Ordering Heuristics for Parallel Graph Coloring

William Hasenplaugh Tim Kaler Tao B. Schardl Charles E. Leiserson
MIT Computer Science and Artificial Intelligence Laboratory
32 Vassar Street
Cambridge, MA 02139

ABSTRACT

This paper introduces the largest-log-degree-first (LLF) and smallest-log-degree-last (SLL) ordering heuristics for parallel greedy graph-coloring algorithms, which are inspired by the largest-degree-first (LF) and smallest-degree-last (SL) serial heuristics, respectively. We show that although LF and SL, in practice, generate colorings with relatively small numbers of colors, they are vulnerable to adversarial inputs for which any parallelization yields a poor parallel speedup. In contrast, LLF and SLL allow for provably good speedups on arbitrary inputs while, in practice, producing colorings of competitive quality to their serial analogs.

We applied LLF and SLL to the parallel greedy coloring algorithm introduced by Jones and Plassmann, referred to here as JP. Jones and Plassman analyze the variant of JP that processes the vertices of a graph in a random order, and show that on an $O(1)$-degree graph $G = (V,E)$, this JP-R variant has an expected parallel running time of $O(\lg V/\lg\lg V)$ in a PRAM model. We improve this bound to show, using work-span analysis, that JP-R, augmented to handle arbitrary-degree graphs, colors a graph $G = (V,E)$ with degree Δ using $\Theta(V+E)$ work and $O(\lg V + \lg\Delta \cdot \min\{\sqrt{E}, \Delta + \lg\Delta \lg V/\lg\lg V\})$ expected span. We prove that JP-LLF and JP-SLL— JP using the LLF and SLL heuristics, respectively — execute with the same asymptotic work as JP-R and only logarithmically more span while producing higher-quality colorings than JP-R in practice.

We engineered an efficient implementation of JP for modern shared-memory multicore computers and evaluated its performance on a machine with 12 Intel Core-i7 (Nehalem) processor cores. Our implementation of JP-LLF achieves a geometric-mean speedup of 7.83 on eight real-world graphs and a geometric-mean speedup of 8.08 on ten synthetic graphs, while our implementation using SLL achieves a geometric-mean speedup of 5.36 on these real-world graphs and a geometric-mean speedup of 7.02 on these synthetic graphs. Furthermore, on one processor, JP-LLF is slightly faster than a well-engineered serial greedy algorithm using LF, and likewise, JP-SLL is slightly faster than the greedy algorithm using SL.

This research was supported in part by the National Science Foundation under Grants CNS-1017058, CCF-1162148, and CCF-1314547 and in part by grants from Intel Corporation and Foxconn Technology Group. Tao B. Schardl was supported in part by an NSF Graduate Research Fellowship.

SPAA'14, June 23–25, 2014, Prague, Czech Republic.
Copyright 2014 ACM 978-1-4503-2821-0/14/06 ...$15.00.
http://dx.doi.org/10.1145/2612669.2612697.

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**]: Concurrent Programming—*parallel programming*; E.1 [**Data Structures**]: *graphs and networks*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*graph algorithms*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*graph labeling*

Keywords

Parallel algorithms; graph coloring; ordering heuristics; Cilk

1. INTRODUCTION

Graph coloring is a heavily studied problem with many real-world applications, including the scheduling of conflicting jobs [4, 25, 44, 51], register allocation [13, 15, 16], high-dimensional nearest-neighbor search [6], and sparse-matrix computation [19, 36, 48], to name just a few. Formally, a ***(vertex)-coloring*** of an undirected graph $G = (V,E)$ is an assignment of a ***color*** *v.color* to each vertex $v \in V$ such that for every edge $(u,v) \in E$, we have $u.color \neq v.color$, that is, no two adjacent vertices have the same color. The ***graph-coloring problem*** is the problem of determining a coloring which uses as few colors as possible.

We were motivated to work on graph coloring in the context of "chromatic scheduling" [1, 7, 37] of parallel "data-graph computations." A ***data graph*** is a graph with data associated with its vertices and edges. A ***data-graph computation*** is an algorithm implemented as a sequence of "updates" on the vertices of a data graph $G = (V,E)$, where ***updating*** a vertex $v \in V$ involves computing a new value associated with v as a function of v's old value and the values associated with the ***neighbors*** of v: the set of vertices adjacent to v in G, denoted $v.adj = \{u \in V : (v,u) \in E\}$. To ensure atomicity of each update, rather than using mutual-exclusion locks or other nondeterministic means of data synchronization, chromatic scheduling first colors the vertices of G and then sequences through the colors, scheduling all vertices of the same color in parallel. The time to perform a data-graph computation thus depends both on how long it takes to color G and on the number of colors produced by the graph-coloring algorithm: more colors means less parallelism. Although the coloring can be performed offline for some data-graph computations, for other computations the coloring must be produced online, and one must accept a trade-off between coloring ***quality*** — number of colors — and the time to produce the coloring.

Although the problem of finding an ***optimal*** coloring of a graph — a coloring using the fewest colors possible — is in NP-complete [26], heuristic "greedy" algorithms work reasonably well in practice. Welsh and Powell [51] introduced the original ***greedy*** coloring algorithm, which iterates over the vertices and assigns each vertex the smallest color not assigned to a neighbor. For a graph $G = (V,E)$, define the ***degree*** of a vertex $v \in V$ by $\deg(v) = |v.adj|$, the number of neighbors of v, and let the ***degree*** of G be $\Delta = \max_{v \in V}\{\deg(v)\}$. Welsh and Powell show that the greedy algorithm colors a graph G with degree Δ using at most $\Delta + 1$ colors.

Ordering heuristics

In practice, however, greedy coloring algorithms tend to produce much better colorings than the $\Delta + 1$ bound implies, and moreover, the order in which a greedy coloring algorithm colors the vertices affects the quality of the coloring.[1] To reduce the number of colors a greedy coloring algorithm uses, practitioners therefore employ ***ordering heuristics*** to determine the order in which the algorithm colors the vertices [2, 11, 35, 45].

The literature includes many studies of ordering heuristics and how they affect running time and coloring quality. Here are six of the more popular heuristics:

- **FF** The ***first-fit*** ordering heuristic [42, 51] colors vertices in the order they appear in the input graph representation.
- **R** The ***random*** ordering heuristic [35] colors vertices in a uniformly random order.
- **LF** The ***largest-degree-first*** ordering heuristic [51] colors vertices in order of decreasing degree.
- **ID** The ***incidence-degree*** ordering heuristic [19] iteratively colors an uncolored vertex with the largest number of colored neighbors.
- **SL** The ***smallest-degree-last*** ordering heuristic [2, 45] colors the vertices in the order induced by first removing all the lowest-degree vertices from the graph, then recursively coloring the resulting graph, and finally coloring the removed vertices.
- **SD** The ***saturation-degree*** ordering heuristic [11] iteratively colors an uncolored vertex whose colored neighbors use the largest number of distinct colors.

The experimental results overviewed in the Appendix (Section 12) indicate that we have listed these heuristics in rough order of coloring quality from worst to best, confirming the findings of Gebremedhin and Manne [27], who also rank the relative quality of R, LF, ID, and SD in this order.

Although an ordering heuristic can be viewed as producing a permutation of the vertices of a graph $G = (V,E)$, we shall find it convenient to think of an ordering heuristic H as producing an injective (1-to-1) ***priority function*** $\rho : V \to \mathbb{R}$.[2] We shall use the notation $\rho \in H$ to mean that the ordering heuristic H produces a priority function ρ.

Figure 1 gives the pseudocode for GREEDY, a greedy coloring algorithm. GREEDY takes a vertex-weighted graph $G = (V,E,\rho)$ as input, where $\rho : V \to \mathbb{R}$ is a priority function produced by some ordering heuristic. Each step of GREEDY simply selects the uncolored vertex with the highest priority according to ρ and colors it with the smallest available color. Generally, for a coloring algorithm A and ordering heuristic H, let A-H denote the coloring algorithm A that runs on vertex-weighted graphs whose priority functions are produced by H. In this way, we separate the behavior of the coloring algorithm from that of the ordering heuristic.

```
GREEDY(G)
1  let G = (V, E, ρ)
2  for v ∈ V in order of decreasing ρ(v)
3      C = {1, 2, ..., deg(v) + 1}
4      for u ∈ v.adj such that ρ(u) > ρ(v)
5          C = C − {u.color}
6      v.color = min C
```

Figure 1: Pseudocode for a serial greedy graph-coloring algorithm. Given a vertex-weighted graph $G = (V,E,\rho)$, where the priority of a vertex $v \in V$ is given by $\rho(v)$, GREEDY colors each vertex $v \in V$ in decreasing order according to $\rho(v)$.

```
JP(G)
 7  let G = (V, E, ρ)
 8  parallel for v ∈ V
 9      v.pred = {u ∈ V : (u, v) ∈ E and ρ(u) > ρ(v)}
10      v.succ = {u ∈ V : (u, v) ∈ E and ρ(u) < ρ(v)}
11      v.counter = |v.pred|
12  parallel for v ∈ V
13      if v.pred == ∅
14          JP-COLOR(v)

JP-COLOR(v)
15  v.color = GET-COLOR(v)
16  parallel for u ∈ v.succ
17      if JOIN(u.counter) == 0
18          JP-COLOR(u)

GET-COLOR(v)
19  C = {1, 2, ..., |v.pred| + 1}
20  parallel for u ∈ v.pred
21      C = C − {u.color}
22  return min C
```

Figure 2: The Jones-Plassman parallel coloring algorithm. JP uses a recursive helper function JP-COLOR to process a vertex once all of its predecessors have been colored. JP-COLOR uses the helper routine GET-COLOR to find the smallest color available to color a vertex v.

GREEDY, using any of these six ordering heuristics, can be made to run in $\Theta(V + E)$ time theoretically. Although some of these ordering heuristics involve more bookkeeping than others, achieving these theoretical bounds for GREEDY-FF, GREEDY-R, GREEDY-LF, GREEDY-ID, and GREEDY-SL is straightforward [29, 45]. Despite conjectures to the contrary [19, 29], GREEDY-SD can also be made to run in $\Theta(V + E)$ time, as we shall show in Section 8.

In practice, to produce a better quality coloring tends to cost more in running time. That is, the six heuristics, which are listed in increasing order of coloring quality, are also listed in increasing order of running time. The only exception is GREEDY-ID, which is dominated by GREEDY-SL in both coloring quality and runtime. The experiments discussed in the Appendix (Section 12) summarize our empirical findings for serial greedy coloring.

Parallel greedy coloring

There is a historical tension between coloring quality and the parallel scalability of greedy graph coloring. While the traditional ordering heuristics FF, LF, ID, and SL are efficient using GREEDY, it can be shown that any parallelization of them requires worst-case span of $\Omega(V)$ for a general graph $G = (V,E)$. Of the various attempts to parallelize greedy coloring [18, 22, 43], the algorithm first proposed by Jones and Plassmann [35] extends the greedy algorithm in a straightforward manner, uses work linear in size of the graph, and is deterministic given a random seed. Jones and Plassmann's original paper demonstrates good parallel performance for

[1] In fact, for any graph $G = (V,E)$, some ordering of V causes a greedy algorithm to color G optimally, although finding such an ordering is NP-hard [46].

