SIAM J. Sci. Comput.
Vol. 32, No. 4, pp. 2418–2446


# DISTRIBUTED-MEMORY PARALLEL ALGORITHMS FOR DISTANCE-2 COLORING AND RELATED PROBLEMS IN DERIVATIVE COMPUTATION*

DORUK BOZDAĞ[†], ÜMİT V. ÇATALYÜREK[‡], ASSEFAW H. GEBREMEDHIN[§], FREDRIK MANNE[¶], ERIK G. BOMAN[‖], AND FÜSUN ÖZGÜNER[**]

**Abstract.** The distance-2 graph coloring problem aims at partitioning the vertex set of a graph into the fewest sets consisting of vertices pairwise at distance greater than 2 from each other. Its applications include derivative computation in numerical optimization and channel assignment in radio networks. We present efficient, distributed-memory, parallel heuristic algorithms for this NP-hard problem as well as for two related problems used in the computation of Jacobians and Hessians. Parallel speedup is achieved through graph partitioning, speculative (iterative) coloring, and a bulk synchronous parallel–like organization of parallel computation. Results from experiments conducted on a PC cluster employing up to 96 processors and using large-size real-world as well as synthetically generated test graphs show that the algorithms are scalable. In terms of quality of solution, the algorithms perform remarkably well—the numbers of colors used by the parallel algorithms are observed to be very close to the numbers used by their sequential counterparts, which in turn are quite often near optimal. Moreover, the experimental results show that the parallel distance-2 coloring algorithm compares favorably with the alternative approach of solving the distance-2 coloring problem on a graph $\mathcal{G}$ by first constructing the square graph $\mathcal{G}^2$ and then applying a parallel distance-1 coloring algorithm on $\mathcal{G}^2$. Implementations of the algorithms are made available via the Zoltan toolkit.

**Key words.** distance-2 graph coloring, distributed-memory parallel algorithms, Jacobian computation, Hessian computation, sparsity exploitation, automatic differentiation, combinatorial scientific computing

**AMS subject classifications.** 05C15, 05C85, 68R10, 68W10

**DOI.** 10.1137/080732158

## 1. Introduction.

A distance-$k$ coloring of a graph is an assignment of colors to vertices such that any two vertices connected by a path consisting of at most $k$ edges receive different colors, and the goal of the associated problem is to use as few colors as possible. Distance-2 coloring (of an appropriate graph) is an archetypal model in the efficient computation of sparse Jacobian and Hessian matrices, whether derivatives

*Received by the editors August 4, 2008; accepted for publication (in revised form) May 17, 2010; published electronically August 5, 2010. This work was supported by the U.S. Department of Energy's Office of Science through the CSCAPES SciDAC Institute; by the U.S. National Science Foundation under grants CCF-0830645, CNS-0643969, OCI-0904809, OCI-0904802, and CNS-0403342; by Ohio Supercomputing Center PAS0052; and by the Norwegian Research Council through the Evita program.

http://www.siam.org/journals/sisc/32-4/73215.html

[†]Department of Biomedical Informatics, The Ohio State University, Columbus, OH 43210 (bozdag.1@osu.edu).

[‡]Departments of Biomedical Informatics and Electrical & Computer Engineering, The Ohio State University, Columbus, OH 43210 (catalyurek.1@osu.edu).

[§]Department of Computer Science, Purdue University, West Lafayette, IN 47907 (agebreme@purdue.edu).

[¶]Department of Informatics, University of Bergen, Bergen, Norway (Fredrik.Manne@ii.uib.no).

[‖]Scalable Algorithms Department, Sandia National Laboratories, Albuquerque, NM 87185 (egboman@sandia.gov). Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin company, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

[**]Department of Electrical & Computer Engineering, The Ohio State University, Columbus, OH 43210 (ozguner@ece.osu.edu).

are calculated using automatic differentiation or estimated using finite differencing [16]. Jacobian and Hessian matrices are needed in many algorithms in scientific computing, including algorithms for nonlinear optimization, differential equations, inverse problems, and sensitivity analysis. Distance-2 coloring also finds applications in other areas, such as in channel assignment problems in radio networks [21, 22]. Distance-1 coloring is used in numerous applications, including in discovering concurrency in parallel scientific computing [18, 19, 27].

In parallel applications where a distance-$k$ coloring is needed, the graph is either already partitioned and mapped or needs to be partitioned and mapped onto the processors of a distributed-memory parallel machine. Under such circumstances, gathering the graph on one processor to perform the coloring sequentially is prohibitively time-consuming or infeasible due to memory constraints. Hence the graph needs to be colored in parallel. Finding a distance-$k$ coloring using the fewest colors is an NP-hard problem [23], but greedy heuristics are effective in practice, as they run fast and provide solutions of acceptable quality [8, 16, 17]. They are, however, inherently sequential and thus challenging to parallelize.

We report in this paper on the design, analysis, implementation, and experimental evaluation of a variety of efficient greedy parallel algorithms that we have developed for distance-2 coloring in distributed-memory environments. We also present appropriate variants of the algorithms tailored for Jacobian and Hessian computation. The algorithms presented here are obtained by extending the parallelization framework that we developed in a recent work in the context of distance-1 coloring [6]. The framework is an iterative, data-parallel, algorithmic scheme that proceeds in two-phased rounds. In the first phase of each round, processors concurrently color the vertices assigned to them in a speculative manner, communicating at a coarse granularity. In the second phase, processors concurrently check the validity of the colors assigned to their respective vertices and identify a set of vertices that needs to be recolored in the next round to resolve any detected inconsistencies. The scheme terminates when every vertex has been colored correctly.

One of the challenges involved in extending the framework outlined above to the distance-2 coloring case is devising an efficient means of information exchange between processors hosting a pair of vertices that are two edges away from each other in the graph. For such pairs of vertices, relying on direct communication between the corresponding processors would incur unduly high communication cost, and locally storing duplicates of distance-2 neighborhoods would require unduly large memory space. Instead, we employ a strategy in which information is relayed via a third processor (the processor owning a mutual distance-1 neighbor of vertices two edges away from each other) as needed. We show that the parallel algorithms designed using this strategy yield good speedup with increasing number of processors while using nearly the same number of colors as a serial greedy algorithm. We also show that the algorithms outperform the alternative approach based on distance-1 coloring of a square graph.

A preliminary version of a small portion of the work presented here has appeared in an earlier conference paper [5]. Compared to [5], the current paper has several new contributions. In terms of the basic distance-2 coloring algorithm for general graphs, the algorithm has been described much more rigorously, its complexity has been analyzed, and possible variations of the algorithm have been outlined; moreover, the experimental performance evaluation of the algorithm is conducted much more thoroughly and is carried out on a larger set of test problems and on a larger number of processors. In addition, new algorithms for distance-2 coloring of bipartite graphs (for Jacobian computation) and restricted star coloring of general graphs (for Hessian computation) have been presented and experimentally evaluated.

To the best of our knowledge, this paper is the first to present demonstrably efficient and scalable parallel algorithms for distance-2 coloring on distributed-memory architectures. Gebremedhin, Manne, and Pothen [15] have developed shared-memory parallel algorithms for distance-2 coloring. They have also provided a comprehensive review of the role of graph coloring in derivative computation in [16], and designed efficient serial algorithms for acyclic and star coloring (which are used in Hessian computation) in [17]. The advantage of using those star and acyclic coloring algorithms in the context of computing Hessians using automatic differentiation was demonstrated in [13]. The literature on algorithmic graph theory features some work related to the distance-2 coloring problem [1, 2, 21, 22]. Readers are referred to section 11.4 of [16] and section 2.3 of [17] for more pointers to theoretical work on distance-$k$ and related coloring problems.

The remainder of this paper is organized as follows. Section 2 provides background: it includes a self-contained review of the coloring models for sparse derivative matrix computation that are relevant for this paper, and a brief discussion of serial greedy coloring algorithms, since they form the foundation for the parallel algorithms presented here. Section 3 sets the stage for a detailed presentation of the parallel distance-2 coloring algorithm for general graphs in section 4 by discussing several algorithm *design* issues. Section 5 shows how the algorithm described in section 4 can be adapted for restricted star coloring of general graphs (for Hessian computation) and distance-2 coloring of bipartite graphs (for Jacobian computation). Section 6 contains a detailed computational evaluation of the performance of the parallel algorithms, and section 7 concludes the paper.

## 2. Background.

### 2.1. Preliminary concepts and notation.

Two distinct vertices in a graph are *distance-$k$* neighbors if a shortest path connecting them consists of at most $k$ edges. We denote the set of distance-$k$ neighbors of a vertex $v$ by $N_k(v)$. The *degree-$k$* of a vertex $v$, denoted by $d_k(v)$, is the number of distinct paths of length at most $k$ edges starting at $v$. Two paths are distinct if they differ in at least one edge. Note that $d_1(v) = |N_1(v)|$, $d_2(v) = \sum_{w \in N_1(v)} d_1(w)$, and, in general, $d_k(v) \geq |N_k(v)|$. We denote the average degree-$k$ in a graph by $\overline{d}_k$.

A *distance-$k$ coloring* of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a mapping $\phi : \mathcal{V} \to \{1, 2, \ldots, q\}$ such that $\phi(v) \neq \phi(w)$ whenever vertices $v$ and $w$ are distance-$k$ neighbors. A distance-$k$ coloring of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is equivalent to a distance-1 coloring of the $k$th *power* graph $\mathcal{G}^k = (\mathcal{V}, \mathcal{F})$, where $(v, w) \in \mathcal{F}$ whenever vertices $v$ and $w$ are distance-$k$ neighbors in $\mathcal{G}$. A distance-$k$ coloring of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ can equivalently be viewed as a partition of the vertex set $\mathcal{V}$ into $q$ *distance-$k$ independent sets*—sets of vertices at a distance greater than $k$ edges from each other. Variants of distance-$k$ coloring are used in modeling partitioning problems in sparse Jacobian and Hessian computation. We review these in the next subsection; for a more comprehensive discussion, see [16, 17].

### 2.2. Coloring models in derivative computation.

The computation of a sparse $m \times n$ derivative matrix $A$ using automatic differentiation (or finite differencing) can be made efficient by first *partitioning* the $n$ columns into $q$ disjoint groups, with $q$ as small as possible, and then evaluating the columns in each group jointly (as a sum) rather than separately. More specifically, the values of the entries of the matrix $A$ are obtained by first evaluating a *compressed* matrix $B \equiv AS$, where $S$ is an $n \times q$ *seed* matrix whose $(j, k)$ entry $s_{jk}$ is such that $s_{jk}$ equals one if and only if column $a_j$ belongs to group $k$ and zero otherwise, and then recovering the entries of $A$ from $B$.

The specific criteria used to define a seed matrix $S$ for a derivative matrix $A$ depends on whether the matrix $A$ is Jacobian (nonsymmetric) or Hessian (symmetric). It also depends on whether the entries of $A$ are to be recovered from the compressed representation $B$ *directly* (without any further arithmetic), via *substitution* (by implicitly solving a set of simple triangular systems of equations), or via *elimination* (by solving a rectangular system of equations). In this paper we focus on only direct methods.

