Parallel Distributed Distance-2 Coloring of Bipartite Graphs

Nedialko S. Nedialkov*,1
Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada, L8S 4K1
Radu Serban2
Xulu Entertainment, 890 Hillview Court, Suite 160 Milpitas, CA 95035

Abstract
We present a method for parallel, partial distance-2 coloring of bipartite graphs on distributed memory machines. Our method uses an independent-set approach for coloring the subgraph induced by boundary vertices at distance two on different processes, and a choice of several sequential coloring heuristics.

This method is implemented in PARCOLIS, a software package for PARallel COLOrizing through Independent Sets. PARCOLIS is designed and tuned to address the problem of efficient Jacobian evaluation through finite differences for the solution of differential equations using implicit integration methods. Numerical experiments show that our approach is scalable and robust, especially for the problem classes of interest.

Key words: graph coloring, distributed computing, sparse Jacobians
PACS: 02.60.Cb, 02.60.Lj
2000 MSC: 65L80, 65L05, 65F05, 68R10, 68N19

1. Introduction
We are interested in efficient, distributed-memory parallel approximations of large, sparse Jacobians using finite differences. Given a continuously differ-

*Corresponding author
Email addresses: nedialk@mcmaster.ca (Nedialko S. Nedialkov), radu@xulu.com (Radu Serban)
1This work was supported in part by the Natural Sciences and Engineering Research Council of Canada. Most of this work was done during N. S. Nedialkov’s sabbatical year at the Lawrence Livermore National Laboratory.
2This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract No. DE-AC52-07NA27344. Most of this work was done while R. Serban was a member of the Center for Applied Scientific Computing at LLNL.

Preprint submitted to Elsevier October 22, 2008
entiable function \( f : \mathbb{R}^n \to \mathbb{R}^n \) and a point \( \mathbf{x} \in \mathbb{R}^n \), the Jacobian of \( f(x) \) at \( \mathbf{x} \) is the \( n \times n \) matrix with \((i, j)\) entry \( \frac{\partial f_i}{\partial x_j}(\mathbf{x}) \), where \( f_i \) is the \( i \)th component of \( f \), and \( x_j \) is the \( j \)th component of \( x \). We denote this Jacobian by \( J \). To compute a finite-differences approximation to \( J \), the \( j \)th column of \( J \) can be approximated by

\[
\frac{\partial f}{\partial x_j}(\mathbf{x}) \approx \frac{1}{\epsilon} \left[ f(\mathbf{x} + \epsilon e_j) - f(\mathbf{x}) \right],
\]

where \( e_j \) is the \( j \)th unit vector, and \( \epsilon > 0 \) is sufficiently small. If each column of \( J \) is computed as in (1), we need \( n \) function evaluations, in addition to computing \( f(\mathbf{x}) \). However, if \( J \) is sparse (and we know its sparsity pattern), we may approximate several columns of \( J \) with the cost of a single function evaluation, as outlined below.

A set of columns of \( J \) is **structurally orthogonal**, if any row in \( J \) contains at most one structural nonzero at the intersection with these columns. If we partition the columns of \( J \) into structurally orthogonal subsets, we can approximate the columns in such a subset with the cost of a single function evaluation. Hence, if we find \( c \) such subsets, we need \( c \) function evaluations—this is attractive when \( c \ll n \) and \( n \) is large.

A standard approach for partitioning the columns of \( J \) is to associate with the sparsity pattern of \( J \) a suitable graph, and color its vertices with as few colors as possible [5, 10]. The columns of \( J \) corresponding to vertices of the same color are then structurally orthogonal and can be approximated with a single function evaluation.

In this paper, we present a parallel method and its implementation—the PARCOLIS, PARallel COLoring through Independent Sets, package—for partitioning the columns of a matrix into groups of structurally orthogonal columns by **partial distance-2 coloring** of the vertices of a bipartite graph associated with this matrix.

**Motivation.** Implicit integration methods for the numerical solution of systems of stiff ordinary differential equations (ODEs) or differential-algebraic equations (DAEs) require efficient solution of linear systems that arise in Newton-type iterations. The matrix in such a linear system is closely related to the Jacobian of the right-hand side of an ODE or the Jacobian of the residual function of a DAE. For large systems, iterative linear solvers are typically applied, but their performance is crucially dependent on the availability of good preconditioners—a good preconditioner may be difficult or even impossible to construct. Indeed, construction of preconditioners often takes advantage of the underlying physical problem and/or the structure of the resulting Jacobian. For many applications (e.g., chemical kinetics), no practical preconditioners may be available, because none of the Jacobian entries can be ignored, or the Jacobian lacks a regular structure, or both.

For large and sparse linear systems, an alternative to an iterative method with preconditioning is to use a direct method for sparse linear systems. There
are several sparse direct linear solvers available, both for sequential and parallel computations \cite{4,15}, but, in the context of an implicit numerical integrator, overall efficiency is heavily dependent on the efficiency with which Jacobians are generated.

Our work is primarily motivated by the need for sparse direct linear algebra support for the solvers in the SUNDIALS suite of nonlinear, ODE, and DAE solvers \cite{11}. The differential equation solvers in SUNDIALS target very large-scale problems, and they all provide support for distributed-memory parallel computations through the Message Passing Interface (MPI).

For our application, we are given \( f : \mathbb{R}^n \rightarrow \mathbb{R}^n \), which is evaluated on multiple processors in a distributed-memory environment. In SUNDIALS, the distribution of a state vector over multiple processors is provided by the user, and it is typically done to minimize communication during the evaluation of the problem-defining function \( f \). As a consequence, for the purposes of Jacobian partitioning, we assume that the distribution of variables and function evaluations among multiple processors is given and fixed. Since we construct a graph representation for the sparsity pattern of the associated Jacobian using the variable and function distribution already done by the user, we do not use specialized graph partitioners to redistribute the resulting graph for the purpose of coloring. That is, we assume that a “good” partitioning of a graph results from the user distribution of variables and function evaluations.

Paper outline. Section 2 describes the graph coloring problem that is the subject of this work. Section 3 gives an outline of the overall method as implemented in PARCOLIS. The following sections are devoted to the algorithms and their implementation in PARCOLIS: Section 4 describes how the associated bipartite graph is constructed; Section 5 presents the parallel phase of PARCOLIS; and Section 6 presents the sequential algorithms employed in this package. Numerical results are provided in Section 7. Related work is summarized in Section 8. We end with conclusions and future work directions, Section 9.

2. Evaluating Jacobians and graph coloring

In §2.1 we illustrate how a sparse Jacobian can be obtained with fewer than \( n \) function evaluations. In §2.2 we show the connection between sets of structurally orthogonal columns and coloring the vertices of a graph. We consider two graph representations of the sparsity pattern of a matrix—a column intersection graph (§2.2.1) and a bipartite graph (§2.2.2)—and discuss briefly (§2.2.3) why we have chosen the latter in PARCOLIS.

2.1. Approximating Jacobians

If variable \( x_j \) does not appear in component \( f_i \), then \( J_{ij} = 0 \) for any value of \( x_j \). We call such \( J_{ij} \) a structural zero; otherwise, \( J_{ij} \) is a structural nonzero. Consider columns \( j \) and \( k \) of \( J \). Let \( J \) be the set of indices for which \( J_{ij} \) is structurally nonzero for all \( i \in J \), and let \( K \) be the set of indices for which
$J_{ik}$ is structurally nonzero for all $i \in K$. If $J \cap K = \emptyset$, columns $j$ and $k$ are structurally orthogonal, and we can perturb components $x_j$ and $x_k$ to evaluate

$$J(e_j + e_k) \approx \frac{1}{\epsilon} \left[ f(\hat{x} + \epsilon(e_j + e_k)) - f(\hat{x}) \right].$$

Given a set of indices of structurally orthogonal columns, let $d$ be the vector with $d_i = 1$, if column $i$ is in this set, and 0 otherwise. Then (2) generalizes to

$$Jd \approx \frac{1}{\epsilon} \left[ f(\hat{x} + \epsilon d) - f(\hat{x}) \right].$$

Example 1. Consider a $7 \times 7$ Jacobian with sparsity pattern (from problem b1_ss in [8])

