17
18 JPColor(v) //JPColor, a routine used in JP
19  C[v] = GetColor(v)
20  for all u ∈ succ[v] in parallel:
21    //Decrement u's counter to reflect that v is now colored:
22    if Join(count[u]) == 0:
23      JPColor(u) //Color u if it has no uncolored predecessors
24
25 GetColor(v) //GetColor, a routine used in JPColor
26  C = {1, 2, ..., |pred[v]| + 1}
27  for all u ∈ pred[v] do in parallel: C = C − {C[u]}
28  return min(C) //Output: the smallest color available.
```

Algorithm 2: **JP**, the Jones-Plassman coloring heuristic. When setting $\rho = \langle \rho_{\text{ADG}}, \rho_{\text{R}} \rangle$, it gives our **JP-ADG** routine that provides $(2(1+\varepsilon)d+1)$-coloring.

Proof. Since ρ defines a k-approximate degeneracy ordering, any $v \in V$ has at most kd neighbors v' with $\rho(v') \geq \rho(v)$ and thus at most kd predecessors in the DAG. Now, we can choose the smallest color available from $\{1, \ldots, kd+1\}$ to color v, when all of its predecessors have been colored. □

Coloring Quality The coloring quality now follows from the properties of the priority function obtained with ADG.

Corollary 1. *With priorities $\rho = \langle \rho_{ADG}, \rho_R \rangle$, JP-ADG colors a graph with at most $2(1+\varepsilon)d+1$ colors, for $\varepsilon > 0$.*

Depth, Work To bound the depth of JP-ADG, we follow the approach by Hasenplaugh et al. [14]. We analyze the expected length of the longest path in a DAG induced by JP-ADG to bound its expected depth. Note that as ρ is a total order on V, the DAG G_ρ is *strongly connected*. Finally, we denote $\overline{\rho} = \max_{v \in V} \{\rho_{\text{ADG}}(v)\}$.

Lemma 5. *For a priority function $\rho = \langle \rho_{ADG}, \rho_R \rangle$, where ρ_{ADG} is a partial k-approximate degeneracy ordering for a constant $k > 1$, ρ_R is a random priority function, the expected length of the longest path in the DAG G_ρ is $O\left(d \log n + \frac{\log d \log^2 n}{\log \log n}\right)$.*

Proof. Let $G_\rho(\ell)$ be the subgraph of G_ρ induced by the vertex set $V(\ell) = \{v \in V \mid \rho_{\text{ADG}} = \ell\}$. Let Δ_ℓ be the maximal degree and $\widehat{\delta}_\ell$ be the average degree of the subgraph $G_\rho(\ell)$.

Since, by the definition of G_ρ, there can be no edges in G_ρ that go from one subgraph $G_\rho(\ell)$ to another $G_\rho(\ell')$ with $\ell' > \ell$, we can see that a longest (directed) path $\mathcal{P}$ in G_ρ will always go through the subgraph $G_\rho(\ell)$ in a monotonically decreasing order with regards to ℓ. Therefore, we can split $\mathcal{P}$ into a sequence of (directed) sub-paths $\mathcal{P}_1, \ldots, \mathcal{P}_{\overline{\rho}}$, where $\mathcal{P}_\ell$ is a path in $G(\ell)$. We have $|\mathcal{P}| = \sum_{i \in \{\rho_{\text{ADG}}(v) | v \in V\}} |\mathcal{P}_i|$ and by Corollary 6 from past work [14], the expected length of a longest sub-path $\mathcal{P}_\ell$ is in $O(\Delta_\ell + \log \Delta_\ell \log n / \log \log n)$, because $G_\rho(\ell)$ is induced by a random priority function. By linearity of expectation, we have for the whole path $\mathcal{P}$:

$$\mathbb{E}\left[|\mathcal{P}|\right] = O\left(\sum_{\ell=1}^{\overline{\rho}} \left(\Delta_\ell + \log \Delta_\ell \cdot \frac{\log n}{\log \log n}\right)\right) \tag{1}$$

Next, since ρ_{ADG} is a partial k-approximate degeneracy ordering, all vertices in $G(\ell)$ have at most kd neighbors in $G(\ell)$. Thus, $\Delta_\ell \leq kd$ holds. This and the fact that $\overline{\rho} \in O(\log n)$ gives:

$$\sum_{i=1}^{\overline{\rho}} \Delta_i \leq \sum_{i=1}^{\overline{\rho}} d \cdot k \in O(d \log n) \tag{2}$$

$$\sum_{i=1}^{\overline{\rho}} \log \Delta_i \in O(\log d \log n) \tag{3}$$

Thus, for the expected length of a longest path in G:

$$\mathbb{E}\left[|\mathcal{P}|\right] = O\left(d \log n + \frac{\log d \log^2 n}{\log \log n}\right) \tag{4}$$

□

Our main result follows by combining our bounds on the longest path $\mathcal{P}$ in the DAG G_ρ and a result by Hasenplaugh et al. [14], which shows that JP has $O(\log n + \log \Delta \cdot |\mathcal{P}|)$ depth.

Theorem 1. *JP-ADG colors a graph G with degeneracy d in expected depth $O(\log^2 n + \log \Delta \cdot (d \log n + \frac{\log d \log^2 n}{\log \log n}))$ and $O(n+m)$ work in the CRCW setting.*

B. Graph Coloring by Silent Conflict Resolution (DEC-ADG)

Our second coloring algorithm takes a radical step to move away from the long line of heuristics based on JP. *The key idea is **to use ADG** to decompose the input graph into low-degree partitions* (thus "**DEC-ADG**"), shown in Algorithm 3. Here, ADG is again crucial to our bounds. Specifically, vertices with the same ADG rank form a *partition* that is "low-degree": it has a bounded number of edges to any other such partitions (by the definition of ADG). Each such partition is then colored separately, with a simple randomized scheme in Algorithm 4. This may generate coloring *conflicts*, i.e., neighboring vertices with identical colors. Such conflicts are resolved "silently" by repeating the coloring on conflicting vertices as many times as needed. As ADG bounds counts of edges between partitions, it also bounds counts of conflicts, improving depth and quality.

We first detail Algorithm 3. A single low-degree partition $G(\ell)$ produced by the iteration ℓ of ADG is the induced subgraph of G over the vertex set R removed in this iteration (Line 10, Alg. 1). Formally, $G(i) = G[R(i)]$ where $R(i) = \{v \in V \mid \rho(v) = i\}$ and ρ is the partial k-approximate degeneracy order produced by ADG (cf. § II-B). Thus, in DEC-ADG, we first run ADG to derive the ordering ρ and also the number $\overline{\rho}$ of low-degree partitions ($\overline{\rho} \in O(\log n)$). Here, we use ADG*, a slightly modified ADG that also records – as an array $\mathcal{G} \equiv [G(1)\ \ldots\ G(\overline{\rho})]$ – each low-degree partition. Then, we iterate over these partitions (starting from $\overline{\rho}$) and color each with SIM-COL ("SIMple coloring", Alg. 4). We discuss SIM-COL in more detail later in this section, its semantics are that it colors a given arbitrary graph G (in our context G is the ℓ-th partition $G(\ell)$) using $(1+\mu)\Delta$ colors, where $\mu > 0$ is an arbitrary value. To keep the coloring consistent with respect to already colored partitions,

```
1 /* Input: G(V, E) (input graph).
2  * Output: An array C, it assigns a color to each vertex. */
3 C = [0 0 ... 0] //Initialize an array of colors
4 //Run ADG* to derive a 2(1 + ε/12)-approximate degeneracy ordering
5 //G ≡ [G(1) ... G(ρ̄)] contains ρ̄ low-degree partitions
6 //We have G(i) = G[R(i)] where R(i) = {v ∈ V | ρ(v) = i}
7 (ρ, G) = ADG*(G)
8
9 //Initialize bitmaps B_v to track colors forbidden for each vertex
10 ∀_{v∈V} B_v = [00...0] //A bitmap has size ⌈2(1 + ε/4)(1 + μ)d⌉ + 1 bits
11 SIM-COL(G(ρ̄), {B_v | v ∈ R(ρ̄)}) //First, we color G(ρ̄)
12
13 for ℓ from ρ̄ − 1 down to 1 do: //For all low-degree partitions
14   Q = R(ρ̄) ∪ ··· ∪ R(ℓ + 1) //A union of already colored partitions
15   for all v ∈ R(ℓ) do in parallel:
16     for all u ∈ N_Q(v) do in parallel: // For v's colored neighbors
17       B_v = B_v ∪ C[u] // Update colors forbidden for v
18   SIM-COL(G(ℓ), {B_v | v ∈ R(ℓ)}) //Run Algorithm 4
```

Algorithm 3: **DEC-ADG**, the second proposed parallel coloring heuristic that provides a $(2(1+\varepsilon)d)$-coloring. Note that we use factors $\varepsilon/4$ and $\varepsilon/12$ for more straightforward proofs (this is possible as ε can be an arbitrary non-negative value).

```
1 /* Input: G(V(ℓ), E(ℓ)) (input graph partition), B_v (a bitmap with
2  * colors forbidden for each v). Output: color C[v] > 0. */
3 U = V(ℓ)
4 while U ≠ ∅ do:
5   //Part 1: all vertices in U are colored randomly:
6   for all v ∈ U do in parallel:
7     choose C[v] u.a.r. from {1, ..., (1 + μ)deg_ℓ(v)}
8   //Part 2: each vertex compares its color to its active neighbors.
9   //If the color is non-unique, the vertex must be re-colored.
10  for all v ∈ U do in parallel:
11    //f_eq(v, u) = (C[v] == C[u]) is the operator in Reduce.
12    if Reduce(N_U(v), f_eq) > 0 || C[v] ∈ B_v: C[v] = 0
13  //Part 3: Update B_v for all neighbors with fixed colors:
14  for all v ∈ U do in parallel:
15    for all u ∈ N_U(v) do in parallel:
16      if C[u] > 0: B_v = B_v ∪ C[u]
17  U = U \ {v ∈ U | C[v] > 0} //Update vertices still to be colored
```

Algorithm 4: **SIM-COL**, our simple coloring routine used by DEC-ADG. It delivers a $((1+\mu)\Delta)$-coloring, where $\mu > 0$ is an arbitrary value. When using SIM-COL as a subroutine in DEC-ADG, we instantiate μ as $\mu = \varepsilon/4$; we use this value in the listing above for concreteness.

we maintain bitmaps B_v that indicate colors already taken by v's neighbors in already colored partitions: If v cannot use a color c, the c-th bit in B_v is set to 1.

How large should B_v be to minimize storage overheads but also ensure that each vertex has enough colors to choose from? We observe that a single bitmap B_v should be able to contain at most as many colors as neighbors of v in a partition currently being colored ($G(\ell)$), and in all partitions that have already been colored ($G(\ell'), \ell' > \ell$). We denote this neighbor count with $deg_\ell(v)$. Observe that any $deg_\ell(v)$ is *at most* $kd = \lceil 2(1+\varepsilon)d \rceil$, as partitions are created according to a partial k-approximate degeneracy order where $k = 2(1+\varepsilon)$. Now, when coloring a partition $G(\ell)$, we know that SIM-COL, by its design, chooses colors for v only in the range of $\{1...(1+\mu)deg_\ell(v)+1\}$ (Algorithm 4, Line 7; as we will show, using such a range will enable the advantageous bounds for DEC-ADG). Thus, it suffices to keep bitmaps of size $\lceil(1+\mu)kd\rceil+1$ for each vertex, where $k = 2(1+\varepsilon/4)$.