[2] If the rule for an ordering heuristic allows for ties in the priority function (the priority function is not injective), we shall assume that ties are broken randomly. Formally, suppose that an ordering heuristic H produces a priority function ρ_H which may contain ties. We extend ρ_H to a priority function ρ that maps each vertex $v \in V$ to an ordered pair $\langle \rho_H(v), \rho_R(v) \rangle$, where the priority function ρ_R is produced by the random ordering heuristic R. To determine which of two vertices $u, v \in V$ has higher priority, we compare the ordered pairs $\rho(u)$ and $\rho(v)$ lexicographically. Notwithstanding this subtlety, we shall still adopt the simplifying convenience of viewing the priority function as mapping vertices to real numbers. In fact, the range of the priority function can be any linearly ordered set.

$O(1)$-degree graphs using the random ordering heuristic R. Unfortunately, in practice, R tends to produce colorings of relatively poor quality relative to the other traditional ordering heuristics. But the other traditional ordering heuristics are all vulnerable to adversarial graph inputs which cause JP to operate in $\Omega(V)$ time and thus exhibit poor parallel scalability. Consequently, there is need for new ordering heuristics for JP that can achieve both good coloring quality and guaranteed fast parallel performance.

Figure 2 gives the pseudocode for JP, which colors a given graph $G = (V, E, \rho)$ in the order specified by the priority function ρ. The algorithm begins in lines 9 and 10 by partitioning the neighbors of each vertex into ***predecessors*** — vertices with larger priorities — and ***successors*** — vertices with smaller priorities. JP uses the recursive JP-COLOR helper function to color a vertex $v \in V$ once all vertices in *v.pred* have been colored. Initially, lines 12–14 in JP scan the vertices of V to find every vertex that has no predecessors and colors each one using JP-COLOR. Within a call to JP-COLOR(v), line 15 calls GET-COLOR to assign a color to v, and the loop on lines 16–18 broadcasts in parallel to all of v's successors the fact that v is colored. For each successor $u \in$ *v.succ*, line 17 tests whether all of u's predecessors have already been colored, and if so, line 18 recursively calls JP-COLOR on u.

Jones and Plassmann analyze the performance of JP-R for $O(1)$-degree graphs. Although they do not discuss using the naive FF ordering heuristic, it is apparent that there exist adversarial input orderings for which their algorithm would fail to scale. For example, if the graph $G = (V, E)$ is simply a chain of vertices and the input order of V corresponds to their in order in the chain, JP-FF exhibits no parallelism. Jones and Plassmann show that a random ordering produced by R, however, allows the algorithm to run in $O(\lg V / \lg\lg V)$ expected time on this chain graph — and on any $O(1)$-degree graph, for that matter. Section 3 of this paper extends their analysis of JP-R to arbitrary-degree graphs.

Although JP-R scales well in theory, as well as in practice, when it comes to coloring quality, R is one of the weaker ordering heuristics, as we have noted. Of the other heuristics, JP-LF and JP-SL suffer from the same problem as FF, namely, it is possible to construct adversarial graphs that cause them to scale poorly, which we explore in Section 4. The ID heuristic tends to produce worse colorings than SL, and since GREEDY-ID also runs more slowly than GREEDY-SL, we have dropped ID from consideration. Moreover, because of our motivation to use the coloring algorithm for online chromatic scheduling, where the performance of the coloring algorithm cannot be sacrificed for marginal improvements in the quality of coloring, we also have dropped the SD heuristic. Since SD produces the best-quality colorings of the six ordering heuristics, however, we see parallelizing it as an interesting opportunity for future research.

Consequently, this paper focuses on alternatives to the LF and SL ordering heuristics that provide comparable coloring quality while exhibiting the same resilience to adversarial graphs that R shows compared with FF. Specifically, we introduce two new randomized ordering heuristics — "largest log-degree first" (LLF) and "smallest log-degree last" (SLL) — which resemble LF and SL, respectively, but which scale provably well when used with JP. We demonstrate that JP-LLF and JP-SLL provide good parallel scalability in theory and practice and are resilient to adversarial graphs.

Figure 3 summarizes our empirical findings. The data suggest that the LLF and SLL ordering heuristics produce colorings that are nearly as good as LF and SL, respectively. With respect to performance, our implementations of JP-LLF and JP-SLL actually operate slightly faster on 1 processor than our highly tuned implementations of GREEDY-LF and GREEDY-SL, respectively, and they scale comparably to JP-R.

H	H'	$\frac{C_{H'}}{C_H}$	$\frac{\text{GREEDY-}H}{\text{JP-}H'_1}$	$\frac{\text{JP-}H'_1}{\text{JP-}H'_{12}}$
FF	R	1.011	0.417	7.039
LF	LLF	1.021	1.058	7.980
SL	SLL	1.037	1.092	6.082

Figure 3: Summary of ordering-heuristic behavior on a suite of 8 real-world graphs and 10 synthetic graphs when run on a machine with 12 Intel Xeon X5650 processor cores. Column H lists three serial heuristics traditionally used for GREEDY, and column H' lists parallel heuristics for JP, of which LLF and SLL are introduced in this paper. Column "$C_{H'}/C_H$" shows the geometric mean of the ratio of the number of colors the parallel heuristic uses compared to the serial heuristic. Column "GREEDY-H/JP-H'_1" shows the geometric mean of the ratio of serial running times of GREEDY with the serial heuristic versus JP with the analogous parallel heuristic when run on 1 processor. Column "JP-H'_1/JP-H'_{12}" shows the geometric mean of the speedup of each parallel heuristic going from 1 processor to 12.

Outline

The remainder of this paper is organized as follows. Section 2 reviews the asynchronous parallel greedy coloring algorithm first proposed by Jones and Plassmann [35]. We show how JP can be extended to handle arbitrary-degree graphs and arbitrary priority functions. Using work-span analysis [21, Ch. 27], we show that JP colors a Δ-degree graph $G = (V, E, \rho)$ in $\Theta(V + E)$ work and $O(L \lg \Delta + \lg V)$ span, where L is the length of the longest path in G along which the priority function ρ decreases. Section 3 analyzes the performance of JP-R, showing that it operates using linear work and $O(\lg V + \lg \Delta \cdot \min\{\sqrt{E}, \Delta + \lg \Delta \lg V / \lg\lg V\})$ span. Section 4 shows that there exist "adversarial" graphs for which JP-LF and JP-SL exhibit limited parallel speedup. Section 5 analyzes the LLF and SLL ordering heuristics. We show that, given a Δ-degree graph G, JP-LLF colors $G = (V, E, \rho)$ using $\Theta(V + E)$ work and $O(\lg V + \lg \Delta(\min\{\Delta, \sqrt{E}\} + \lg^2 \Delta \lg V / \lg\lg V))$ expected span, while JP-SLL colors $G = (V, E, \rho)$ using same work and an additive $\Theta(\lg \Delta \lg V)$ additional span. Section 6 evaluates the performance of JP-LLF and JP-SLL on a suite of 8 real-world and 10 synthetic benchmark graphs. Section 7 discusses the software engineering techniques used in our implementation of JP-R, JP-LLF, and JP-SLL. Section 8 introduces an algorithm for computing the SD ordering heuristic using $\Theta(V + E)$ work. Section 9 discusses related work, and Section 10 offers some concluding remarks. The Appendix (Section 12) presents some experimental results for serial ordering heuristics.

2. THE JONES-PLASSMANN ALGORITHM

This section reviews JP, the parallel greedy coloring algorithm introduced by Jones and Plassmann [35], whose pseudocode is given in Figure 2. We first review the dag model of dynamic multithreading and work-span analysis [21, Ch. 27]. Then we describe how JP can be modified from Jones and Plassmann's original algorithm to handle arbitrary-degree graphs and arbitrary priority functions. We analyze JP with an arbitrary priority function ρ and show that on a Δ-degree graph $G = (V, E, \rho)$, JP runs in $\Theta(V + E)$ work and $O(L \lg \Delta + \lg V)$ span, where L is the longest path in the "priority dag" of G induced by ρ.

The dag model of dynamic multithreading

We shall analyze the parallel performance of JP using the dag model of dynamic multithreading introduced by Blumofe and Leiserson [9, 10] and described in tutorial fashion in [21, Ch. 27]. The dag model views the executed computation resulting from running a parallel algorithm as a ***computation dag*** A, in which each vertex denotes an instruction, and edges denote parallel control dependencies between instructions. Although the model encompasses other parallel control constructs, for our purposes, we need only understand that the execution of a **parallel for** loop can be modeled as a balanced binary tree of vertices in the dag, where the leaves of the tree denote the initial instructions of the loop iterations.

To analyze the performance of a dynamic multithreading program theoretically, we assume that the program executes on an ***ideal parallel computer***: each instruction executes in unit time, the computer has ample memory bandwidth, and the computer supports concurrent writes and read-modify-write instructions [33] without incurring overheads due to contention.

Given a dynamic multithreading program whose execution is modeled as a dag A, we can bound the parallel running time $T_P(A)$ of the computation as follows. The ***work*** $T_1(A)$ is the number of strands in the computation dag A. The ***span*** $T_\infty(A)$ is the length of the longest path in A. A deterministic algorithm with work T_1 and span T_∞ can always be executed on P processors in time T_P satisfying $\max\{T_1/P, T_\infty\} \le T_p \le T_1/P + T_\infty$ [9,10,12,24,32]. The ***speedup*** of an algorithm on P processors is T_1/T_P, which is at most P in theory, since $T_P \ge T_\infty$. The ***parallelism*** T_1/T_∞ is the greatest theoretical speedup possible for any number P of processors.

Analysis of JP

To analyze the performance of JP, it is convenient to think of the algorithm as coloring the vertices in the partial order of a "priority dag," similar to the priority dag described by Blelloch *et al.* [8]. Specifically, on a vertex-weighted graph $G = (V, E, \rho)$, the priority function ρ induces a ***priority dag*** $G_\rho = (V, E_\rho)$, where $E_\rho = \{(u,v) \in V \times V : (u,v) \in E \text{ and } \rho(u) > \rho(v)\}$. Notice that G_ρ is a dag, because ρ is an injective function and thus induces a total order on the vertices V. We shall bound the span of JP running on a graph G in terms of the ***depth*** of G_ρ, that is, the length of the longest path through G_ρ. We analyze JP in two steps.

First, we bound the work and span of calls during the execution of JP to the helper routine GET-COLOR(v), which returns the minimum color not assigned to any vertex $u \in v.pred$.

LEMMA 1. *The helper routine* GET-COLOR*, shown in Figure 2, can be implemented so that during the execution of* JP *on a graph $G = (V, E, \rho)$, a call to* GET-COLOR(v) *for a vertex $v \in V$ costs $\Theta(k)$ work and $\Theta(\lg k)$ span, where $k = |v.pred|$.*

PROOF. Implement the set C in GET-COLOR as an array whose ith entry initially stores the value i. The ith element from this array can be removed by setting the ith element to ∞. With this implementation, lines 20–21 execute in $\Theta(k)$ work and $\Theta(\lg k)$ span. The min operation on line 22 can be implemented as a parallel minimum reduction in the same bounds. □

Second, we show that JP colors a graph $G = (V, E, \rho)$ using work $\Theta(V + E)$ and span linear in the depth of the priority dag G_ρ.