**2.2.1. Structurally orthogonal partition.** Curtis, Powell, and Reid [10] showed that a *structurally orthogonal* partition of a Jacobian matrix $A$—a partition of the columns of $A$ in which no two columns in a group share a nonzero at the same row index—gives a seed matrix $S$ where the entries of $A$ can be directly recovered from the compressed representation $B \equiv AS$. The structure of a Jacobian matrix $A$ can be represented by the *bipartite* graph $\mathcal{G}_b(A) = (\mathcal{V}_1, \mathcal{V}_2, \mathcal{E})$, where $\mathcal{V}_1$ is the row vertex set, $\mathcal{V}_2$ is the column vertex set, and $(r_i, c_j) \in \mathcal{E}$ whenever the matrix entry $a_{ij}$ is nonzero. A partitioning of the columns of the matrix $A$ into groups consisting of structurally orthogonal columns is equivalent to a *partial distance-*2 *coloring* of the bipartite graph $\mathcal{G}_b(A)$ on the vertex set $\mathcal{V}_2$ [16]. The coloring is called "partial" because the row vertex set $\mathcal{V}_1$ is left uncolored.

A structurally orthogonal column partition could also be used in computing a Hessian via a direct method, albeit that symmetry is not exploited. Specifically, McCormick [23] showed that a structurally orthogonal partition of a Hessian is equivalent to a distance-2 coloring of its *adjacency graph*. The adjacency graph $\mathcal{G}_a(A)$ of a Hessian $A$ has a vertex for each column, and an edge joins column vertices $c_i$ and $c_j$ whenever the entry $a_{ij}$, $i \neq j$, is nonzero; the diagonal entries in $A$ are assumed to be nonzero, and they are not explicitly represented by edges in the graph $\mathcal{G}_a(A)$. Figure 2.1 illustrates how a structurally orthogonal column partition of a matrix is modeled by a distance-2 coloring in the appropriate graph. The rightmost subfigures show the equivalent distance-1 coloring formulations in the appropriate square graph.

**2.2.2. Symmetry-exploiting partition.** Powell and Toint [26] were the first to introduce a symmetry-exploiting technique for computing a Hessian via a direct method. When translated to a coloring $\phi$ of the adjacency graph, the partition Powell and Toint suggested for a direct Hessian computation requires that (1) $\phi$ be a distance-1 coloring, and (2) in every path $v, w, x$ on three vertices, the terminal vertices $v$ and $x$ be allowed to have the same color, but only if the color of the middle vertex $w$ is lower in value. A coloring that satisfies these two requirements has been called a *restricted star coloring* in [17].

Coleman and Moré [9] showed that a *symmetrically orthogonal partition* of a Hessian is sufficient for a direct recovery, and established that such a partition is equivalent to a *star coloring* of the adjacency graph of the Hessian. A star coloring is a distance-1 coloring where, in addition, every path on four vertices uses at least three colors. The name is due to the fact that in a star-colored graph, a subgraph induced by any two color classes is a *collection of stars*. Note that the three coloring models for direct Hessian computation discussed here can be ranked in increasing order of restriction (decreasing order of accuracy) as *star coloring, restricted star coloring, distance-*2 *coloring*.

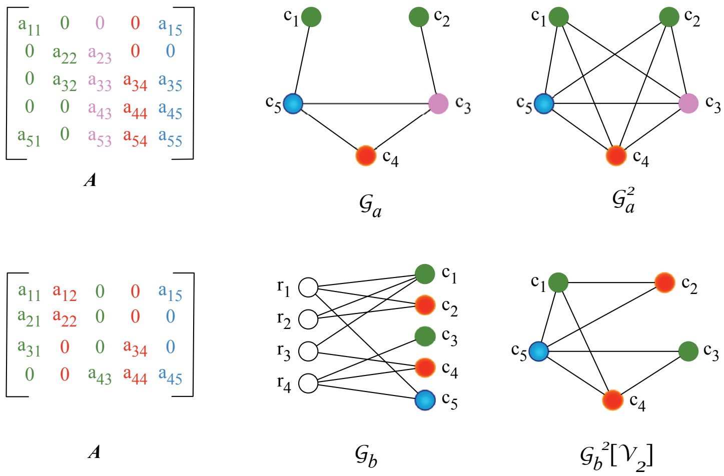



FIG. 2.1. *Equivalence among structurally orthogonal column partition of a matrix $A$, distance-2 coloring of the graph $\mathcal{G}(A)$ of $A$, and distance-1 coloring of $\mathcal{G}^2(A)$. Top: symmetric case. Bottom: nonsymmetric case, where $\mathcal{G}_b^2[\mathcal{V}_2]$ denotes the subgraph of the square graph $\mathcal{G}_b^2$ induced by the vertex set $\mathcal{V}_2$.*

**2.3. Greedy coloring algorithms.** An optimization problem associated with distance-$k$, restricted star, or star coloring asks for an appropriate coloring with the *fewest* colors, and each is known to be NP-hard [9, 17, 23]. In practice, *greedy* algorithms have been found effective in delivering good suboptimal solutions for these problems quickly [8, 17]. A greedy algorithm for each of these problems progressively extends a partial coloring of a graph by processing one vertex at a time, in some order; there exist a number of effective ordering techniques that are based on some variation of vertex degree [8, 17]. In the step where a vertex $v$ is colored, first, a set $F$ of *forbidden colors* for the vertex $v$ is obtained by exploring the appropriate neighborhood of $v$. Then, the *smallest* allowable color (not included in $F$) is chosen and assigned to $v$.

In the case of distance-1 coloring, such a greedy algorithm uses at most $\Delta + 1$ colors, where $\Delta$ is the maximum degree-1 in the graph. The quantity $\Delta + 1$ is a lower bound on the optimal number of colors needed in a distance-2 coloring. Furthermore, the number of colors used by a greedy distance-2 coloring algorithm is bounded from above by $\min\{\Delta^2 + 1, n\}$, where $n$ is the number of vertices in the input graph. Using this bound, McCormick [23] showed that the greedy distance-2 coloring algorithm is an $O(\sqrt{n})$-approximation algorithm.

Greedy algorithms for distance-1 and distance-2 coloring can be implemented such that their respective complexities are $O(n\overline{d}_1)$ and $O(n\overline{d}_2)$. Gebremedhin et al. [17] have developed $O(n\overline{d}_2)$-time greedy algorithms for star and restricted star coloring. Their algorithm for star coloring takes advantage of the structure of the two-colored induced subgraphs—the collection of stars—and uses fairly complex data structures to maintain them. In this paper we develop parallel versions of the greedy algorithms for distance-2 and restricted star coloring. We considered the simpler variant, restricted star coloring, instead of star coloring, since restricted star coloring can be derived via a simple modification of the parallel algorithm for the distance-2 coloring problem, which is the main focus of this paper.

**3. Design issues.** The parallel distance-2 coloring algorithm proposed in this paper will be presented in detail in section 4. Here we discuss the major issues that arise in the design of the algorithm and the assumptions and decisions we made around them.

**3.1. Data distribution.** The way in which the input graph is partitioned and mapped to processors has implications for both load balance and interprocessor communication. A graph could be partitioned among processors either by partitioning the vertex set or by partitioning the edge set. Traditionally, vertex partitioning has been the most commonly used strategy for mapping graphs (or matrices) to processors [4, 7, 12]. When a vertex partitioning is used, edges are implicitly mapped to processors, with every crossing edge essentially being duplicated on the two processors to which its endpoints are mapped. For matrices, this corresponds to mapping of entire columns or rows to processors. To make our algorithm and its implementation more readily usable in other parallel codes, we assume that a vertex partitioning is used in distributing the graph among processors. A vertex partitioning classifies the vertices of the graph into two categories: *interior* and *boundary*. An interior vertex is a vertex all of whose distance-1 neighbors are mapped onto the same processor as itself. A boundary vertex has at least one distance-1 neighbor mapped onto a different processor.

**3.2. Data duplication and communication mechanism.** The next design issue is data duplication and its impact on information exchange.

As stated earlier, when a vertex partitioning is used, every crossing edge is *duplicated*, and that was the approach used in our earlier work on distance-1 coloring [6]. In such a mapping strategy, it makes sense for each processor to store the colors of the off-processor endpoints of its crossing edges, as this would require storing at most one extra color per crossing edge. In terms of communication, such a storage scheme necessitates that each processor send the colors of its boundary vertices to neighboring processors as soon as the colors become available. Each receiving processor could then store the information and use it later while coloring its own vertices. In this way, the color of a boundary vertex is sent at most once for each of its incident crossing edges.

In the distance-2 coloring case, for each vertex, color information about vertices that are two edges away is also needed. One way of acquiring this information would be for each processor to keep a local copy of the subgraph induced by off-processor vertices that are within a distance of two edges from its own boundary vertices. Then, each processor could store and have access to all the needed color information as soon as the information was received from neighboring processors. Thus, as in the distance-1 coloring case, color information could be sent as soon as it became available. This would in turn allow for a flexible coloring order on each processor, since the order in which vertices are colored can be freely determined as the algorithm proceeds. However, this flexibility comes at the expense of extra storage. For relatively dense graphs, large portions of the input graph may need to be duplicated on each processor. In fact, this could happen even if there were just one high degree boundary vertex. For this reason, we chose to duplicate just the boundary vertices and their colors, and no more.

With this design decision in place, each processor will gain local access to distance-1 color information just as in the distance-1 coloring algorithm, and the information can be exchanged among the processors at the earliest possible time. But a mechanism for exchanging color information among vertices that are two edges apart still needs to be devised. Since such colors are not going to be stored permanently on the receiving processor, each color will have to be resent every time it is needed. Here, there are two basic ways in which the communication can be coordinated: one can use either a *request-based protocol* or a *precomputed schedule*. In a request-based protocol, each processor would send a message to its neighboring processors asking for specific color information whenever it needs the information. It then receives the information, uses it, and discards it. With a precomputed schedule, each processor would know the order (at least partially) in which its neighbor processors are going to color their vertices. Thus a processor could itself determine what color information to send to its neighbors and when to do so. A request-based protocol gives rise to more communication than a precomputed schedule, but, on the other hand, it is more flexible as it allows for a completely independent coloring order on each processor. In our algorithm we chose to use a precomputed schedule. Even with a precomputed schedule, there exists an opportunity for using ordering techniques at a local level on each processor, but we forego a detailed study of such ordering techniques to limit the scope of this paper.