In **SIM-COL**, we color a single low-degree partition $G(\ell) = (V(\ell), E(\ell))$. SIM-COL takes two arguments: (1) the partition to be colored (it can be an arbitrary graph $G = (V, E)$ but for clarity we explicitly use $G(\ell) = (V(\ell), E(\ell))$ that denotes a partition from a given iteration ℓ in DEC-ADG) and (2) bitmaps associated with vertices in a given partition $R(i)$. By design, SIM-COL delivers a $((1+\mu)\Delta)$-coloring; $\mu > 0$ can be an arbitrary value. However, in all the following discussions, for concreteness, we set $\mu = \varepsilon/4$; this value will enable deriving our final bounds for DEC-ADG. U are vertices still to be colored, initialized as $U = V(\ell)$. In each iteration, vertices in U are first colored randomly. Then, each vertex v compares its color $C[v]$ to the colors of its *active* (not yet colored) neighbors in N_U and checks if $C[v]$ is not already taken by other neighbors inside *and outside* of $V(\ell)$ (by checking B_v), see Lines 8–12. The goal is to identify *whether at least one such neighbor has the same color as* v. For this, we use $Reduce$ over $N_U(v)$ with the operator f defined as $f_{eq}(u) = (C[v] == C[u])$ (the "==" operator works analogously to the same-name operator in C++) and a simple lookup in B_v. If v and u have the same color, $f_{eq}(u)$ equals 1. Thus, if *any* of v's neighbors in U have the same color as v, $Reduce(N_U(v), f_{eq}) > 0$. This enables us to attempt to re-color v by setting $C[v] = 0$. If a vertex gets colored, we remove it from U (Line 17) and update the bitmaps of its neighbors (Line 16). We iterate until U is empty.

Depth, Work We now prove the time complexity of DEC-ADG. **The key observation** is that the probability that a particular vertex becomes *inactive* (permanently colored) is constant regardless of the coloring status of its neighbors. **The key proof technique** is to use Markov and Chernoff Bounds. We provide full proofs in the report (the link on page 1) and only sketch the intuition in the following.

Before we proceed with the analysis, we provide some definitions. For each round ℓ of SIM-COL (Algorithm 4), we define an indicator random variable X_v to refer to the event in which a vertex v gets removed from U (i.e., becomes colored and thus inactive) in this specific round ℓ. The vertex v is removed if and only if the color $C[v]$, which is selected on Line 7, is *not* used by some neighbor of v (i.e., this color is not in B_v) and no active neighbor chose $C[v]$ in this round. The random variable $\overline{X}_v$ indicates the complement of event X_v (i.e., a vertex v is not removed from U in a given round). Next, let Z be a Bernoulli random variable with probability $Pr[Z = 1] \equiv p = 1 - \frac{1}{1+\mu}$, and let $\overline{Z}$ be the complement of Z. In the following, we show that the event of an arbitrary vertex v becoming deactivated ($X_v = 1$) is at least as probable as $Z = 1$. This will enable us to use these *mutually independent* variables Z to analyze the time complexity of SIM-COL and DEC-ADG.

Claim 1. *In every iteration, for every vertex v, the probability that the vertex v becomes inactive is at least* $1 - \frac{1}{1+\mu}$.

Proof. The probability that v becomes inactive in any iteration ($Pr[X_v = 1]$) is at least $1 - \frac{i}{(1+\mu)deg_\ell(v)}$, where i is the number of distinct colors in B_v *and* received from neighbors in this round. This is because, in each iteration, while v connects to vertices with a total of i distinct colors, the total number of colors to be selected from is $(1+\mu)deg_\ell(v)$. Now, as v can have at most $deg_\ell(v)$ colored neighbors, we get $1 - \frac{i}{(1+\mu)deg_\ell(v)} \geq 1 - \frac{deg_\ell(v)}{(1+\mu)deg_\ell(v)} = 1 - \frac{1}{(1+\mu)}$, which shows that $Pr[X_v = 1] \geq Pr[Z = 1]$ holds for all active v. □

Thus, in expectation, a constant fraction of the vertices be-

comes inactive in every iteration. Now, in the next step, we will apply Markov and Chernoff bounds to an appropriately chosen binomial random variable, showing that the number of vertices that are removed is concentrated around its expectation. Hence, the algorithm terminates after $O(\log n)$ iterations.

First, to proof the intuitive fact that Z can be used to approximate the number of vertices removed in each round, we use the technique of *coupling* together with a handy equivalence between *stochastic dominance* and the coupling of two random variables [58], all details are in the report.

Lemma 6. *The random variable $X = \sum_{v \in U} X_v$ stochastically dominates $Y = \sum_{i=1}^{|U|} Z$.*

We also need an equivalent lemma for a complement event:

Lemma 7. *A random variable $\overline{X} = \sum_{v \in U} \overline{X}_v$ is stochastically dominated by a random variable $\overline{Y} = \sum_{i=1}^{|U|} \overline{Z}$.*

With these two lemmas and the Markov inequality, we show that with probability at least $1 - \frac{1}{\mu}$, at least $\frac{\mu}{1+\mu}|U|$ vertices are permanently colored in each round, giving:

Lemma 8. *SIM-COL performs $O(\log n)$ iterations w.h.p. (for constant $\mu > 0$).*

To bound the work of SIM-COL, we can observe that similarly to the number of vertices, the number of edges incident to at least one active vertex also decreases by a constant factor in each iteration with high probability. The work in an iteration is bounded by this number of edges and every iteration has depth $O(\Delta)$ (in the CREW setting) and $O(\log \Delta)$ (in the CRCW setting). Hence, we conclude:

Lemma 9. *SIM-COL takes $O(\Delta \log n)$ depth (in the CREW setting) or $O(\log \Delta \log n)$ depth (in the CRCW setting), and it has $O(n + m)$ work w.h.p in the CREW setting.*

Next, we turn our attention back to DEC-ADG. As DEC-ADG decomposes the edges of the input graph into $O(\log n)$ disjoint subgraphs of maximum degree $O(d)$, we get:

Lemma 10. *DEC-ADG takes $O(\log d \log^2 n)$ depth and $O(n + m)$ work w.h.p. in the CRCW setting.*

Coloring Quality Finally, we prove the coloring quality.

Claim 2. *DEC-ADG produces a $(2+\varepsilon)d$ coloring $0 < \varepsilon < 2$.*

Proof. Since we use ADG to partition the graph into $(\rho, \mathcal{G})$ ($\mathcal{G} \equiv [G(1)\ ...\ G(\overline{\rho})]$) on Line 7, we know that ρ is a partial $2(1 + \varepsilon/6)$-approximate degeneracy ordering. Therefore, we also know that each vertex $v \in G(i)$ has at most $2(1+\varepsilon/6)d$ neighbors in partitions $G(i')$ with $i' \geq i$. This implies, that if we run SIM-COL on each partition $G(i)$, we will color each partition with at most $(1 + \varepsilon/4)2(1 + \varepsilon/6)d$ colors, which is smaller or equal to $(2 + \varepsilon)d$ for $0 \leq \varepsilon \leq 2$, as $(1+\varepsilon/4)2(1+\varepsilon/6)d = 2 + \frac{10\varepsilon+\varepsilon^2}{12} \leq 2(1 + \varepsilon)$, for $\varepsilon \leq 2$. □

C. Enhancing Existing Coloring Algorithms

Finally, we illustrate that ADG does not only provide new provably efficient algorithms, *but also can be used to enhance existing ones*. For this, we seamlessly replace our default SIM-COL routine with a recent speculative coloring heuristic, ITR, by Çatalyürek et al. [23]. The result, **DEC-ADG-ITR**, is similar to DEC-ADG, except that the used SIM-COL differs in Line 7 from the default Algorithm 4: colors are not picked randomly, but we choose the smallest color not in B_v.

Using ADG enables deriving similar bounds on coloring quality $(2(1 + \varepsilon)d + 1)$ as before. However, deriving good bounds on work and depth is hard because the lack of randomization (when picking colors) prevents us from using techniques such as Chernoff bounds. We were still able to provide new results, detailed in a full report (also cf. Table III).

D. Comparison to Other Coloring Algorithms

We exhaustively compare JP-ADG and DEC-ADG to other algorithms in Table III. We consider: non-JP parallel schemes, the best sequential greedy schemes, and parallel JP algorithms. We consider depth (time), work, used model, quality, generality, randomized design, work-efficiency, and scalability. We also use past empirical analyses [14], [16], [20] and our results (§ VI) to summarize run-times and coloring qualities of algorithms used in practice, focusing on modern real-world graphs as input. Details are in the caption of Table III.

As explained in Section I, *only our algorithms work for arbitrary graphs, deliver strong bounds for depth and work and quality, and are often competitive in practice*. Now, JP-SL may deliver a little higher quality colorings, as it uses the *exact* degeneracy ordering (although without explicitly naming it) and its quality is (provably) $d+1$. However, JP-SL comes with much lower performance. On the other hand, most recent JP-LLF and JP-SLL only provide the straightforward $\Delta+1$ bound for coloring quality. These two are however inherently parallel as they depend linearly on $\log n$, while JP-ADG depends on $\log^2 n$. Yet, *JP-ADG has a different advantage in depth*: JP-SLL and JP-LLF depend linearly on $\sqrt{m}$ or Δ while JP-ADG on d. In today's graphs, d is usually much (i.e., orders of magnitude) smaller than $\sqrt{m}$ and Δ [36]. In the full report, we also provide a small lemma showing that $d/2 \leq \sqrt{m}$. This further illustrates that our bounds on depth in JP-ADG offer an interesting tradeoff compared to JP-LF and JP-LLF.

We finally observe that the design of ADG combined with the parametrization using ε enables a *tunable parallelism-quality tradeoff*. When $\varepsilon \to 0$, coloring quality in JP-ADG approaches $2d + 1$, only $\approx 2\times$ more than JP-SL. On the other hand, for $\varepsilon \to \infty$, ρ_{ADG} becomes irrelevant and the derived final ordering $\rho = \langle \rho_{\text{ADG}}, \rho_X \rangle$ converges to ρ_X. Now, X could be the random order R but also the low-depth LF and LLF orders based on largest degrees. This enables JP-ADG to increase parallelism *tunably*, depending on user's needs.

We also compare JP-ADG to works based on speculation and conflict resolution [21], [23], [26], [28], based on an early scheme by Gebremedhin [20]. A direct comparison is difficult because these schemes do no offer detailed theoretical investigations. Simple bounds on coloring quality, depth, and work are – respectively – $\Delta+1$, $O(\Delta I)$, and $O(\Delta I P)$, where I is #iterations and P is #processors. Here, we illustrate

GC Algorithm	Theory											Practice			Remarks
	Time (in PRAM) or Depth (in W–D), Work		Model	Quality	F	G	R	W	S	Q	Perf.	Quality	Code used	
Class 1: Parallel coloring algorithms *not based on JP* [9]. Many are not used in practice (except for work by Gebremedhin [20] and related ones [21], [23], [26], [28]); we include them for completeness of our analysis.														
