Paper Presentation
High-Performance Parallel Graph Coloring with Strong Guarantees on Work, Depth, and Quality by Besta et. al.

Presentation by Viktor Urvantsev
Graph Coloring

- Also known as vertex coloring
- **vertex coloring**: an assignment of vertices where no two neighboring vertices share the same color
- **k-coloring**: a vertex coloring of a graph using k distinct colors
- **chromatic number (χ(G))**: the lowest number of colors k that yields a successful k-coloring
- **optimal coloring (or coloring problem)**: the problem of finding χ(G) given a graph G
- optimal coloring is an NP-complete problem
Heuristics

- Used to compute colorings
- **greedy heuristic**: for all colors, choose the smallest color not already chosen by a neighbor for vertex v
  - Gives a guarantee for coloring of a graph G with at most $\Delta + 1$ colors, where $\Delta$ is the maximum degree of the graph
- **vertex ordering heuristic**: a heuristic which decides in which order the vertices are colored. Some examples are
  - *first fit (FF)*: coloring via the default order in the vertices
  - *largest-degree-first (LF)*: orders vertices based on degree in descending order
  - *random (R)*: chooses vertices uniformly at random
  - *incidence-degree (ID)*: largest number of uncolored neighbors first
  - *saturation-degree (SD)*: largest number of distinct colors used by neighbors first
  - *smallest-degree-last (SL)*: removes lowest degree vertices, colors remaining graph, colors removed vertices
- Sadly, with all of these combined with greedy, these algorithms have **no parallelism**
Parallel Graph Coloring Algorithms

- Jones and Plassman combined past heuristic work with parallel maximum independent set deriving algorithms to make **JP**, a parallel graph coloring algorithm
  - Colors a vertex once all of its neighbors that come later in the ordering have been colored
- JP combined with random vertex ordering (**JP-R**) ran with $O(n+m)$ work, $O(\log n/ \log \log n)$ depth for constant degree graphs ($n =$ number of vertices, $m =$ number of edges)
- Hasenplaugh et. al. use approximate LF and SL to create **JP-LLF** and **JP-SLL**, which improve upon work and depth bounds over JP-R
- Another approximate SL ordering was used to create **JP-ASL**, but offers no work or depth bounds (scary)
Problems With Aforementioned Algorithms

- None have strong guarantees of quality!
  - Quality: the closeness of a result to the chromatic number (true optimal coloring)
  - JP-R may yield poor quality
  - JP-LF and JP-SL yield better quality, but can balloon in performance (run in $\Omega(n)$ or $\Omega(\Delta^2)$ in some instances)

- None of the above algorithms have any upper bound coloring quality guarantees
What The Paper Offers

- The first graph coloring algorithms with provably good bounds on work, depth, and quality
- These algorithms do this using a new vertex ordering heuristic: provably approximate degeneracy ordering (ADG)
- An algorithm using this heuristic with JP, called JP-ADG, with proven strong bounds on work, depth and quality
- An algorithm using this ordering with speculative coloring, called DEC-ADG, with proven strong bounds on work, depth and quality
- “A use case of how ADG can seamlessly enhance an existing state-of-the-art graph coloring scheme (DEC-ADG-ITR)”
- Extensive theoretical analysis of and experimentation with graph coloring algorithms
- Superior coloring quality in practice
Fundamental Concepts: Graph Model and Representation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G$</td>
<td>A graph $G = (V, E)$; $V$ and $E$ are sets of vertices and edges.</td>
</tr>
<tr>
<td>$G[U]$</td>
<td>$G[U] = (U, E[U])$ is a subgraph of $G$ induced on $U \subseteq V$.</td>
</tr>
<tr>
<td>$n, m$</td>
<td>Numbers of vertices and edges in $G$; $</td>
</tr>
<tr>
<td>$\Delta, \delta, \bar{\delta}$</td>
<td>Maximum degree, minimum degree, and average degree of $G$.</td>
</tr>
<tr>
<td>$d$</td>
<td>The degeneracy of $G$.</td>
</tr>
<tr>
<td>$\text{deg}(v), N(v)$</td>
<td>The degree and the neighborhood of a vertex $v \in V$.</td>
</tr>
<tr>
<td>$\text{deg}_U(v)$</td>
<td>The degree of $v$ in a subgraph induced by the vertex set $U \subseteq V$.</td>
</tr>
<tr>
<td>$N_U(v)$</td>
<td>The neighborhood of $v$ in a subgraph induced by $U \subseteq V$.</td>
</tr>
<tr>
<td>$\rho_X(v)$</td>
<td>A priority function $V \rightarrow \mathbb{R}$ associated with vertex ordering $X$.</td>
</tr>
<tr>
<td>$P$</td>
<td>The number of processors (in a given PRAM machine).</td>
</tr>
</tbody>
</table>