**4. Parallel distance-2 coloring of general graphs.** We are now ready to present the new parallel distance-2 coloring algorithm for a general graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. We begin in section 4.1 by providing an overview of the algorithm, and then present its details layer-by-layer in sections 4.2–4.4. The complexity of the algorithm is analyzed in section 4.5, and a brief discussion of possible variations of the algorithm is given in section 4.6. In section 5 we will show how the algorithm needs to be modified to solve the restricted star coloring problem, again on a general graph $\mathcal{G}$, and the partial distance-2 coloring problem on a bipartite graph $\mathcal{G}_b = (\mathcal{V}_1, \mathcal{V}_2, \mathcal{E})$.

**4.1. Overview of the algorithm.** Initially, the input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is assumed to be vertex-partitioned and distributed among the $p$ available processors. The set $V_i$ of vertices in the partition $\{V_1, \ldots, V_p\}$ of $\mathcal{V}$ is assigned to and colored by processor $P_i$. We say $P_i$ *owns* $V_i$. In addition, processor $P_i$ stores the adjacency lists of its vertices and the identities of the processors owning them. This initial data distribution classifies each set $V_i$ into sets of interior and boundary vertices ($V_i = I_i \cup B_i$). We call two processors $P_i$ and $P_j$ *neighbors* if at least one boundary vertex owned by processor $P_i$ has a distance-1 neighbor vertex owned by processor $P_j$.

Clearly, any two interior vertices owned by two different processors can safely be colored concurrently in a distance-2 coloring. In contrast, a concurrent coloring of a pair of boundary vertices or a pair consisting of one boundary and one interior vertex may not be safe, as the constituents of the pair, while being distance-2 neighbors, may receive the same color and therefore result in a *conflict*. We avoid the latter situation for a potential conflict (due to a pair consisting of one boundary and one interior vertex) by requiring that interior vertices be colored strictly before or strictly after boundary vertices have been colored. Then, a conflict can occur only for pairs of boundary vertices. Thus, the central part of the algorithm being presented is concerned with how the coloring of the boundary vertices is performed in parallel.

The main idea is to perform the coloring of the boundary vertices concurrently in a speculative manner and then detect and rectify conflicts that may have arisen. The algorithm (iteratively) proceeds in *rounds*, each consisting of a *tentative coloring* and a *conflict detection* phase. Both of these phases are performed in parallel. To reduce the frequency of communication among processors, the tentative coloring phase is organized in a sequence of *supersteps*, a term borrowed from the literature on the bulk synchronous parallel model [4] and used here in a loose sense. Specifically, in each superstep, each processor colors a prespecified number $s$ of the vertices that

**Algorithm 1.** Overview of the parallel distance-2 coloring algorithm.

**procedure** ParallelColoring($\mathcal{G} = (\mathcal{V}, \mathcal{E}), s$)
  *Data distribution*: $\mathcal{G}$ is divided into $p$ subgraphs $G_1 = (V_1, E_1), \ldots, G_p = (V_p, E_p)$, where $V_1, \ldots, V_p$ is a partition of the set $\mathcal{V}$ and $E_i = \{(v, w) : v \in V_i, (v, w) \in \mathcal{E}\}$. Processor $P_i$ owns the vertex set $V_i$ and stores the edge set $E_i$ and the ID's of the processors owning the other endpoints of $E_i$.
  **on each** processor $P_i$, $i \in P = \{1, \ldots, p\}$,
    $I_i \leftarrow$ interior vertices in $V_i$
    $B_i \leftarrow$ boundary vertices in $V_i$ ▷ $V_i = I_i \cup B_i$
    Color the vertices in $I_i$.
    Assign each vertex $v \in B_i \cup N_1(B_i)$ a random number $rand(v)$, generated using $v$'s ID as *seed*.
    $U_i \leftarrow B_i$ ▷ $U_i$ is the current set of boundary vertices to be colored by $P_i$
    **while** $\exists j \in P$, $U_j \neq \emptyset$ **do**
      TentativelyColor($U_i, s$)
      $U_i \leftarrow$ DetectConflicts()

it owns in a sequential manner, using forbidden color information available at the beginning of the superstep, and only thereafter sends recent color information to *neighboring* processors. In this scenario, if two boundary vertices that are either adjacent or at a distance of exactly two edges from each other are colored during the same superstep, they may receive the same color and thus cause a conflict. The purpose of the subsequent detection phase is to discover such conflicts in the current round and accumulate a list of vertices on each processor that needs to be recolored in the next round to resolve the conflicts.

Given a pair of vertices involved in a conflict, it suffices to recolor only one of them to resolve the conflict. The vertex to be recolored is determined by making use of a *global* random function defined over all boundary vertices. In particular, each processor $P_i$ assigns a random number to each vertex in the set $B_i \cup N_1(B_i)$, where $B_i$ is the set of boundary vertices owned by $P_i$ and $N_1(B_i) = \bigcup_{w \in B_i} N_1(w)$. Each random number $rand(v)$ associated with the vertex $v$ is computed by a pseudorandom generator using the *global ID* of the vertex $v$ as a seed. This avoids the potential need for exchange of random values among processors, since the number $rand(v)$ computed on one processor for a given vertex $v$ would be the same as the number $rand(v)$ computed on another processor for the same vertex. The algorithm terminates when no more vertices are left to be recolored. A high-level structure of the algorithm is given in Algorithm 1. The routines TentativelyColor and DetectConflicts called in Algorithm 1 will be discussed in detail in sections 4.3 and 4.4, respectively. But first we discuss a few fundamental techniques employed in Algorithm 1.

**4.2. Conflict detection and relaying distance-2 color information.** In addition to the design issues on data distribution, data duplication, and communication protocol discussed in section 3, the way in which conflicts are detected is a major issue in the design of Algorithm 1. We employed a strategy in which, for every path $v, w, x$ on three vertices, the processor on which the vertex $w$ resides is responsible for detecting not only conflicts that involve the vertex $w$ and an adjacent vertex in $N_1(w)$ but also a conflict involving the vertices $v$ and $x$. We call the former (involving adjacent vertices) *type* 1 conflicts, and the latter (involving vertices two edges apart) *type* 2 conflicts. A type 1 conflict is detected by both of the implied processors, whereas a type 2 conflict is detected by the processor owning the middle vertex. Clearly, this

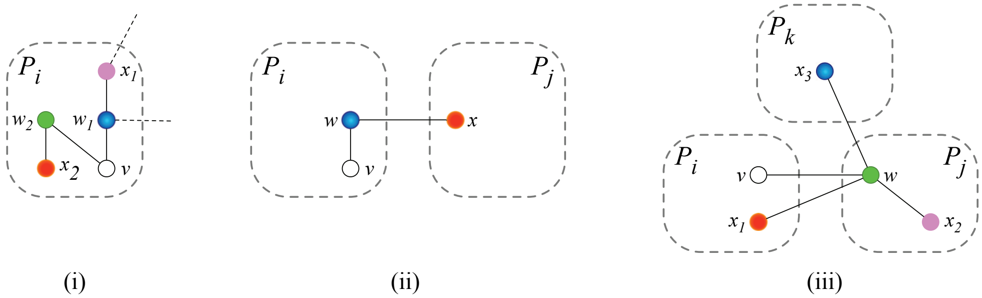



FIG. 4.1. *Scenarios depicting the distribution of the distance-2 neighbors of vertex $v$ across processors.*

way of detecting a type 2 conflict is more efficient than the alternative in which the conflict is detected by both of the processors owning the terminal vertices $v$ and $x$.

As the termination condition of the while-loop in Algorithm 1 indicates, even if a processor has no more vertices left to be recolored in a round, i.e., $U_i = \emptyset$, it could still be active in that round, as the processor may need to provide color information to other processors, participate in detecting conflicts on other processors, or both.

Another basic ingredient in the design of Algorithm 1 is the technique used to *build* the list of forbidden colors for a given vertex $v$ in a given superstep. The technique is directly related with the strategies on data duplication and communication protocol employed in the design of the algorithm (these were discussed in section 3.2). The next two paragraphs discuss elements of this technique.

Let $P_i$ be the processor that owns the vertex $v$. The list of forbidden colors for the vertex $v$ consists of colors assigned to adjacent vertices—those in the set $N_1(v)$—and colors assigned to vertices exactly two edges away from $v$. These colors are assigned either in a previous superstep (for boundary vertices) or prior to the iterative coloring (for interior vertices). We classify these colors as *local* or *nonlocal* relative to processor $P_i$ on the onset of the superstep. A color of a vertex $u$ is local to processor $P_i$ if $P_i$ owns either the vertex $u$ or some distance-1 neighbor of $u$ (in which case $P_i$ would store a copy of $u$'s color information, which is computed and sent by $u$'s owner). In contrast, a color is nonlocal to $P_i$ if the information is not locally stored and hence needs to be relayed via an "intermediate" processor.

Figure 4.1 shows the three scenarios in which the vertices on a path $v, w, x$ may be distributed among processors. Case (i) corresponds to the situation where both of the vertices $w$ and $x$ ($w_1$ and $x_1$ or $w_2$ and $x_2$ in the figure) are owned by processor $P_i$. Clearly, in this case, the colors of the vertices $w$ and $x$ are both local to $P_i$. Case (ii) shows the situation where vertex $w$ is owned by processor $P_i$ and vertex $x$ is owned by processor $P_j$, $j \neq i$. In this case, again, the colors of both vertices $w$ and $x$ are local to $P_i$. Case (iii) shows the situation where vertex $w$ is owned by processor $P_j$, and vertices $v$ and $x$ do not have a common distance-1 neighbor owned by processor $P_i$. As depicted in the figure, vertex $x$ may be owned by any one of the three processors $P_i$, $P_j$, or $P_k$, $i \neq j \neq k$ (shown as vertices $x_1$, $x_2$, and $x_3$, respectively). In this third case, if the vertex $x$ is owned by either of the processors $P_j$ or $P_k$ (shown as $x_2$ and $x_3$), then the color of $x$ is nonlocal to processor $P_i$ and needs to be relayed to $P_i$ through processor $P_j$. On the other hand, if the vertex $x$ is owned by processor $P_i$ (shown as $x_1$), then the color of $x$ is, strictly speaking, local to $P_i$ and needs not be relayed via $P_j$. However, in the algorithm being described, since processor $P_j$ does