(MIS) Alon [59]	$\mathbb{E}\,O(\Delta \log n)$	$O\left(m\Delta^2 \log n\right)$	CRCW	$\Delta+1$	👎	👍	👍	👎	👍†	👎	—	—	—	†Depth depends on Δ, $P = m\Delta$
(MIS) Goldberg [60]	$O\left(\Delta \log^4 n\right)$	$O\left((n+m)\log^4 n\right)$	EREW	$\Delta+1$	👍	👍	👎	👎	👍†	👎	—	—	—	†Depth depends on Δ, $P = (m+n)/\Delta$
(MIS) Goldberg [61], [62]	$O(\log^* n)$	$O(n \log^* n)$	EREW	$\Delta+1$	👍	👎†	👍†	👍†	👍	👎	—	—	—	†Graphs with $\Delta \in O(1)$. $P = n$.
(MIS) Goldberg [62]	$O(\Delta \log \Delta(\Delta + \log^* n))$	$O((n+m)\Delta \log \Delta(\Delta + \log^* n))$	EREW	$\Delta+1$	👍	👍	👎	👎	👍†	👎	—	—	—	†Depth depends on Δ. $P = n + m$.
Luby [63]	$O\left(\log^3 n \log\log n\right)$	$O\left((n+m)(\log^3 n \log\log n)\right)$	CREW	$\Delta+1$	👍	👍	👎	👎	👍	👎	—	—	—	$P = n + m$
(MIS) Luby [13] (as implemented in ColPack)	$\mathbb{E}\,O(\Delta \log n)$	$O(m\Delta \log n)$	CRCW	$\Delta+1$	👎	👍	👎	👎	👍†	👎	★★☆☆☆	★★☆☆☆	ColPack	†Depth depends on Δ, $P = m$
Gebremedhin [20] (as implemented in ColPack)	$\mathbb{E}\,O\left(\frac{\Delta n}{P}\right)$	$O(\Delta n)$	CREW	—	👍	👍	👎	👎	👎	👎	★★☆☆☆	★★☆☆☆	ColPack	Assuming $P \leq \frac{n}{2\sqrt{m}}$.
Gebremedhin [20] (as implemented in ColPack)	$\mathbb{E}\,O\left(\frac{\Delta P m}{n}\right)$	$O\left(\frac{\Delta P^2 m}{n}\right)$	CREW	—	👍	👍	👍	👍†	👎	👎	★★☆☆☆	★★☆☆☆	ColPack	Assuming $P > \frac{n}{2\sqrt{m}}$. †Work can be $\Omega(n+m)$ (for some P).
ITRB (Boman et al. [21]) (as implemented in Zoltan)	$O(\Delta \cdot I)$ ✪	$O(\Delta \cdot I \cdot P)$ ✪	❓	—	👍	👍	👎	👎†	❓†	👎	★☆☆☆☆	★★★½☆	Zoltan	†No detailed bounds available
ITR (Çatalyürek et al. [23] and others [26], [28])	$O(\Delta \cdot I)$ ✪	$O(\Delta \cdot I \cdot P)$ ✪	❓	—	👍	👍	👎	👎†	❓†	👎	★★★★☆	★★½☆☆	ColPack	†No detailed bounds available
ITR-ASL (Patwary et al. [15])	$O(n \cdot I)$ ✪	$O(n \cdot I \cdot P)$ ✪	❓	—	👍	👍	👎	👎†	❓†	👎	★★★☆☆	★★☆☆☆	ColPack	†No detailed bounds available
Class 2: Coloring algorithms that are *not parallel* and *based on the Greedy coloring scheme* [8]. We include them as comparison baselines that deliver best-known coloring quality in practice.														
Greedy-ID [1]	$O(n+m)$	$O(n+m)$	Seq.	$\Delta+1$	👍	👍	👎	👎	👎	👎	★☆☆☆☆	★★★★½	ColPack, GBBS	—
Greedy-SD [10], [14]	$O(n+m)$	$O(n+m)$	Seq.	$\Delta+1$	👍	👍	👎	👎	👎	👎	★☆☆☆☆	★★★★★	ColPack, GBBS	—
Class 3: *Parallel* heuristics that constitute the largest line of work into parallel graph coloring and are fast in theory and practice. Most are based on JP [9].														
JP-FF [8], [14]	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	W–D	$\Delta+1$	👎	👍	👎	👍	👎†	👎	★★★★½	☆☆☆☆☆	ColPack, GBBS	†No general bounds
JP-LF [14]	No general bounds; $\Omega(\Delta^2)$ for some graphs	$O(n+m)$	W–D	$\Delta+1$	👎	👍	👍	👎†	👎	👎	★★★★½	★★☆☆☆	Col., GBBS, HP	†No general bounds
JP-SL [14]	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	W–D	$d+1$	👎	👍	👍	👎	👍‡	👎	★☆☆☆☆	★★★★½	Col., GBBS, HP	†No general bounds. ‡Often, $d \ll \Delta$.
JP-R [9]	$\mathbb{E}\,O\left(\frac{\log n}{\log\log n}\right)$	$O(n+m)$	W–D	$\Delta+1$	👎	👎†	👍	👍	👍	👎	★★★★½	★☆☆☆☆	Col., GBBS, HP	†Graphs with $\Delta \in O(1)$
JP-R [14]	$\mathbb{E}\,O\left(\log n + \log\Delta \cdot \min\left\{\sqrt{m}, \Delta + \frac{\log\Delta \log n}{\log\log n}\right\}\right)$	$O(n+m)$	W–D	$\Delta+1$	👎	👍	👍	👍	👍†	👎	★★★★½	★☆☆☆☆	Col., GBBS, HP	†Depth depends on $\sqrt{m}$ or Δ
JP-LLF [14]	$\mathbb{E}\,O\left(\log n + \log\Delta \cdot \left(\min\left\{\Delta, \sqrt{m}\right\} + \frac{\log^2\Delta \log n}{\log\log n}\right)\right)$	$O(n+m)$	W–D	$\Delta+1$	👎	👍	👍	👍	👍†	👎	★★★★½	★★★☆☆	GBBS, HP	†Depth depends on $\sqrt{m}$ or Δ
JP-SLL [14]	$\mathbb{E}\,O\left(\log\Delta \log n + \log\Delta \cdot \left(\min\left\{\Delta, \sqrt{m}\right\} + \frac{\log^2\Delta \log n}{\log\log n}\right)\right)$	$O(n+m)$	W–D	$\Delta+1$	👎	👍	👍	👍	👍†	👎	★★½☆☆	★★★★☆	GBBS, HP	†Depth depends on $\sqrt{m}$ or Δ
JP-ASL [15], [30]	$O(n \cdot I)$ ✪	$O(n \cdot I \cdot P)$ ✪	W–D	$\Delta+1$	👎	👍	👍	❓†	👍†	👎	★☆☆☆☆	★☆☆☆☆	ColPack	†No detailed bounds available
JP-ADG [This Paper]	$\mathbb{E}\,O\left(\log^2 n + \log\Delta \cdot \left(d\log n + \frac{\log d \cdot \log^2 n}{\log\log n}\right)\right)$	$O(n+m)$	W–D	$2(1+\varepsilon)d+1$	👎	👍	👍	👍	👍†	👍†	★★½☆☆	★★★★☆	[This work]	†Often, $d \ll \Delta$
JP-ADG-M [This Paper] (a variant described in § V)	$\mathbb{E}\,O\left(\log^2 n + \log\Delta \cdot \left(d\log n + \frac{\log d \cdot \log^2 n}{\log\log n}\right)\right)$	$O(n+m)$	W–D	$4d+1$	👎	👍	👍	👍	👍†	👍†	★★½☆☆	★★★★☆	[This work]	†Often, $d \ll \Delta$
DEC-ADG [This Paper]	$O\left(\log d \log^2 n\right)$ w.h.p.	$O(n+m)$ (w.h.p.)	W–D	$(2+\varepsilon)d$	👍*	👍	👍	👍†	👍	👍	★★☆☆☆	★★★★☆	[This work]	*Using CREW gives $\mathbb{E}O(nd+m)$ work. †Work efficient in expectation.
DEC-ADG-M [This Paper] (a variant described in § V)	$O\left(\log d \log^2 n\right)$ w.h.p.	$O(n+m)$ (w.h.p.)	W–D	$(4+\varepsilon)d$	👍*	👍	👍	👍†	👍	👍	★★☆☆☆	★★★★☆	[This work]	*Using CREW gives $\mathbb{E}O(nd+m)$ work. †Work efficient in expectation.
DEC-ADG-ITR [This Paper]	$O\left(I \cdot d\log n\right)$, [work bound is complex, details in text (§ IV-C)]		W–D	$2(1+\varepsilon)d+1$	👍	👍	👎	👎	👎	👍	★★★☆☆	★★★★☆	[This work]	I is the number of iterations in [23]

TABLE III: **Comparison of parallel graph coloring algorithms. "Greedy-X"** is a the Greedy sequential coloring scheme [8], with ordering X. **"JP-X"** is a Jones and Plassmann scheme [9], with ordering X. **"(MIS)"** indicates that a given algorithm solves the Minimum Independent Set (MIS) problem, but it can be easily converted into a parallel GC algorithm. **"EREW, CREW, and CRCW"** are well-known variants of the PRAM model. **"Performance"** and **"Quality"** summarize the run-times and coloring qualities, based on the extensive existing evaluations [14], [16], [20] and our analysis (§ VI), focusing on modern real-world graphs as input sets (we exclude Class 1 as it is not relevant for practical purposes). **"Seq."** is a simple sequential (RAM) model. **"W–D"** indicates work-depth. **"F. (Free)?":** Is the heuristic provably free from using concurrent writes? **"G. (General)?":** Does the algorithm work for general graphs? **"R. (Randomized)?":** Is the algorithm randomized? **"W. (Work-efficient)?":** Is the algorithm provably work-efficient (i.e., does it take $O(n+m)$ work)? **"S. (Scalable)?":** Is the algorithm provably scalable (i.e., does it take $O(a \log^b n)$, where $a \ll n$ and $b \in \mathbb{N}$)? **"Q. (Quality)?":** Does the algorithm provably ensure the coloring quality better than the trivial $\Delta+1$ bound? **Code used:** source code evaluated in this work; respective repositories are ColPack [30], [31], Zoltan [18], [32]–[35], original code by Hasenplaugh et al. (HP) [14], and Graph Based Benchmark Suite with Ligra [36]–[38]. "👍": full support, "👍": partial support, "👎": no support, "❓": unknown, "✪": bounds based on our best-effort derivation; I is the number of iterations. "**w.h.p.**": with high probability. All symbols are explained in Table I. ‡ $\log^* n$ grows very slowly. *The proposed schemes are the only ones with provably good work **and** depth **and** quality.*

that using ADG in combination with these works simplifies deriving better bounds for such algorithms, as seen by the example of DEC-ADG-ITR, see § IV-B.

V. Optimizations and Implementation

Here, the main driving question that we followed was *how to maximize the practical performance of the proposed coloring algorithms while maintaining all the theoretical guarantees*? All theorems, proofs, and detailed listings are in the full report.

Representation of U and R The first key optimization (ADG, Alg. 1) is to maintain set U (vertices still to be assigned the rank ρ) and sets $R(\cdot)$ (vertices removed from U) *together in the same contiguous array* such that all elements in $R(\cdot)$ *precede* all elements in U. In iteration i, this gives an array $[R(1)\ \ldots\ R(i)\ \texttt{index}\ U]$, where `index` points to the first element of U (initially, `index` is 0). Each $R(i)$ is kept sorted by (increasing) vertex degrees. On one hand, this requires sorting $R(i)$ at every iteration, which we tackle with linear time integer sort. The benefit – among others – is that removing $R(i)$ from U (Line 12) only takes $O(1)$ time by simply moving the `index` pointer by $|R|$ positions "to the right".