THEOREM 2. *Given a Δ-degree graph $G = (V, E, \rho)$ for some priority function ρ, let G_ρ be the priority dag induced on G by ρ, and let L be the depth of G_ρ. Then* JP(G) *runs in $\Theta(V + E)$ work and $O(L \lg \Delta + \lg V)$ span.*

PROOF. Let us first bound the work and span of JP-COLOR excluding any recursive calls. For a single call to JP-COLOR on a vertex $v \in V$, Lemma 1 shows that line 15 takes $\Theta(\deg(v))$ work and $\Theta(\lg(\deg(v)))$ span. The JOIN operation on line 17 can be implemented as an atomic decrement-and-fetch operation [33] on the specified counter. Hence, excluding the recursive call, the loop on lines 16–18 performs $\Theta(\deg(v))$ work and $\Theta(\lg(\deg(v)))$ span to decrement the counters of all successors of v.

Because JP-COLOR is called once per vertex, the total work that JP spends in calls to JP-COLOR is $\Theta(V + E)$. Furthermore, the span of JP-COLOR is the length of any path of vertices in G_ρ, which is at most L, times $\Theta(\lg \Delta)$. Finally, the loop on lines 8–11 executes in $\Theta(V + E)$ work and $\Theta(\lg V + \lg \Delta)$ span, and the parallel loop on lines 12–14, excluding the call to JP-COLOR, executes in $\Theta(V + E)$ work and $\Theta(\lg V)$ span. □

3. JP WITH RANDOM ORDERING

This section bounds the depth of a priority dag G_ρ induced on a Δ-degree graph $G = (V, E, \rho)$ by a random priority function ρ in R. We show that the expected depth of G_ρ is $O(\min\{\sqrt{E}, \Delta + \lg \Delta \lg V / \lg\lg V\})$. Combined with Theorem 2, this bound implies that the expected span of JP-R is $O(\lg V + \lg \Delta \cdot \min\{\sqrt{E}, \Delta + \lg \Delta \lg V / \lg\lg V\})$. This bound extends Jones and Plassmann's $O(\lg V / \lg\lg V)$ bound for the depth of G_ρ when $\Delta = \Theta(1)$ [35].

To bound the depth of a priority dag G_ρ induced on a graph G by $\rho \in$ R, let us start by bounding the number of length-k paths in G_ρ. Each path in G_ρ corresponds to a unique ***simple*** path in G, that is, a path in which each vertex in G appears at most once. The following lemma bounds the number of length-k simple paths in G.

LEMMA 3. *The number of length-k simple paths in any Δ-degree graph $G = (V, E)$ is at most $|V| \cdot \min\{\Delta^{k-1}, (2|E|/(k-1))^{k-1}\}$.*

PROOF. Consider selecting a length-k simple path $p = \langle v_1, \ldots, v_k \rangle$ in G. There are $|V|$ choices for v_1, and for all $i \in \{1, \ldots, k-1\}$, given a choice of $\langle v_1, \ldots, v_i \rangle$, there are at most $\deg(v_i)$ choices for v_{i+1}. Hence there are at most $J = |V| \cdot \prod_{i=1}^{k-1} \deg(v_i)$ simple paths in G of length k. Let $V_k \subseteq V$ denote some set of $k-1$ vertices in V, and let $\delta = \max_{V_{k-1}}\{\sum_{v \in V_{k-1}} \deg(v)/(k-1)\}$ be the maximum average degree of any such set. Then we have $J \le |V| \cdot \delta^{k-1}$.

The proof follows from two upper bounds on δ. First, because $\deg(v) \le \Delta$ for all $v \in V$, we have $\delta \le \Delta$. Second, for all $V_{k-1} \subseteq V$, we have $\sum_{v \in V_{k-1}} \deg(v) \le \sum_{v \in V} \deg(v) = 2|E|$ by the handshaking lemma [21, p. 1172–3], and thus $\delta \le 2|E|/(k-1)$. □

Intuitively, the bound on the expected depth of G_ρ follows by arguing that although the number of simple length-k paths in a graph G might be exponential in k, for sufficiently large k, the probability is tiny that any such path is a path in G_ρ. To formalize this argument, we make use of the following technical lemma.

LEMMA 4. *Define the function $g(\alpha, \beta)$ for $\alpha, \beta > 1$ as*

$$g(\alpha, \beta) = e^2 \frac{\ln \alpha}{\ln \beta} \ln\left(e \frac{\beta \ln \alpha}{\alpha \ln \beta}\right) .$$

Then for all $\beta \ge e^2$, $\alpha \ge 2$, and $\beta \ge \alpha$, we have $g(\alpha, \beta) \ge 1$.

PROOF. We consider the cases when $\alpha \ge e^2$ and when $\alpha < e^2$ separately.

When $\alpha > e^2$, the partial derivative of $g(\alpha, \beta)$ with respect to β is

$$\begin{aligned} \frac{\partial g(\alpha, \beta)}{\partial \beta} &= e^2 \frac{\ln \alpha}{\beta \ln^2 \beta} \ln\left(\frac{\alpha}{e^2} \frac{\ln \beta}{\ln \alpha}\right) \\ &\ge 0 , \end{aligned}$$

since $\alpha \ln \beta / e^2 \ln \alpha \ge 1$ when $\alpha \ge e^2$ and $\beta \ge \alpha$. Thus, $g(\alpha, \beta)$ is a nondecreasing function in its second argument when $\alpha \ge e^2$ and $\beta \ge \alpha$. Since we have

$$\begin{aligned} g(\alpha, \alpha) &= e^2 (\ln \alpha / \ln \alpha) \ln(e(\alpha \ln \alpha)/(\alpha \ln \alpha)) \\ &\ge 1 , \end{aligned}$$

it follows that $g(\alpha,\beta) \geq 1$ for $\alpha \geq e^2$ and $\beta \geq \alpha$.

When $e^2 > \alpha \geq 2$, we make use of the fact that $2\beta/e\ln\beta > \sqrt{\beta}$ for all $\beta > e^2$:

$$\begin{aligned} g(\alpha,\beta) &\geq (e^2\ln 2/\ln\beta)\ln(2\beta/(e\ln\beta)) \\ &\geq (e^2\ln 2/\ln\beta)\ln\left(\sqrt{\beta}\right) \\ &\geq (e^2\ln 2\ln\beta)/(2\ln\beta) \\ &\geq 1 . \quad \square \end{aligned}$$

The following theorem applies Lemmas 3 and 4 to establish the bound on the depth of G_ρ.

THEOREM 5. *Let $G = (V,E)$ be a Δ-degree graph, let $n = |V|$ and $m = |E|$, and let G_ρ be a priority dag induced on G by a random priority function $\rho \in$ R. For any constant $\varepsilon > 0$ and sufficiently large n, with probability at most $n^{-\varepsilon}$, there exists a directed path of length $e^2 \cdot \min\{\Delta, \sqrt{m}\} + (1+\varepsilon)\min\{e^2\ln\Delta\ln n/\ln\ln n, \ln n\}$ in G_ρ.*

PROOF. Let $p = \langle v_1,\ldots,v_k\rangle$ be a length-k simple path in G. Because ρ is a random priority function, ρ induces each possible permutation among $\{v_1,\ldots,v_k\}$ with equal probability. If p is a directed path in G_ρ, then we must have that $\rho(v_1) < \rho(v_2) < \cdots < \rho(v_k)$. Hence, p is a length-k path in G_ρ with probability at most $1/k!$. If J is the number of length-k simple paths in G, then by the union bound, the probability that a length-k directed path exists in G_ρ is at most $J/k!$, which is at most $J(e/k)^k$ by Stirling's approximation [21, p. 57].

We consider cases when $\Delta < \ln n$ and $\Delta \geq \ln n$ separately. First, suppose that $\Delta < \ln n$. By Lemma 3, the number of length-k simple paths in G is at most $n\Delta^{k-1} \leq n\Delta^k$. By the union bound, the probability that a length-k path exists in G_ρ is at most $n(e\Delta/k)^k$. We assume, without loss of generality, that $\Delta > 2$, since the theorem holds for $O(1)$-degree graphs as a result of [35].

For $\Delta \geq 2$, observe that, by Lemma 4, the function $g(\alpha,\beta) = e^2(\ln\alpha/\ln\beta)\ln(\beta\ln\alpha/\alpha\ln\beta)$ is at least 1 for all $\alpha \geq 2$ and $\beta \geq e^2$. Letting $\alpha = \Delta$, $\beta = \ln n$, and $k = e^2(\Delta + (1+\varepsilon)\ln\Delta\ln n/\ln\ln n)$, we conclude that

$$\begin{aligned} n(e\Delta/k)^k &= n\cdot\exp(-k\ln(k/e\Delta)) \\ &\leq n\cdot\exp\left(-e^2(1+\varepsilon)\ln n\frac{\ln\Delta}{\ln\ln n}\ln\left(e\frac{\ln n\ln\Delta}{\Delta\ln\ln n}\right)\right) \\ &= n\cdot\exp(-(1+\varepsilon)(\ln n)\cdot g(\Delta,\ln n)) \\ &\leq ne^{-(1+\varepsilon)\ln n} \\ &= n^{-\varepsilon} . \end{aligned}$$

Next, given $\Delta \geq \ln n$, consider the cases when $\Delta < \sqrt{m}$ and $\Delta \geq \sqrt{m}$, separately. When $\Delta < \sqrt{m}$, letting $k = e^2\Delta + (1+\varepsilon)\ln n$, the theorem follows from the facts that $k \geq (1+\varepsilon)\ln n$ and $k \geq e^2\Delta$. When $\Delta \geq \sqrt{m}$, let $k = e^2\sqrt{m} + (1+\varepsilon)\ln n$. By Lemma 3, the number of length-k simple paths is at most $n(2m/(k-1))^{k-1} \leq n(4m/k)^k$, and thus the probability that a length-k path exists in G_ρ is at most $n(4em/k^2)^k$. The theorem follows from the facts that $k \geq (1+\varepsilon)\ln n$ and $k^2 \geq e^4 m$. $\square$

COROLLARY 6. *Given a graph $G = (V,E,\rho)$, where $\rho \in$ R is a random priority function, the expected depth of the priority dag G_ρ is $O(\min\{\sqrt{E}, \Delta + \lg\Delta\lg V/\lg\lg V\})$, and thus* JP-R *colors all vertices of G with $O(\lg V + \lg\Delta\cdot\min\{\sqrt{E}, \Delta + \lg\Delta\lg V/\lg\lg V\})$ expected span.*

PROOF. Theorems 2 and 5 imply the corollary. $\square$

4. THE LF AND SL HEURISTICS

This section shows that the largest-first (LF) and smallest-last (SL) ordering heuristics can inhibit parallel speedup when used by JP. We examine a "clique-chain" graph and show that JP-LF incurs $\Omega(\Delta^2)$ span to color a Δ-degree clique-chain graph $G = (V,E)$, whereas JP-R colors G incurring only $O(\Delta\lg\Delta + \lg^2\Delta\lg V/\lg\lg V)$ expected span. We formally review the SL ordering heuristic and observe that this formulation of SL means that JP-SL requires $\Omega(V)$ span to color a path graph $G = (V,E)$.

The LF ordering heuristic

The LF ordering heuristic colors the vertices of a graph $G = (V,E,\rho)$ for some ρ in LF in order of decreasing degree. Formally, $\rho \in$ LF is defined for a vertex $v \in V$ as $\rho(v) = \langle\deg(V), \rho_R(v)\rangle$, where ρ_R is randomly chosen from R.