<table>
<thead>
<tr>
<th></th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
<th>x5</th>
<th>x6</th>
<th>x7</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>f2</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>f3</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>f4</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>f5</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>f6</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>f7</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>

We can group the above columns as

(a) $\begin{pmatrix}
  x1 & x2 & x3 & x4 & x5 & x6 & x7 \\
  x  & x  & x  & x  & x  & x  & x  \\
  x  & x  & x  & x  & x  & x  & x  \\
  x  & x  & x  & x  & x  & x  & x  \\
  x  & x  & x  & x  & x  & x  & x  \\
  x  & x  & x  & x  & x  & x  & x  \\
  x  & x  & x  & x  & x  & x  & x  \\
\end{pmatrix}$ or

(b) $\begin{pmatrix}
  x1 & x2 & x3 & x4 & x5 & x6 & x7 \\
  x  & x  & x  & x  & x  & x  & x  \\
  x  & x  & x  & x  & x  & x  & x  \\
  x  & x  & x  & x  & x  & x  & x  \\
  x  & x  & x  & x  & x  & x  & x  \\
  x  & x  & x  & x  & x  & x  & x  \\
  x  & x  & x  & x  & x  & x  & x  \\
\end{pmatrix}$

In both cases, we can evaluate a Jacobian with the above sparsity pattern using three function evaluations.

We have the following general problem statement:

**Problem 1.** Given the sparsity pattern of an $m \times n$ matrix $A$, partition its columns into groups of structurally orthogonal columns, such that the number of groups is as small as possible.

This column partitioning problem can be modeled as **distance-1**, or **d-1**, coloring of the vertices of the *column intersection* graph associated with $A$ [6]. It can also be modeled as **partial distance-2 coloring** [10] of the vertices of the bipartite graph corresponding to $A$. We discuss these two approaches in the next subsection.

For convenience in the notation and to simplify the presentation, we assume square matrices throughout this paper. The described techniques can be readily extended to rectangular matrices as well.
2.2. Graphs and coloring

2.2.1. Column intersection graph

Given an \( n \times n \) matrix \( A \), its column intersection graph is the undirected graph \( G = (V, E) \) with vertices \( V = \{ v_1, v_2, \ldots, v_n \} \), where \( v_j \) corresponds to column \( j \) in \( A \), and edges

\[
E = \{ (v_i, v_j) \mid \exists k \text{ for which } A_{ki} \neq 0 \text{ and } A_{kj} \neq 0 \}.
\]

That is, there is an edge between \( v_i \) and \( v_j \), if and only if columns \( i \) and \( j \) are not structurally orthogonal \([6, 10]\).

We can color the vertices of \( G \) such that no two adjacent vertices are of the same color. This is also referred to as distance-1, or d-1, coloring. The columns corresponding to vertices of the same color are structurally orthogonal. Therefore, our partitioning problem reduces to finding the fewest number of colors such that adjacent vertices in a column intersection graph are of distinct colors.

**Remark 1.** In practice, we typically have the sparsity pattern of \( A \) and can write

\[
E = \{ (v_i, v_j) \mid \exists k \text{ for which } A_{ki} \text{ and } A_{kj} \text{ are structurally nonzero} \}.
\]

**Example 2.** In Figure 1 we show two colored instances of the column intersection graph corresponding to (3). The partition in (4-a) can be derived from the graph in Figure 1(a), and the partition in (4-b) can be derived from the graph in Figure 1(b).

![Figure 1: Colored column intersection graphs corresponding to (3).](image)

2.2.2. Bipartite graph

A **bipartite graph** is an undirected graph \( G = (V, E) \) in which the set of vertices \( V \) can be partitioned into two sets \( V_1 \) and \( V_2 \) such that \((u, v) \in E\) implies either \( u \in V_1 \) and \( v \in V_2 \) or \( v \in V_1 \) and \( u \in V_2 \).
Given an $n \times n$ matrix $A$, we associate with it a bipartite graph $G = (V, E)$ with $V_1 = \{w_1, w_2, \ldots, w_n\}$, $V_2 = \{v_1, v_2, \ldots, v_n\}$, where $w_i$ corresponds to row $i$, $v_j$ corresponds to column $j$, and

$$E = \{ (w_i, v_j) \mid A_{ij} \neq 0 \}.$$

Vertices $v_i \in V_2$ and $v_j \in V_2$, $i \neq j$, are distance-two, or $d$-2, neighbors, if there is a $w_k \in V_1$ such that $(w_k, v_i) \in E$ and $(w_k, v_j) \in E$. Columns $i$ and $j$ are structurally orthogonal, if and only if $v_i$ and $v_j$ are at a distance greater than two in the corresponding bipartite graph.

We are interested in $d$-2 coloring of $V_2$. Namely, our goal is to find the smallest number of colors such that any $d$-2 neighbors in $V_2$ are colored with distinct colors. This is also called partial distance-2 coloring \cite{10}, as it does not color the vertices in $V_1$. Hence, if we find a $d$-2 coloring of $V_2$, a set of structurally orthogonal columns contains those columns for which the corresponding $V_2$ vertices are of the same color.

**Example 3.** In Figure 2, we show partial distance-2 colorings of the bipartite graph corresponding to (3). The partitions in (4-a) and (4-b) can be derived from (a) and (b), respectively.

![Figure 2: Bipartite graphs corresponding to (3) with colored $V_2$ vertices.](image)

Finally, we note that finding the smallest number of colors in either $d$-1 coloring of a column intersection graph or partial $d$-2 coloring of a bipartite graph is an NP-complete problem; for related references see \cite{10}.
2.2.3. Bipartite versus column intersection graph

In PARCOLIS, we use bipartite representation and perform d-2 coloring of the vertices corresponding to the columns of a matrix. A major advantage of the bipartite representation over the column intersection graph is the amount of memory needed, as summarized below from [10].

For a given $n \times m$ matrix $A$, it can be shown that its column intersection graph is isomorphic to the adjacency graph of $A^T A$, which graph contains $n$ vertices and $(\text{nnz}(A^T A) - n)/2$ edges, where $\text{nnz}(A^T A)$ is the number of nonzeros in $A^T A$. However, a bipartite graph corresponding to $A$ contains $n + m$ vertices and $\text{nnz}(A)$ edges. Although $\text{nnz}(A^T A)$ and $\text{nnz}(A)$ depend on the sparsity pattern of $A$, rough analysis in [10] shows that, for sparse matrices of practical interest, the column intersection graph is likely to have many more edges than a bipartite graph. Furthermore, larger amount of memory typically gives rise to larger access times due to the memory hierarchy of modern processors.

Generally, a bipartite representation is easier and faster to construct than a column intersection representation, but the d-1 coloring can be done faster on the column intersection graph than d-2 coloring on a bipartite graph. However, it is shown in [10] that constructing and d-1 coloring of a column intersection graph is of the same complexity as constructing and d-2 coloring of a bipartite graph. Moreover, empirical results in [10] show that overall execution time (graph construction plus coloring) for d-2 coloring of a bipartite graph can be much smaller than overall execution time for d-1 coloring of a column intersection graph. In passing, we note that the results in [10] are for sequential algorithms and implementation.

From our motivating application, namely finite-difference Jacobian approximation for sparse direct linear solvers within an implicit method for differential equation solvers, the bipartite representation is more natural and can be generated by directly transcribing the dependencies of the right-hand side function on the ODE states.

3. Method outline

We outline the method for partitioning the columns of a matrix as implemented in PARCOLIS. Details are presented in sections 4 to 6.