**Algorithm 2.** Tentative coloring phase of Algorithm 1 run on processor $P_i$.

1: **procedure** TENTATIVELYCOLOR$(U_i, s)$
2: Partition $U_i$ into $n_i$ subsets $U_{i,1}, U_{i,2}, \ldots, U_{i,n_i}$, each of size $s$ ▷ $n_i = \lceil \frac{|U_i|}{s} \rceil$. Vertices in $U_{i,\ell}$ will be colored in the $\ell$th superstep by processor $P_i$.
3: **for each** processor-superstep pair $(j, \ell) \in \{\{1, \ldots, p\} \times \{1, \ldots, n_i\}\}$, $j \neq i$, **do**
4: $U^j_{i,\ell} \leftarrow \{v | v \in U_{i,\ell} \text{ and } N_1(v) \cap V_j \neq \emptyset\}$ ▷ processor $P_j$ is neighbor to processor $P_i$
5: Send schedules $U^j_{i,\ell}$ to each neighbor processor $P_j$.
6: Receive schedules $U^i_{j,\ell}$ from each neighbor processor $P_j$.
7: $X_{i,\ell} \leftarrow \bigcup_{j=1}^{j=p} U^i_{j,\ell}$ ▷ Vertices in $X_{i,\ell}$ will be colored in $\ell$th step by processors $P_j, j \neq i$
8: **for each** $v \in X_i \cup U_i$, where $X_i = \bigcup_{\ell=1}^{\ell=n_i} X_{i,\ell}$, **do**
9: $color(v) \leftarrow 0$ ▷ (re)initialize colors
10: $L \leftarrow \max_{1 \leq j \leq p} \{n_j\}$ ▷ $L$ is the max number of supersteps over all processors
11: **for** $\ell \leftarrow 1$ **to** $L$ **do** ▷ each $\ell$ corresponds to a superstep
12: Build lists of forbidden colors for vertices in $U^i_{j,\ell}$.
13: Send the lists to each neighboring processor $P_j$, where $\ell \leq n_j$.
14: **if** $\ell \leq n_i$, **then** ▷ $P_i$ has not finished coloring $U_i$
15: Receive lists of forbidden colors for vertices in $U^j_{i,\ell}$ from each neighboring $P_j$.
16: Merge the lists of forbidden colors.
17: Update the lists of forbidden colors with local color information.
18: **for each** $v \in U_{i,\ell}$ **do**
19: $color(v) \leftarrow c$ such that $c > 0$ is the smallest "permissible" color for $v$
20: Send updated colors of vertices in $U^j_{i,\ell}$ to each neighboring $P_j$.
21: Receive updated colors of vertices in $U^i_{j,\ell}$ from each neighboring $P_j$.

not store the adjacency lists of the vertices owned by processor $P_i$, it would treat the color of $x_1$ as if it were nonlocal to $P_i$ and send the color information to $P_i$. In other words, for every edge $(v, w)$ in which vertex $v$ is owned by processor $P_i$ and $w$ is owned by $P_j$, processor $P_j$ takes the responsibility of building a list of colors used by vertices two edges away from the vertex $v$ (a partial list of forbidden colors to $v$) and sending the list to processor $P_i$.

**4.3. The tentative coloring phase.** Algorithm 2 outlines in detail the routine TENTATIVELYCOLOR run on each processor $P_i$. The routine starts off by processor $P_i$ determining a coloring-schedule—a breakdown of its current set $U_i$ of vertices to be colored into supersteps (line 2). Processor $P_i$ then computes and sends schedule information to each of its neighboring processors (lines 3–5). Similarly, processor $P_i$ receives analogous schedules from each of its neighboring processors (line 6). This enables each processor to know the distance-2 color information that it needs to send in each superstep. In particular, using the schedules, for each superstep $\ell$, processor $P_i$ constructs a list $X_{i,\ell}$ of vertices that will be colored by some other processor $P_j$ in superstep $\ell$ and for which it must supply forbidden color information (line 7). Thus with the knowledge of each $X_{i,\ell}$, processor $P_i$ can be "proactive" in building up lists of relevant forbidden color information and sending these to neighboring processors in superstep $\ell$.

Before the coloring of the vertices in the set $U_i$ by processor $P_i$ commences, impermissible colors assigned in a previous superstep need to be cleared. These consist of colors assigned to vertices in $U_i$ (by processor $P_i$) and colors assigned to vertices in $X_i$, which are to be colored by other processors in the current round (lines 8 and 9).

Since for different processors $P_i$ the number of vertices $|U_i|$ to be colored could differ, the number of supersteps required to color these vertices, $n_i = \lceil \frac{|U_i|}{s} \rceil$, would also vary. Processor $P_i$ needs $n_i$ supersteps to color its vertices, but it may need to supply forbidden color information to a neighboring processor that has not finished coloring its vertices. Therefore, the algorithm overall needs $L = \max_{1 \le j \le p} \{n_j\}$ supersteps (see line 10).

In each superstep, before processor $P_i$ begins to color the set of vertices it owns, it proactively builds and sends relevant color information to neighboring processors (lines 12 and 13). Further, to perform the coloring of its own vertices in a superstep, a processor first gathers color information from other processors to build a partial list of forbidden colors for each of its boundary vertices scheduled to be colored in the current superstep. After the processor has received the partial lists of forbidden colors from all of its neighboring processors, it merges these lists and augments them with local color information to determine a complete list of forbidden colors for its vertices scheduled to be colored in the current superstep. Using this information, the processor then speculatively colors the vertices of the current superstep. At the end of the superstep, the new color information is sent to neighboring processors. These actions are performed in the piece of code in lines 14–20. Regardless of whether a processor has finished coloring all of its vertices or not, it needs to receive updated color information from neighboring processors (see line 21). This information is needed to enable the processor to compile forbidden color information to be sent to other processors in the next superstep.

**4.4. The conflict detection phase.** Algorithm 3 outlines the conflict detection routine DetectConflicts executed on each processor $P_i$. The routine has two major parts. In the first part, the routine finds a subset $W_i \subset V_i$ of vertices that processor $P_i$ needs to *examine* to detect both type 1 and type 2 conflicts. Whenever a conflict is detected, one of the involved vertices is selected to be recolored in the next round to resolve the conflict. The selection makes use of the random values assigned to boundary vertices in Algorithm 1. In the second part, the routine *determines* and returns a set of vertices to be recolored by processor $P_i$ in the next round.

Two vertices would be involved in a conflict only if they are colored in the same superstep. Thus the vertex set $W_i$ needs consist only of (1) every vertex $v \in U_i$ that has at least one distance-1 neighbor on a processor $P_j$, $j \neq i$, colored in the same superstep as $v$, and (2) every vertex $v \in V_i$ that has at least two distance-1 neighbors on different processors that are colored in the same superstep, since these might be assigned the same color. To obtain elements of the set $W_i$ that satisfy one or both of these two conditions in an efficient manner, in Algorithm 3 relevant vertices on processor $P_i$ are *traversed a superstep at a time*, using the schedule computed in Algorithm 2. In each superstep $\ell$, first each vertex in $U_{i,\ell}$ and its neighboring boundary vertices are marked. Then, for each vertex $v \in X_{i,\ell}$ the vertices in the set $N_1(v)$ owned by processor $P_i$ are marked. If this causes some vertex to be marked twice during the same superstep, then the vertex is added to $W_i$. The determination of the set $W_i$ is achieved by the piece of code in lines 3–7 of Algorithm 3; details are omitted for brevity.

Turning to the second part of Algorithm 3, processor $P_i$ accumulates a list $R_{i,j}$ of vertices to be recolored by each processor $P_j$ in the next round. To detect conflicts around a vertex $w$ in the set $W_i$, we need to look for vertices in the set $\bar{N}_1(w) = N_1(w) \cup \{w\}$ that have the same color. In a valid distance-2 coloring, every vertex in the set $\bar{N}_1(w)$ needs to have a distinct color. If several vertices with the same color are found, we let the vertex with the lowest random value keep its color and recolor the rest. To perform these tasks efficiently, we use two color-indexed, one-dimensional,

**Algorithm 3.** Conflict detection phase of Algorithm 1 run on processor $P_i$.

1: **function** DetectConflicts
2: $W_i \leftarrow \emptyset$ ▷ $W_i$ is the set of vertices $P_i$ examines to detect conflicts
3: **for** $\ell \leftarrow 1$ **to** $L$ **do** ▷ uses schedules computed in Algorithm 2
4: **for each** $w \in U_{i,\ell}$, where $w$ has at least one neighbor in $X_{i,\ell}$, **do**
5: $W_i \leftarrow W_i \cup \{w\}$ ▷ $w$ is used for detecting type 1 conflicts
6: **for each** $w \in V_i$, where $w$ has at least two neighbors in $X_{i,\ell} \cup U_{i,\ell}$ on different processors, **do**
7: $W_i \leftarrow W_i \cup \{w\}$ ▷ $w$ is used for detecting type 2 conflicts
8: **for each** $j \in P = \{1, \ldots, p\}$ **do**
9: $R_{i,j} \leftarrow \emptyset$ ▷ $R_{i,j}$ is a set of vertices $P_i$ notifies $P_j$ to recolor
10: **for each** $w \in W_i$ **do**
11: $encountered[color(w)] \leftarrow w$
12: $lowest[color(w)] \leftarrow w$
13: **for each** $x \in N_1(w)$ **do**
14: **if** $encountered[color(x)] = w$, **then**
15: $v \leftarrow lowest[color(x)]$
16: **if** $rand(v) \leq rand(x)$, **then** ▷ $rand(u)$: random number assigned to $u$
17: **if** $v \neq w$, **then**
18: $R_{i,Id(x)} \leftarrow R_{i,Id(x)} \cup \{x\}$ ▷ $Id(u)$: ID of processor owning $u$
19: **else**
20: $R_{i,Id(v)} \leftarrow R_{i,Id(v)} \cup \{v\}$
21: $lowest[color(x)] \leftarrow x$
22: **else**
23: $encountered[color(x)] \leftarrow w$
24: $lowest[color(x)] \leftarrow x$
25: **for each** $j \in P$, $j \neq i$ **do**
26: Send $R_{i,j}$ to processor $P_j$.
27: **for each** $j \in P$, $j \neq i$ **do**
28: Receive $R_{j,i}$ from processor $P_j$.
29: $R_{i,i} \leftarrow R_{i,i} \cup R_{j,i}$
30: **return** $R_{i,i}$

arrays—*encountered* and *lowest*—that store vertices. The values stored in the two arrays encode information that is updated and used in a for-loop that iterates over each vertex $w \in W_i$. The context in each iteration in turn is a visit through the neighborhood $N_1(w)$ of the vertex $w$. For each vertex $w$, $encountered[c] = w$ indicates that at least one vertex in the set $\bar{N}_1(w)$ having the color $c$ has been encountered, and $lowest[c]$ stores the vertex with the lowest random value among these. Initially, both $encountered[color(w)]$ and $lowest[color(w)]$ are set to be $w$. This ensures that any conflict involving the vertex $w$ and one of the vertices in the set $N_1(w)$ would be discovered. To detect conflicts involving the neighbors of $w$, the algorithm checks whether a given color used by a vertex in $N_1(w)$ has been encountered more than once, and, if so, the vertex to be recolored is determined using the random values assigned to the vertices and the array *lowest* is updated accordingly. See the for-loop in lines 10–24 for details.

In Algorithm 3, a type 1 conflict involving adjacent vertices is detected by both of the implied processors. Thus the if-test in line 17 is included to avoid sending unnecessary notification from one processor to the other. Note also that in line 13 it would have been sufficient to check for conflicts using only vertices ${N_1}'(w) \subseteq N_1(w)$ that belong to either $U_i$ or $X_i$. However, since determining the subset ${N_1}'(w)$ takes more time than testing for a conflict, we use the larger set $N_1(w)$ in line 13.