**TABLE I:** Selected symbols used in the paper. When we use a symbol in the context of a specific loop iteration $\ell$, we add $\ell$ in brackets or as subscript (e.g., $\hat{\delta}_\ell$ is $\hat{\delta}$ in iteration $\ell$).
Fundamental Concepts: Degeneracy, Coreness, and Related Concepts

- **s-degenerate**: a graph is s-degenerate if, in each of its induced subgraphs, there is a vertex with a degree of at most s.
- **degeneracy**: the degeneracy d of a graph is the smallest s where the graph is still s-degenerate
- **degeneracy ordering**: an ordering where there are at most d neighbors of each vertex ordered higher than it
- **k-approximate degeneracy ordering**: same as exact, but at most k * d neighbors
- **partial k-approximate degeneracy ordering**: same as k-approximate, but certain vertices can be ranked equally
- **k-core**: A connected component that is left over after iteratively removing vertices with degree less than k from G
- **coreness**: the largest possible k, such that v is part of a subgraph S of G with minimum degree k
Fundamental Concepts: Models For Algorithm Analysis

- Compute model: DAG model of dynamic multithreading
- Machine model: Ideal parallel computer
- Work-Depth Analysis for bounding runtimes
  - Uses both concurrent reads exclusive writes (CREW) and concurrent reads concurrent writes (CRCW)
Fundamental Concepts: Compute Primitives

- **Reduce**
  - Given a set $S$, it finds and returns the sum of the set in $O(n)$ work and $O(\log n)$ depth
  - Used to implement Count($S$), which simply returns the size of $S$

- **DecrementAndFetch (DAF)**
  - Atomically decrements its operand and returns a new value
  - Used to implement Join to synchronize processors
Fundamental Concepts: Randomization

- **Monte Carlo Algorithms**: algorithms which return the correct result w.h.p.
- **Las Vegas Algorithms**: algorithms that always return the correct answer but have probabilistic runtime bounds
  - JP is a Las Vegas algorithm
- **Coupling**: A coupling of two random variables $X$ and $Y$ is defined as a new variable $(X', Y')$ over the joint probability space, such that the marginal distribution of $X'$ and $Y'$ coincides with the distribution of $X$ and $Y$ respectively.
Parallel Approximate Degeneracy Ordering

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

- D is an array
- U and R are n-bit dense bitmaps
  - Updates and contains() functions run in $O(1)$ time
  - Constructing R takes $O(1)$ depth and $O(|U|)$ work
  - $U = U \setminus R$ takes $O(1)$ depth and $O(|R|)$ work
  - Average degree calculation takes $O(\log n)$ depth and $O(|U|)$ work
Parallel Approximate Degeneracy Ordering: Depth

- $O(\log n)$ depth of contents in the while loop
- While loop runs $O(\log n)$ iterations
- Overall depth: $O(\log^2 n)$

**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 $\ell$ of the algorithm we can have at most $\frac{n}{1+\varepsilon}$ vertices with a degree larger than $(1 + \varepsilon)\hat{d}_\ell$. 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 $\hat{d}_\ell$. For vertices with a degree exactly $(1 + \varepsilon)\hat{d}_\ell$ we get $\frac{n}{1+\varepsilon} \cdot (1 + \varepsilon)\hat{d}_\ell = n\hat{d}_\ell$, 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 $\leq (1 + \varepsilon)\hat{d}_\ell$, 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. To see this explicitly, one can define a simple recurrence relation for the number of iterations $T(n) \leq 1 + T\left(\frac{\log n}{1+\varepsilon}\right)$, $T(1) = 1$; solving it gives $T(n) \leq \left\lceil \frac{\log n}{\log (1+\varepsilon)} + 1 \right\rceil \in O(\log n)$. $\square$
Parallel Approximate Degeneracy Ordering: Work