Given an $n \times n$ matrix $A$, denote by $G = (V_1, V_2, E)$ the bipartite graph with

$$V_1 = \{w_i, \ i = 1, \ldots, n\}, \quad V_2 = \{v_j, \ j = 1, \ldots, n\}, \quad \text{and} \quad E = \{(w_i, v_j) \mid A_{ij} \neq 0\}. \quad (5)$$

(In the context of Jacobian evaluation, we can consider $w_i$ corresponding to $f_i$, $v_j$ corresponding to $x_j$, and $E = \{(w_i, v_j) \mid x_j \text{ occurs in } f_i\}$.) Our goal is to distribute $G$ among processes and perform in parallel partial d-2 coloring of the $V_2$ vertices in the subgraph of $G$ on each process.
We assume we have $P$ processes with process identifiers $\{1, \ldots, n\}$. Consider a row-wise partitioning of $A$ among $P$ processes, where process $p$ stores $I_p$ rows of $A$. Each process $p$ builds a subgraph $G_p = (V_1, V_2, E)$ of $G$ corresponding to the submatrix of $A$ formed from these $I_p$ rows. For $u, v \in V_2$, let $pe(u)$ and $pe(v)$ be the processes on which $u$ and $v$ are distributed to. If $u$ and $v$ are $d$-2 neighbors in $G$, and $pe(u) \neq pe(v)$, then $u$ and $v$ are boundary vertices on $pe(u)$ and $pe(v)$, respectively; otherwise they are internal vertices.

When building each $G_p$, we also construct a communication pattern, which is used to “complete” $G_p$ by constructing $G^*_p$, such that each $V_2$ vertex in $G^*_p$ “knows” all its $d$-2 neighbors. That is, if $u, v \in V_2$ are boundary $d$-2 neighbors in $G$, then they are also $d$-2 neighbors in $G^*_{pe(u)}$ and $G^*_{pe(v)}$ (but they may not be $d$-2 neighbors in $G_{pe(u)}$ and $G_{pe(v)}$).

In a parallel phase, each process colors its boundary vertices. The coloring proceeds iteratively, where on each iteration, processes color sequentially and exchange vertex and color information. After all boundary vertices are colored, then, in a sequential phase, each process colors its internal vertices. Finally, the color information is collected, to determine a partitioning of the columns of the matrix. This is summarized as:

1. Build a subgraph on each process (Section 4).
2. Color boundary vertices in parallel (Section 5).
3. Color internal vertices sequentially (Section 6).
4. Collect color information from processes to determine partitioning of columns (trivial step, details omitted).

4. Constructing bipartite graphs

We introduce notation for mapping of global to local indices (§4.1), show how initial graphs are constructed on each process (§4.2), and then describe how these graphs are completed (§4.3, §4.4).

4.1. Global and local indices

Let $\mathcal{M}$ be a set of global indices, and let $\mathcal{I}_p$ be a set of local indices on process $p$, $p = 1, \ldots, P$, where $|\mathcal{M}| = \sum_{i=1}^{P} |\mathcal{I}_i|$. We denote by $\mu$ a bijective map of a global index $m \in \mathcal{M}$ to a pair of indices $(p, i)$, where $p$ is process identifier and $i$ is local index:

$$\mu : \mathcal{M} \rightarrow \{ (p, i) \mid 1 \leq p \leq P \text{ and } i \in \mathcal{I}_p \}$$

and

$$\mu^{-1} : \{ (p, i) \mid 1 \leq p \leq P \text{ and } i \in \mathcal{I}_p \} \rightarrow \mathcal{M}.$$

Given an $n \times n$ matrix $A$, we use the same mapping of global indices for row and column distribution of $A$. We set $\mathcal{M} = \{1, \ldots, n\}$, and assuming each process stores $I_p$ rows of $A$, we take $\mathcal{I}_p = \{1, \ldots, I_p\}$.

Typically these identifiers are 0, $\ldots, n-1$ in practice, but the notation is more convenient here, if they start from 1.
Example 4. A possible distribution and local indexing of the rows and columns for the sparsity pattern in (3) is

\[
\begin{array}{cccccc}
(1, 1) & (1, 2) & (1, 3) & (2, 1) & (2, 2) & (3, 1) & (3, 2) \\
(1, 1) & \times & \times & \times & \ \\
(1, 2) & \times & \times & \ \\
(1, 3) & \times & \ \\
(2, 1) & \times & \times & \times \\
(2, 2) & \times & \times & \ \\
(3, 1) & \times & \times & \ \\
(3, 2) & \times & \times & \ \\
\end{array}
\]

(6)

Remark 2. In our application, we use the same mapping \( \mu \) for distribution of variables and functions evaluations. That is, function \( f_i \), for which \( \mu(i) = (p, \alpha) \), is evaluated on process \( p \) and its local index is \( \alpha \) on this process. Similarly, variable \( x_j \), for which \( \mu(j) = (q, \beta) \), is stored on process \( q \) and its local index is \( \beta \) on this process.

4.2. Initial graphs

On each process \( p \), we construct a bipartite graph \( G_p = (V_{1p}, V_{2p}, E_p) \) as follows. We create vertices \( w_{p,i}, i = 1, \ldots, I_p \). For each \( A_{i,j} \neq 0 \) with \( \mu(i) = (p, \alpha) \) and \( \mu(j) = (q, \beta) \), we create a vertex \( v_{q,\beta} \) and an edge \((w_{p,\alpha}, v_{q,\beta})\). More formally,

\[
V_{1p} = \{ w_{p,i} | i = 1, \ldots, I_p \}, \quad V_{2p} = \{ v_{q,\beta} | \exists A_{i,j} \neq 0 \text{ with } \mu(i) = (p, \cdot) \text{ and } \mu(j) = (q, \beta) \}, \quad \text{and} \quad E_p = \{ (w_{p,\alpha}, v_{q,\beta}) | A_{\mu^{-1}(p,\alpha),\mu^{-1}(q,\beta)} \neq 0 \}.
\]

Example 5. For the sparsity pattern in (6) and distribution (6), the initial graphs \( G_1, G_2, \) and \( G_3 \) are shown in Figure 3.

4.3. Communication pattern for exchanging edges

We associate with a process \( p \) a “send” vector \( s^{(p)} \) of size \( P \) such that, for process \( q \neq p \), \( s^{(p)}_q = 1 \) if there is \( A_{i,j} \neq 0 \) with \( \mu(i) = (p, \cdot) \) and \( \mu(j) = (q, \cdot) \); and \( s^{(p)}_q = 0 \) otherwise (\( s^{(p)}_p = 0 \)).

Similarly, we associate with process \( p \) a “receive” vector \( r^{(p)} \) of size \( P \) such that, for process \( q \neq p \), \( r^{(p)}_q = 1 \) if there is \( A_{i,j} \neq 0 \) with \( \mu(i) = (q, \cdot) \) and \( \mu(j) = (p, \cdot) \); and \( r^{(p)}_q = 0 \) otherwise (\( r^{(p)}_p = 0 \)).

Example 6. For the distribution in (6), we have

\[
\begin{array}{c|cc}
p & s^{(p)} & r^{(p)} \\
1 & (0, 1, 1) & (0, 1, 1) \\
2 & (1, 0, 1) & (1, 0, 0) \\
3 & (1, 0, 0) & (1, 1, 0) \\
\end{array}
\]

Notice that, if we collect the vectors \( r^{(p)} \) in a matrix \( R \), such that \( R_{p,i} = r^{(p)}_i \), and the vectors \( s^{(p)} \) in a matrix \( S \), such that \( S_{p,i} = s^{(p)}_i \), then \( S = R^T \).
4.4. Completing an initial graph

After the initial graphs and the communication pattern are constructed, if \( s_q^{(p)} = 1 \), then process \( p \) has stored at least one edge \((w_{p,\alpha}, v_{q,\beta}) \in E_p\) for some \( \alpha, \beta \), but process \( q \) has not stored this edge (in its \( G_q \)) and should receive \((w_{p,\alpha}, v_{q,\beta})\) from \( p \) and add it to its \( G_q \).

For processes \( p \) and \( q \), \( p \neq q \), let \( X_{p,q} \subseteq V_{1p} \) be the set of vertices on process \( p \) that are adjacent to at least one vertex \( v \in V_{2p} \) with \( \text{pe}(v) = q \). Then \( p \) sends to \( q \) the edges

\[
Z_{p,q} = \{ (u, v) | (u, v) \in E_p \text{ and } u \in X_{p,q} \},
\]
which process \( q \) adds to its graph.