When the lists $R_{i,j}$ that processor $P_i$ accumulates are complete, processor $P_i$ sends each list $R_{i,j}$ to processor $P_j$, $j \neq i$, to notify the latter to do the recoloring (lines 25–26). Processor $P_i$ itself is responsible for recoloring the vertices in $R_{i,i}$ and therefore adds to $R_{i,i}$ notifications $R_{j,i}$ received from each neighboring processor $P_j$ (lines 27–29).

**4.5. Complexity analysis.** In Algorithm 2, the overall sequential work carried out by processor $P_i$ and its neighboring processors in order to perform the coloring of the vertices in $U_i$ is $O(\Sigma_{v \in U_i} d_2(v))$. Summing over all processors, the total work (excluding communication cost) involved in coloring the vertices in the set $U = \bigcup_{i=1}^{i=p} U_i$ is $O(\Sigma_{v \in U} d_2(v))$, which is equivalent to the complexity of a sequential algorithm for coloring the vertex set $U$.

Turning to the communication cost involved in Algorithm 2, note that for each vertex $v \in U_i$ every neighboring processor sends to processor $P_i$ the union of the colors used by vertices at exactly two edges from the vertex $v$, while the color of the vertex $v$ is sent to every processor that owns a distance-1 neighbor of $v$. Thus the total size of data exchanged while coloring the vertex $v$ is bounded by $O(d_2(v))$, which in turn gives the bound $O(\Sigma_{v \in U} d_2(v))$, where $U = \bigcup_{i=1}^{i=p} U_i$, on the overall communication cost of Algorithm 2.

In Algorithm 3, in determining the set $W_i$ (in lines 3–7), at most $|V_i|$ vertices are processed. The per-vertex work involved in this process is proportional to the degree-1 of the vertex. Hence, the time needed to determine $W_i$ is bounded by $O(|V_i|\overline{d}_1)$, where $\overline{d}_1$ is the average degree-1 in the input graph $\mathcal{G}$. Further, in each iteration of the for-loop over the set $W_i$ in lines 10–24, the set of vertices to be recolored is determined in time $O(d_1(w))$, which gives a complexity of $O(|W_i|\overline{d}_1)$ for the entire for-loop. Since $|W_i| \leq |V_i|$ clearly holds, the overall complexity of Algorithm 3 is $O(|V_i|\overline{d}_1)$.

With the complexities of Algorithms 2 and 3 just established and assuming that the number of *rounds* required in Algorithm 1 is sufficiently small, the total work carried out by all of the $p$ processors in coloring the input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is $O(|\mathcal{V}|\overline{d}_2)$, which is the same as the complexity of the sequential algorithm.

The assumption that the number of rounds needed by Algorithm 1 is small is realistic. It is supported empirically by the experimental results to be presented in section 6, where the number of rounds for test graphs with *millions* of edges and representing a wide variety of structures and processor count in the order of a *hundred* is observed to be in the order of *ten*. It is hard to provide a reasonable *analytical* estimate on the number of rounds needed, as such an analysis would involve several complicated factors, including the size and structure of the input graph, the manner in which vertices are partitioned among processors, the order in which vertices are colored, and the random numbers assigned to vertices.

Meanwhile, a very loose upper bound on the number of rounds is $s \cdot p$, where $s$ is the superstep size and $p$ is the number of processors. And here is why the bound holds. First, note that a total of $s \cdot p$ vertices are colored in one superstep, and recall that conflicts occur only between vertices colored in the same superstep. At the end of a round, only vertices involved in conflicts (and in fact only some of those) are marked to be recolored. If a vertex is not marked to be recolored at the end of a round, it will never be involved in conflicts in subsequent rounds (since its color will be forbidden to neighbors), and hence its color does not change. Algorithm 3 ensures that, among the vertices involved in conflicts, those vertices with the smallest random numbers in their neighborhood retain their colors, i.e., are not marked for recoloring. As a result, at the very least, one vertex among the $s \cdot p$ vertices colored in the same superstep is guaranteed to receive its final color in a round. Therefore, the number of rounds can never exceed $s \cdot p$.

In practice, the number of vertices that get their final colors in a round is likely to be much much greater than one, making the bound $s \cdot p$ extremely unlikely to ever be attained. In fact, as the experimental results in section 6 will show, typically, more than 90% of the vertices of an input graph get their final colors in the *first* round. And the remaining 10% of the vertices get their final colors in a small number of rounds since the number of conflicts drops rapidly between successive rounds.

**4.6. Variations.** Algorithm 1 and its subroutines Algorithms 2 and 3 could be specialized along several axes to result in a variety of derived algorithms. We discuss three of these axes.

First, in Algorithm 1, interior vertices ($I_i$) are colored *before* boundary vertices ($B_i$), but the reverse order could also be considered.

Second, while coloring the vertices in a superstep on each processor (see line 18 of Algorithm 2), the *natural* ordering of the vertices, a *random* ordering, or any other *degree-based ordering* technique could be used [17].

Third, the choice of a color for a vertex in a superstep (see line 19 of Algorithm 2) could be done in several different ways. For example, a *first fit* (FF), a *staggered first fit* (SFF), or a *randomized* coloring strategy could be used [6]. In the FF strategy, each processor chooses the *smallest* allowable color for a vertex, starting from color 1. In the SFF strategy, each processor $P_i$ chooses the *smallest* permissible color from the set $\{\lceil \frac{iK}{p} \rceil, \ldots, K\}$, where the initial estimate $K$ is set to be, for example, equal to the lower bound $\Delta + 1$ on the distance-2 chromatic number. If no such color exists, then the smallest permissible color in $\{1, \ldots, \lfloor \frac{iK}{p} \rfloor\}$ is chosen. If there still is no such color, the smallest permissible color greater than $K$ is chosen. Since the search for a color in SFF starts from different "base colors" for each processor, SFF is likely to result in fewer conflicts than FF.

**5. Parallel restricted star and partial distance-2 coloring.** The algorithms presented in the previous section need to be modified only slightly to solve the two related problems of our concern, restricted star coloring on a general graph (for Hessian computation) and partial distance-2 coloring on a bipartite graph (for Jacobian computation). In this section we point out the specific changes that need to be made in Algorithms 1–3 to address these two problems.

**5.1. Restricted star coloring.** As the definition given in section 2.2 implies, in a restricted star coloring of a graph, the color assigned to a vertex $v$ needs to satisfy conditions that concern the distance-2 neighbors $N_2(v)$ of the vertex $v$. The exact same neighborhood is consulted in assigning a color for the vertex $v$ in a distance-2 coloring of the graph. Therefore, the greedy algorithms for the two variants of coloring (as developed in [17]) differ only in the way the set of forbidden colors for the vertex $v$ is determined. We describe this difference below in the sequential setting; the additional differences that arise in the parallel setting are fairly straightforward to implement, and their discussion is omitted.

Consider the step of a greedy algorithm (distance-2 or restricted star coloring) in which the vertex $v$ is to be colored, and let $v, w, x$ be a path in the distance-2 neighborhood of the vertex $v$. In the distance-2 coloring algorithm, both $color(w)$ and $color(x)$ are forbidden to the vertex $v$, since the path needs to have three distinct colors. In the restricted star coloring algorithm, on the other hand, $color(w)$ would always be forbidden to $v$, whereas $color(x)$ may or may not be forbidden. The decision in the latter case is made based on further tests:

- If $color(w) = 0$ (i.e., vertex $w$ is not yet colored), then $color(x)$ is forbidden to $v$. This ensures that any extension $v, w, x, y$ of the path $v, w, x$ would end up using at least three colors, as desired, since in the step in which the vertex $w$ is colored the distance-1 coloring requirement guarantees that the vertex $w$ gets a color distinct from $color(v)$ and $color(x)$.
- If $color(w) \neq 0$, then $color(x)$ is forbidden to $v$ only if $color(w) > color(x)$.

**5.2. Partial distance-2 coloring.** Here, the input is a bipartite graph $\mathcal{G}_b = (\mathcal{V}_1, \mathcal{V}_2, \mathcal{E})$, and only the vertex set $\mathcal{V}_2$ needs to be colored satisfying the condition that a pair of vertices from $\mathcal{V}_2$ sharing a common distance-1 neighbor in $\mathcal{V}_1$ receive different colors. If not already distributed, the graph $\mathcal{G}_b$ needs to be partitioned among the processors in a manner that minimizes boundary vertices (and hence the likelihood of conflicts and the size of overall communication). Assuming that a vertex partitioning is used, let $V_{1,1}, \ldots, V_{1,p}$ denote the partitioning of the set $\mathcal{V}_1$, and let $V_{2,1}, \ldots, V_{2,p}$ denote the partitioning of the set $\mathcal{V}_2$. The subgraph assigned to processor $P_i$ would then be $G_{b,i} = (V_{1,i}, V_{2,i}, E_i)$, where $E_i$ is the set of edges incident on vertices in $V_{1,i} \cup V_{2,i}$. In terms of the underlying matrix, such a partitioning means that each processor owns a subset of the rows as well as a subset of the columns; this is in contrast with the case where either only the columns or only the rows are partitioned. With such a partitioning in place, the only change that needs to be made in Algorithm 1 is to replace the two references to $V_i$ (in lines 4 and 5) with $V_{2,i}$.

The changes that need to be made in Algorithms 2 and 3 are minimal as well, and they are consequences of the basic difference between distance-2 coloring in a general graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and partial distance-2 coloring in a bipartite graph $\mathcal{G}_b = (\mathcal{V}_1, \mathcal{V}_2, \mathcal{E})$ illustrated by the code fragment in Algorithm 4.

**Algorithm 4.** Basic difference between distance-2 and partial distance-2 coloring.

**procedure** D2COLORING($\mathcal{G} = (\mathcal{V}, \mathcal{E})$)
  **for each** $v \in \mathcal{V}$ **do**
    **for each** $w \in N_1(v)$ **do**
      Forbid $color(w)$ to $v$.
      **for each** $x \in N_1(w)$ **do**
        Forbid $color(x)$ to $v$.
    Assign a color to vertex $v$.

**procedure** PD2COLORING($\mathcal{G}_b = (\mathcal{V}_1, \mathcal{V}_2, \mathcal{E})$)
  **for each** $v \in \mathcal{V}_2$ **do**
    **for each** $w \in N_1(v)$ **do**
      $\triangleright$ $w \in \mathcal{V}_1$ never receives a color
      **for each** $x \in N_1(w)$ **do**
        Forbid $color(x)$ to $v$. $\triangleright$ $x \in \mathcal{V}_2$
    Assign a color to vertex $v$.