- Each iteration of the while loop does $O((\text{deg } v \text{ for all } v \text{ in } R) + U_i)$ work.
- $(\text{deg } v \text{ for all } v \text{ in } R)$ for all $i = 1$ to $k = O(m)$
- $U_i$ from $i=1$ to $k$ can be bound by a geometric series, implying that it is still $O(n)$
- Overall runtime: $O(m + n)$

Work

The proof of work is similar; it also uses the fact that a constant fraction of vertices is removed in each iteration. Intuitively, (1) we show that each while loop iteration performs $O(\left(\sum_{v \in R} \text{deg}(v)\right) + |U_i|)$ work (where $U_i$ is the set $U$ in iteration $i$), and (2) we bound $\sum_{i=1}^{k} |U_i|$ by a geometric series, implying that it is still in $O(n)$.

Lemma 2. For a constant $\varepsilon > 0$, ADG does $O(n + m)$ work in the CRCW setting.

Proof. Let $k$ be the number of iterations we perform and let $U_i$ be the set $U$ in iteration $i$. To calculate the total work performed by ADG, we first consider the work in one iteration. As explained in “Design Details”, deriving the average degree takes $O(|U_i|)$ work in one iteration. Initializing $R$ takes $O(|U_i|)$ and removing $R$ from $U_i$ takes $O(|R|)$. UPDATE takes $O\left(\sum_{v \in R} \text{deg}(v)\right)$ work. Thus, the total work in one iteration is in $O\left(\sum_{v \in R} \text{deg}(v)\right) + |U_i|$. As each vertex becomes included in $R$ in a single unique iteration, this gives $\sum_{i=1}^{k} \sum_{v \in R_i} \text{deg}(v) \in O(m)$. Moreover, since we remove a constant number of vertices in each iteration from $U$ (at least $\frac{\varepsilon}{1+\varepsilon}$ as shown above in the proof of the depth of ADG), we can bound $\sum_{i=1}^{k} |U_i|$ by a geometric series, implying that it is still in $O(n)$. This can be seen from the fact that $\sum_{i=1}^{k} |U_i| \leq \sum_{i=0}^{k} \left(\frac{1}{1+\varepsilon}\right)^i n \leq \sum_{i=0}^{\infty} \left(\frac{1}{1+\varepsilon}\right)^i n = \frac{(1+\varepsilon)n}{\varepsilon}$ if $\frac{1}{1+\varepsilon} < 1$ (which holds as $\varepsilon > 0$). Ultimately, we have $O\left(\sum_{i=0}^{k} \left(\sum_{v \in R_i} \text{deg}(v)\right) + |U_i|\right) \in O(m + O(n) \in O(m + n)$.
Parallel Approximate Degeneracy Ordering: Quality

- Approximation ratio of $2(1+\varepsilon)$
- All vertices removed in a step have a degree of at most $(1+\varepsilon) \times$ the average degree of vertices in the subgraph at that point, which is upper bounded by $2d$
- Therefore, each vertex has at most $2(1+\varepsilon)d$ neighbors that are ranked equal or higher.

**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 3.** 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 $\text{deg}_U(v) \leq d$. 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 $\hat{\delta}(G[U]) = \frac{1}{|U|} \sum_{v \in U} \text{deg}_U(v) \leq 2d$.  

**Lemma 4.** ADG computes a partial $2(1 + \varepsilon)$-approximate degeneracy ordering of $G$.

*Proof.* By the definition of $R$ (Line 13), all vertices removed in step $\ell$ have a degree of at most $(1 + \varepsilon)\hat{\delta}_\ell$, where $\hat{\delta}_\ell$ is the average degree of vertices in subgraph $U$ in step $\ell$. From Lemma 3, we know that $\hat{\delta}_\ell \leq 2d$. 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.
Parallel Approximate Degeneracy Ordering: Extras

- Graph to the right shows ADG compared to other ordering heuristics.
- In the CREW model, depth stays the same, while work increases to $O(m + nd)$, after redesigning the update function.