**Example 7.** We have \( X_{1,2} = \{ w_{1,1}, w_{1,2} \} \) and \( X_{1,3} = \{ w_{1,3} \} \). Process 1 sends to process 2 the edges

\[
Z_{1,2} = \{ (w_{1,1}, v_{1,2}), (w_{1,1}, v_{1,3}), (w_{1,1}, v_{2,1}), (w_{1,2}, v_{1,2}), (w_{1,2}, v_{2,2}) \},
\]
and sends to process 3 the edges

\[
Z_{1,3} = \{ (w_{1,3}, v_{1,3}), (w_{1,3}, v_{3,1}) \},
\]

Similarly, \( X_{2,1} = \{ w_{2,2} \} \) and \( X_{2,3} = \{ w_{2,1} \} \). Process 2 sends to process 1 the edges

\[
Z_{2,1} = \{ (w_{2,2}, v_{1,1}), (w_{2,2}, v_{2,2}) \},
\]
and sends to process 3 the edges
\[ \mathcal{Z}_{2,3} = \{ (w_{2,1}, v_{2,1}), (w_{2,1}, v_{3,2}) \}. \]

Finally, \( \mathcal{X}_{3,1} = \{ w_{3,1}, w_{3,2} \} \), and process 3 sends to process 1 the edges
\[ \mathcal{Z}_{3,1} = \{ (w_{3,1}, v_{1,1}), (w_{3,1}, v_{3,1}), (w_{3,2}, v_{1,1}), (w_{3,2}, v_{3,2}) \}. \]

The bipartite graphs after this communication are shown in Figure 4.

![Bipartite Graphs](image)

(a) process 1, \( G^*_1 \)  
(b) process 2, \( G^*_2 \)  
(c) process 3, \( G^*_3 \)

Figure 4: Graphs after communicating edges. Edges received from process 1 are in red (longest dashed lines in monochrome), edges received from process 2 are in green (shortest dashed lines), and edges received from process 3 are in blue.

The above considerations are summarized in Algorithm 4.1. It is not difficult to show that, after it terminates, each \( V_2 \) vertex on any process knows all its \( d-2 \) neighbors.

5. Parallel coloring phase

We begin in §5.1 by introducing additional notation. We then present our method for coloring boundary vertices (§5.2), explain ranking of vertices (§5.3) and exchanging of color information (§5.4), and finally give the overall parallel algorithm for coloring boundary vertices (§5.5). An example is provided as an illustration of this algorithm. In the last subsection (§5.6), we elaborate on the number of iterations in the parallel phase.
Algorithm 4.1 (Complete graph on process $p$).

**INPUT**

\[ G_p = (V_{1p}, V_{2p}, E_p) \]

\[ r(p), s(p) \]

**OUTPUT**

\[ G^*_p = (V_{1p}^*, V_{2p}^*, E_p^*) \]

**COMPUTE**

\[ E \leftarrow \emptyset \]

for each process $q \neq p$

if $s_q(p) = 1$ construct $Z_{p,q}$ and send it to $q$

if $r_q(p) = 1$ receive $Z_{q,p}$ from $q$

\[ E \leftarrow E \cup Z_{q,p} \]

\[ E_p^* \leftarrow E_p \cup E \]

5.1. Notation

The coloring works on the graphs $G^*_p = (V_{1p}^*, V_{2p}^*, E_p^*)$, $p = 1, \ldots, P$. For simplicity in the notation, we shall omit "*" and denote these graphs by $G_p = (V_{1p}, V_{2p}, E_p)$.

Unless stated otherwise, when we say a vertex, we shall mean a $V_2$ vertex. We say that a process $p$ owns a vertex $v_{l,\gamma} \in V_{2p}$ if $l = p$. The process owning a vertex $v$ is $pe(v)$. The set of boundary vertices owned by process $p$ will be denoted by $V_{2p}^b$.

Let $adj_2(v)$ be the set of $d$-2 neighbors of a vertex $v$. The $d$-2 degree of $v$ is $\text{deg}_2(v) = |adj_2(v)|$. For a boundary vertex $u$, we denote by $\text{bndr}_l(u)$ the set of $d$-2 neighbors of $u$ that are owned by $l \neq pe(u)$:

\[ \text{bndr}_l(u) = \{ v \mid v \in \text{adj}_2(u) \text{ and } pe(v) = l \}, \]

and by $\text{bndr}(u)$ the union of these $\text{bndr}_l(u)$:

\[ \text{bndr}(u) = \bigcup_{l \neq pe(u)} \text{bndr}_l(u). \]

The color of a vertex $v$ will be denoted by $\text{col}(c)$ and will be encoded by $0, 1, 2, \ldots$.

**Example 8.** In Figure 3 $\text{adj}_2(v_{1,3}) = \{ v_{1,2}, v_{2,1}, v_{3,1} \}$, $\text{bndr}_2(v_{1,3}) = \{ v_{2,1} \}$, $\text{bndr}_3(v_{1,3}) = \{ v_{3,1} \}$, and $\text{bndr}(v_{1,3}) = \{ v_{2,1}, v_{3,1} \}$.

5.2. Method outline

In the parallel phase, each process colors the boundary vertices it owns. To avoid conflicts, namely (boundary) $d$-2 neighbors colored with the same color by different processes, we adapt the independent set-based approach from [12].

With each boundary vertex $v$, we associate a unique number, which we refer to as $\text{rank}$ and denote it by $\text{rank}(v)$. (The calculation of ranks is discussed in
To avoid conflicts, given d-2 neighbors $u$ and $v$ with $\text{pe}(u) \neq \text{pe}(v)$, $\text{pe}(u)$ colors $u$ before $\text{pe}(v)$ colors $v$ if
\[
\text{rank}(u) > \text{rank}(v).
\]

We also associate with each boundary vertex $v$ a nonnegative wait number $\text{wait}(v)$. Initially, $\text{wait}(v)$ is the number of $u \in \text{bndr}(v)$ for which $\text{rank}(u) > \text{rank}(v)$:
\[
\text{wait}(v) = |\{ u \mid u \in \text{bndr}(v) \text{ and } \text{rank}(u) > \text{rank}(v) \}|.
\]
That is, $\text{wait}(v)$ is the number of d-2 boundary vertices that need to be colored before $v$ is colored.

The parallel coloring proceeds iteratively, where on iteration $i$, process $p$ colors the set of vertices
\[
C_p^{(i)} = \{ v \in V_{2p}^b \mid \text{wait}(v) = 0 \text{ and } v \text{ is not colored} \}.
\]
(7)

At each iteration, processes communicate vertex and color information. In particular, after a vertex $v$ is colored, an encoding of it and its color is sent to all processes that own uncolored $u \in \text{bndr}(v)$ with $\text{rank}(u) < \text{rank}(v)$, after which the wait numbers of those $u$ are decremented by one.

5.3. Rank calculation

In PARCOLIS, we have implemented two rankings of boundary vertices:
(a) $\text{rank}(v) = \text{pseudo random number in } (0,1)$; and
(b) $\text{rank}(v) = \text{deg}_2(v) + \text{pseudo random number in } (0,1)$.

In the context of distance-1 coloring, the former is advocated in [12, 13], while the latter corresponds to the ranking proposed in [1, 16], where a vertex is colored first if its degree is larger than the degrees of its adjacent vertices; ties are resolved at random. In [1, 16], it is shown empirically that using (b) typically results in a smaller number of colors than using (a). We note also that (b) is related to the largest-degree-first ordering (see [7, 10] for example), where vertices of largest degree are selected first.

Our empirical studies show (Section 7) that, for some problems, (a) results in fewer iterations (in the parallel phase) and number of colors than (b), and on other problems, the opposite occurs.

Example 9. Consider the graphs in Figure 4. In Figure 5, we show the same graphs with assigned ranks and calculated wait numbers, the two columns after the $V_2$ vertices. The integer part of each rank of a vertex owned by a process is the d-2 degree of this vertex. Note that, for example, the d-2 degree of $v_{1,1}$ is 1 in the graph on process 2, but we store on this process the actual d-2 degree of this vertex, which is 3.
Figure 5: Graphs with assigned ranks and calculated wait numbers for $V_2$ vertices.
5.4. Communication pattern for exchanging color information

For exchanging color information, we set a process send and receive vectors $\mathbf{s}(p)$ and $\mathbf{r}(p)$, respectively, as follows. Consider a boundary $u$ owned by process $p$. Process $p$ sends color information of $u$ (an encoding of $u$ and its color) to process $q$

* after $u$ is colored, and
* if process $q$ owns uncolored d-2 neighbors $v$ of $u$ with $\text{rank}(v) < \text{rank}(u)$.