Linear Sorting We additionally explored different schemes for fast parallel integer sort used to maintain the above-described representation of $U \cup R$. We tried different algorithms (radix sort [64], counting sort [65], and quicksort [66]).

Combining JP and ADG We observe that Part 1 of JP-ADG (Lines 6–9, Algorithm 2), where one derives predecessors and successors in a given ordering to construct the DAG G_ρ, can also be implemented as a part of UPDATE in ADG, in Algorithm 1. To maintain the output semantics of ADG, we calculate a new priority function ρ' that simultaneously specifies the needed DAG structure. For each $v \in V$, ρ' is defined as the number of neighbors $u \in N(v)$, for which $\rho(u) > \rho(v)$ holds. There is no change in theory results.

Median We also use degree median instead of degree average in ADG, to derive R: $R = \{u \in U \mid D[u] \leq (1+\varepsilon)\delta_m\}$, where δ_m is median of degrees of vertices in U. This enables minor speedups for some graphs.

Push vs. Pull Computing ρ' can be implemented either in the push or the pull style (pushing updates to a shared state or pulling updates to a private state) [19]. We analyzed with both and found that, while pushing needs atomics, pulling incurs more work. Both options ultimately give similar performance.

Infrastructure Details We integrated our algorithms with GBBS [36], a recent platform for testing parallel graph algorithms. We use OpenMP [67] for parallelization.

VI. Evaluation

In evaluation, *we found that empirical results follow theoretical predictions*, already scrutinized in Table III and Section IV.

Fig. 1: **Run-times (1st and 3rd columns) and coloring quality (2nd and 4th columns)**. Two plots next to each other correspond to the same graph. Graphs are representative (other results follow similar patterns). Parametrization: 32 cores (all available), $\varepsilon = 0.01$, sorting: Radix sort, direction-optimization: push, JP-ADG variant based on average degrees $\widehat{\delta}$. SL and SLL are excluded from run-times in the right column (for larger graphs) because they performed consistently worse than others. We exclude DEC-ADG for similar reasons and because it is of mostly theoretical interest; instead, we focus on DEC-ADG-ITR, which is based on core design ideas in DEC-ADG. Numbers in bars for color counts are numbers of used colors. **"SC"**: results for the class of algorithms based on **speculative coloring** (ITR, DEC-ADG-ITR). **"JP"**: results for the class of algorithms based on the **Jones and Plassman** approach (**color scheduling**, JP-*). A vertical line in each plot helps to separate these two classes of algorithms. DEC-ADG-ITR uses dynamic scheduling. JP-ADG uses linear time sorting of R. Any schemes that are always outperformed in a given respect (e.g., Zoltain in runtimes or ColPack in qualities) are excluded from the plots.

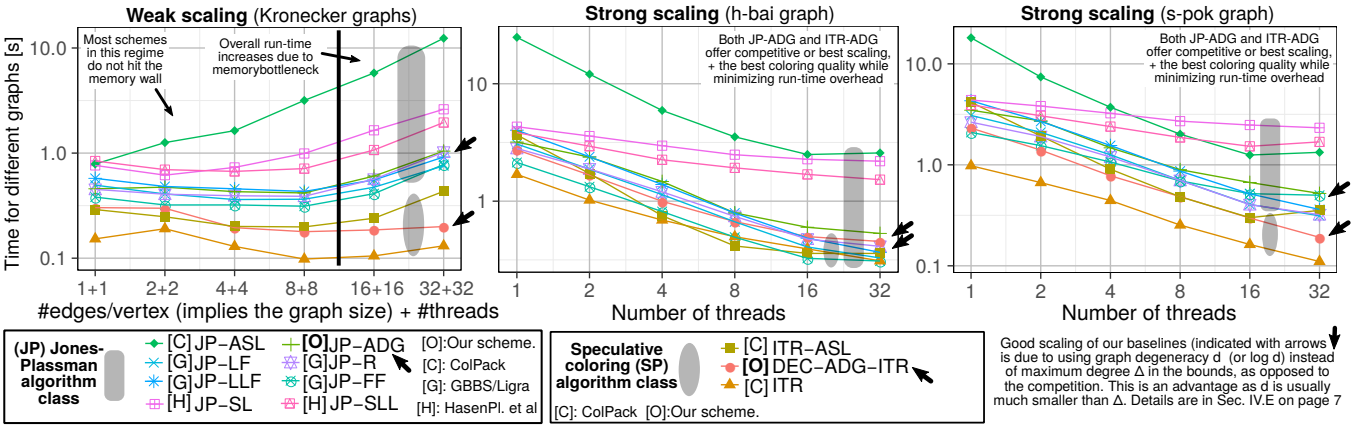

Fig. 2: **Weak and strong scaling**. Graphs are representative (other results follow similar patterns). Parametrization: $\varepsilon = 0.01$, sorting: Radix sort, direction-optimization: push, JP-ADG variant based on average degrees $\widehat{\delta}$. DEC-ADG-ITR uses dynamic scheduling. JP-ADG uses linear time sorting of R. In weak scaling, we use $n = 1$M vertices.

Thus, for brevity, we now summarize the most important observations. A comprehensive comparison of run-times and coloring qualities of different algorithms is in Table III (together with abbreviations of used comparison baselines).

Used Architectures In the first place, we use **Einstein**, an in-house Dell PowerEdge R910 server with an Intel Xeon X7550 CPUs @ 2.00GHz with 18MB L3 cache, 1TiB RAM, and 32 cores per CPU (grouped in four sockets). We also conducted experiments on **Ault** (a CSCS server with Intel Xeon Gold 6140 CPU @ 2.30GHz, 768 GiB RAM, 18 cores, and 24.75MB L3) and **Fulen** (a CSCS server with Intel Skylake @ 2GHz, 1.8 TiB RAM, 52 cores, and 16MB L3).

Methodology We provide absolute runtimes when reporting speedups. In our measurements, we exclude the first measured 1% of performance data as warmup. We derive enough data to obtain the mean and 95% non-parametric confidence intervals. Data is summarized with arithmetic means.

Algorithms & Datasets We focus on modern heuristics from Table III. For each scheme, we always pick the most competitive implementation (i.e., fewest colors used and smallest performance overheads), selecting from existing repositories (**ColPack** [30], [31], **Zoltan** [18], [32]–[35], original code by **Hasenplaugh et al. (HP)** [14], **GBBS with Ligra** [36]–[38]), and our implementation. Detailed parametrizations are in the reproducibility appendix. We use real-world graphs from SNAP [68], KONECT [69], DIMACS [70], and WebGraph datasets [71]; see Table IV for details. We analyze synthetic power-law graphs (generated with the Kronecker model [72]). *This gives a **large evaluation space**; we only summarize selected findings.*

The results are in Figure 1. Following past analyses [14], we consider *separately* two distinctive families of algorithms: those based on **speculative coloring (SC)**, and the ones with the **Jones and Plassman** structure (**color scheduling**). These two classes of algorithms – especially for larger datasets – are often *complementary*, i.e., whenever one class achieves lower performance, the other thrives, and vice versa. This is especially visible for larger graphs, such as h-dsk, h-wdb, or s-gmc. The reason is that the structure of some graphs (e.g., with dense clusters) entails many coloring conflicts which may need many re-coloring attempts, giving long tail run-times.

Summary Our algorithms almost always offer superior coloring quality. Only JP-SL, JP-SLL (HP), and sometimes ITRB by Boman et al. [21] (Zoltan) use comparably few colors, but they are at least 1.5× and 2× slower, respectively. Simultaneously, run-times of our algorithms are comparable or marginally higher than the competition (in the class of algorithms with speculative coloring) and within at most 1.3-1.4× of the competition (in the class of JP baselines). Thus, *we offer the best coloring quality at the smallest required run-time overhead*. Finally, our routines are the only ones with theoretical guarantees on work, depth, and quality.

Run-Times with Full Parallelism We analyze **run-times** using all the available cores. Whenever applicable, we show fractions due to **reordering** (preprocessing, e.g., the "ADG" phase in JP-ADG) and the actual **coloring** (e.g., the "JP" phase in JP-ADG). JP-SL, JP-SLL (HP), and JP-ASL (ColPack) are the slowest as they offer least parallelism. JP-LF, JP-LLF, and JP-R (GBBS/Ligra) are very fast, as their depth is in $O(\log n)$. We also analyze speculative coloring from ColPack and Zoltan; we summarize the most competitive variants. ITR does not come with clear bounds on depth, but its simple and parallelizable structure makes it very fast. ITRB schemes are >2× slower than other baselines and are thus excluded from run-time plots. We also consider an additional variant of ITR based on ASL [30], ITR-ASL. In several cases, it approaches the performance of ITR.

The coloring run-times of JP-ADG are comparable to JP-LF, JP-LLF, and others. This is not surprising, as this phase

Friendships: Friendster (**s-frs**, 64M, 2.1B), Orkut (**s-ork**, 3.1M, 117M), LiveJournal (**s-ljn**, 5.3M, 49M), Flickr (**s-flc**, 2.3M, 33M), Pokec (**s-pok**, 1.6M, 30M), Libimseti.cz (**s-lib**, 220k, 17M), Catster/Dogster (**s-cds**, 623k, 15M), Youtube (**s-you**, 3.2M, 9.3M), Flixster (**s-flx**, 2.5M, 7.9M),
Hyperlink graphs: GSH domains (**h-dgh**, 988M, 33.8B), SK domains (**h-dsk**, 50M, 1.94B), IT domains (**h-dit**, 41M, 1.15B), Arabic domains (**h-dar**, 22M, 639M), Wikipedia/DBpedia (en) (**h-wdb**, 12M, 378M), Indochina domains (**h-din**, 7.4M, 194M), Wikipedia (en) (**h-wen**, 18M, 172M), Wikipedia (it) (**h-wit**, 1.8M, 91.5M), Hudong (**h-hud**, 2.4M, 18.8M), Baidu (**h-bai**, 2.1M, 17.7M), DBpedia (**h-dbp**, 3.9M, 13.8M),
Communication: Twitter follows (**m-twt**, 52.5M, 1.96B), Stack Overflow interactions (**m-stk**, 2.6M, 63.4M), Wikipedia talk (en) (**m-wta**, 2.39M, 5.M),
Collaborations: Actor collaboration (**l-act**, 2.1M, 228M), DBLP co-authorship (**l-dbl**, 1.82M, 13.8M), Citation network (patents) (**l-cit**, 3.7M, 16.5M), Movie industry graph (**l-acr**, 500k, 1.5M)
Various: UK domains time-aware graph (**v-euk**, 133M, 5.5B), Webbase crawl (**v-wbb**, 118M, 1.01B), Wikipedia evolution (de) (**v-ewk**, 2.1M, 43.2M), USA road network (**v-usa**, 23.9M, 58.3M), Internet topology (Skitter) (**v-skt**, 1.69M, 11M),

TABLE IV: Considered real graphs from established datasets [68]–[71]. Graph are sorted by m in each category. For each graph, we show its "(**symbol used**, n, m)".

is dominated by the common JP skeleton (with some minor differences from varying schedules of vertex coloring). However, the reordering run-time in JP-ADG comes with certain overheads because it depends on $\log^2 n$. This is expected, as JP-ADG – by its design – performs several sequential iterations, the count of which is determined by ε (i.e., how well the degeneracy order is approximated). Importantly, JP-ADG is consistently faster (by more than 1.5×) than JP-SL and JP-SLL that also focus on coloring quality.