Although LF has been used in parallel greedy graph-coloring algorithms in the past [2,29], Figure 4 illustrates a Δ-degree "clique-chain" graph $G = (V,E)$ for which JP-LF incurs $\Omega(\Delta^2)$ span to color, but JP-R colors with only $O(\Delta\lg\Delta + \lg^2\Delta\lg V/\lg\lg V)$ expected span. Conceptually, the ***clique-chain*** graph comprises a set of cliques of increasing size that are connected in a "chain" such that JP-LF is forced to color these cliques sequentially from largest to smallest. Figure 4 illustrates a Δ-degree clique-chain graph $G = (V,E)$, where 3 evenly divides Δ. This clique-chain graph contains a sequence of cliques $\mathscr{K} = \{K_1, K_4, \ldots, K_{\Delta-2}\}$ of increasing size, each pair of which is separated by two additional vertices forming a linear chain. Specifically, for $r \in \{1,4,\ldots,\Delta-2\}$, each vertex $u \in K_r$ is connected to each vertex $u \in K_{r+3}$ by a path $\langle u, x_{r+1}, x_{r+2}, v\rangle$ for distinct vertices $x_{r+1}, x_{r+2} \in V$. Additional vertices, shown above the chain in Figure 4, ensure that the degree of each vertex in K_r is $r+2$, and the degrees of the vertices x_{r+1} and x_{r+2} are $r+3$ and $r+4$, respectively. Clique-chain graphs of other degrees are structured similarly.

THEOREM 7. *For any $\Delta > 0$, there exists a Δ-degree graph $G = (V,E)$ such that* JP-LF *colors G in $\Omega(\Delta^2)$ span and* JP-R *colors G in $O(\Delta\lg\Delta + \lg^2\Delta\lg V/\lg\lg V)$ expected span.*

PROOF. Assume without loss of generality that 3 evenly divides Δ and that G is a clique-chain graph. The span of JP-R follows from Corollary 6. Because JP-LF trivially requires $\Omega(1)$ span to process each vertex in G, the span of JP-LF on G can be bounded by showing that the length of the longest path p in the priority dag G_ρ induced on G by any priority function ρ in LF is $\Delta^2/6 + \Delta/2 + 2$. Because LF assigns higher priority to higher-degree vertices, p starts at some vertex in $K_{\Delta-2}$, which has degree Δ, and passes through the $\Delta - 2$ vertices in $K_{\Delta-2}$ followed by $x_{\Delta-3}$ and $x_{\Delta-4}$.[3] The remainder of p is a longest path through the clique-chain graph G' of degree $\Delta - 3$ in the remaining graph $G - K_{\Delta-2} - \{x_{\Delta-3}, x_{\Delta-4}\}$, which has a longest path p' of length $|p'| = (\Delta-3)^2/6 + (\Delta-3)/2 + 2$ by induction. The length of p is thus $\Delta + |p'| = \Delta^2/6 + \Delta/2 + 2$. $\square$

The SL ordering heuristic

We focus on the formulation of the SL ordering heuristic due to Allwright *et al.* [2], because our experiments indicate that it gives colorings using fewer colors than other formulations [45].

Given a graph $G = (V,E)$, the SL ordering heuristic produces a priority function ρ via an iterative algorithm that assigns priorities to the vertices V in rounds to induce an ordering on V. For $i \geq 0$, let $G_i = (V_i, E_i)$ denote the subgraph of G remaining at the start of round i, and let δ_i denote an upper bound on the

[3]Notice that it does not matter how ties are broken in the priority function.

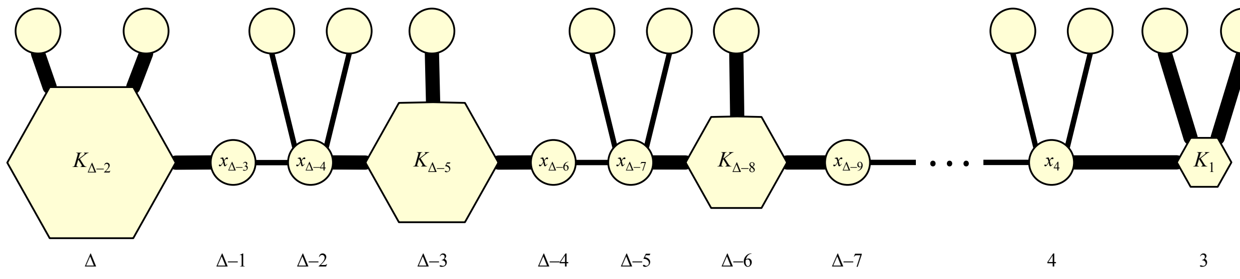

Figure 4: A Δ-degree clique-chain graph G, which Theorem 7 shows is adversarial for JP-LF. This graph contains $\Theta(\Delta^2)$ vertices arranged as a chain of cliques. Each hexagon labeled K_r represents a clique of r vertices, and circles represent individual vertices. A thick edge between an individual vertex and a clique indicates that the vertex is connected to every vertex within the clique. A label below an individual vertex indicates the degree of the associated vertex, and a label below a clique indicates the degree of every vertex within that clique.

smallest degree of any vertex $v \in V_i$. Assume that $\delta_0 = 1$. At the start of round i, remove all vertices $v \in V_i$ such that $\deg(v) \leq \max\{\delta_{i-1}, \min_{v\in V_i}\{\deg(v)\}\}$. For a vertex v removed in round i, a priority function $\rho \in \text{SL}$ is defined as $\rho(v) = \langle i, \rho_{\text{R}}(v)\rangle$ where $\rho_{\text{R}} \in \text{R}$ is a random priority function.

The following theorem shows that there exist graphs for which JP-SL incurs a large span, whereas JP-R incurs only a small span.

THEOREM 8. *There exists a class of graphs such that for any $G = (V,E,\rho)$ in the class and for any priority function $\rho \in$ SL,* JP-SL *incurs $\Omega(V)$ span and* JP-R *incurs $O(\lg V/\lg\lg V)$ span.*

PROOF. Consider the algorithm to compute the priority function ρ for all vertices in a path graph G. By induction over the rounds, the graph G_i at the start of round i is a path with $|V| - 2i + 2$ vertices, and in round i the 2 vertices at the endpoints of G_i will be removed. Hence $\lceil |V|/2\rceil$ rounds are required to assign priorities for all vertices in G. A similar argument shows that the resulting priority dag G_ρ contains a path of length $|V|/2$ along which the priorities strictly decrease. JP-SL trivially incurs $\Omega(1)$ span through each vertex in the longest path in G_ρ. Since there are $\Theta(V)$ total vertices along the path and by Corollary 6 with $\Delta = \Theta(1)$, the theorem follows. □

We shall see in Section 5 that it is possible to achieve coloring quality comparable to LF and SL, but with guaranteed parallel performance comparable to JP-R.

5. LOG ORDERING HEURISTICS

This section describes the largest-log-degree-first (LLF) and smallest-log-degree-last (SLL) ordering heuristics. Given a Δ-degree graph G, we show that the expected depth of the priority dag G_ρ induced on G by a priority function $\rho \in \text{LLF}$ is $O(\min\{\Delta, \sqrt{E}\} + \lg^2\Delta \lg V/\lg\lg V)$. The same bound applies to the depth of a priority dag G_ρ induced on a graph G by a priority function $\rho \in \text{SLL}$, though $O(\lg\Delta\lg V)$ additional span is required to calculate ρ using the method given in Figure 5. Combined with Theorem 2, these bounds imply that the expected span of JP-LLF is $O(\lg V + \lg\Delta(\min\{\Delta, \sqrt{E}\} + \lg^2\Delta\lg V/\lg\lg V))$ and the expected span of JP-SLL is $O(\lg\Delta\lg V + \lg\Delta(\min\{\Delta, \sqrt{E}\} + \lg^2\Delta\lg V/\lg\lg V))$.

The LLF ordering heuristic

The **LLF** ***ordering heuristic*** orders the vertices in decreasing order by the logarithm of their degree. More precisely, given a graph $G = (V,E,\rho)$ for some $\rho \in \text{LLF}$, the priority of each $v \in V$ is equal to $\rho(v) = \langle\lceil\lg(\deg(v))\rceil, \rho_{\text{R}}(v)\rangle$, where $\rho_{\text{R}} \in \text{R}$ is a random priority function and $\lg x$ denotes $\log_2 x$. [4] For a given graph G, the following theorem bounds the depth of the priority dag G_ρ induced by $\rho \in \text{LLF}$.

THEOREM 9. *Let $G = (V,E)$ be a Δ-degree graph, and let G_ρ be the priority dag induced on G by a priority function $\rho \in$ LLF. The expected length of the longest directed path in G_ρ is $O(\min\{\Delta, \sqrt{E}\} + \lg^2\Delta\lg V/\lg\lg V)$.*

PROOF. Consider a length-k path $p = \langle v_1, \ldots, v_k\rangle$ in G_ρ. Let $G(\ell) \subseteq G_\rho$ be the subdag of G_ρ induced by those vertices $v \in V$ for which $\rho(v) = \lceil\lg(\deg(v))\rceil = \ell$. Suppose that $v_i \in G(\ell)$ for some $v_i \in p$. Since $\lceil\lg(\deg(v_{i-1}))\rceil \geq \lceil\lg(\deg(v_i))\rceil$ for all $i > 1$, we have $v_{i-1} \in G(\ell')$ for some $\ell' \geq \ell$. We can therefore decompose p into a sequence of paths $p = \langle p_{\lceil\lg\Delta\rceil}, \ldots, p_0\rangle$ such that each subpath $p_\ell \in p$ is a path through $G(\ell)$. By definition of LLF, the subdag $G(\ell)$ is a dag induced on a graph with degree 2^ℓ by a random priority function.

By Corollary 6, the expected length of p_ℓ is $O(2^\ell + \ell\lg V/\lg\lg V)$. Linearity of expectation therefore implies that

$$\begin{aligned} \text{E}[|p|] &= \sum_{\ell=0}^{\lceil\lg\Delta\rceil} O\Big(2^\ell + \ell\lg V/\lg\lg V\Big) \\ &= O\Big(\Delta + \lg^2\Delta\lg V/\lg\lg V\Big) . \end{aligned}$$

To establish the $\sqrt{E}$ bound, observe that at most $E/2^\ell$ vertices have degree at least 2^ℓ. Consequently, for $\ell > \lg\sqrt{E}$, the depth of $G(\ell)$ can be at most $E/2^\ell$. Hence we have

$$\begin{aligned} \text{E}[|p|] &\leq \sum_{\ell=0}^{\lceil\lg\sqrt{E}\rceil} O\Big(2^\ell\Big) + \sum_{\ell=\lceil\lg\sqrt{E}\rceil}^{\infty} E/2^\ell \\ &\qquad + \sum_{\ell=0}^{\lceil\lg\Delta\rceil} O(\ell\lg V/\lg\lg V) \\ &= O\Big(\sqrt{E} + \lg^2\Delta\lg V/\lg\lg V\Big). \quad \square \end{aligned}$$

COROLLARY 10. *Given a graph $G = (V,E,\rho)$ for some $\rho \in$ LLF,* JP-LLF *colors all vertices in G with expected span $O(\lg V + \lg\Delta(\min\{\sqrt{E}, \Delta\} + \lg^2\Delta\lg V/\lg\lg V))$.* □

[4]The theoretical results in this section assume only that the base b of the logarithm is a constant. In practice, however, it is possible that the choice of b could have impact on the coloring quality or runtime of JP-LLF. We studied this trade-off and found that there is only a minor dependence on b. In general, the coloring quality and runtime of JP-LLF smoothly transitions from the behavior of JP-LF for small b and the behavior of JP-R for large b, sweeping out a Pareto-efficient frontier of reasonable choices. We chose $b = 2$ for our experiments, because $\log_2 x$ can be calculated conveniently by native instructions on modern architectures.

```
SLL-ASSIGN-PRIORITIES($G, r$)
23  let $G = (V, E)$
24  $i = 1$
25  $U = V$
26  let $\Delta$ be the degree of $G$
27  let $\rho_R \in R$ be a random priority function
28  for $d = 0$ to $\lg\Delta$
29      for $j = 1$ to $r$
30          $Q = \{u \in U : |u.adj \cap U| \leq 2^d\}$
31          parallel for $v \in Q$
32              $\rho(v) = \langle i, \rho_R(v)\rangle$
33          $U = U - Q$
34          $i = i + 1$
35  return $\rho$
```

Figure 5: Pseudocode for SLL-ASSIGN-PRIORITIES, which computes a priority function $\rho \in$ SLL for the input graph. The input parameter r denotes the maximum number of times SLL-ASSIGN-PRIORITIES is permitted to remove vertices of at most a particular degree 2^d on lines 29–34.