Initially, for all $q \neq p$,

$$\mathbf{s}(p) = \left\{ u \in V_{2p}^b \mid \exists v \in \text{bndr}_q(u) \text{ with } \text{rank}(v) < \text{rank}(u) \right\}$$

($\mathbf{s}(p) = 0$). That is, $\mathbf{s}(p)$ is the total number of vertices that have to be sent (along with their colors) from $p$ to $q$.

Similarly, process $p$ receives from process $q \neq p$

$$\mathbf{r}(p) = \left\{ u \in V_{2q}^b \mid \exists v \in \text{bndr}_p(u) \text{ with } \text{rank}(v) < \text{rank}(u) \right\}$$

($\mathbf{r}(p) = 0$) vertices with their colors.

5.5. Coloring boundary vertices

When coloring boundary vertices, each process $p$ colors at iteration $i$ the set $C_p(i)$ in $\mathbf{4}$. If $v \in C_p(i)$ has at least one uncolored d-2 neighbor $u$ owned by process $q \neq p$ with $\text{rank}(u) < \text{rank}(v)$, then $p$ sends the color information of $v$ to $q$. Hence, process $p$ needs to construct the set of colored vertices that will be sent to $q$:

$$\mathbf{S}_{p,q}^{(i)} = \left\{ v \mid v \in C_p^{(i)} \text{ and } \exists \text{ uncolored } u \in \text{bndr}_q(v) \text{ with } \text{rank}(u) < \text{rank}(v) \right\}$$

After $q$ receives a vertex $v$ and its color from $p$, $q$ colors its copy of $v$ (the vertex with the same process identifier and local index as those of $v$) and decrements the wait numbers of all uncolored d-2 neighbors of $v$ that it owns.

Finally, this parallel phase is summarized in Algorithm 5.1. We illustrate it with the example that follows.

Example 10. We show how the iterations would proceed on the graph in Figure 4. For the illustrations, we encode the red color with 0, green with 1, and blue with 2.

**Iteration 1.** Process 1 colors $v_1, 1$, encodes process number, vertex number, and color, for example by the triple $(1, 1, 0)$ and sends it to processes 2 and 3. Similarly, process 2 colors $v_2, 1$ and sends $(2, 1, 0)$ to processes 1 and 3. This is summarized as:

<table>
<thead>
<tr>
<th>process</th>
<th>colors</th>
<th>sends to</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$v_{1,1}$</td>
<td>2, 3</td>
<td>$(1, 1, 0)$</td>
</tr>
<tr>
<td>2</td>
<td>$v_{2,1}$</td>
<td>1, 3</td>
<td>$(2, 1, 0)$</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

15
Algorithm 5.1 (Color boundary vertices on process $p$).

**INPUT**

- $G_p = (V_{1p}, V_{2p}, E_p)$  
  with set $\text{rank}(v)$ and $\text{wait}(v)$ for all $v \in V_{2p}$  
  $\mathcal{S}(p), \mathcal{P}(p)$

**OUTPUT**

- $G'_p = (V_{1p}, V_{2p}, E_p)$ with colored boundary vertices

**COMPUTE**

- $i \leftarrow 0$
- while $\sum_{q=1}^{p} (\mathcal{S}_q^{(p)} + \mathcal{P}_q^{(p)}) > 0$
  - $i \leftarrow i + 1$
  - construct $\mathcal{C}_i^{(i)}_p$ and color it sequentially
  - $\mathcal{S} \leftarrow \emptyset$
  - for each process $q \neq p$
    - if $\mathcal{S}_q^{(p)} > 0$
      - construct $\mathcal{S}_{p,q}$ and send it to $q$
      - $\mathcal{S}_q^{(p)} \leftarrow \mathcal{S}_q^{(p)} - |\mathcal{S}_{p,q}|$
    - if $r_q^{(p)} > 0$
      - receive $\mathcal{S}_{q,p}$ from $q$
      - $r_q^{(p)} \leftarrow r_q^{(p)} - |\mathcal{S}_{q,p}|$
      - $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{S}_{q,p}$
  - for each $u \in \mathcal{S}$
    - color copy of $u$ with $\text{col}(u)$
    - for each uncolored $v \in \text{bndr}_p(u)$
      - if $\text{rank}(v) < \text{rank}(u)$
        - $\text{wait}(v) \leftarrow \text{wait}(v) - 1$

When process 1 receives, for example $(2, 1, 0)$, it determines the corresponding copy of $v_{2,1}$, colors it with the received color, and decrements the wait numbers of the $d-2$ neighbors of $v_{2,1}$ that it owns, here $v_{1,2}, v_{1,3}$. This “processing” of vertices can be summarized in the following table (“decrements $v$” means wait($v$) decremented by one):

<table>
<thead>
<tr>
<th>process</th>
<th>receives</th>
<th>from</th>
<th>decrements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$(2, 1, 0)$</td>
<td>2</td>
<td>$v_{1,2}, v_{1,3}$</td>
</tr>
<tr>
<td>2</td>
<td>$(1, 1, 0)$</td>
<td>1</td>
<td>$v_{2,2}$</td>
</tr>
<tr>
<td>3</td>
<td>$(1, 1, 0)$</td>
<td>1</td>
<td>$v_{3,1}, v_{3,2}$</td>
</tr>
<tr>
<td></td>
<td>$(2, 1, 0)$</td>
<td>2</td>
<td>$v_{3,2}$</td>
</tr>
</tbody>
</table>

The resulting graphs are shown in Figure 6.

*Iteration 2.* Similarly, we have the tables
Figure 6: Graphs after first iteration.
<table>
<thead>
<tr>
<th>process</th>
<th>colors</th>
<th>sends to</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$v_{1,2}$</td>
<td>2</td>
<td>(1,2,1)</td>
</tr>
<tr>
<td></td>
<td>$v_{1,3}$</td>
<td>3</td>
<td>(1,3,2)</td>
</tr>
<tr>
<td>2</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>3</td>
<td>$v_{3,2}$</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

and

<table>
<thead>
<tr>
<th>process</th>
<th>receives from</th>
<th>decrements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>2</td>
<td>(1, 2, 1)</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>(1, 3, 2)</td>
<td>1</td>
</tr>
</tbody>
</table>

The resulting graphs are shown in Figure 7.

**Iteration 3.** Process 2 colors $v_{2,2}$ and process 3 colors $v_{3,2}$. The resulting graphs are shown in Figure 8.

### 5.6. Number of iterations

Consider a bipartite graph $G = (V_1, V_2, E)$ with unique ranks for all $V_2$ vertices. Let the $V_2$ vertices $v_{i_1}, v_{i_2}, \ldots, v_{i_j}$ be such that $v_{i_r}$ and $v_{i_{r+1}}$ are d-2 neighbors, for all $r = 1, \ldots, j - 1$, and

$$\text{rank}(v_{i_1}) > \text{rank}(v_{i_2}) > \cdots > \text{rank}(v_{i_j}).$$

We say that these vertices are on a monotone path. We define its length as the number of $V_2$ vertices in it.

Define $\lambda(G)$ as the largest integer, such that for any ranking of the vertices in $V_2$, there is a monotone path of length $\lambda(G)$. Define the chromatic number $\chi_2(G)$ as the least number of colors that are needed for partial d-2 coloring of the $V_2$ vertices in $G$. Then, translating a result from [4] to a bipartite graph and monotone paths as defined above, it can be shown that

$$\lambda(G) = \chi_2(G). \quad (8)$$

Consider the graph $G$ defined in [6]. Let $G^b$ be the subgraph of $G$ induced by boundary vertices and their incident edges. The number of iterations in Algorithm 5.1 is the length of the longest monotone path in $G^b$, formed from boundary vertices, where no d-2 neighbors in this path are owned by the same process. Since we are also interested in minimizing the number of these iterations (in addition to minimizing the number of colors), we wish to have an assignment of ranks such that the length of the largest monotone path is as small as possible.