**6. Experiments.** We have implemented the algorithms presented in the previous two sections in the programming language C and using the message-passing interface (MPI) library. We report in this section on experimental evaluation of the performance of the algorithms. In particular, we present in sections 6.4 and 6.5 detailed experimental results on the performance of the parallel distance-2 coloring algorithm for general graphs, and in sections 6.6 and 6.7 results on restricted star coloring of general graphs and partial distance-2 coloring of bipartite graphs, respectively. In section 6.8 we present results on the alternative approach of obtaining a distance-2 coloring of a graph by a distance-1 coloring of the square graph. We begin the section with a discussion of the test platform and test graphs used (largely for the experiments on the parallel algorithm for distance-2 coloring of general graphs) in section 6.1, a comment on the performance of the underlying sequential algorithms in section 6.2, and a discussion of the choice of a superstep size for the parallel algorithms in section 6.3. Some of the experiments on the restricted star coloring and

TABLE 6.1

*Structural properties of the* application *(top), and the* synthetic *(bottom) test graphs used in many of the experiments. The last column shows the number of edges in $\mathcal{G}^2 = (\mathcal{V}, \mathcal{F})$ compared to $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. (ST–structural engineering* [11]*, SH–ship section* [3]*, CA–linear car analysis* [3]*, AU–automotive* [11]*, CE–civil engineering* [11]*.)*

| App/ class | Name | $\lvert\mathcal{V}\rvert$ | $\lvert\mathcal{E}\rvert$ | Degree Max | Degree Avg | Colors $d1$ | Colors $d2$ | Exec. time (s) $d1$ | Exec. time (s) $d2$ | $\frac{\lvert\mathcal{F}\rvert}{\lvert\mathcal{E}\rvert}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| ST | nasasrb | 54,870 | 1,311,227 | 275 | 48 | 41 | 276 | 0.049 | 2.237 | 3.2 |
| | ct20stif | 52,329 | 1,323,067 | 206 | 51 | 49 | 210 | 0.063 | 2.581 | 3.8 |
| | pwtk | 217,918 | 5,708,253 | 179 | 52 | 48 | 180 | 0.229 | 10.335 | 2.9 |
| SH | shipsec8 | 114,919 | 3,269,240 | 131 | 57 | 54 | 150 | 0.128 | 6.776 | 3.5 |
| | shipsec1 | 140,874 | 3,836,265 | 101 | 55 | 48 | 126 | 0.143 | 7.457 | 3.1 |
| | shipsec5 | 179,860 | 4,966,618 | 125 | 55 | 50 | 140 | 0.190 | 9.852 | 3.2 |
| CA | bmw7st1 | 141,347 | 3,599,160 | 434 | 51 | 54 | 435 | 0.167 | 6.730 | 3.3 |
| | bmw3_2 | 227,362 | 5,530,634 | 335 | 49 | 48 | 336 | 0.274 | 10.077 | 3.2 |
| | inline1 | 503,712 | 18,156,315 | 842 | 72 | 51 | 843 | 0.925 | 55.217 | 7.0 |
| AU | hood | 220,542 | 5,273,947 | 76 | 48 | 42 | 103 | 0.277 | 9.407 | 3.2 |
| | msdoor | 415,863 | 9,912,536 | 76 | 48 | 42 | 105 | 0.520 | 17.438 | 3.2 |
| | ldoor | 952,203 | 22,785,136 | 76 | 48 | 42 | 112 | 1.197 | 40.180 | 3.2 |
| CE | pkustk10 | 80,676 | 2,114,154 | 89 | 52 | 42 | 126 | 0.091 | 3.904 | 2.9 |
| | pkustk11 | 87,804 | 2,565,054 | 131 | 58 | 66 | 198 | 0.103 | 6.041 | 4.2 |
| | pkustk13 | 94,893 | 3,260,967 | 299 | 69 | 57 | 303 | 0.155 | 9.302 | 6.0 |
| planar | plan-1 | 4,072,937 | 12,218,805 | 40 | 6 | 9 | 41 | 3.435 | 18.880 | 3.4 |
| random | rand-1 | 400,000 | 2,002,202 | 27 | 10 | 9 | 41 | 0.644 | 6.676 | 11 |
| random | rand-2 | 400,000 | 4,004,480 | 45 | 20 | 12 | 101 | 1.242 | 21.977 | 21 |
| s. world | sw-1 | 400,000 | 1,998,542 | 468 | 10 | 18 | 469 | 0.345 | 13.909 | 31 |
| s. world | sw-2 | 400,000 | 3,993,994 | 880 | 20 | 27 | 882 | 0.632 | 50.954 | 59 |

partial distance-2 coloring experiments are conducted on additional test graphs, and these graphs will be presented in sections 6.6 and 6.7.

### 6.1. Experimental setup.

*Test platform.* The experiments are largely carried out on a 96-node PC cluster equipped with dual 2.4 GHz Intel P4 Xeon CPUs and 4 GB memory. The nodes of the cluster are interconnected via a switched 10 Gbps Infiniband network.

*Test graphs.* Our first test set consists of 15 *real-world* graphs obtained from five different application areas: structural engineering, ship section design, linear car analysis, automotive design, and civil engineering [14, 28, 11, 3]. The top part of Table 6.1 lists the number of vertices, number of edges, maximum degree-1, and average degree-1 in each of these test graphs. The graphs are classified according to the application area from which they are drawn. Table 6.1 also lists the number of colors and the execution time in seconds used by greedy sequential distance-1 and distance-2 coloring algorithms when each is run on a single node of our test platform. The last column lists the ratio between the number of edges in the square graph $\mathcal{G}^2$ and the number of edges in $\mathcal{G}$. These computed quantities will be used to gauge the performance of the proposed parallel coloring algorithms on the input graphs $\mathcal{G}$.

To be able to study the scalability of the proposed algorithms on a wider range of graph structures, we have also run experiments on *synthetically* generated graphs, which constitute our second test set. To represent extreme cases, we considered synthetic graphs drawn from three different classes: *random*, *planar*, and *small-world* graphs. (Loosely speaking, a small-world graph is a graph in which the distance between any pair of nonadjacent vertices is fairly small [25].) The random and small-world graphs are generated using the GTgraph synthetic graph generator suite [30].

The planar graphs are maximally planar—the degree of every vertex is at least 5—and are generated using the expansion method described in [24]; the generator is a part of the graph suite for the Second DIMACS Challenge [29]. The structural properties of these graphs as well as the number of colors and runtime used by sequential distance-1 and distance-2 coloring algorithms run on them are listed in the bottom part of Table 6.1.

In the runtimes reported in Table 6.1 as well as in later figures in this section, each individual test is an average of five runs. In the timing of the parallel coloring code, we assume that the graph is initially partitioned and distributed among the nodes of the parallel machine, and thus the times reported concern only coloring. In all of the experiments, the input graph is partitioned using the tool MeTiS [20].

**6.2. Performance of the sequential algorithms.** Before presenting results on the performance of the proposed parallel coloring algorithms, it is worth pointing out that the underlying sequential greedy algorithms, in the distance-1 as well as distance-2 coloring cases, performed remarkably well in terms of number of colors used on both the application and synthetic test graphs. Specifically, as Table 6.1 shows, the number of colors used by the greedy sequential distance-1 coloring algorithm is in most cases slightly below the average degree and far below the maximum degree, which is an *upper bound* on the optimal solution (the distance-1 chromatic number). Even more remarkably, the number of colors used by the greedy sequential distance-2 coloring algorithm in most cases is observed to be very close to the maximum degree, which is a *lower bound* on the optimal solution (the distance-2 chromatic number). This implies that the solution obtained by the greedy distance-2 coloring algorithm is in most cases either optimal or just a few colors more. In both the distance-1 and distance-2 coloring greedy algorithms, vertices were colored in the natural *order* in which they appear in the input graphs; i.e., no reordering techniques were used.

**6.3. Choice of superstep size.** Our first set of experiments is conducted to study the dependence of the performance of the parallel distance-2 coloring algorithm (Algorithm 1) on the choice of the superstep size $s$, the number of vertices colored in a superstep before communication takes place, for a fixed number of processors. The results of this set of experiments are given in Figure 6.1. The experiments are conducted using the *application* test graphs, and the reported results are *averages* over each class. The number of processors used in the experiments is 32; we obtained similar results while experimenting with various other numbers of processors, but for space considerations we report results only for 32 processors.

Figure 6.1(a) shows the number of conflicts, normalized relative to the total number of vertices in the graph, as a function of superstep size. Figure 6.1(b) shows a similar plot of the number of rounds required by the algorithm. As one would expect, the normalized number of conflicts and the number of rounds increased as the superstep size $s$ increased, but the rate of increase remained fairly low once the value of $s$ passed a few hundred. Furthermore, it can be seen that both of these quantities are small in magnitude: the normalized number of conflicts was at most about 15%, and the number of rounds was at most around 16. Figure 6.1(c) shows that, for values of $s$ above a few hundred, further increase in $s$ does not significantly influence speedup. Similarly, as shown in Figure 6.1(d), the number of colors does not vary significantly with $s$ and stays within 12% of the number of colors used by the sequential algorithm. In general, an "optimal" value for $s$ is likely to depend on both the properties of the graph being colored (size and density) and the platform on which the algorithm is

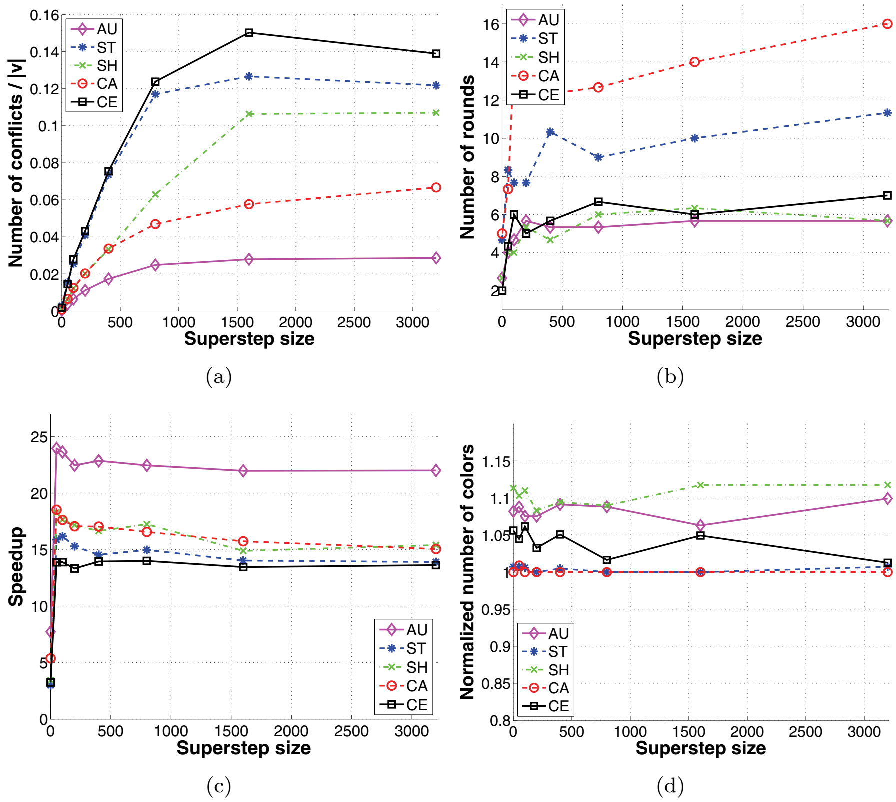



FIG. 6.1. *Normalized number of conflicts* (a)*, number of rounds* (b)*, speedup* (c)*, and normalized number of colors* (d) *in the parallel distance-2 coloring algorithm while the superstep size $s$ for the application test graphs listed in Table* 6.1 *is varied. In all cases, the number of processors was fixed at $p = 32$.*

executed. Based on observations from our parallel experiments, we used a superstep size of $s = 100$ for the scalability studies we report on in the rest of this section.