PROOF. The corollary follows from Theorem 2. □

The SLL ordering heuristic

To understand the **SLL *ordering heuristic***, it is convenient to consider in isolation how to compute its priority function. The pseudocode in Figure 5 for SLL-ASSIGN-PRIORITIES describes algorithmically how to perform this computation on a given graph $G = (V, E)$. As Figure 5 shows, a priority function $\rho \in$ SLL can be computed by iteratively removing low-degree vertices from G in rounds. The priority of a vertex $v \in V$ is the round number in which v is removed, with ties broken randomly. As with SL, SLL colors the vertices of G in the reverse order in which they are removed, but SLL-ASSIGN-PRIORITIES determines when to remove a vertex using a degree bound that grows exponentially. SLL-ASSIGN-PRIORITIES considers each degree bound for a maximum of r rounds. Effectively, a vertex is removed from G based on the logarithm of its degree in the remaining graph.

We can formalize the behavior of SLL as follows. Given a graph G, let $G_i = (V_i, E_i)$ denote the subgraph of G remaining at the start of round i. As Figure 5 shows, for each $d \in \{0, 1, \ldots, \lg\Delta\}$, SLL-ASSIGN-PRIORITIES executes r rounds in which it removes vertices $v \in V_i$ such that $\deg(v) \leq 2^d$ in G_i.[5]

For a given graph G, the following theorem bounds the depth of the priority dag G_ρ induced by a priority function $\rho \in$ SLL.

THEOREM 11. *Let $G = (V, E)$ be a Δ-degree graph, and let G_ρ be the priority dag induced on G by a random priority function $\rho \in$ SLL. The expected length of the longest directed path in G_ρ is $O(\min\{\Delta, \sqrt{E}\} + \lg^2\Delta \lg V / \lg\lg V)$.*

PROOF. We begin with an argument similar to the proof of Theorem 9. Let $p = \langle v_1, \ldots, v_k\rangle$ be a length-k path in G_ρ, and let $G(\ell) \subseteq G_\rho$ be the subdag of G_ρ induced by those vertices $v \in V$, where $\rho(v) = \ell$. Since lines 29–34 of SLL-ASSIGN-PRIORITIES remove vertices with degree at most 2^d exactly r times for each $d \in [0, \ldots, \lg\Delta]$, we have that $\lfloor \rho(v)/r \rfloor = d$, and thus the degree of $G(\ell)$ is at most $2^{\lfloor \ell/r \rfloor}$. Suppose that $v_i \in G(\ell)$ for some $v_i \in p$. Since $\rho(v_{i-1}) \leq \rho(v_i)$ for all $i > 1$, we have $v_{i-1} \in G(\ell')$ for some $\ell' \geq \ell$. We can therefore decompose p into a sequence of paths $p = \langle p_{\lceil r\lg\Delta\rceil}, \ldots, p_0\rangle$ where each $p_\ell \in p$ is a path in $G(\ell)$. By definition of SLL, the subdag $G(\ell)$ is a dag induced on a subgraph with degree at most $2^{\lfloor \ell/r\rfloor}$ by a random priority function.

By Corollary 6, the expected length of p_ℓ is $O(2^{\lfloor \ell/r\rfloor} + \lfloor \ell/r\rfloor \lg V/\lg\lg V)$. Linearity of expectation therefore implies that

$$\begin{aligned}\mathrm{E}[|p|] &= \sum_{\ell=0}^{\lceil r\lg\Delta\rceil} O\left(2^{\lfloor \ell/r\rfloor} + \lfloor \ell/r\rfloor \lg V/\lg\lg V\right) \\ &= O\left(\Delta + \lg^2\Delta\lg V/\lg\lg V\right) .\end{aligned}$$

Next, because at most $E/2^{\lfloor \ell/r\rfloor}$ vertices can have degree at least $2^{\lfloor \ell/r\rfloor}$, we have for $\ell > r\lg\sqrt{E}$ that the longest path through the subdag $G(\ell)$ is no longer than $E/2^{\lfloor \ell/r\rfloor}$. We thus conclude that

$$\begin{aligned}\mathrm{E}[|p|] &\leq \sum_{\ell=0}^{\lceil r\lg\sqrt{E}\rceil} O\left(2^{\lfloor \ell/r\rfloor}\right) + \sum_{\ell=\lceil r\lg\sqrt{E}\rceil}^{\infty} E/2^{\lfloor \ell/r\rfloor} \\ &\quad + \sum_{\ell=0}^{\lceil r\lg\Delta\rceil} O(\lfloor \ell/r\rfloor \lg V/\lg\lg V) \\ &= O\left(\sqrt{E} + \lg^2\Delta\lg V/\lg\lg V\right) .\end{aligned}$$ □

COROLLARY 12. *Given a graph $G = (V, E, \rho)$ for some $\rho \in$ SLL, JP-SLL colors all vertices in G with expected span $O(\lg\Delta\lg V + \lg\Delta(\min\{\sqrt{E}, \Delta\} + \lg^2\Delta\lg V/\lg\lg V))$.*

PROOF. The procedure SLL-ASSIGN-PRIORITIES calls the parallel loop on line 31 $O(\lg\Delta)$ times, each of which has expected span $O(\lg V)$. The proof then follows from Theorems 2 and 11. □

6. EMPIRICAL EVALUATION

This section evaluates the LLF and SLL ordering heuristics empirically using a suite of eight real-world and ten synthetic graphs. We describe the experimental setup used to evaluate JP-R, JP-LLF, and JP-SLL, and we compare their performance with GREEDY-FF, GREEDY-LF, and GREEDY-SL. We compare the ordering heuristics in terms of the quality of the colorings they produce and their execution times. We conclude that LLF and SLL produce colorings with quality comparable to LF and SL, respectively, and that JP-LLF and JP-SLL scale well. We also show that the engineering quality of our implementations appears to be competitive with COLPACK [28], a publicly available graph-coloring library. Our source code and data are available from `http://supertech.csail.mit.edu`.

Experimental setup

To evaluate the ordering heuristics, we implemented JP using Intel Cilk Plus [34] and engineered it to use the parallel ordering heuristics R, LLF, and SLL. To compare these parallel codes against their serial counterparts, we implemented GREEDY in C to use the FF, LF, or SL ordering heuristics. In order to empirically evaluate the potential parallel performance of the serial ordering heuristics, we also engineered JP to use FF, LF, or SL. We evaluated our implementations on a dual-socket Intel Xeon X5650 with a total of 12 processor cores operating at 2.67-GHz (hyperthreading disabled); 49 GB of DRAM; 2 12-MB L3-caches, each shared between 6 cores; and private L2- and L1-caches with 128 KB and 32 KB, respectively. Each measurement was taken as the median of 7 independent trials, and the averages of those measurements reported in Figure 7 were taken across 5 independent random seeds.

[5] As with LLF, the degree cutoff 2^d on line 30 of Figure 5 could be b^d for an arbitrary constant base b with no harm to the theoretical results. We explored the choice of base empirically, but found that there was only a minor dependence on b. Generally, JP-SLL smoothly transitions from the behavior of JP-SL for small b to the behavior of JP-R and for large b. We therefore chose $b = 2$ for our experiments because of its implementation simplicity.