From [8], the following readily follows. First, finding a ranking such that (5) holds is as difficult as finding the chromatic number, which is an NP-complete problem. Second, the number of iterations in Algorithm 5.1 is at least $\chi_2(G^b)$. Therefore, the best we can hope for is a ranking heuristic that gives rise to

- a longest monotone path in $G^b$ not much larger than $\chi_2(G^b)$ and
- number of colors not much larger than $\chi_2(G^b)$.
Figure 7: Graphs after second iteration.
(a) process 1, $G_1$

(b) process 2, $G_2$

(c) process 3, $G_3$

Figure 8: Graphs after third iteration.
6. Sequential coloring heuristics

In this section, we describe the sequential coloring heuristics implemented in PARCOLIS, namely, greedy (§6.1), largest-degree-first (LDF) ordering (§6.2), incidence-degree (ID) ordering (§6.3), and saturation-degree (SD) ordering (§6.4). Each of them can be used in both the parallel and sequential phases of coloring.

We assume that a bipartite graph \( G = (V_1, V_2, E) \) and a set of uncolored vertices \( U \subseteq V_2 \) are given, and that the vertices in \( V_2 \setminus U \) are not necessarily all uncolored.

We denote the largest degree in the vertex set \( V_1 \) by \( \Delta(V_1) \), the largest degree in the vertex set \( V_2 \) by \( \Delta(V_2) \), and set \( \Delta = \max\{ \Delta(V_1), \Delta(V_2) \} \). We denote by \( \delta \) the largest \( d-2 \) degree of a vertex in \( V_2 \), and obviously \( \delta \leq \Delta^2 \). We use colors \( 0, 1, \ldots \).

We show the complexity of the implementation of each of these heuristics and give the amount of additional memory needed by each of them, in addition to the memory used for storing \( G \) and \( U \). In particular, we show that to color \( N = |U| \) vertices in partial \( d-2 \) coloring, the work and required additional memory are:

<table>
<thead>
<tr>
<th></th>
<th>work</th>
<th>additional mem.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>( O(N \Delta^2) )</td>
<td>( O(\delta) )</td>
</tr>
<tr>
<td>LDF</td>
<td>( O(N \log N + N\Delta^2) )</td>
<td>( O(\delta) )</td>
</tr>
<tr>
<td>ID</td>
<td>( O(N\Delta^2) )</td>
<td>( O(\delta + N) )</td>
</tr>
<tr>
<td>SD</td>
<td>( O(N \log N \cdot \Delta^2) )</td>
<td>( O(\delta N) )</td>
</tr>
</tbody>
</table>

6.1. Greedy

In this approach, in a loop over all vertices, each is assigned the smallest available color; see Algorithm 6.1. The array \( c \) is for keeping track of the smallest available color.

Algorithm 6.1 (Greedy coloring).

**INPUT**

\( G = (V_1, V_2, E) \)

uncolored \( U \subseteq V_2 \)

**OUTPUT**

colored \( U \)

**COMPUTE**

for each \( u \in U \)

set array \( c \) with \( c[i] = 0 \) for \( i = 0, \ldots, \delta - 1 \)

for each colored \( v \in \text{adj}_2(u) \)

\( c[\text{col}(v)] \leftarrow 1 \)

\( \text{col}(u) \leftarrow \) smallest \( i \) such that \( c[i] = 0 \)
Complexity. Finding the colored d-2 neighbors of a vertex is done in $O(\Delta(V_1) \cdot \Delta(V_2)) = O(\Delta^2)$. Finding the smallest available color (per vertex) takes $O(\delta) = O(\Delta^2)$. The total amount of work is $O(N\Delta^2)$. The extra memory, in addition to storing $G$ and $U$, is $O(\delta)$.

In the remaining algorithms, the smallest available color will be determined as in this algorithm.

6.2. Largest-degree-first ordering

The d-2 degree of a vertex $v$, denoted as $\text{deg}_2(v)$, is the number of its d-2 neighbors, that is, $\text{deg}_2(v) = |\text{adj}_2(v)|$. In this heuristic, we sort the vertices from $U$ in non-increasing order of their d-2 degrees and then color them in order, where colors are selected in a greedy manner. This is realized in Algorithm 6.2.

Algorithm 6.2 (LDF coloring).

\textbf{Input} \\
\hspace{1em} $G = (V_1, V_2, E)$ \\
\hspace{1em} uncolored $U \subseteq V_2$ \\
\hspace{1em} with $\text{deg}_2(u)$ set for all $u \in U$ \\
\textbf{Output} \\
\hspace{1em} colored $U$ \\
\textbf{Compute} \\
\hspace{1em} sort the vertices in $U$ by non-increasing d-2 degree \\
\hspace{1em} for each vertex $u$ in order \\
\hspace{2em} col($u$) ← smallest available color

\textbf{Complexity.} We use quick sort to sort the vertices in place; hence $O(N \log N)$. The for loop is executed in $O(N\Delta^2)$ as in the greedy approach. Hence, the work is $O(N \log N + N\Delta^2)$, and the additional memory required is as in the greedy algorithm, $O(\delta)$.

\textbf{Remark 3.} Since we are mainly interested in problems for which typically $\delta \ll N$, one can use a linear time sorting algorithm, but such an algorithm needs extra memory of the order of $O(\delta + N)$, while quicksort sorts in place. With a linear sorting algorithm, the amount of work be $O(N\Delta^2)$.

6.3. Incidence degree (ID) ordering

By an incidence degree of a vertex $u$, denoted here by $\text{id}(u)$, we mean the number of colored d-2 neighbors of $u$. In PARCOLIS, we color a set of (uncolored) vertices $U \subseteq V_2$ using the following ID-based heuristic:

1. select a vertex from $U$ of largest ID; 
2. if there is more than one such vertex, try to select (as explained below) a vertex of largest d-2 degree among these vertices.
In our implementation (see Algorithm 6.3) we assume that $id(u)$ and $deg_2(u)$ are known for all $u \in U$. During coloring, we have to keep track of IDs of uncolored d-2 neighbors of each vertex that is colored. Initially, each $u \in U$ is inserted at the beginning of a doubly-linked list $L_{id(u)}$ that stores vertices of the same ID. We denote by $L$ the linked list corresponding to the vertex with the largest ID.

**Coloring and updating IDs.** Assume that we have selected vertex $u$ for coloring. We find the colored and uncolored d-2 neighbors of $u$.

- From the colored neighbors, we determine the smallest available color and color (as in greedy and LDF) $u$ with it.
- For each uncolored d-2 neighbor $v$ of $u$, if $v \in U$, we delete $v$ from $L_{id(v)}$ and insert it into $L_{id(v)+1}$. Then we increment $v$’s ID.

If a new list is created, then $L$ becomes this list. If no vertices remain in $L$, $L$ becomes the list of second largest IDs.

**Vertex selection.** We start coloring the vertices from $L$. On the first iteration of the while loop in Algorithm 6.3, we select the first vertex in $L$ of largest d-2 degree. On subsequent iterations, we select the first vertex of largest d-2 degree among the first $\min\{\delta, \text{length}(L)\}$ vertices in $L$.

The motivation for this approach is to keep the time for searching in $L$ as $O(\delta)$. Suppose that $L$ contains vertices $v_1, \ldots, v_l$, and $v_j$ is selected for coloring. Denote the d-2 neighbors of $v_j$ by $w_1, w_2, \ldots w_k$, $k \leq \delta$. When $v_j$ is colored, the IDs of all $w_i$ vertices are incremented by 1. There are two possibilities:

1. If $\{v_1, \ldots, v_{j-1}, v_{j+1}, \ldots, v_l\} \cap \{w_1, w_2, \ldots, w_k\} = \emptyset$, then some, all, or none of the $w_i$’s may be inserted in $L$. Hence length($L$) may increase, but because of (7), we search at most $\delta$ items in $L$.

2. If $\{v_1, \ldots, v_{j-1}, v_{j+1}, \ldots, v_l\} \cap \{w_1, w_2, \ldots, w_k\} \neq \emptyset$, the number of vertices with largest ID is at most $\delta$, and length($L$) $\leq \delta$.

Our experiments show that usually length($L$) $\ll \delta$ and rarely length($L$) $> \delta$.