### Ordering heuristics

<table>
<thead>
<tr>
<th>Ordering heuristic</th>
<th>Time / Depth</th>
<th>Work</th>
<th>F.? B., Approx.?</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF (first-fit) [25]</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>n/a n/a</td>
</tr>
<tr>
<td>R (random) [26], [31]</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>n/a</td>
</tr>
<tr>
<td>ID (incidence-degree) [1]</td>
<td>$O(n + m)$</td>
<td>$O(n + m)$</td>
<td>n/a n/a</td>
</tr>
<tr>
<td>SD (saturation-degree) [27], [31]</td>
<td>$O(n + m)$</td>
<td>$O(n + m)$</td>
<td>n/a n/a</td>
</tr>
<tr>
<td>LF (largest-degree-first) [31]</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>n/a</td>
</tr>
<tr>
<td>LLF (largest-log-degree-first) [31]</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>n/a</td>
</tr>
<tr>
<td>SLL (smallest-log-degree-last) [31]</td>
<td>$O(\log \Delta \log n)$</td>
<td>$O(n + m)$</td>
<td>n/a n/a</td>
</tr>
<tr>
<td>SL (smallest-degree-last) [28], [31]</td>
<td>$O(n)$</td>
<td>$O(m)$</td>
<td>exact</td>
</tr>
<tr>
<td>ASL (approximate-SL) [32]</td>
<td>$O(n)$</td>
<td>$O(m)$</td>
<td>n/a</td>
</tr>
</tbody>
</table>

| ADG [approx. degeneracy] | $O\left(\log^2 n\right)$ | $O(n + m)$ | $\Theta \left(2(1 + \varepsilon)\right)$ |

### Algorithm 2

1. UPDATE($U_i$, $R_i$, $D$): //$U_i$ and $R_i$ are sets $U$ and $R$ in iteration $i$
2. for all $v \in U_i$ do in parallel:
3.  $D[v] = D[v] - \text{Count}(NU_i(v) \cap R_i)$

**Algorithm 2:** The modified version of the UPDATE routine from ADG (Algorithm 1) that works in the CREW setting.
Algorithm 3: JP, the Jones-Plaxton coloring heuristic. With $\rho = \langle \rho_{ADG}, \rho_R \rangle$, it gives JP-ADG that provides $(2(1 + \varepsilon)d + 1)$-coloring.
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 $\varepsilon > 0$.

*Proof.* Since $\langle \rho_{ADG}, \rho_R \rangle$ is a $2(1+\varepsilon)$-approximate degeneracy ordering, the result follows from Lemma 6 and 4. $\square$
JP-ADG: Depth, Work

- Expected longest path in DAG $G_p$ is $O(d \log n + \log d \log^2 n / \log \log n)$
- Expected depth is $O(\log^2 n + \log \Delta \ast \text{longest path})$
- Work is $O(n+m)$ since both ADG and JP take $O(n+m)$ work

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

**Proof.** Since $\rho_{ADG}$ is a partial $2(1+\epsilon)$-approximate degeneracy ordering (Lemma 4) and since ADG performs at most $O(\log n)$ iterations (Lemma 1), the depth follows from Lemma 7, Lemma 1 and past work [31], which shows that JP runs in $O(\log n + \log \Delta \cdot |P|)$ depth. As both JP and ADG perform $O(n+m)$ work, so does JP-ADG. □

**Proof.** Let $G_p(ℓ)$ be the subgraph of $G_p$ induced by the vertex set $V(ℓ) = \{v \in V \mid \rho_{ADG} = ℓ\}$. Let $\Delta_\ell$ be the maximal degree and $d_\ell$ be the average degree of the subgraph $G_p(ℓ)$.