DEC-ADG-ITR – similarly to JP-ADG – entails ordering overheads because it precomputes the ADG low-degree decomposition. However, total run-times are only marginally higher, and in several cases *lower* than those in ITR. This is because the low-degree decomposition that we employ, despite enforcing some sequential steps in preprocessing, *reduces counts of coloring conflict*, translating to performance gains.

Coloring Quality Coloring quality also follows the theoretical predictions: JP-SL outperforms JP-SLL, JP-LF, and JP-LLF (by up to 15%), as it strictly follows the degeneracy order. Overall, all four schemes (GBBS/Ligra, HP) are competitive. As expected, JP-FF and JP-R come with *much* worse coloring qualities because they do not focus on minimizing color counts. As observed before [14], ITR (ColPack) outperforms JP-FF and JP-R but falls behind JP-LF, JP-LLF, JP-SLL, and JP-SL. JP-ASL and ITR-ASL (ColPack) offer low (often the lowest) quality. ITRB (Zoltan) sometimes approaches the quality of JP-SL, JP-SLL, DEC-ADG-ITR, and JP-ADG.

The coloring quality of our schemes outperforms others in almost all cases. Only JP-SL, JP-SLL (GBBS/Ligra, HP), and sometimes ITRB (Zoltan) are competitive, but they are always much slower. In some cases, JP-ADG (e.g., in s-ork) and DEC-ADG-ITR (e.g., in s-gmc) *are better than JP-SL and JP-SLL* (by 3-10%). Hence, while the strict degeneracy order is in general beneficial when scheduling vertex coloring, it does *not* always give best qualities. JP-ADG consistently outperforms others, reducing used color counts by even up to 23% compared to JP-LLF (for m-wta). Finally, DEC-ADG-ITR always ensures much better quality than ITR, up to 40% (for s-lib). Both DEC-ADG-ITR and JP-ADG offer similarly high coloring qualities across all comparison targets.

Strong Scaling We also investigate strong scaling (i.e., run-times for the increasing thread counts). Relative performance differences between baselines do not change significantly, except for SLL that becomes more competitive when the thread count approaches 1, due to the tuned sequential implementation that we used [14]. Representative results are in Figure 2; all other graphs result in analogous performance patterns. Most variants from ColPack, Zoltan, GBBS/Ligra, and HP scale well (we still exclude Zoltan due to high runtimes). Importantly, scaling of our baselines is also *advantageous* and comparable to others. This follows theoretical predictions, as the $\log^2 n$ factor in our depth bounds is alleviated by the presence of the degeneracy d (or $\log d$) instead of Δ, as opposed to the competition; see § IV-D on page 7 for details.

Weak Scaling Weak scaling is also shown in Figure 2. We use Kronecker graphs [72] of the increasing sizes by varying the number of edges/vertex; this *fixes the used graph model*. JP-ADG scales comparably to other JP baselines; DEC-ADG-ITR scales comparably or better than ITR or ITR-ASL.

Impact of ε Representative results of the impact of ε are in Fig. 3. As expected, larger ε offers more parallelism and thus lower runtimes, but coloring qualities might decrease. Still, the decrease is *minor*, and the qualities remain the highest or competitive across almost the whole spectrum of ε.

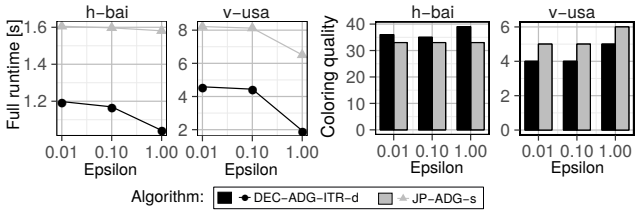

Fig. 3: **Impact of ε on run-times and coloring quality**. Parameters: 32 cores, sorting: Radix sort, direction-optimization: push, JP-ADG variant based on average degrees $\widehat{\delta}$. DEC-ADG-ITR uses dynamic scheduling. JP-ADG uses linear time sorting of R.

Memory Pressure We also investigate the pressure on the memory bus, see Figure 4. For this, we use PAPI [73] to gather data about idle CPU cycles and L3 cache misses. Low ratios of L3 misses or idle cycles indicate high locality and low pressure on the memory bus. Overall, our routines have comparable or best ratios of active cycles and L3 hits.

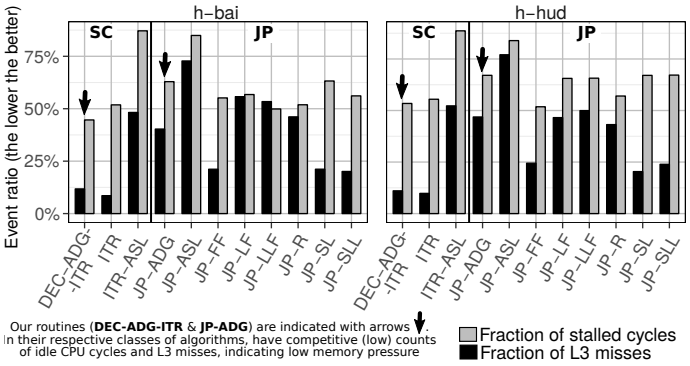

Fig. 4: **Fractions of L3 misses (out of all L3 accesses) and idle (stalled) CPU cycles (out of all CPU cycles) in each algorithm execution**. Parametrization: graph h-hud, 32 cores, sorting: Radix sort, direction-optimization: push, JP-ADG uses average degrees $\widehat{\delta}$. DEC-ADG-ITR uses dynamic scheduling. JP-ADG uses linear time sorting of R.

Performance Profiles We also summarize the results from Figure 1 using *performance profiles* [74], see Figure 5 for a representative profile for coloring quality. Details on using performance profiles are in the extended report due to space constraints; intuitively, such a profile shows cumulative distributions for a selected performance metric (e.g., a color count). The summary in Figure 5 confirms the previous insights: DEC-ADG-ITR, JP-ADG, and JP-SL offer the best colorings.

Additional Analyses We also vary push vs. pull and other parameters from Section V; this does not impact the previous insights (details are in the extended report).

VII. Related Work

We already exhaustively analyzed a large body of **sequential & parallel graph coloring heuristics** [1]–[6], [8]–[14], [16], [20], [59]–[63], [75] in Section I and in Table III. Almost all of them have theoretical guarantees based on the work-depth [76]

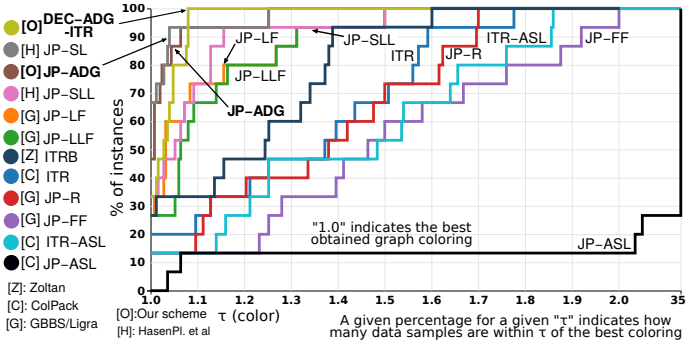

Fig. 5: **Color qualities from Figure 1 summarized with a performance profile**.

or the PRAM model [77]. *We build on and improve on these works in several dimensions, as explained in detail in Section I.*

Many works exist in the **theory of distributed graph coloring** based on models such as LOCAL or CONGESTED-CLIQUE [78]–[92]. These algorithms are highly theoretical and do not come with any implementations. Moreover, they come with assumptions that are unrealistic in HPC settings, for example distributed LOCAL and CONGEST algorithms do not initially know the interconnection graph, or the message size in LOCAL algorithms can be unbounded. Finally, they cannot be directly compared to Work-Depth or PRAM algorithms. Thus, they are *of little relevance to our work*.

Next, many **practical parallel and distributed approaches** have recently been proposed. They often use different *speculative schemes* [17]–[29], where vertices are colored speculatively and potential conflicts are resolved in a second pass. Some of these schemes were implemented within frameworks or libraries [30], [93]–[95]. Another line of schemes incorporates *GPUs and vectorization* [24], [27], [28], [96]–[100]. Other schemes use *recoloring* [101], [102] in which one improves an already existing coloring. Patidar and Chakrabarti use *Hadoop* to implement graph coloring [103]. Alabandi et al. [104] illustrate how to *increase parallelism* of coloring heuristics. *These works are orthogonal to this paper*: they do not provide theoretical analyses, but they usually offer numerous architectural and design optimizations that can be combined with our algorithms for further performance benefits. As we focused on theoretical guarantees and its impact on performance, and *not* on architecture-related optimizations, *we leave integration with these optimizations as future work.*

There are works on **coloring specific graph classes**, such as planar graphs [105]–[108]. Some works impose **additional restrictions**, for example coloring *balance*, which limits differences between numbers of vertices with different colors [109]–[111]. Other lines of related work also exist, for example on **edge coloring** [112], **dynamic** or **streaming coloring** [113]–[118], k-distance-coloring and other generalizations [119]–[121], and **sequential exact coloring** [122]–[124]. There are even works on solving graph coloring with evolutionary and genetic algorithms [125]–[127] and with machine learning methods [128]–[131]. All these works are unrelated as *we focus on unrestricted, parallel, and 1-distance vertex coloring with provable guarantees on performance and quality*, targeting *general, static, and simple graphs*.

The **general structure of our ADG algorithm**, based on iteratively removing vertices with degrees in certain ranges defined by the approximation parameter ε, was also used to solve other problems, for example the $(2+\varepsilon)$-approximate maximal densest subgraph algorithms by Dhulipala et al. [36]. Finding more applications of ADG is left for future work.

We note that, while our ADG scheme is *the first parallel algorithm for deriving approximate degeneracy ordering with a provable approximation factor*, two algorithms in the *streaming setting* exist [132], [133].

Graph coloring has been targeted in several recent works related to **broad graph processing paradigms, abstractions, and frameworks** [19], [134]–[141]. Several HPC works [142]–[144] consider distributed graph coloring in the context of high-performance RDMA networks and RMA programming [145]–[150]. Different coloring properties of graphs were also analyzed in the context of **graph compression** and **summarization** [151], [152].

VIII. Conclusion

We develop graph coloring algorithms with strong theoretical guarantees on all three key aspects of parallel graph coloring: work, depth, and coloring quality. No other existing algorithm provides such guarantees.

One algorithm, JP-ADG, is often superior in coloring quality to all other baselines, even including the tuned SL and SLL algorithms specifically designed to reduce counts of used colors [14]. It also offers low run-times for different input graphs. As we focus on algorithm design and analysis, one could combine JP-ADG with many orthogonal optimizations, for example in the GPU landscape, to achieve more performance without sacrificing quality. Another algorithm, DEC-ADG, is of theoretical interest as it is the first routine – in a line of works based on speculative coloring – with strong theoretical bounds. While being less advantageous in practice, we use its underlying design to enhance a recent coloring heuristic [23] obtaining DEC-ADG-ITR, an algorithm with (1) strong quality bounds and (2) competitive performance, for example up to 40% fewer colors used then compared to the base design [23].

Our algorithms use a very simple (but rich in outcome) idea of provably relaxing the strict vertex degeneracy order, to maximize parallelism when deriving this order. This idea, and our corresponding parallel ADG algorithm, are of separate interest, and could enhance other algorithms that rely on vertex ordering, for example in mining maximal cliques [39], [40].

We provide the most extensive theoretical study of parallel graph coloring algorithms. This analysis can be used by other researchers as help in identifying future work.