**Complexity.** The first for loop executes in $O(N)$. Consider the body of the whole loop. In the first iteration, selecting a vertex is in $O(N)$, and on subsequent iterations, we have $O(\delta)$ for selecting a vertex. Finding the colored and uncolored d-2 neighbors of a vertex takes $O(\Delta^2)$. Processing the colored vertices, to determine available color, and uncolored vertices, to update incidence degrees, takes $O(\delta)$. The body of the for loop is done in $O(1)$, by maintaining appropriate pointers between vertices (details omitted). Hence, a bound for the running time of this algorithm is $O(N\Delta^2)$.

We organize the linked lists using an array of size $N$ for storing pointers to vertices, and an array of size $\delta$ for storing pointers to the beginning of lists.
Algorithm 6.3 (ID coloring).

**INPUT**
\[ G = (V_1, V_2, E) \]
uncolored \( U \subseteq V_2 \) with \( \text{id}(u) \) and \( \text{deg}_2(u) \) for all \( u \in U \)

**OUTPUT**
colored \( U \)
updated IDs of uncolored vertices in \( V_2 \setminus U \)

**COMPUTE**

\[
\text{compute } \quad \text{for each } u \in U \\
\quad \text{insert } u \text{ in } L_{\text{id}(u)} \\
\quad L \leftarrow \text{list of vertices with largest ID} \\
\quad \text{while } L \text{ is not the empty list} \\
\quad \quad \text{if first iteration} \\
\quad \quad \quad u \leftarrow \text{first vertex of largest d-2 degree in } L \\
\quad \quad \quad \text{else} \\
\quad \quad \quad \quad u \leftarrow \text{first vertex of largest d-2 degree among the} \\
\quad \quad \quad \quad \quad \quad \text{first } \min\{\delta, \text{length}(L)\} \text{ elements in } L \\
\quad \quad \quad \text{remove } u \text{ from } L \\
\quad \quad \quad \text{find colored and uncolored d-2 neighbors of } u \\
\quad \quad \quad \text{col}(u) \leftarrow \text{smallest available color} \\
\quad \quad \quad \text{for each uncolored d-2 neighbor } v \text{ of } u \\
\quad \quad \quad \quad \text{if } v \in U \\
\quad \quad \quad \quad \quad \text{remove } v \text{ from } L_{\text{id}(v)} \\
\quad \quad \quad \quad \quad \text{insert } v \text{ in } L_{\text{id}(v)+1} \\
\quad \quad \quad \quad \quad \text{id}(v) \leftarrow \text{id}(v) + 1 \\
\quad \quad \quad \quad \quad \text{update } L \text{ if necessary} \\
\]

with pointers to vertices of the same ID. For storing pointers to colored and uncolored d-2 neighbors, we use another array of size \( \delta \). Hence the extra memory is \( O(\delta + N) \)

6.4. Saturation degree (SD) ordering

By a saturation degree of a vertex \( u \), denoted here by \( \text{sd}(u) \), we mean the number of distinctly colored d-2 neighbors of \( u \). Obviously, \( \text{sd}(u) \leq \text{id}(u) \leq \text{deg}_2(u) \). Our SD-based ordering heuristics is (Algorithm 6.4):

1. select a vertex with largest SD;
2. if there is more than one such vertex, select a vertex with largest ID;
3. if there is more than one such vertex, select the first vertex in “order”, see below, with largest d-2 degree.

Initially, we build a max-heap of size \( N \), where each element in this heap is a pointer to a vertex from \( U \). We build such a heap using an ordering of the
**Algorithm 6.4 (SD coloring).**

**INPUT**
\[ G = (V_1, V_2, E) \]
uncolored \( U \subseteq V_2 \)
with \( \text{sd}(u), \text{id}(u) \) and \( \text{deg}_2(u) \) for all \( u \in U \)

**OUTPUT**
colored \( U \)
updated SDs and IDs of uncolored vertices in \( V_2 \setminus U \)

**COMPUTE**
build max-heap
while heap-size > 0
\[ \begin{align*}
    u & \leftarrow \text{vertex from top of heap} \\
    \text{heap-size} & \leftarrow \text{heap-size} - 1, \text{ remove top of heap} \\
    \text{find} \text{ colored and uncolored } d-2 \text{ neighbors of } u \\
    \text{col}(u) & \leftarrow \text{smallest available color} \\
    \text{for each uncolored } d-2 \text{ neighbor } v \text{ of } u \\
    \text{id}(v) & \leftarrow \text{id}(v) + 1 \\
    \text{if } \text{col}(u) \text{ not in } \text{colhash}(v) \\
    \text{insert } \text{col}(u) \text{ in } \text{colhash}(v) \\
    \text{sd}(v) & \leftarrow \text{sd}(v) + 1 \\
    \text{heap-increase-key}
\end{align*} \]

vertices defined by
\[ u \prec v \text{ iff } \begin{cases} 
    \text{sd}(u) < \text{sd}(v) \\
    \text{sd}(u) = \text{sd}(v) \text{ and } \text{id}(u) < \text{id}(v) \\
    \text{sd}(u) = \text{sd}(v) \text{ and } \text{id}(u) = \text{id}(v) \text{ and } \text{deg}_2(u) < \text{deg}_2(v) \\
\end{cases} \text{ or } \begin{cases} 
    \text{id}(u) < \text{id}(v) \\
    \text{id}(u) = \text{id}(v) \text{ and } \text{deg}_2(u) < \text{deg}_2(v) \\
\end{cases} \]
\[ u \equiv v \text{ iff } \text{sd}(u) = \text{sd}(v), \text{id}(u) = \text{id}(v), \text{ and } \text{deg}_2(u) = \text{deg}_2(v). \]

(For a discussion of heaps and heap operations, see for example [2].) That is, the top of the heap points to a vertex with largest SD, ID, and d-2 degrees. If there is more than one vertex with largest SD, ID, and d-2 degrees, it will be selected at a later stage of coloring (when the pointer to this vertex is at the top of the heap).

After a vertex is colored, we update the IDs of its uncolored neighbors, by incrementing each of these IDs by one. To keep track of the saturation degree of an uncolored vertex, we maintain a “color” hash table that stores distinct colors of d-2 neighbors. For an uncolored vertex \( u \), we denote its color hash table by \( \text{colhash}(u) \).

The variable heap-size contains the number of elements in the heap. After an ID or SD, or both, are incremented we rebuild the heap using the heap increase operation, see e.g. [2].

**Complexity.** Building a heap is in \( O(N) \). Finding the colored and uncolored d-2 neighbors of a vertex takes \( O(\Delta^2) \). Processing the colored vertices, to determine available color takes \( O(\delta) \).
The heap increase operation in the for loop is in $O(\log N)$. The hash operations are in $O(1)$. Since this loop is executed at most $O(\Delta^2)$ times, the total amount of work in the for loop is $O(\log N \cdot \Delta^2)$. Hence the total amount of work in this algorithm is $O(N\Delta^2 + N \log N \cdot \Delta^2) = O(N \log N \cdot \Delta^2)$.

Since each hash table is of size at most $\delta$, the extra memory that is needed is $O(N \delta)$.

**Remark 4.** In our implementation, the ID- and SD-based heuristics do not result in “true” ID- and SD-type colorings, since during the parallel phase, incidence and saturation degrees are not communicated between processes.

### 7. Numerical Results

We have tested Parcolis on a variety of problems, available from sparse matrix collections [2, 8], arising in molecular dynamics simulations [17], as well as for ODE and DAE sparse Jacobian matrices.

For space considerations, we only provide results for a few of the problems that we used as our test set. We begin with a subset of the graphs used to compare Parcolis and the algorithm presented in [3] (subsequently denoted by HPCC-05) which is, to our knowledge, the only other existing algorithm for distributed parallel distance-2 coloring. Next, we present results for two problems typical of the class targeted by the Sundials integrators, namely coloring of graphs associated to Jacobians of differential equations (ODE or DAE) obtained through semi-discretization (either with finite differences or finite elements) of time-dependent partial differential equations.

We note that, even though all matrices below are symmetric for the purpose of being used within Sundials, the Parcolis implementation does not take advantage of symmetry. A planned future extension of Parcolis to explicitly take symmetry into account is expected to perform much better on such problems.