Since, by the definition of $G_p$, there can be no edges in $G_p$ that go from one subgraph $G_p(ℓ)$ to another $G_p(ℓ')$ with $ℓ' > ℓ$, we can see that a longest (directed) path $P$ in $G_p$ will always go through the subgraph $G_p(ℓ)$ in a monotonically decreasing order with regards to $ℓ$. Therefore, we can split $P$ into a sequence of (directed) sub-paths $P_1, \ldots, P_{\tilde{p}}$, where $P_\ell$ is a path in $G(ℓ)$. We have $|P| = \sum_{\ell=1}^{\tilde{p}} |P_\ell|$ and by Corollary 6 from past work [31], the expected length of a longest sub-path $P_\ell$ is in $O(\Delta_\ell + \log \Delta_\ell n / \log \log n)$, because $G_p(ℓ)$ is induced by a random priority function. By linearity of expectation, we have for the whole path $P$:

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

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

$$\sum_{i=1}^{\tilde{p}} \Delta_\ell \leq \sum_{i=1}^{\tilde{p}} d \cdot k \in O(d\log n) \tag{2}$$

$$\sum_{i=1}^{\tilde{p}} \log \Delta_\ell \in O(d\log \log n) \tag{3}$$

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

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

Our main result follows by combining our bounds on the longest path $P$ in the DAG $G_p$, and a result by Hasenplaugh et al. [31], which shows that JP has $O(\log n + \log \Delta \cdot |P|)$ depth.
DEC-ADG

- Employs the use of a function called SIM-COL (Simple Coloring), which colors a subgraph using $(1+\mu)\Delta$ colors
Algorithm 5: **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 \(\mu\) as \(\mu = \varepsilon / 4\); we use this value in the listing above for concreteness.
SIM-COL: Work, Depth

- SIM-COL performs $O(\log n)$ iterations w.h.p.
- The work in each iteration has depth $O(\Delta)$ in CREW and $O(\log \Delta)$ in CRCW.
- Thus, depth is $O(\Delta \log n)$ and $O(\log \Delta \log n)$ in CREW and CRCW respectively.
- Work is $O(n + m)$.
**DEC-ADG: Work, Depth, and Quality**

- $O(\log d \log^2 n)$ depth in CRCW
- $O(n + m)$ work w.h.p
- Coloring quality: $(2+\epsilon)d$ coloring of the graph for $0 < \epsilon \leq 8$.
- Work and depth proofs of this algorithm and SIM-COL made possible by use of Markov Inequalities and Chernoff Bounds, taking advantage of randomness in conjunction with constants, to create high probability bounds on size and computation.
DEC-ADG-ITR

- Same as DEC-ADG, but instead of choosing a random color in SIM-COL line 7, it chooses the smallest color not taken by a vertex v’s neighbors.

Selecting colors can be done in $O(\bar{\Delta})$ depth and $O(\Delta \cdot |U_i|)$ work, where $U_i$ is the set $U$ in iteration $i$ (of the while loop in SIM-COL) and $\bar{\Delta} = \max_{u \in V(\ell)} \deg(\ell)(v)$. For the modified SIM-COL, we get, since all other operations are equal, $O(\Delta I)$ depth ($I$ is the number of iterations of ITR); the work is

$$O \left( \sum_{i=1}^{I} \left[ \Delta \cdot |U_i| - \sum_{v \in U_i} \deg(v) \right] \right).$$

Depth and work in DEC-ADG-ITER are, respectively

$$O(Id \log n)$$

and (as a simple sum over all iterations $I$)

$$O \left( \sum_{\ell=1}^{I} \left( \sum_{i=1}^{I} \left[ \Delta \cdot |U_i| - \sum_{v \in U_i} \deg(v) \right] + \sum_{v \in R(\ell)} \deg(v) \right) \right).$$

Note that these bounds are valid in the CREW setting.
Using Concurrent Reads (CREW)

- Adds a (small) factor of $d$ to a subcomponent of ADG, yielding a new work of $O(dn + m)$ for ADG and DEC-ADG
Optimizations