Acknowledgements: We thank Hussein Harake, Colin McMurtrie, Mark Klein, Angelo Mangili, and the whole CSCS team granting access to the Ault and Daint machines, and for their excellent technical support. We thank Timo Schneider for immense help with computing infrastructure at SPCL, and Jakub Czajka for help in the early project stage.

REFERENCES

[1] T. F. Coleman and J. J. Moré, "Estimation of sparse jacobian matrices and graph coloring problems," *SIAM Journal on Numerical Analysis*, vol. 20, no. 1, pp. 187–209, 1983.

[2] M. T. Jones and P. E. Plassmann, "Scalable iterative solution of sparse linear systems," *Parallel Computing*, vol. 20, no. 5, pp. 753–773, 1994.

[3] A. H. Gebremedhin, F. Manne, and A. Pothen, "What color is your jacobian? graph coloring for computing derivatives," *SIAM Review*, vol. 47, no. 4, pp. 629–705, 2005.

[4] E. M. Arkin and E. B. Silverberg, "Scheduling jobs with fixed start and end times," *Discrete Applied Mathematics*, vol. 18, no. 1, pp. 1–8, 1987.

[5] D. Marx, "Graph colouring problems and their applications in scheduling," *Periodica Polytechnica Electrical Engineering (Archives)*, vol. 48, no. 1-2, pp. 11–16, 2004.

[6] R. Ramaswami and K. Parhi, "Distributed scheduling of broadcasts in a radio network," in *Proceedings of the Eighth Annual Joint Conference of the IEEE Computer and Communications Societies*, ser. IEEE INFOCOM '89. IEEE, 1989.

[7] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified NP-complete problems," in *Proceedings of the sixth annual ACM symposium on Theory of computing*, ser. STOC'74. ACM, 1974, pp. 47–63.

[8] D. J. A. Welsh and M. B. Powell, "An upper bound for the chromatic number of a graph and its application to timetabling problems," *The Computer Journal*, vol. 10, no. 1, pp. 85–86, 1967.

[9] M. T. Jones and P. E. Plassmann, "A parallel graph coloring heuristic," *SIAM Journal on Scientific Computing*, vol. 14, no. 3, pp. 654–669, 1993.

[10] D. Brélaz, "New methods to color the vertices of a graph," *Communications of the ACM*, vol. 22, no. 4, 1979.

[11] D. W. Matula and L. L. Beck, "Smallest-last ordering and clustering and graph coloring algorithms," *Journal of the ACM*, vol. 30, no. 3, pp. 417–427, 1983.

[12] R. M. Karp and W. Avi, "A fast parallel algorithm for the maximal independent set problem," *JACM*, vol. 32, no. 4, pp. 762–773, 1985.

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

[14] W. Hasenplaugh, T. Kaler, T. B. Schardl, and C. E. Leiserson, "Ordering heuristics for parallel graph coloring," in *Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures*, ser. SPAA'14. ACM, 2014, pp. 166–177.

[15] M. M. A. Patwary, A. H. Gebremedhin, and A. Pothen, "New multithreaded ordering and coloring algorithms for multicore architectures," in *Euro-Par 2011 Parallel Processing*, E. Jeannot, R. Namyst, and J. Roman, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 250–262.

[16] J. Allwright, R. Bordawekar, D. Coddington, K. Dincer, and C. L. Martin, "A comparison of parallel Graph coloring algorithms," 1995.

[17] Ü. V. Çatalyürek, F. Dobrian, A. Gebremedhin, M. Halappanavar, and A. Pothen, "Distributed-memory parallel algorithms for matching and coloring," in *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*. IEEE, 2011, pp. 1971–1980.

[18] D. Bozdağ, A. H. Gebremedhin, F. Manne, E. G. Boman, and U. V. Catalyurek, "A framework for scalable greedy coloring on distributed-memory parallel computers," *Journal of Parallel and Distributed Computing*, vol. 68, no. 4, pp. 515–535, 2008.

[19] M. Besta, M. Podstawski, L. Groner, E. Solomonik, and T. Hoefler, "To push or to pull: On reducing communication and synchronization in graph computations," in *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*, 2017, pp. 93–104.

[20] A. H. Gebremedhin and F. Manne, "Scalable parallel graph coloring algorithms," *Concurrency: Practice and Experience*, vol. 12, no. 12, pp. 85–120, 2000.

[21] E. G. Boman, D. Bozdağ, U. Catalyurek, A. H. Gebremedhin, and F. Manne, "A scalable parallel graph coloring algorithm for distributed memory computers," in *European Conference on Parallel Processing*. Springer, 2005, pp. 241–251.

[22] A. H. Gebremedhin, I. G. Lassous, J. Gustedt, and J. A. Telle, "Graph coloring on a coarse grained multiprocessor," in *International Workshop on Graph-Theoretic Concepts in Computer Science*. Springer, 2000, pp. 184–195.

[23] Ü. V. Çatalyürek, J. Feo, A. H. Gebremedhin, M. Halappanavar, and A. Pothen, "Graph coloring algorithms for multi-core and massively multithreaded architectures," *Parallel Computing*, vol. 38, no. 10-11, pp. 576–594, 2012.

[24] E. Saule and Ü. V. Çatalyürek, "An early evaluation of the scalability of graph algorithms on the intel mic architecture," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*. IEEE, 2012, pp. 1629–1639.

[25] A. E. Sariyüce, E. Saule, and Ü. V. Çatalyürek, "Scalable hybrid implementation of graph coloring using mpi and openmp," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*. IEEE, 2012, pp. 1744–1753.

[26] G. Rokos, G. Gorman, and P. H. Kelly, "A fast and scalable graph coloring algorithm for multi-core and many-core architectures," in *European Conference on Parallel Processing*. Springer, 2015, pp. 414–425.

[27] A. V. P. Grosset, P. Zhu, S. Liu, S. Venkatasubramanian, and M. Hall, "Evaluating graph coloring on gpus," in *Proceedings of the 16th ACM symposium on Principles and practice of parallel programming*, 2011, pp. 297–298.

[28] M. Deveci, E. G. Boman, K. D. Devine, and S. Rajamanickam, "Parallel graph coloring for manycore architectures," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2016, pp. 892–901.

[29] I. Finocchi, A. Panconesi, and R. Silvestri, "An experimental analysis of simple, distributed vertex coloring algorithms," *Algorithmica*, vol. 41, no. 1, pp. 1–23, 2005.

[30] A. H. Gebremedhin, D. Nguyen, M. M. A. Patwary, and A. Pothen, "Colpack: Software for graph coloring and related problems in scientific computing," *ACM Transactions on Mathematical Software (TOMS)*, vol. 40, no. 1, pp. 1–31, 2013.

[31] A. H. Gebremedhin, D. Nguyen, M. Patwary, and A. Pothen, "Colpack: Graph coloring software for derivative computation and beyond," *Submitted to ACM TOMS*, 2010.

[32] E. G. Boman, Ü. V. Çatalyürek, C. Chevalier, and K. D. Devine, "The zoltan and isorropia parallel toolkits for combinatorial scientific computing: Partitioning, ordering and coloring," *Scientific Programming*, vol. 20, no. 2, pp. 129–150, 2012.

[33] K. D. Devine, E. G. Boman, L. A. Riesen, U. V. Catalyurek, and C. Chevalier, "Getting started with zoltan: A short tutorial," in *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2009.

[34] S. Rajamanickam and E. G. Boman, "Parallel partitioning with zoltan: Is hypergraph partitioning worth it?" *Graph Partitioning and Graph Clustering*, vol. 588, pp. 37–52, 2012.

[35] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps *et al.*, "An overview of the trilinos project," *ACM Transactions on Mathematical Software (TOMS)*, vol. 31, no. 3, pp. 397–423, 2005.

[36] L. Dhulipala, G. E. Blelloch, and J. Shun, "Theoretically efficient parallel graph algorithms can be fast and scalable," *arXiv:1805.05208v4*, 2018.

[37] J. Shun and G. E. Blelloch, "Ligra: a lightweight graph processing framework for shared memory," in *ACM Sigplan Notices*, vol. 48, no. 8. ACM, 2013, pp. 135–146.

[38] L. Dhulipala, J. Shi, T. Tseng, G. E. Blelloch, and J. Shun, "The graph based benchmark suite (gbbs)," in *Proceedings of the 3rd Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*, 2020, pp. 1–8.

[39] F. Cazals and C. Karande, "A note on the problem of reporting maximal cliques," *Theoretical Computer Science*, vol. 407, no. 1-3, pp. 564–568, 2008.

[40] D. Eppstein, M. Löffler, and D. Strash, "Listing all maximal cliques in sparse graphs in near-optimal time," in *Algorithms and Computation - 21st International Symposium, ISAAC 2010, Jeju Island, Korea, December 15-17, 2010, Proceedings, Part I*, 2010, pp. 403–414. [Online]. Available: https://doi.org/10.1007/978-3-642-17517-6_36

[41] E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," *Theor. Comput. Sci.*, vol. 363, no. 1, pp. 28–42, 2006. [Online]. Available: https://doi.org/10.1016/j.tcs.2006.06.015

[42] M. Farach-Colton and M. Tsai, "Computing the degeneracy of large graphs," in *LATIN 2014: Theoretical Informatics - 11th Latin American Symposium, Montevideo, Uruguay, March 31 - April 4, 2014. Proceedings*, 2014, pp. 250–260. [Online]. Available: https://doi.org/10.1007/978-3-642-54423-1_22

[43] M. Chrobak and D. Eppstein, "Planar orientations with low out-degree and compaction of adjacency matrices," *Theoretical Computer Science*, vol. 86, no. 2, pp. 243–266, 1991.

[44] P. Erdős and A. Hajnal, "On chromatic number of graphs and set-systems," *Acta Mathematica Hungarica*, vol. 17, no. 1-2, pp. 61–99, 1966.

[45] L. M. Kirousis and D. M. Thilikos, "The linkage of a graph," *SIAM Journal on Computing*, vol. 25, no. 3, pp. 626–647, 1996.

[46] E. C. Freuder, "A sufficient condition for backtrack-free search," *Journal of the ACM (JACM)*, vol. 29, no. 1, pp. 24–32, 1982.

[47] G. D. Bader and C. W. Hogue, "An automated method for finding molecular complexes in large protein interaction networks," *BMC bioinformatics*, vol. 4, no. 1, p. 2, 2003.

[48] R. D. Blumofe and C. E. Leiserson, "Scheduling multithreaded computations by work stealing," *Journal of the ACM (JACM)*, vol. 46, no. 5, pp. 720–748, 1999.

[49] ——, "Space-efficient scheduling of multithreaded computations," *SIAM Journal on Computing*, vol. 27, no. 1, pp. 202–229, 1998.

[50] G. Bilardi and A. Pietracaprina, *Models of Computation, Theoretical*. Boston, MA: Springer US, 2011, pp. 1150–1158.

[51] G. E. Blelloch and B. M. Maggs, *Parallel Algorithms*, 2nd ed. Chapman & Hall/CRC, 2010, p. 25.

[52] J. F. JaJa, *PRAM (Parallel Random Access Machines)*. Boston, MA: Springer US, 2011, pp. 1608–1615.

[53] A. Aggarwal, A. K. Chandra, and M. Snir, "On communication latency in pram computations," in *Proceedings of the first annual ACM symposium on Parallel algorithms and architectures*, 1989, pp. 11–21.

[54] F. E. Fich, *The complexity of computation on the parallel random access machine*. Department of Computer Science, University of Toronto, 1993.