We carried out our experiments on the Jacquard cluster at NERSC. Jacquard has 356 dual-processor nodes available for scientific calculations. Each processor runs at a clock speed of 2.2GHz, and has a theoretical peak performance of 4.4 GFlop/s. Processors on each node share 6GB of memory. The nodes are interconnected with a high-speed InfiniBand network.

#### 7.1. Standard sparse matrices

Table 4 displays the relevant structural properties of the test graphs used here. We provide the total number of vertices (number of columns in the matrix) and edges in the column intersection graph associated with the sparse matrices as well as the maximum and average distance-2 vertex degree.

---

4We are indebted to D. Bozdag who graciously provided us with the data for their numerical results.

### Table 1: Structural properties of the standard sparse matrices.

| Matrix      | |V| | |E| | D2 dgr. |
|-------------|-----------------|-----------------|-----------------|
|             | max  | avg  |                 |                 |                 |
| apo1-4      | 92,224 | 2,355,096 | 194 | 122.67 |
| pkustk13    | 94,893 | 6,616,827 | 1136 | 415.34 |
| shipsec1    | 140,874 | 7,813,404 | 341 | 169.98 |
| ldoor       | 952,203 | 46,522,475 | 258 | 151.72 |

(a) NAMD test data [17] (courtesy of P. Hovland, ANL)
(b) U. Florida Sparse Matrix Collection [8]
(c) PARASOL test data [2]

Consistent with the results presented in [3], all matrices used in these tests (of which the 4 presented in Table 1 are a subset) were first partitioned with METIS [14] by applying edge reduction on the adjacency graph. Note that, when used within an implicit integrator, this partitioning step is typically not required. Indeed, the states of the differential equation system are already distributed over processes, usually in such a manner as to minimize inter-process communication at each function evaluation. This is practically equivalent to a partition which minimizes the size of the subgraph induced by the boundary vertices.

Figure 9 shows the string speedup results for different choices of the underlying sequential coloring method. In this and next figures, “R greedy” refers to the greedy algorithm with a random permutation of the vertices before colored.

We note that the, in general, hpcc-05 outperforms PARCOLIS, but we attribute this to the fact that hpcc-05 does take advantage of symmetry and expect that a variant of PARCOLIS that does the same will show significant improvements.

On the other hand, the quality of the resulting coloring is, in general, better with PARCOLIS as shown in Figure 10. As we observed with other test matrices, the independent-set approach implemented in PARCOLIS results in fewer colors than the conflict detection/resolution approach taken in hpcc-05, regardless of the sequential coloring heuristics used (with SD providing on average the best results).

### 7.2. Partial differential equation examples

**Semi-discretized PDE.** As our first example in this section, we use the Jacobian matrices arising in solving a 2-D, time-dependent advection-reaction-diffusion PDE using the method of lines with varying spatial resolutions. The PDE is defined on a square with Dirichlet boundary conditions on the left and right and periodic boundary conditions on top and bottom. We use a 5-point internal stencil (central finite differences) to semi-discretize the PDE in space. We impose the boundary conditions as algebraic relations, thus obtaining a DAE system.

To obtain a sequence of problems for weak parallel scaling experiments, we increase the spatial refinement so that the size of the global DAE systems in-
creases proportionally with the number of processes (or, in other words, the local problem size remains constant). The largest problem solved (on 128 processes) has $|V| = 51.2 \cdot 10^6$ and $|E| \approx 2.56 \cdot 10^9$. A characteristic sparsity pattern (for the problem solved on 4 processes) is shown in Figure 11.

Weak parallel scaling results, and the number of colors obtained with Parcolis for the different choices of sequential coloring heuristics are shown in Figure 12.

Unstructured finite element method problems. Table 2 presents the structural characteristics of the graphs associated to a series of related matrices, distributed over 1, 2, 4, 8, 16, 32, and 64 processes, respectively. This sequence of problems

---

6Problems provided by T. Kolev and P. Vassilevski, LLNL
corresponds to refinements of the same physical PDE (Laplace equation on unit cube) using linear finite elements. However, it should be noted that the intermediate refinements (on number of processors not equal to perfect cubes) are obtained by bisection, and hence the resulting matrices have different topological structures. Therefore, equivalent problems are: \{ufe01, ufe08, ufe64\}, \{ufe02, ufe16\}, and \{ufe04, ufe32\}. Table 2 provides the size and distance-2 degrees in the global graph, the distribution over processes (number of processes and resulting number of vertices with distance-2 neighbors on a different process), as well as the size of the largest local problem (i.e.; the size of the subgraph owned by one process with largest number of vertices).

For illustration, we provide in Figure 13 the actual physical distribution of the finite elements and the sparsity pattern of the resulting matrix for the problem ufe04.
Figure 11: Characteristic sparsity pattern for the Jacobian matrix corresponding to a semi-discretization of a 2-D advection-reaction-diffusion PDE to a system of DAE.

Figure 12: Weak parallel results and resulting number of colors for the semi-discretized PDE problems.

<table>
<thead>
<tr>
<th>D2 dgr.</th>
<th>Distrib.</th>
<th>Largest local problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>V</td>
<td>$, $</td>
</tr>
<tr>
<td>ufe01 124 77.67 1 0</td>
<td>68,705 885,741</td>
<td>0</td>
</tr>
<tr>
<td>ufe02 40 31.45 2 11.928</td>
<td>170,081 1,265,993</td>
<td>86,351 643,678 5,992</td>
</tr>
<tr>
<td>ufe04 24 21.25 4 31.525</td>
<td>274,625 1,751,093</td>
<td>70,955 453,289 8,209</td>
</tr>
<tr>
<td>ufe08 124 85.52 8 101.470</td>
<td>536,769 7,467,021</td>
<td>68,956 969,979 14,984</td>
</tr>
<tr>
<td>ufe16 40 33.47 16 225.646</td>
<td>1,335,489 10,433,193</td>
<td>87,604 684,353 20,274</td>
</tr>
<tr>
<td>ufe32 24 22.59 32 433.061</td>
<td>2,146,689 14,340,213</td>
<td>71,567 475,902 18,242</td>
</tr>
<tr>
<td>ufe64 124 89.68 64 1,244,655</td>
<td>4,243,841 61,309,005</td>
<td>70,460 1,021,135 26,903</td>
</tr>
</tbody>
</table>

Table 2: Structural properties of the graphs associated with the sequence of 7 unstructured FEM problems.
Weak parallel scaling results and the number of colors obtained with PARCOLIS for the different choices of sequential coloring heuristics are shown in Figure 14.

(a) Physical distribution  
(b) Sparsity pattern

Figure 13: Characteristic structure of unstructured FEM problems.

Figure 14: Weak parallel scaling and coloring results for the unstructured FEM problems.

8. Related work

An excellent and comprehensive survey of graph coloring algorithm for evaluating Jacobians is [10]. An implementation of parallel distributed distance-2 graph coloring is reported in [3].
9. Conclusions

We have introduced Parcolis, an independent-set algorithm for partial distance-2 coloring of bipartite graphs on distributed parallel architectures. Parcolis was designed and tuned to address the related problem of efficient Jacobian evaluation through finite differences for differential equations (ODE or DAE) solved with implicit methods.

The Parcolis package is written in a modular manner and provides various options for vertex ranking, sequential coloring, etc. Numerical experiments show that the algorithm is scalable and robust, especially for the problem classes of interest.

Future work will focus on integrating Parcolis within the Sundials suite of integrators. Even though, for the purpose of use within Sundials, we are interested in general (non-symmetric) Jacobian matrices, we will extend Parcolis to use tailored structures for symmetric matrices which will result in improved performance (both in terms of memory and computational efficiency) for parallel distance-2 coloring of the associated graphs. Finally, we plan on investigating the potential benefits of a staggered scheme for finite-difference Jacobian approximations in which we would use Parcolis only for the coloring (in parallel) of the subgraph induced by the boundary vertices, evaluate the corresponding Jacobian columns with inter-process communication turned on in the function evaluation, and then, on each process separately and independently, produce a coloring of the subgraph induced by the internal vertices only and evaluate (simultaneously on all processes) the remaining Jacobian columns with inter-process communication turned off in the function evaluation. This would obviously result in an incomplete graph coloring, but may be a more effective approach to the desired goal of Jacobian evaluation.

References