				Greedy	JP						JP			
Graph		H	C_H	T_S	T_1	T_{12}	T_S/T_1	T_1/T_{12}	H'	$C_{H'}$	T_1	T_{12}	T_S/T_1	T_1/T_{12}
com-orkut	$\|E\| = 117.2$M	FF	175	2.23	4.16	0.817	0.54	5.09	R	132	4.44	0.817	0.50	5.43
	$\|E\|/\|V\| = 38.1$	LF	87	3.54	6.43	1.067	0.55	6.02	LLF	98	5.74	0.846	0.62	6.79
	$\Delta = 33{,}313$	SL	83	10.59	12.94	8.264	0.82	1.57	SLL	84	9.90	1.865	1.07	5.31
soc-LiveJournal1	$\|E\| = 42.9$M	FF	352	0.89	1.69	0.275	0.52	6.15	R	330	2.08	0.231	0.43	8.98
	$\|E\|/\|V\| = 8.8$	LF	323	2.34	2.89	0.365	0.81	7.91	LLF	326	2.23	0.286	1.05	7.80
	$\Delta = 20{,}333$	SL	322	4.69	4.76	2.799	0.98	1.70	SLL	327	4.03	0.704	1.16	5.73
europe-osm	$\|E\| = 36.0$M	FF	5	1.32	∞	∞	∞	∞	R	5	4.04	0.391	0.33	10.34
	$\|E\|/\|V\| = 0.7$	LF	4	17.15	5.16	0.587	3.33	8.79	LLF	4	4.93	0.473	3.48	10.41
	$\Delta = 9$	SL	3	19.87	∞	∞	∞	∞	SLL	3	7.28	1.232	2.73	5.91
cit-Patents	$\|E\| = 16.5$M	FF	17	0.50	0.99	0.152	0.50	6.47	R	21	1.08	0.163	0.46	6.67
	$\|E\|/\|V\| = 2.7$	LF	14	2.00	1.52	0.211	1.31	7.22	LLF	14	1.46	0.160	1.37	9.11
	$\Delta = 793$	SL	13	3.21	3.05	1.579	1.05	1.93	SLL	14	2.90	0.519	1.11	5.58
as-skitter	$\|E\| = 11.1$M	FF	103	0.24	0.55	0.109	0.45	5.00	R	81	0.58	0.114	0.42	5.07
	$\|E\|/\|V\| = 1.0$	LF	71	2.43	0.69	0.133	3.51	5.21	LLF	72	0.63	0.106	3.84	5.99
	$\Delta = 35{,}455$	SL	70	2.79	1.19	0.733	2.35	1.62	SLL	71	1.04	0.269	2.67	3.88
wiki-Talk	$\|E\| = 4.7$M	FF	102	0.09	0.23	0.046	0.38	4.99	R	85	0.28	0.053	0.31	5.28
	$\|E\|/\|V\| = 1.9$	LF	72	0.49	0.37	0.073	1.30	5.12	LLF	70	0.34	0.050	1.43	6.78
	$\Delta = 100{,}029$	SL	56	0.61	0.57	0.293	1.08	1.93	SLL	62	0.55	0.124	1.12	4.43
web-Google	$\|E\| = 4.3$M	FF	44	0.09	0.20	0.036	0.47	5.62	R	44	0.21	0.029	0.44	7.44
	$\|E\|/\|V\| = 4.7$	LF	45	0.25	0.29	0.042	0.88	6.85	LLF	44	0.27	0.030	0.94	8.92
	$\Delta = 6{,}332$	SL	44	0.47	0.53	0.278	0.89	1.92	SLL	44	0.50	0.093	0.94	5.44
com-youtube	$\|E\| = 3.0$M	FF	57	0.06	0.16	0.027	0.39	6.07	R	46	0.18	0.026	0.36	6.86
	$\|E\|/\|V\| = 2.6$	LF	32	0.25	0.24	0.040	1.03	6.12	LLF	33	0.22	0.028	1.11	7.97
	$\Delta = 28{,}754$	SL	28	0.35	0.36	0.181	0.98	1.99	SLL	28	0.35	0.073	1.01	4.75
constant1M-50	$\|E\| = 50.0$M	FF	33	0.90	1.70	0.230	0.53	7.40	R	32	1.93	0.255	0.47	7.55
	$\|E\|/\|V\| = 50.0$	LF	32	1.16	2.96	0.386	0.39	7.68	LLF	32	2.70	0.323	0.43	8.35
	$\Delta = 100$	SL	34	2.96	5.09	2.023	0.58	2.52	SLL	32	4.63	0.610	0.64	7.59
constant500K-100	$\|E\| = 50.0$M	FF	52	0.74	1.26	0.286	0.59	4.42	R	52	1.50	0.190	0.49	7.89
	$\|E\|/\|V\| = 99.9$	LF	52	0.84	2.55	0.444	0.33	5.73	LLF	52	2.01	0.273	0.42	7.34
	$\Delta = 200$	SL	53	1.97	3.50	1.435	0.56	2.44	SLL	52	3.33	0.498	0.59	6.69
graph500-5M	$\|E\| = 49.1$M	FF	220	1.83	2.86	0.560	0.64	5.11	R	220	2.99	0.558	0.61	5.35
	$\|E\|/\|V\| = 5.9$	LF	159	3.69	3.99	0.649	0.92	6.15	LLF	160	3.74	0.542	0.99	6.89
	$\Delta = 121{,}495$	SL	158	8.43	9.45	5.576	0.89	1.69	SLL	162	7.63	1.056	1.10	7.23
graph500-2M	$\|E\| = 19.2$M	FF	206	0.52	0.98	0.208	0.53	4.72	R	208	1.01	0.212	0.51	4.77
	$\|E\|/\|V\| = 9.2$	LF	153	0.98	1.34	0.221	0.73	6.06	LLF	154	1.24	0.151	0.79	8.19
	$\Delta = 70{,}718$	SL	153	2.22	2.72	1.559	0.81	1.75	SLL	156	2.25	0.324	0.99	6.94
rMat-ER-2M	$\|E\| = 20.0$M	FF	12	0.47	1.11	0.169	0.42	6.60	R	12	1.25	0.149	0.37	8.40
	$\|E\|/\|V\| = 9.5$	LF	11	1.07	1.72	0.204	0.62	8.45	LLF	12	1.63	0.198	0.66	8.25
	$\Delta = 44$	SL	11	2.22	3.07	1.362	0.72	2.25	SLL	11	3.13	0.506	0.71	6.18
rMat-G-2M	$\|E\| = 20.0$M	FF	27	0.48	0.88	0.130	0.55	6.74	R	27	0.91	0.144	0.53	6.33
	$\|E\|/\|V\| = 9.5$	LF	15	1.18	1.42	0.200	0.83	7.09	LLF	17	1.34	0.204	0.88	6.54
	$\Delta = 938$	SL	15	2.59	3.09	1.712	0.84	1.81	SLL	15	2.75	0.432	0.94	6.36
rMat-B-2M	$\|E\| = 19.8$M	FF	105	0.50	0.84	0.151	0.60	5.53	R	105	0.86	0.149	0.58	5.78
	$\|E\|/\|V\| = 9.4$	LF	67	1.00	1.28	0.191	0.79	6.68	LLF	68	1.18	0.149	0.85	7.94
	$\Delta = 14{,}868$	SL	67	2.41	2.84	1.691	0.85	1.68	SLL	68	2.38	0.376	1.01	6.31
big3dgrid	$\|E\| = 29.8$M	FF	4	0.41	1.68	0.173	0.24	9.69	R	7	1.66	0.178	0.25	9.31
	$\|E\|/\|V\| = 3.0$	LF	7	4.07	1.53	0.198	2.66	7.72	LLF	7	1.89	0.216	2.15	8.76
	$\Delta = 6$	SL	7	4.77	2.60	1.074	1.83	2.42	SLL	7	2.63	0.307	1.81	8.57
clique-chain-400	$\|E\| = 3.6$M	FF	399	0.05	0.09	0.224	0.51	0.40	R	399	0.09	0.012	0.50	7.77
	$\|E\|/\|V\| = 132.4$	LF	399	0.05	∞	∞	∞	∞	LLF	399	0.12	0.015	0.41	7.70
	$\Delta = 400$	SL	399	0.08	0.14	0.265	0.55	0.54	SLL	399	0.16	0.024	0.47	6.70
path-10M	$\|E\| = 10.0$M	FF	2	0.18	∞	∞	∞	∞	R	3	0.85	0.074	0.21	11.54
	$\|E\|/\|V\| = 1.0$	LF	3	2.49	0.76	0.092	3.26	8.27	LLF	3	0.98	0.083	2.54	11.87
	$\Delta = 2$	SL	2	2.58	∞	∞	∞	∞	SLL	3	1.36	0.169	1.90	8.04

Figure 7: Performance measurements for a set of real-world graphs taken from Stanford's SNAP project [40] are included above the center line. Five classes of synthetically generated graph are included below the center line: constant degree, rMat, 3D grid, clique chain and path. The column heading H denotes that the priority function used for the experiment in a particular row was produced by the ordering heuristic listed in the column. The average number of colors used by the corresponding ordering heuristic and graph is C_H. The time in seconds of Greedy, JP with 1 worker and with 12 workers is given by T_S, T_1 and T_{12}, respectively, where a value of ∞ indicates that the program crashed due to excessive stack usage. Details of the experimental setup and graph suite can be found in Section 6.

Graph	$\|V\|$	a	b	c	d
graph500-5M	5M	0.57	0.19	0.19	0.05
graph500-2M	2M	0.57	0.19	0.19	0.05
rMat-ER-2M	2M	0.25	0.25	0.25	0.25
rMat-G-2M	2M	0.45	0.15	0.15	0.25
rMat-B-2M	2M	0.55	0.15	0.15	0.15

Figure 6: Parameters for the generation of rMat graphs [17], where $a+b+c+d=1$ and $b=c$, when the desired graph is undirected. An rMat graph is built by adding $|E|$ edges independently at random using the following rule: Let k be the number of 1's in a binary representation of i. As each edge is added, the probability that the ith vertex v_i is selected as an endpoint is $(a+c)^k(b+d)^{\lg n-k}$.

These implementations were run on a suite of eight real-world graphs and ten synthetic graphs. The real-world graphs came from the Large Network Dataset Collection provided by Stanford's SNAP project [40]. The synthetic graphs consist of the adversarial graphs described in Section 4 and a set of graphs from three classes: constant degree, 3D grid, and "recursive matrix" (rMat) [14, 17]. The adversarial graphs — clique-chain-400 and path-10M — are described in Figure 4 with $\Delta = 400$ and Theorem 8 with $|V| = 10,000,000$, respectively. The constant-degree graphs — constant1M-50 and constant500K-100 — have 1M and 500K vertices and constant degrees of 100 and 200, respectively. These graphs were generated such that every pair of vertices is equally likely to be connected and every vertex has the same degree. The graph big3dgrid is a 3-dimensional grid on 10M vertices. The rMat graphs were generated using the parameters in Figure 6.

Coloring quality of R, LLF, and SLL

Figure 7 presents the coloring quality of the three parallel ordering heuristics R, LLF, and SLL alongside that of their serial counterparts FF, LF, and SL.

The number of colors used by LLF was comparable to that used by LF on the vast majority of the 18 graphs. Indeed, LLF produced colorings that were within 2 colors of LF on all synthetic graphs and all but 2 real-world graphs: com-orkut and soc-LiveJournal. Similarly, SLL produced colorings that were within 3 colors of SL for all synthetic graphs and all but 2 real-world graphs: soc-LiveJournal and wiki-Talk.

The soc-LiveJournal graph appears to benefit little from the ordering heuristics we considered. Every heuristic uses more than 300 colors, and the biggest difference between the number of colors used by any heuristic is less than 10.

The wiki-Talk and com-orkut graphs appear to benefit from ordering heuristics and illustrate what we believe is a coarse hierarchy of coloring quality in which FF < R < LLF < LF < SLL < SL. On com-orkut, LLF produced a coloring of size 98, which was better than the 175 and 132 colors used by FF and R, respectively, but not as good as the 87 colors used by LF. In contrast, SLL nearly matched the superior coloring quality of SL, producing a coloring of size 84. On wiki-Talk, SLL produced a coloring of size 62, which was better than LF, LLF, R, and FF by a margin of between 8 to 40 colors, but not as good as SL, which used only 56 colors. These trends appear to exist, in general, for most of the graphs in the suite.

Scalability of JP-R, JP-LLF, and JP-SLL

The parallel performance of JP was measured by computing the speedup it achieved on 12 cores and by comparing the 1-core runtimes of JP to an optimized serial implementation of GREEDY. These results are summarized in Figure 7.

Overall, JP-LLF obtains a geometric-mean speedup — the ratio of the runtime on 1 core to the runtime on 12 cores — of 7.83 on the eight real-world graphs and 8.08 on the ten synthetic graphs. Similarly, JP-SLL obtains a geometric-mean speedup of 5.36 and 7.02 on the real-world and synthetic graphs, respectively.

Figure 7 also includes scalability data for JP-FF, JP-LF, and JP-SL. Historically, JP-LF has been used with mixed success in practical parallel settings [2, 29, 35, 49]. Despite the fact that it offers little in terms of theoretical parallel performance guarantees, we have measured its parallel performance for our graph suite, and indeed JP-LF scales reasonably well: $\text{JP-LF}_1/\text{JP-LF}_{12} = 6.8$ as compared to $\text{JP-LLF}_1/\text{JP-LLF}_{12} = 8.0$ in geometric mean, not including clique-chain-400, which is omitted since JP-LF crashes due to excessive stack usage on clique-chain-400. The omission of clique-chain-400 highlights the dangers of using algorithms without good performance guarantees: it is difficult to know if the algorithm will behave badly given any particular input. In this respect, JP-FF is particularly vulnerable to adversarial inputs, as we can see by the fact that it crashes on europe-osm, which is not even intentionally adversarial. We also see this vulnerability with JP-SL, as well as generally poor scalability on the entire suite.

To measure the overheads introduced by using a parallel algorithm, the runtime T_1 of JP on 1 core was compared with the runtime T_S of an optimized implementation of GREEDY. This comparison was performed for each of the three parallel ordering heuristics we considered: R, LLF, and SLL. The serial runtime of GREEDY using FF is 2.5 times faster than JP-R on 1 core for the eight real-world graphs and 2.3 times faster on the ten synthetic graphs. We conjecture that GREEDY gains its advantage due to the spatial-locality advantage that results from processing the vertices in the linear order they appear in the graph representation. JP-LLF and JP-SLL on 1 core, however, are actually faster than GREEDY with LF and SL by 43.3% and 19% on the eight real-world graphs and 6% and 3% on the whole suite, respectively.