[55] R. P. Brent, "The parallel evaluation of general arithmetic expressions," *Journal of the ACM (JACM)*, vol. 21, no. 2, pp. 201–206, 1974.

[56] R. E. Ladner and M. J. Fischer, "Parallel prefix computation," *Journal of the ACM*, vol. 27, no. 4, pp. 831–838, 1980.

[57] M. Snir, *Reduce and Scan*. Boston, MA: Springer US, 2011, pp. 1728–1736. [Online]. Available: https://doi.org/10.1007/978-0-387-09766-4_120

[58] S. Roch, "Modern discrete probability: An essential toolkit," University Lecture, 2015.

[59] N. Alon, L. Babai, and I. Alon, "A fast and simple randomized parallel algorithm for the maximal independent set problem," *Journal of Algorithms*, vol. 7, no. 4, pp. 567–583, 1986.

[60] M. Goldberg and S. Thomas, "A new parallel algorithm for the maximal independent set problem ," *SIAM journal on coputing*, vol. 18, no. 2, pp. 419–427, 1989.

[61] A. V. Goldberg and S. A. Plotkin, "Parallel $(\Delta + 1)$-coloring of constant-degree graphs," *Information Processing Letters*, vol. 25, no. 4, pp. 341–345, 1987.

[62] A. Goldberg, S. Plotkin, and G. Shannon, "Parallel symmetry-breaking in sparse graphs," in *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, ser. STOC '87. ACM, 1987, pp. 315–324.

[63] M. Luby, "Removing randomness in parallel computation without a processor penalty," *Journal of Computer and System Sciences*, vol. 74, no. 2, pp. 250–286, 1993.

[64] P. M. McIlroy, K. Bostic, and M. D. McIlroy, "Engineering radix sort," *Computing systems*, vol. 6, no. 1, pp. 5–27, 1993.

[65] H. H. Seward, "Information sorting in the application of electronic digital computers to business operations," Ph.D. dissertation, Massachusetts Institute of Technology. Department of Electrical Engineering, 1954.

[66] C. A. Hoare, "Quicksort," *The Computer Journal*, vol. 5, no. 1, pp. 10–16, 1962.

[67] R. Chandra, L. Dagum, D. Kohr, R. Menon, D. Maydan, and J. McDonald, *Parallel programming in OpenMP*. Morgan kaufmann, 2001.

[68] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.

[69] J. Kunegis, "Konect: the koblenz network collection," in *Proc. of Intl. Conf. on World Wide Web (WWW)*. ACM, 2013, pp. 1343–1350.

[70] C. Demetrescu, A. V. Goldberg, and D. S. Johnson, *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*. American Math. Soc., 2009, vol. 74.

[71] P. Boldi and S. Vigna, "The webgraph framework i: compression techniques," in *Proceedings of the 13th international conference on World Wide Web*. ACM, 2004, pp. 595–602.

[72] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker graphs: An approach to modeling networks," *Journal of Machine Learning Research*, vol. 11, no. Feb, pp. 985–1042, 2010.

[73] P. J. Mucci, S. Browne, C. Deane, and G. Ho, "Papi: A portable interface to hardware performance counters," in *Proceedings of the department of defense HPCMP users group conference*, vol. 710, 1999.

[74] E. D. Dolan and J. J. Moré, "Benchmarking optimization software with performance profiles," *Mathematical programming*, vol. 91, no. 2, pp. 201–213, 2002.

[75] D. W. Matula, G. Marble, and J. D. Isaacson, "Graph coloring algorithms," in *Graph theory and computing*. Elsevier, 1972, pp. 109–122.

[76] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[77] G. E. Blelloch, "Programming parallel algorithms," *Communications of the ACM*, vol. 39, no. 3, pp. 85–97, 1996.

[78] N. Linial, "Distributive graph algorithms global solutions from local data," in *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, 1987, pp. 331–335.

[79] ——, "Locality in distributed graph algorithms," *SIAM Journal on Computing*, vol. 21, no. 1, pp. 193–201, 1992.

[80] L. Barenboim and M. Elkin, "Distributed graph coloring: Fundamentals and recent developments," 2013.

[81] L. Barenboim, M. Elkin, S. Pettie, and J. Schneider, "The locality of distributed symmetry breaking," *Journal of the ACM*, vol. 63, no. 3, 2016.

[82] L. Barenboim and M. Elkin, "Deterministic distributed vertex coloring in polylogarithmic time," *J. ACM*, vol. 58, no. 5, 2011.

[83] J. Schneider and R. Wattenhofer, "A new technique for distributed symmetry breaking," in *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, ser. PODC '10. New York, NY, USA: ACM, 2010, p. 257–266. [Online]. Available: https://doi.org/10.1145/1835698.1835760

[84] L. Barenboim and M. Elkin, "Distributed $(\delta + 1)$-coloring in linear (in δ) time," in *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, ser. STOC '09. New York, NY, USA: ACM, 2009, p. 111–120.

[85] D. G. Harris, J. Schneider, and H.-H. Su, "Distributed (δ+1)-coloring in sublogarithmic rounds," in *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '16. New York, NY, USA: ACM, 2016, p. 465–478.

[86] Y.-J. Chang, W. Li, and S. Pettie, "An optimal distributed (δ+1)-coloring algorithm?" in *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, ser. STOC 2018. ACM, 2018, p. 445–456.

[87] A. Panconesi and A. Srinivasan, "Improved distributed algorithms for coloring and network decomposition problems," in *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, ser. STOC '92. New York, NY, USA: ACMy, 1992, p. 581–592.

[88] L. Barenboim, M. Elkin, and U. Goldenberg, "Locally-iterative distributed (δ+ 1): -coloring below szegedy-vishwanathan barrier, and applications to self-stabilization and to restricted-bandwidth models," in *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, ser. PODC '18. New York, NY, USA: ACM, 2018, p. 437–446.

[89] Y.-J. Chang, T. Kopelowitz, and S. Pettie, "An exponential separation between randomized and deterministic complexity in the local model," *SIAM Journal on Computing*, vol. 48, no. 1, pp. 122–143, 2019.

[90] Öjvind Johansson, "Simple distributed δ-coloring of graphs," *Information Processing Letters*, vol. 70, no. 5, pp. 229 – 232, 1999.

[91] D. A. Grable and A. Panconesi, "Fast distributed algorithms for brooks–vizing colorings," *Journal of Algorithms*, vol. 37, no. 1, pp. 85–120, 2000.

[92] F. Kuhn and R. Wattenhofer, "On the complexity of distributed graph coloring," in *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, 2006, pp. 7–15.

[93] D. Gregor and A. Lumsdaine, "The parallel bgl: A generic library for distributed graphcomputations," 2005.

[94] D. Bozdağ, A. H. Gebremedhin, F. Manne, E. G. Boman, and U. V. Catalyurek, "A framework for scalable greedy coloring on distributed-memory parallel computers ," *Journal of Parallel and Distributed Computing*, vol. 68, no. 4, pp. 515–535, 2008.

[95] M. Naumov, M. Arsaev, P. Castonguay, J. Cohen, J. Demouth, J. Eaton, S. Layton, N. Markovskiy, I. Reguly, N. Sakharnykh, V. Sellappan, and R. Strzodka, "Amgx: A library for gpu accelerated algebraic multigrid and preconditioned iterative methods," *SIAM Journal on Scientific Computing*, vol. 37, no. 5, pp. 602–626, 2015.

[96] M. Naumov, L. Chien, P. Vandermersch, and U. Kapasi, "Cusparse library," in *GPU Technology Conference*, 2010.

[97] J. Cohen and P. Castonguay, "Efficient graph matching and coloring on the gpu," in *GPU Technology Conference*, 2012, pp. 1–10.

[98] X. Chen, P. Li, J. Fang, T. Tang, Z. Wang, and C. Yang, "Efficient and high-quality sparse graph coloring on gpus," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 10, p. e4064, 2017.

[99] S. Che, G. Rodgers, B. Beckmann, and S. Reinhardt, "Graph coloring on the gpu and some techniques to improve load imbalance," in *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*. IEEE, 2015, pp. 610–617.

[100] M. Naumov, P. Castonguay, and J. Cohen, "Parallel graph coloring with applications to the incomplete-lu factorization on the gpu," *Nvidia White Paper*, 2015.

[101] J. Culberson, "Iterated greedy graph coloring and the difficulty landscape," 1992.

[102] A. E. Sarlyüce, E. Saule, and U. V. Çatalyürek, "Improving graph coloring on distributed-memory parallel computers," in *2011 18th International Conference on High Performance Computing*. IEEE, 2011.

[103] N. M. Gandhi and R. Misra, "Performance comparison of parallel graph coloring algorithms on bsp model using hadoop," in *2015 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2015, pp. 110–116.

[104] G. Alabandi, E. Powers, and M. Burtscher, "Increasing the parallelism of graph coloring via shortcutting," in *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2020, pp. 262–275.

[105] K. Diks, "A fast parallel algorithm for six-colouring of planar graphs," in *International Symposium on Mathematical Foundations of Computer Science*. Springer, 1986, pp. 273–282.

[106] D. W. Matula, Y. Shiloach, and R. E. Tarjan, "Two linear-time algorithms for five-coloring a planar graph," STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, Tech. Rep., 1980.

[107] J. F. Boyar and H. J. Karloff, "Coloring planar graphs in parallel," *Journal of Algorithms*, vol. 8, no. 4, pp. 470–479, 1987.

[108] T. Hagerup, M. Chrobak, and K. Diks, "Optimal parallel 5-colouring of planar graphs," *SIAM Journal on Computing*, vol. 18, no. 2, pp. 288–300, 1989.

[109] M. T. Gjertsen Jr., Robert K. ane Jones and P. E. Plassmann, "Parallel heuristics for improved, balanced graph colorings," *Journal of Parallel and Distributed Computing*, vol. 37, no. 2, pp. 171 – 186, 1996.

[110] H. Lu, M. Halappanavar, D. Chavarría-Miranda, A. Gebremedhin, and A. Kalyanaraman, "Balanced coloring for parallel computing applications," in *2015 IEEE International Parallel and Distributed Processing Symposium*, 2015, pp. 7–16.

[111] A. H. Gebremedhin, F. Manne, and A. Pothen, "Parallel Distance-k Coloring Algorithms for Numerical Optimization," in *Euro-Par 2002 Parallel Processing Proceedings*. Springer, Berlin, Heidelberg, 2002, pp. 912–921.

[112] I. Holyer, "The np-completeness of edge-coloring," *SIAM Journal on computing*, vol. 10, no. 4, pp. 718–720, 1981.

[113] S. Sallinen, K. Iwabuchi, S. Poudel, M. Gokhale, M. Ripeanu, and R. Pearce, "Graph colouring as a challenge problem for dynamic graph processing on distributed systems," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2016, pp. 347–358.

[114] L. Yuan, L. Qin, X. Lin, L. Chang, and W. Zhang, "Effective and efficient dynamic graph coloring," *Proceedings of the VLDB Endowment*, vol. 11, no. 3, pp. 338–351, 2017.

[115] J. Bossek, F. Neumann, P. Peng, and D. Sudholt, "Runtime analysis of randomized search heuristics for dynamic graph coloring," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019, pp. 1443–1451.

[116] S. Bhattacharya, D. Chakrabarty, M. Henzinger, and D. Nanongkai, "Dynamic algorithms for graph coloring," in *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2018, pp. 1–20.