### 6.4. Results on parallel distance-2 coloring of general graphs.

**6.4.1. Strong scaling results.** Our second set of experiments is concerned with the strong scalability of the parallel distance-2 coloring algorithm as the number of processors used is varied. This set of experiments was conducted on both the application and synthetic test graphs listed in Table 6.1. The results from the experiments on the application graphs are summarized in Figure 6.2, and those from the synthetic graphs are summarized in Figure 6.3. In the results shown in Figure 6.2, for each application category, the *largest* graph from the category was used. A more detailed set of results on *all* of the application test graphs (for the distance-2 as well as the restricted star coloring algorithm) is included in a table in the appendix.

*Results on application test graphs.* The results in Figures 6.2(a) and (b) show that the number of conflicts and the number of rounds increase with increasing numbers of processors. However, the rate of growth as well as the actual values of both of these quantities were observed to be fairly small. For the graphs *pkustk*13 and *inline*1, the densest graphs in our test set, the numbers are relatively larger. The reason for the

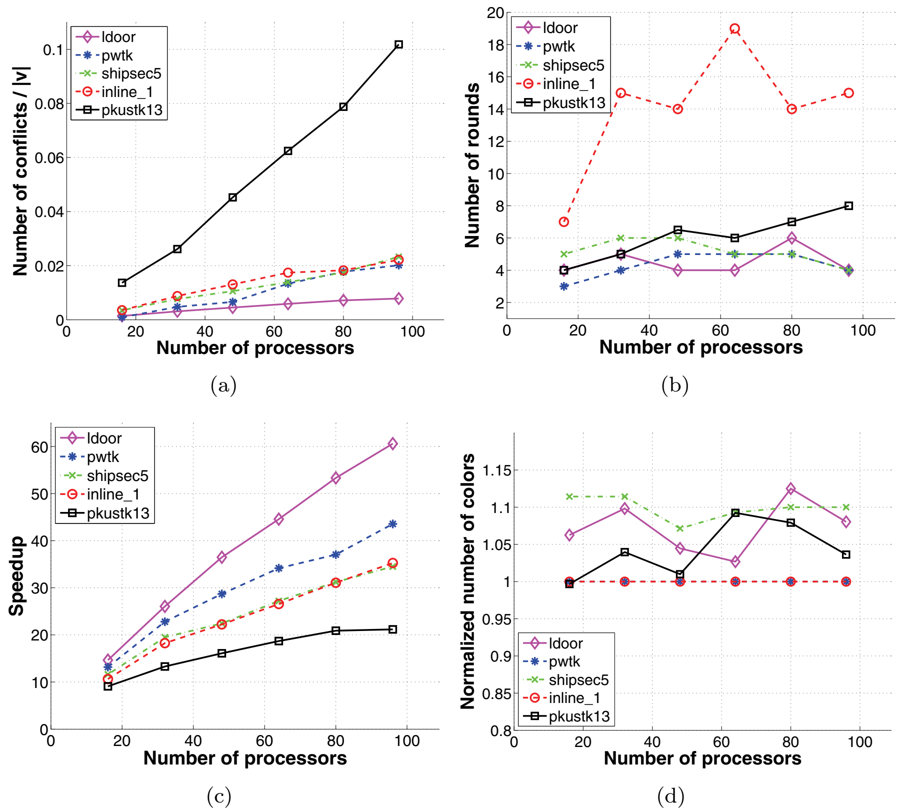



Fig. 6.2. *Normalized number of conflicts* (a)*, number of rounds* (b)*, speedup* (c)*, and normalized number of colors* (d) *in the parallel distance-2 coloring algorithm while the number of processors for the application test graphs listed in Table* 6.1 *is varied. In all cases, superstep size* $s = 100$ *was used.*

relatively large number of rounds required in coloring the graph *inline*1 is the existence of vertices with very large degree-1 values. This phenomenon will be explained further with the help of synthetic graphs.

The metrics *number of conflicts* and *number of rounds* are in some sense "intermediate" performance measures for our parallel coloring algorithms. They are included in the reports to help explain results on the two ultimate performance metrics—*speedup* and *number of colors used*. The lower row of Figure 6.2 shows results on the latter two metrics. We will use a similar set of four metrics in several other experiments reported in this section.

The speedup results in Figure 6.2(c) demonstrate that our algorithm in general scales well with increasing numbers of processors. As expected, the obtained speedups are relatively poorer in the cases where the number of conflicts and the number of rounds are relatively higher. The proposed algorithm also performed well in terms of the number of colors used. The results in Figure 6.2(d) show that the number of colors used by our parallel distance-2 algorithm, when as many as 96 processors are used, is at most 12% more than that used by the sequential algorithm. Recall that the solution obtained by the sequential algorithm in most cases is very close to the maximum degree in the graph—a lower bound on the optimal solution—and hence nearly optimal.

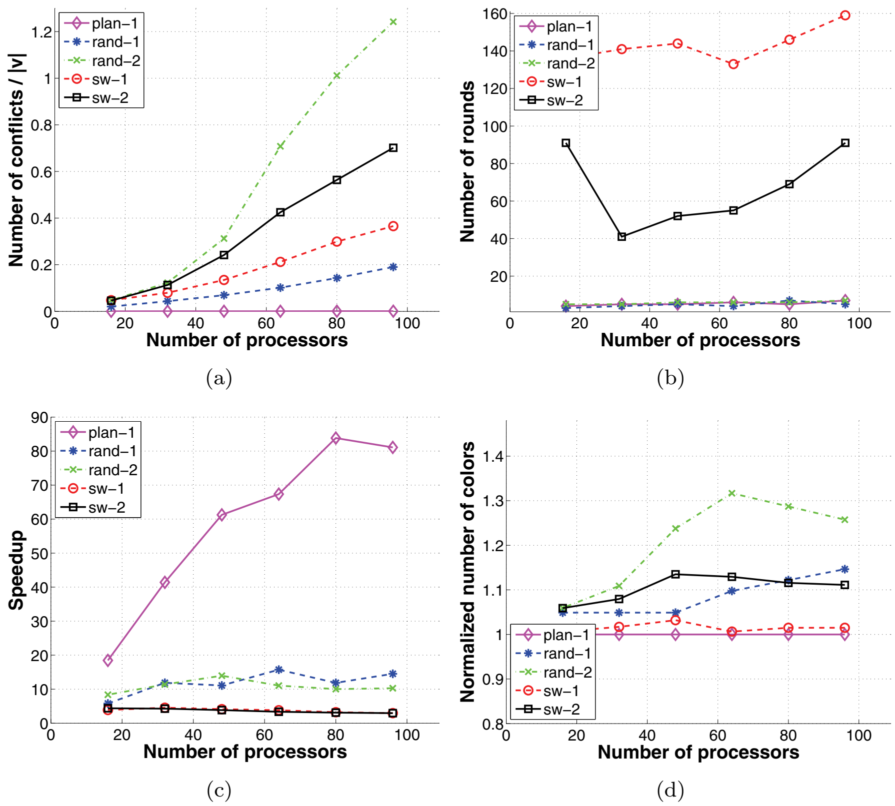



FIG. 6.3. *Normalized number of conflicts* (a), *number of rounds* (b), *speedup* (c), *and normalized number of colors* (d) *in the parallel distance-*2 *coloring algorithm while the number of processors for the synthetic test graphs in Table* 6.1 *is varied. In all cases, superstep size* $s = 100$ *was used.*

*Number of rounds.* Notice that the number of rounds in Figure 6.2(b) is observed to be at most 19 (achieved when 64 processors are used), which is very small compared to the bound $s \cdot p$ (which is equal to $100 \times 64 = 6{,}400$) discussed in section 4.5. The major reason for the huge difference between the bound $s \cdot p$ and the actual number of rounds is the fact that a vast majority of the vertices colored in a round get their final colors in the same round, and only a small fraction need to be recolored in a subsequent round. In other words, the number of conflicts detected in a round is small, and out of that small number only a portion will necessitate recoloring of vertices. To provide evidence for this fact, we present Figure 6.4, which shows results on the five application test graphs used in the experiments of Figure 6.2. Figure 6.4(a) shows the fraction of boundary vertices that get their final colors in each round of the distance-2 coloring algorithm until a valid coloring is achieved. For each test graph in the figure, the leftmost bar corresponds to the first round, the next bar to the right corresponds to the second round, and so on. It can be seen that typically more than 90% of the boundary vertices get their final colors in the very first round, and the remaining vertices eventually get their final colors in a matter of just a few rounds. Figure 6.4(b) shows how the number of vertices colored in a round translates into execution time. The bottom portion of the bars in this figure shows time spent in the tentative coloring phase, and the top portion shows the time spent in the conflict

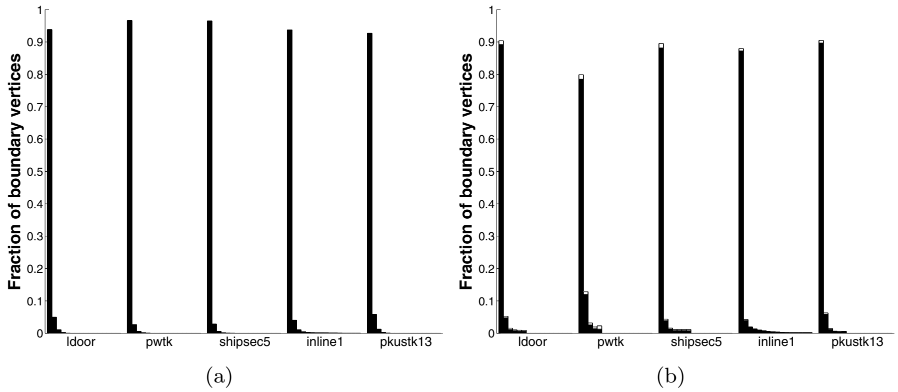



FIG. 6.4. *Fraction of boundary vertices colored* (a) *and fraction of time spent* (b) *in each round of the parallel distance-2 coloring algorithm. For each test graph, the leftmost bar corresponds to the first round, the next bar to the right corresponds to the second round, and so on. Each bar in* (b) *is divided into time spent on coloring (black) and time spent on conflict detection (white). In all cases, the number of processors was fixed at $p = 32$, and superstep size was $s = 100$.*

detection phase of each round of the distance-2 coloring algorithm. The results show that most of the time in a round is spent in the tentative coloring phase, and the total time spent in a round is proportional to the number of vertices colored in that round.