In order to validate that our implementation of GREEDY is a credible baseline, we compared it with a publicly available graph-coloring library, COLPACK [28], developed by Gebremedhin *et al.* and found that the two implementations appeared to achieve similar performance. For example, using the SL ordering heuristic, GREEDY is 19% faster than COLPACK in geometric-mean across the graph suite, though GREEDY is slower on 5 of the 16 graphs and as much 2.22 times slower for as-skitter.

7. IMPLEMENTATION TECHNIQUES

This section describes the techniques we employed to implement JP and GREEDY for the evaluation in Section 6. We describe three techniques — join-trees [23], bit-vectors, and software prefetching — that improve the practical performance of JP. Where applicable, these same techniques were used to optimize the implementation of GREEDY. Overall, applying these techniques yielded a speedup of between 1.6 and 2.9 for JP and a speedup of between 1.2 and 1.6 for GREEDY on the rMat-G-2M, rMat-B-2M, web-Google, and as-skitter graphs used in Section 6.

Join trees for reducing memory contention

Although the theoretical analysis of JP in Section 2 does not concern itself with contention, the implementation of JP works to mitigate overheads due to contention. The pseudocode for JP in Figure 2 shows that each vertex u in the graph has an associated counter *u.counter*. Line 17 of JP-COLOR executes a JOIN operation on *u.counter*. Although Section 2 describes how JOIN can treat *u.counter* as a join counter [20] and update *u.counter* using an atomic decrement and fetch operation, the cache-coherence protocol [47] on the machine serializes such atomic operations, giving rise to potential memory contention. In particular, memory con-

```
GREEDY-SD($G$)
36  let $G = (V, E)$
37  for $v \in V$
38      $v.adjColors = \emptyset$
39      $v.adjUncolored = v.adj$
40      PUSHORADDKEY($v, Q[0][|v.adjUncolored|]$)
41  $s = 0$
42  while $s \geq 0$
43      $v$ = POPORDELKEY($Q[s][\max \text{KEYS}(Q[s])]$)
44      $v.color = \min(\{1, 2, \ldots, |v.adjUncolored| + 1\} - v.adjColors)$
45      for $u \in v.adjUncolored$
46          REMOVEORDELKEY($u, Q[|u.adjColors|][|u.adjUncolored|]$)
47          $u.adjColors = u.adjColors \cup \{v.color\}$
48          $u.adjUncolored = u.adjUncolored - \{v\}$
49          PUSHORADDKEY($u, Q[|u.adjColors|][|u.adjUncolored|]$)
50          $s = \max\{s, |u.adjColors|\}$
51      while $s \geq 0$ and $Q[s] == \emptyset$
52          $s = s - 1$
```

Figure 8: The GREEDY-SD algorithm computes a coloring for the input graph $G = (V, E)$ using the SD heuristic. Each uncolored vertex $v \in V$ maintains a set *v.adjColors* of colors used by its neighbors and a set *v.adjUncolored* of uncolored neighbors of v. The PUSHORADDKEY method adds a specified key, if necessary, and then adds an element to that key's associated set. The POPORDELKEY and REMOVEORDELKEY methods remove an element from a specified key's associated set, deleting that key if the set becomes empty. The variable s maintains the maximum saturation degree of G.

tention may harm the practical performance of JP on graphs with large-degree vertices.

Our implementation of JP mitigates overheads due to contention by replacing each join counter *u.counter* with a join tree having $\Theta(|u.pred|)$ leaves. In particular, each join tree was sized such that an average of 64 predecessors of u map to each leaf through a hash function that maps predecessors to random leaves. We found that the join tree reduces T_1 for JP by a factor of 1.15 and reduces T_{12} for JP by between 1.1 and 1.3.

Bit vectors for assigning colors

To color vertices more efficiently, the implementation of JP uses vertex-local bit vectors to store information about the availability of low-numbered colors. Because JP assigns to each vertex the lowest-numbered available color, vertices tend to be colored with low-numbered colors. To take advantage of this observation, we store a 64-bit word per vertex u to track the colors in the range $\{1, 2, \ldots, 64\}$ that have already been assigned to a neighbor of u. The bit vector on *u.vec* is computed as a "self-timed" OR reduction that occurs during updates on u's join tree. Effectively, as each predecessor v of u executes JOIN on u's join tree, if *v.color* is in $\{1, 2, \ldots, 64\}$, then v OR's the word $2^{v.color-1}$ into *u.vec*. When GET-COLOR(u) subsequently executes, GET-COLOR first scans for the lowest unset bit in *u.vec* to find the minimum color in $\{1, 2, \ldots, 64\}$ not assigned to a neighbor of u. Only when no such color is available does GET-COLOR(u) scan its predecessors to assign a color to u.

We discovered that a large fraction of vertices in a graph can be colored efficiently using this practical optimization. We found that this optimization improved T_{12} for JP by a factor of 1.4 to 2.2, and a similar optimization sped up the implementation of GREEDY by a factor of 1.2 to 1.6.

Software prefetching

We used software prefetching to improve the latency of memory accesses in JP. In particular, JP uses software prefetching to mitigate the latency of the indirect memory access encountered when accessing the join trees of the successors of a vertex v on line 16 of JP-COLOR in Figure 2. This optimization improves T_{12} for JP by a factor of 1.2 to 1.5.

Interestingly, our implementation of GREEDY did not appear to benefit from using software prefetching in a similar context, specifically, to access the predecessors of a vertex on line 4 of GREEDY in Figure 1. We suspect that because GREEDY only reads the predecessors of a vertex on this line and does not write them, the processor hardware is able to generate many such reads in parallel, thereby mitigating the latency penalty introduced by cache misses.

8. THE SD HEURISTIC

Our experiments with serial heuristics detailed in the Appendix (Section 12) indicate that the SD heuristic tends to provide colorings with higher quality than the other heuristics we have considered, confirming similar findings by Gebremedhin and Manne [27]. Although we leave the problem of devising a good parallel algorithm for SD as an open question, we were able to devise a linear-time serial algorithm for the problem, despite conjectures in the literature [19, 29] that superlinear time is required. This section briefly describes our linear-time serial algorithm for SD.

Figure 8 gives pseudocode for the GREEDY-SD algorithm, which implements the SD heuristic. Rather than trying to define a priority function for SD, the figure gives the coloring algorithm GREEDY-SD itself, since the calculation of such a priority function would color the graph as a byproduct. At any moment during the execution of the algorithm, the ***saturation degree*** of a vertex v as the number $|v.adjColors|$ of distinct colors of v's neighbors, and the ***effective degree*** of v as $|v.adjUncolored|$, its degree in the as yet uncolored graph.

The main loop of GREEDY-SD (lines 42–52) first removes a vertex v of maximum saturation degree from Q (line 43) and colors it (line 44). It then updates each uncolored neighbor $u \in v.adjUncolored$ of v (lines 45–50) in three steps. First, it removes u from Q (line 46). Next, it updates the set *u.adjUncolored* of u's ***effective neighbors*** — u's uncolored neighbors in G — and the set *u.adjColors* of colors used by u's neighbors (lines 47–48). Finally, it enqueues u in Q based on u's updated information (lines 49–50).

The crux of GREEDY-SD lies in the operation of the queue data structure Q, which is organized as an array of ***saturation tables***, each of which supports the three methods PUSHORADDKEY, POPORDELKEY, and REMOVEORDELKEY described in the caption of Figure 8. A saturation table can support these operations in $\Theta(1)$ time and allow its keys K to be read in $\Theta(K)$ time. At the start of each main loop iteration, entry $Q[i]$ stores the uncolored vertices in the graph with saturation degree i in a saturation table. The PUSHORADDKEY, POPORDELKEY, and REMOVEORDELKEY methods maintain the invariant that, for each table $Q[i]$, each key $j \in \text{KEYS}(Q[i])$ is associated with a nonempty set of vertices, such that each vertex $v \in Q[i][j]$ has saturation degree i and effective degree j.

THEOREM 13. GREEDY-SD *colors a graph* $G = (V, E)$ *according to the* SD *ordering heuristic in* $\Theta(V + E)$ *time.*

PROOF. PUSHORADDKEY, POPORDELKEY, and REMOVEORDELKEY operate in $\Theta(1)$ time, and a given saturation table's key set K can be read in $\Theta(K)$ time. Line 43 can thus find a vertex v with maximum saturation degree s in $\Theta(|\text{KEYS}(Q[s])|)$ time. Line 44 can color v in $\Theta(\deg(v))$ time, and lines 50–52 maintain s in $\Theta(s)$ time. Because $s + |\text{KEYS}(Q[s])| \leq \deg(v)$, lines 42–52 evaluate v in $\Theta(\deg(v))$ time. The handshaking lemma [21, p. 1172–3] implies the theorem, because each vertex in V is evaluated once. □

	C							T_S						
Graph	FF	R	LF	ID	SL	SD	*Spark*	FF	R	LF	ID	SL	SD	*Spark*
com-orkut	175	132	87	86	83	76		2.23	3.39	3.54	44.13	10.59	46.60	
soc-LiveJournal1	352	330	323	325	322	326		0.89	2.05	2.34	17.93	4.69	19.75	
europe-osm	5	5	4	4	3	3		1.32	13.36	17.15	48.59	19.87	52.73	
cit-Patents	17	21	14	14	13	12		0.50	1.62	2.00	9.82	3.21	10.08	
as-skitter	103	81	71	72	70	70		0.24	1.70	2.43	9.41	2.79	9.94	
wiki-Talk	102	85	72	57	56	51		0.09	0.35	0.49	2.79	0.61	2.90	
web-Google	44	44	45	45	44	44		0.09	0.22	0.25	1.68	0.47	1.77	
com-youtube	57	46	32	28	28	26		0.06	0.19	0.25	1.50	0.35	1.55	
constant1M-50	33	32	32	34	34	26		0.90	1.13	1.16	16.07	2.96	17.23	
constant500K-100	52	52	52	55	53	44		0.74	0.88	0.84	14.20	1.97	15.51	
graph500-5M	220	220	159	157	158	147		1.83	3.14	3.69	25.19	8.43	35.29	
graph500-2M	206	208	153	152	153	141		0.52	0.77	0.98	8.09	2.22	11.68	
rMat-ER-2M	12	12	11	11	11	8		0.47	0.93	1.07	10.10	2.22	9.13	
rMat-G-2M	27	27	15	15	15	11		0.48	0.92	1.18	9.17	2.59	9.07	
rMat-B-2M	105	105	67	67	67	59		0.50	0.83	1.00	8.44	2.41	8.64	
big3dgrid	4	7	7	4	7	5		0.41	3.34	4.07	13.61	4.77	15.30	
clique-chain-400	399	399	399	399	399	399		0.05	0.05	0.05	0.81	0.08	2.06	
path-10M	2	3	3	2	2	2		0.18	1.95	2.49	7.34	2.58	7.96	

Figure 9: Performance measurements for six serial ordering heuristics used by GREEDY, where measurements for real-world graphs appear above the center line and those for synthetic graphs appear below. The columns under the heading *C* present the average number of colors obtained by each ordering heuristic. The columns under the heading T_S present the average serial running time for each heuristic. The "*Spark*" columns under the *C* and T_S headings contain bar graphs that pictorially represent the coloring quality and serial running time, respectively, for each of the ordering heuristics. The height of the bar for the coloring quality C_H of ordering heuristic *H* is proportional to C_H. The bar heights are similar for T_S except that the log of times are used. Section 6 details the experimental setup and graph suite used.