[117] S. Solomon and N. Wein, "Improved dynamic graph coloring," *arXiv preprint arXiv:1904.12427*, 2019.

[118] S. K. Bera and P. Ghosh, "Coloring in graph streams," *arXiv preprint arXiv:1807.07640*, 2018.

[119] H. Lu, M. Halappanavar, D. Chavarría-Miranda, A. H. Gebremedhin, A. Panyala, and A. Kalyanaraman, "Algorithms for balanced graph colorings with applications in parallel computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 5, pp. 1240–1256, 2017.

[120] D. Bozdağ, Ü. i. t. V. Çatalyürek, A. H. Gebremedhin, F. Manne, E. G. Boman, and F. Özgüner, "Distributed-memory parallel algorithms for distance-2 coloring and related problems in derivative computation," *SIAM Journal on Scientific Computing*, vol. 32, no. 4, pp. 2418–2446, 2010.

[121] D. Bozdağ, U. Catalyurek, A. H. Gebremedhin, F. Manne, E. G. Boman, and F. Özgüner, "A parallel distance-2 graph coloring algorithm for distributed memory computers," in *International Conference on High Performance Computing and Communications*. Springer, 2005, pp. 796–806.

[122] J. Lin, S. Cai, C. Luo, and K. Su, "A reduction based method for coloring very large graphs." in *IJCAI*, 2017, pp. 517–523.

[123] A. Verma, A. Buchanan, and S. Butenko, "Solving the maximum clique and vertex coloring problems on very large sparse networks," *INFORMS Journal on computing*, vol. 27, no. 1, pp. 164–177, 2015.

[124] E. Hébrard and G. Katsirelos, "A hybrid approach for exact coloring of massive graphs," in *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 2019, pp. 374–390.

[125] R. Abbasian and M. Mouhoub, "An efficient hierarchical parallel genetic algorithm for graph coloring problem," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, 2011, pp. 521–528.

[126] A. E. Eiben, J. K. Van Der Hauw, and J. I. van Hemert, "Graph coloring with adaptive evolutionary algorithms," *Journal of Heuristics*, vol. 4, no. 1, pp. 25–46, 1998.

[127] C. Fleurent and J. A. Ferland, "Genetic and hybrid algorithms for graph coloring," *Annals of Operations Research*, vol. 63, no. 3, pp. 437–461, 1996.

[128] Y. Zhou, B. Duval, and J.-K. Hao, "Improving probability learning based local search for graph coloring," *Applied Soft Computing*, vol. 65, pp. 542–553, 2018.

[129] H. Lemos, M. Prates, P. Avelar, and L. Lamb, "Graph colouring meets deep learning: Effective graph neural network models for combinatorial problems," *arXiv preprint arXiv:1903.04598*, 2019.

[130] Y. Zhou, J.-K. Hao, and B. Duval, "Reinforcement learning based local search for grouping problems: A case study on graph coloring," *Expert Systems with Applications*, vol. 64, pp. 412–422, 2016.

[131] N. Musliu and M. Schwengerer, "Algorithm selection for the graph coloring problem," in *International Conference on Learning and Intelligent Optimization*. Springer, 2013, pp. 389–403.

[132] M. Farach-Colton and M.-T. Tsai, "Tight approximations of degeneracy in large graphs," in *LATIN 2016: Theoretical Informatics*. Springer, 2016, pp. 429–440.

[133] ——, "Computing the degeneracy of large graphs," in *Latin American Symposium on Theoretical Informatics*. Springer, 2014, pp. 250–260.

[134] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," in *Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)*, 2012, pp. 17–30.

[135] M. Besta and T. Hoefler, "Accelerating irregular computations with hardware transactional memory and active messages," in *ACM HPDC*, 2015.

[136] M. Besta, D. Stanojevic, T. Zivic, J. Singh, M. Hoerold, and T. Hoefler, "Log (graph): a near-optimal high-performance graph representation," in *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*. ACM, 2018, p. 7.

[137] L. Gianinazzi, P. Kalvoda, A. De Palma, M. Besta, and T. Hoefler, "Communication-avoiding parallel minimum cuts and connected components," in *Proceedings of the 23rd ACM SIGPLAN Symposium on*

Principles and Practice of Parallel Programming. ACM, 2018, pp. 219–232.

[138] M. Besta, M. Fischer, V. Kalavri, M. Kapralov, and T. Hoefler, "Practice of streaming and dynamic graphs: Concepts, models, systems, and parallelism," *arXiv preprint arXiv:1912.12740*, 2019.

[139] M. Besta, E. Peter, R. Gerstenberger, M. Fischer, M. Podstawski, C. Barthels, G. Alonso, and T. Hoefler, "Demystifying graph databases: Analysis and taxonomy of data organization, system designs, and graph queries," *arXiv preprint arXiv:1910.09017*, 2019.

[140] M. Besta, D. Stanojevic, J. D. F. Licht, T. Ben-Nun, and T. Hoefler, "Graph processing on fpgas: Taxonomy, survey, challenges," *arXiv preprint arXiv:1903.06697*, 2019.

[141] M. Besta, M. Fischer, T. Ben-Nun, D. Stanojevic, J. D. F. Licht, and T. Hoefler, "Substream-centric maximum matchings on fpga," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 13, no. 2, pp. 1–33, 2020.

[142] L. Thebault, "Scalable and efficient algorithms for unstructured mesh computations," Ph.D. dissertation, 2016.

[143] J. S. Firoz, M. Zalewski, A. Lumsdaine, and M. Barnas, "Runtime scheduling policies for distributed graph algorithms," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2018, pp. 640–649.

[144] D. Gregor and A. Lumsdaine, "The parallel bgl: A generic library for distributed graph computations," *Parallel Object-Oriented Scientific Computing (POOSC)*, vol. 2, pp. 1–18, 2005.

[145] M. Besta and T. Hoefler, "Active access: A mechanism for high-performance distributed data-centric computations," in *ACM ICS*, 2015.

[146] ——, "Fault tolerance for remote memory access programming models," in *ACM HPDC*, 2014, pp. 37–48.

[147] R. Gerstenberger, M. Besta, and T. Hoefler, "Enabling Highly-scalable Remote Memory Access Programming with MPI-3 One Sided," in *ACM/IEEE Supercomputing*, ser. SC '13, 2013, pp. 53:1–53:12.

[148] ——, "Enabling highly scalable remote memory access programming with mpi-3 one sided," *Communications of the ACM*, vol. 61, no. 10, pp. 106–113, 2018.

[149] H. Schweizer, M. Besta, and T. Hoefler, "Evaluating the cost of atomic operations on modern architectures," in *IEEE PACT*, 2015, pp. 445–456.

[150] P. Schmid, M. Besta, and T. Hoefler, "High-performance distributed RMA locks," in *ACM HPDC*, 2016, pp. 19–30.

[151] M. Besta, S. Weber, L. Gianinazzi, R. Gerstenberger, A. Ivanov, Y. Oltchik, and T. Hoefler, "Slim graph: practical lossy graph compression for approximate graph processing, storage, and analytics," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2019, p. 35.

[152] M. Besta and T. Hoefler, "Survey and taxonomy of lossless graph compression and space-efficient graph representations," *arXiv preprint arXiv:1806.01799*, 2018.

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

In our experiments, we evaluate an extensive number of graph coloring algorithms. The algorithms come with different variants and parametrizations, details are in the paper or in the attached package with the implementations.

Used Architectures: We used different high-performance servers for a broad evaluation and to analyze and ensure performance portability. Einstein is a Dell PowerEdge R910 with an Intel Xeon X7550 CPUs @ 2.00GHz with 18MB L3 cache, 1TiB RAM, and 32 cores per CPU (grouped in four sockets). Ault has Intel Xeon Gold 6140 CPU @ 2.30GHz, 768 GiB RAM, 18 cores, and 24.75MB L3. Fulen has Intel Skylake @ 2GHz, 1.8 TiB RAM, 52 cores, and 16MB L3.

Algorithms & Datasets We focus on modern heuristics from Table III. For each scheme, we always pick the most competitive implementation (i.e., fewest colors used and smallest performance overheads), selecting from existing repositories (ColPack [30], [31], Zoltan [18], [32]–[35], original code by Hasenplaugh et al. (HP) [14], GBBS with Ligra [36]–[38]), and our implementation. Specifically, we exhaustively evaluate the combinations of (a) coloring scheme (e.g., "MTJP" or "GMMP" from ColPack) and (b) the ordering of vertices (e.g., "ASL" from ColPack). Detailed parametrizations are in the attached package with code. We use real-world graphs from SNAP [60], KONECT [61], DIMACS [62], and WebGraph datasets [63]; see Table IV for details. We analyze synthetic power-law graphs (generated with the Kronecker model [64]). This gives a very large evaluation space and we only summarize selected findings; full data is in the extended report.

Datasets are described in detail in Section VI.

ARTIFACT AVAILABILITY

Software Artifact Availability: All author-created software artifacts are maintained in a public repository under an OSI-approved license.

Hardware Artifact Availability: There are no author-created hardware artifacts.

Data Artifact Availability: All author-created data artifacts are maintained in a public repository under an OSI-approved license.

Proprietary Artifacts: None of the associated artifacts, author-created or otherwise, are proprietary.

Author-Created or Modified Artifacts:

```
Persistent ID: https://www.dropbox.com/s/fd7r1z031x5
↪ b6hk/pgc.zip?dl=0
Artifact name: PGD repository
```

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: Einstein is a Dell PowerEdge R910 with an Intel Xeon X7550 CPUs @ 2.00GHz with 18MB L3 cache, 1TiB RAM, and 32 cores per CPU (grouped in four sockets)

Operating systems and versions: Debian with kernel release 4.19.0-1-amd64,

Compilers and versions: gcc 8.3.0

Applications and versions: GBBS, initial release (2018), https://github.com/ParAlg/gbbs; GAPBS, the original release (2015), https://github.com/sbeamer/gapbs

Libraries and versions: OpenMP version number 201511

Key algorithms: Parallel graph coloring heuristic

Input datasets and versions: Graph publc datasets: SNAP, KONECT, DIMACS, and WebGraph

URL to output from scripts that gathers execution environment information.

```
https://www.dropbox.com/s/htqts4yjdd68rkg/pgc\_confi
↪ g?dl=0
```

ARTIFACT EVALUATION

Verification and validation studies: We validated the results of the execution of algorithms by using simple verification routines that verify the correctness of colorings. These routines are simple sequential implementations, thoroughly tested. To facilitate the whole benchmarking process, we integrated these methods into the execution toolchain of GBBS and GAPBS (used infrastructure for executing graph algorithms). Finally, we used the memory checker valgrind for all the routines to ensure execution without any memory-related issues.

Accuracy and precision of timings: We measured the time with the standard std::chrono c++ library. The precision of chrono was set to microseconds, which is a much higher precision than necessary for our experiments.

Used manufactured solutions or spectral properties: n/a

Quantified the sensitivity of results to initial conditions and/or parameters of the computational environment: n/a

Controls, statistics, or other steps taken to make the measurements and analyses robust to variability and unknowns in the system. Each of the performance experiments (execution of graph algorithms) were performed 10 times (for small graphs) and 6 times (for large graphs). The first run was always discarded due to the setup overhead. For the next runs, we computed arithmetic mean and 95% confidence intervals. The limited number of runs was dictated by large scale of the experiments (graphs with up to 500 GB in size).