- **Representation of U and R**
  - R should be listed in the same array as U, where R(.) precedes U
  - Done so “the actual removal of R(i + 1) from U takes O(1) time by simply moving the index pointer by |R(i+1)| positions “to the right”, giving – at the end of iteration i + 1 – a representation [R(0) R(1) ... R(i) R(i + 1) index U]”

- **Explicit Ordering in R(.)**
  - “often enhances the accuracy of the obtained approximate degeneracy ordering, which in turn consistently improves the ultimate coloring accuracy”

- **Combining JP and ADG**
  - “where one derives predecessors and successors in a given ordering to construct the DAG Gp [in JP], can also be implemented as a part of UPDATE in ADG, in Algorithm 1”
  - Doesn’t improve theoretical results, but saves time in implementation
Optimizations Pt. 2

- **Usage of degree median instead of degree average**
  - Takes $O(1)$ time in a sorted array
  - Variants with “-M” at the end use this variation
  - “However, the whole U has to be sorted in each pass. We incorporate linear-time integer sorting, which was shown to be fast in the context of sorting vertex IDs [86] ADG-M only differs from Algorithm 1 in that”
  - “Additionally we limit R to half the size of U (+1 if $|U|$ is odd). We refer to the priority function produced by ADG-M as $\rho_{ADG-M}$”

- **Push vs Pull style updating**
  - “(pushing updates to a shared state or pulling updates to a private state)”
  - While pushing needs atomics, pulling takes more updates
  - Similar performance in practice

- **Caching Sums of Degrees**
  - When computing average degree, one can just subtract edges from existing sum instead of fully recomputing from all edges
  - Slightly improves runtime (~1%)
Implementation

- Integrated with GBBS and GAP benchmark suite
- Implemented with optimizations
- **ADG-M**
  - Work: $O(n + m)$
  - Depth: $O(\log^2 n)$, from half of the vertices being removed each iteration, and each iteration taking $O(\log n)$ work
  - ADG-M computes a partial 4-approximate degeneracy ordering of $G$
- **JP-ADG-M**
  - Same work and depth as JP-ADG
Evaluations

- Coloring Quality is superior
- Algorithm runtime is comparable or marginally higher
- “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.”
Evaluations: Runtime and Quality

**Smaller graphs** (used in online execution scheduling, ...)

<table>
<thead>
<tr>
<th>SC</th>
<th>JP</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Graph 1" /></td>
<td><img src="image2.png" alt="Graph 2" /></td>
</tr>
</tbody>
</table>

**Larger graphs** (used in offline data analytics, ...)

<table>
<thead>
<tr>
<th>SC</th>
<th>JP</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image3.png" alt="Graph 3" /></td>
<td><img src="image4.png" alt="Graph 4" /></td>
</tr>
</tbody>
</table>
Evaluations: Runtime and Quality
Evaluations: Runtime and Quality

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 $\delta$. 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 Plassmann 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.
Evaluations: Scaling

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 $\delta$. DEC-ADG-ITR uses dynamic scheduling. JP-ADG uses linear time sorting of $R$. In weak scaling, we use $n = 1M$ vertices.
Evaluations: Impact of $\varepsilon$

Fig. 3: Impact of $\varepsilon$ on run-times and coloring quality. Parametrization: 32 cores, sorting: Radix sort, direction-optimization: push, JP-ADG variant based on average degrees $\delta$. DEC-ADG-ITR uses dynamic scheduling.
H. Memory Pressure and Idle Cycles

We also investigate the pressure on the memory bus, see Figure 4. For this, we use PAPI [102] 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 $\delta$. DEC-ADG-ITR uses dynamic scheduling. JP-ADG uses linear time sorting of $R$. 
Evaluations: Color Quality Frequency

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

- Overall very strong paper
- Provides ground breaking results, explains concepts well, provided reader knowledge from ground up, thorough experimentation
- Nothing for me to really nitpick after reading
Discussion Questions

● Other practical optimizations that can be made? Theoretical?
● Is recoloring worthwhile?
● Applications?
● Potential approaches for exact parallel colorings?
Bibliography

- Image on slide 2 is from wikipedia’s graph coloring page https://en.wikipedia.org/wiki/Graph_coloring
- All other images come from the reviewed paper