9. RELATED WORK

Parallel coloring algorithms have been explored extensively in the distributed computing domain [3,5,30,31,35,38,39,41]. These algorithms are evaluated in the message-passing model, where nodes are allowed unlimited local computation and exchange messages through a sequence of synchronized rounds. Kuhn [38] and Barenboim and Elkin [5] independently developed $O(\Delta + \lg^* n)$-round message passing algorithms to compute a deterministic greedy coloring.

Several greedy coloring algorithms have been described in synchronous PRAM models. Goldberg *et al.* [30] describe an algorithm for finding a greedy coloring of $O(1)$-degree graphs in $O(\lg n)$ time in the EREW PRAM model using a linear number of processors. They observe that their technique can be applied recursively to color Δ-degree graphs in $O(\Delta \lg \Delta \lg n)$ time. Their strategy incurs $\Omega(\lg \Delta(V+E))$ (superlinear) work, however.

Catalyurek *et al.* [14] present the algorithm ITERATIVE, which first speculatively colors a graph *G* and then fixes coloring conflicts, that is, corrects the coloring where two adjacent vertices are assigned the same color. The process of fixing conflicting colors can introduce new conflicts, though the authors observe empirically that comparatively few iterations suffice to find a valid coloring. We ran ITERATIVE on our test system and found that JP-LLF uses 13% fewer colors and takes 19% less time in geometric mean of number of colors and relative time, respectively, over all graphs in our test suite. Furthermore, we found that JP-SLL uses 17% fewer colors, but executes in twice the time of ITERATIVE. We do not know the extent to which the optimizations enjoyed by our algorithms could be adopted by speculative-coloring algorithms, however, and so it is likely too soon to draw conclusions about comparisons between the strategies.

10. CONCLUSION

Because of the importance of graph coloring, considerable effort has been invested over the years to develop ordering heuristics for serial graph-coloring algorithms. For the traditional "serial" LF and SL ordering heuristics, we have developed "parallel" analogs — the LLF and SLL heuristics, respectively — which approximate the traditional orderings, generating colorings of comparable quality while offering provable guarantees on parallel scalability. The correspondence between serial ordering heuristics and their parallel analogs is fairly direct for LF and LLF . LLF colors any two vertices whose degrees differ by more than a factor of 2 in the same order as LF. In this sense, LLF can be viewed as a simple coarsening of the vertex ordering used by LF. Although SLL is inspired by SL, and both heuristics tend to color vertices of smaller degree later, the correspondence between SL and SLL is not as straightforward. We relied on empirical results to determine the degree to which SLL captures the salient properties of SL.

We had hoped that the coarsening strategy LLF and SLL embody would generalize to the other serial ordering heuristics, and we are disappointed that we have not yet been able to devise parallel analogs for the other ordering heuristics, and in particular, for SD. Because the SD heuristic appears to produce better colorings in practice than all of the other serial ordering heuristics, SD appears to capture an important phenomenon that the others miss.

The problem with applying the coarsening strategy to SD stems from the way that SD is defined. Because SD determines the order to color vertices while serially coloring the graph itself, it seems difficult to parallelize, and it is not clear how SD might correspond to a possible parallel analog. Thus, it remains an intriguing open question as to whether a parallel ordering heuristic exists that captures the same "insights" as SD while offering provable guarantees on scalability.

11. ACKNOWLEDGMENTS

Thanks to Guy Blelloch of Carnegie Mellon University for sharing utility functions from his Problem Based Benchmark Suite with us [50]. Thanks to Aydın Buluç of Lawrence Berkeley Laboratory for helping us in our search for collections of large sparse graphs. Thanks to Mahantesh Halappanavar of Pacific Northwest National Laboratory for providing us with the code for ITERATIVE [14]. Thanks to Assefaw Gebremedhin for input regarding the publicly available graph-coloring library ColPack [28]. Thanks to Jack Dennis of MIT CSAIL for helping us track down early work on parallel sorting and join counters. Thanks to Jeremy Fineman for helpful discussions on the amortized analysis of SD. Thanks to Angelina Lee and Justin Zhang of MIT CSAIL and Julian Shun and Harsha Vardhan Simhadri of Carnegie Mellon University for several helpful discussions.

12. APPENDIX: PERFORMANCE OF SERIAL ORDERING HEURISTICS

Figure 9 summarizes our empirical evaluation of Greedy run on our suite of real-world and synthetic graphs using the six ordering heuristics from Section 1. The measurements were taken using the same machine and methodology as was used for Figure 7. As Figure 9 shows, we found that, in order, FF, R, LF, SL, and SD generally produce better colorings at the cost of greater running times. ID was outperformed in both time and quality by SL. The figure indicates that LF tends to produce better colorings than FF and R at some performance cost, and SL produces better colorings than LF at additional cost. We found that SD produces the best colorings overall, at the cost of a 4.5 geometric-mean slowdown versus SL.

13. REFERENCES

[1] L. Adams and J. Ortega. A multi-color SOR method for parallel computation. In *ICPP*, 1982.

[2] J. R. Allwright, R. Bordawekar, P. D. Coddington, K. Dincer, and C. L. Martin. A comparison of parallel graph coloring algorithms. Technical report, Northeast Parallel Architecture Center, Syracuse University, 1995.

[3] N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Algorithms*, 1986.

[4] E. M. Arkin and E. B. Silverberg. Scheduling jobs with fixed start and end times. *Discrete Applied Mathematics*, 1987.

[5] L. Barenboim and M. Elkin. Distributed $(\Delta+1)$-coloring in linear (in Δ) time. In *ACM STOC*, 2009.

[6] S. Berchtold, C. Böhm, B. Braunmüller, D. A. Keim, and H.-P. Kriegel. Fast parallel similarity search in multimedia databases. In *ACM SIGMOD Int. Conf. on Management of Data*, 1997.

[7] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, 1989.

[8] G. E. Blelloch, J. T. Fineman, and J. Shun. Greedy sequential maximal independent set and matching are parallel on average. In *ACM SPAA*, 2012.

[9] R. D. Blumofe and C. E. Leiserson. Space-efficient scheduling of multithreaded computations. *SICOMP*, 1998.

[10] R. D. Blumofe and C. E. Leiserson. Scheduling multithreaded computations by work stealing. *JACM*, 1999.

[11] D. Brélaz. New methods to color the vertices of a graph. *CACM*, 1979.

[12] R. P. Brent. The parallel evaluation of general arithmetic expressions. *JACM*, 1974.

[13] P. Briggs. *Register allocation via graph coloring*. PhD thesis, Rice University, 1992.

[14] Ü. V. Çatalyürek, J. Feo, A. H. Gebremedhin, M. Halappanavar, and A. Pothen. Graph coloring algorithms for muti-core and massively multithreaded architectures. *CoRR*, 2012.

[15] G. J. Chaitin. Register allocation & spilling via graph coloring. In *ACM SIGPLAN Notices*, 1982.

[16] G. J. Chaitin, M. A. Auslander, A. K. Chandra, J. Cocke, M. E. Hopkins, and P. W. Markstein. Register allocation via coloring. *Computer Languages*, 1981.

[17] D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-MAT: A recursive model for graph mining. In *SDM*. SIAM, 2004.

[18] R. Cole and U. Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Inf. Control*, 1986.

[19] T. Coleman and J. Moré. Estimation of sparse Jacobian matrices and graph coloring problems. *SIAM J. Numer. Anal.*, 1983.

[20] M. E. Conway. A multiprocessor system design. In *AFIPS*, 1963.

[21] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, third edition, 2009.

[22] K. Diks. A fast parallel algorithm for six-colouring of planar graphs. In *Mathematical Foundations of Computer Science*. 1986.

[23] C. Dwork, M. Herlihy, and O. Waarts. Contention in shared memory algorithms. In *STOC*, 1993.

[24] D. L. Eager, J. Zahorjan, and E. D. Lazowska. Speedup versus efficiency in parallel systems. *IEEE Trans. Comput.*, 1989.

[25] M. Fischetti, S. Martello, and P. Toth. The fixed job schedule problem with spread-time constraints. *Operations Research*, 1987.

[26] M. Garey, D. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1976.

[27] A. H. Gebremedhin and F. Manne. Scalable parallel graph coloring algorithms. *Concurrency: Practice and Experience*, 2000.

[28] A. H. Gebremedhin, D. Nguyen, M. M. A. Patwary, and A. Pothen. ColPack: Software for graph coloring and related problems in scientific computing. *ACM Trans. on Mathematical Software*, 2013.

[29] R. K. Gjertsen Jr., M. T. Jones, and P. E. Plassmann. Parallel heuristics for improved, balanced graph colorings. *JPDC*, 1996.

[30] A. V. Goldberg, S. A. Plotkin, and G. E. Shannon. Parallel symmetry-breaking in sparse graphs. In *SIAM J. Disc. Math*, 1987.

[31] M. Goldberg and T. Spencer. A new parallel algorithm for the maximal independent set problem. *SICOMP*, 1989.

[32] R. L. Graham. Bounds for certain multiprocessing anomalies. *The Bell System Technical Journal*, 1966.

[33] M. Herlihy and N. Shavit. *The Art of Multiprocessor Programming*. Morgan Kaufmann Publishers Inc., 2008.

[34] Intel. Intel Cilk Plus. Available from `http://software.intel.com`, 2013.

[35] M. T. Jones and P. E. Plassmann. A parallel graph coloring heuristic. *SIAM Journal on Scientific Computing*, 1993.

[36] M. T. Jones and P. E. Plassmann. Scalable iterative solution of sparse linear systems. *Parallel Computing*, 1994.

[37] T. Kaler, W. Hasenplaugh, T. B. Schardl, and C. E. Leiserson. Executing dynamic data-graph computations deterministically using chromatic scheduling. In *SPAA*, 2014.

[38] F. Kuhn. Weak graph colorings: distributed algorithms and applications. In *ACM SPAA*, 2009.

[39] F. Kuhn and R. Wattenhofer. On the complexity of distributed graph coloring. In *PODC*, 2006.

[40] J. Leskovec. SNAP: Stanford Network Analysis Platform. Available from `http://snap.stanford.edu/data/index.html`, 2013.

[41] N. Linial. Locality in distributed graph algorithms. *SICOMP*, 1992.

[42] L. Lovász, M. Saks, and W. T. Trotter. An on-line graph coloring algorithm with sublinear performance ratio. *Discrete Math.*, 1989.

[43] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 1986.

[44] D. Marx. Graph colouring problems and their applications in scheduling. *John von Neumann Ph.D. Students Conf.*, 2004.

[45] D. W. Matula and L. L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *JACM*, 1983.

[46] J. Mitchem. On various algorithms for estimating the chromatic number of a graph. *The Computer Journal*, 1976.

[47] M. S. Papamarcos and J. H. Patel. A low-overhead coherence solution for multiprocessors with private cache memories. In *ISCA*, 1984.

[48] Y. Saad. *SPARSKIT: A basic toolkit for sparse matrix computations*. Research Institute for Advanced Computer Science, NASA Ames Research Center, 1990.

[49] A. Sariyuce, E. Saule, and U. Catalyurek. Improving graph coloring on distributed-memory parallel computers. In *HiPC*, 2011.

[50] J. Shun, G. E. Blelloch, J. T. Fineman, P. B. Gibbons, A. Kyrola, H. V. Simhadri, and K. Tangwongsan. Brief announcement: the Problem Based Benchmark Suite. In *SPAA*, 2012.

[51] 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*, 1967.