*Results on synthetic test graphs.* We now turn our attention to results on the synthetic graphs, which are designed to capture "extreme cases." The planar graph *plan*-1 represents partitionable graphs extremely well—almost all of the vertices of *plan*-1 became interior vertices in a partition obtained using the tool MeTiS [20]. For such graphs, parallel speedup in the context of Algorithm 1 comes largely from graph partitioning (data distribution) as opposed to the iterative coloring part. As Figure 6.3 shows, this results in a very small number of conflicts and good speedup.

The random graphs *rand*-1 and *rand*-2 represent the opposite extreme—almost all of the vertices in these graphs become boundary vertices regardless of how the graph is partitioned. As expected, Figure 6.3 shows that the number of conflicts is considerably larger for the random graphs. Nonetheless, our algorithm achieved some speedup (a constant around 10) even under such an extreme case. Note that the speedup in this case comes solely from the iterative coloring part.

As mentioned earlier, graphs having vertices with very large degrees comprise a particularly difficult set of instances for the proposed parallel distance-2 coloring algorithm. Resolving conflicts involving the neighbors of such vertices requires a large number of rounds, which in turn degrades speedup. The third category in our synthetic graphs, the small-world graphs *sw*-1 and *sw*-2, is included to represent such a class of pathological instances for our approach. As Figure 6.3 shows, even though the number of conflicts for small-world graphs is not necessarily larger than that for random graphs with similar sizes, coloring small-world graphs requires a significantly larger number of rounds. Consequently, the speedup achieved was poorer—it was around 3 for all numbers of processors used in our experiments.

As can be seen from Figure 6.3(d), the normalized number of colors for all the synthetic graphs except *rand*-2 is within 12% of the sequential algorithm. In the case of the graph *rand*-2, the algorithm involved relatively many conflicts but few rounds, implying that many processors had to recolor a fairly large number of vertices in each

| Name | $\|\mathcal{V}\|$ | $\|\mathcal{E}\|$ | Degree | |
|---|---|---|---|---|
| | | | Max | Avg |
| plan-2 | 1,625,972 | 4,877,910 | 37 | 6 |
| plan-3 | 3,250,939 | 9,752,811 | 43 | 6 |
| plan-4 | 4,888,479 | 14,665,431 | 43 | 6 |
| plan-5 | 6,513,589 | 19,540,761 | 38 | 6 |
| plan-6 | 8,150,267 | 24,450,795 | 39 | 6 |
| rand-3 | 160,000 | 1,600,528 | 45 | 20 |
| rand-4 | 320,000 | 3,201,327 | 46 | 20 |
| rand-5 | 480,000 | 4,803,946 | 43 | 20 |
| rand-6 | 640,000 | 6,403,242 | 44 | 20 |
| rand-7 | 800,000 | 8,005,505 | 45 | 20 |

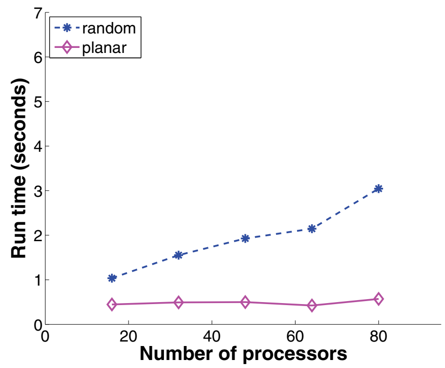



FIG. 6.5. *Properties of the synthetic test graphs used in weak scaling tests (left) and results for the parallel distance-*2 *coloring algorithm (right).*

round to resolve the conflicts. Over multiple rounds, this resulted in a significant increase in the number of colors used. This phenomenon is in contrast to that in small-world graphs, where many fewer vertices are recolored in many more rounds. The latter resembles sequential coloring; hence the number of colors increases only slightly.

**6.4.2. Weak scaling.** Our next set of experimental results is on weak scalability, where problem size and number of processors are increased in proportion so as to result in nearly constant runtime. The experiments were conducted on five instances from each of the random and planar graph classes. The structural properties of these graphs and the experimental results obtained using them are summarized in Figure 6.5. It can be seen that the algorithm behaved almost ideally for planar graphs—the runtime remained nearly constant as the problem size and the number of processors were increased in proportion. For the random graphs, the runtime grew, but it did so fairly slowly.

**6.5. Variations of the algorithm.** In the results presented thus far in this section, a variant of the proposed distance-2 coloring algorithm with the following combination of parameters was used: interior vertices were colored *before* boundary vertices; the *natural* ordering of the vertices was used while coloring; and an FF color choice strategy was employed. In addition to this "default" variant, we experimented with *seven* other variations obtained via various combinations in which interior vertices are colored *after* boundary vertices, a *random* vertex ordering is used while coloring, and an SFF color choice strategy (as discussed in section 4.6) is used. The experiments showed that the only variant that resulted in better performance in the majority of the test cases compared to the default variant was a variant that used the SFF coloring strategy. Figure 6.6 shows the performance of the SFF-coloring–based variant on the synthetic test graphs. Results on the application graphs are omitted, as no significant improvement over the FF-based variant was observed.

As Figure 6.6 shows, the performance improvement with SFF was especially significant for the random and small-world graphs. The number of conflicts and the number of rounds were much smaller for these graphs when SFF is used instead of FF; hence better speedup was achieved. For small-world graphs, the SFF strategy required significantly more colors than FF, whereas for random graphs SFF required

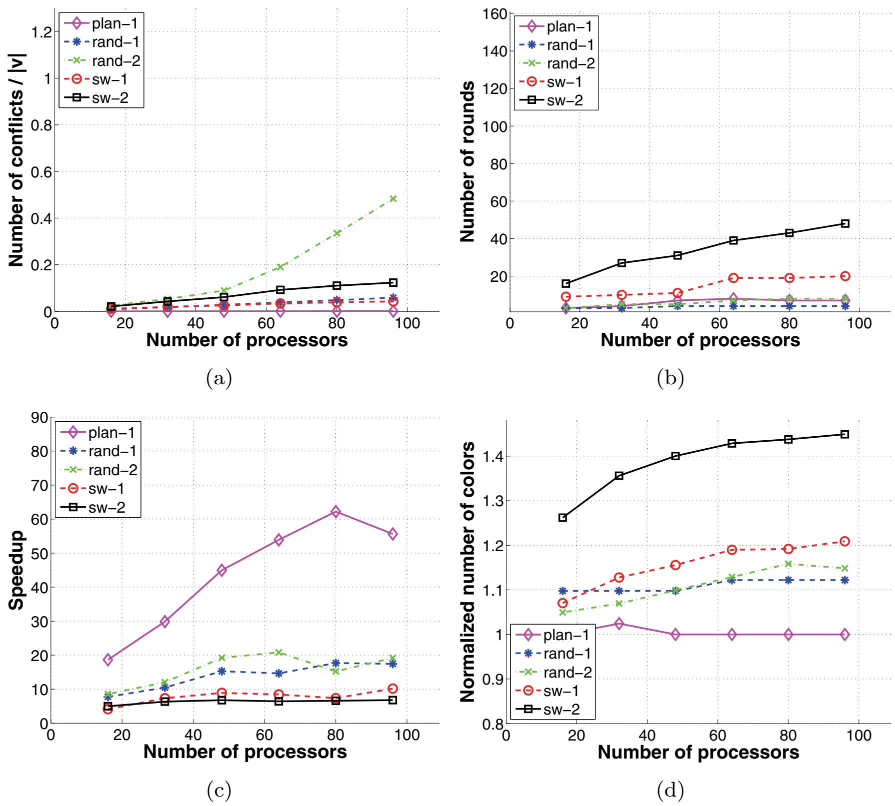
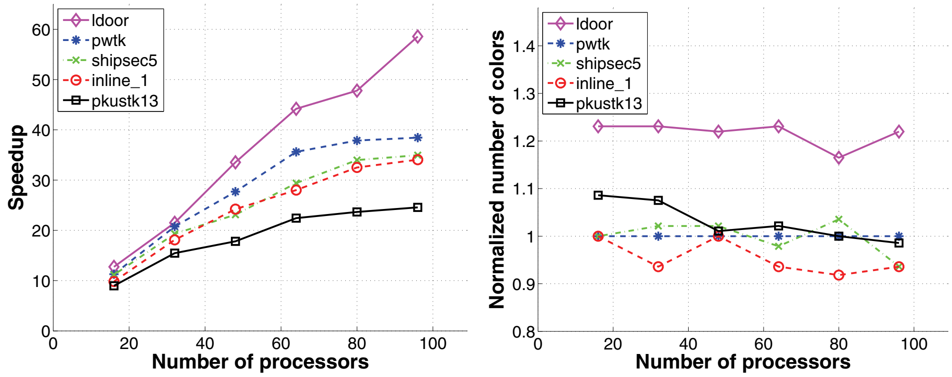



FIG. 6.6. *Normalized number of conflicts* (a)*, number of rounds* (b)*, speedup* (c)*, and normalized number of colors* (d) *in the parallel distance-2 coloring algorithm using an SFF color selection strategy and while the number of processors for the synthetic graphs listed in Table* 6.1 *was varied. In all cases, superstep size* $s = 100$ *was used.*

about the same number of colors for the random graph *rand*-1 and fewer colors for the denser random graph *rand*-2.

**6.6. Results on parallel restricted star coloring.** As discussed in section 5, a parallel restricted star coloring of a graph can be obtained via a slight modification of the parallel distance-2 coloring algorithm. We performed the necessary modifications and carried out experiments on the scalability of the resulting restricted star coloring algorithm. Figure 6.7(a) shows speedup results obtained when the restricted star coloring algorithm was run on the largest graph from each of the application graph classes listed in Table 6.1. As expected, these speedup results are very similar to those for distance-2 coloring (Figure 6.2(c)). The number of colors used by the parallel restricted star coloring algorithm normalized by the respective numbers used by the sequential restricted star coloring algorithm are given in Figure 6.7(b), and it can be seen that the increase in four of the five graphs is below 10%. As the table in Figure 6.7(c) shows, restricted star coloring of a graph could use significantly fewer colors than a distance-2 coloring, demonstrating the advantage of exploiting symmetry in Hessian computation. (For a more detailed listing of the number of colors and other information concerning restricted star and distance-2 coloring of *all* of the application graphs in Table 6.1, see Table A.1 in the appendix.)




(a) (b)

| | ldoor | pwtk | shipsec5 | inline1 | pkustk13 |
|---|---|---|---|---|---|
| *r.star* (nr. colors) | 91 | 180 | 141 | 843 | 279 |
| *distance*-2 (nr. colors) | 112 | 180 | 140 | 843 | 303 |

(c)

FIG. 6.7. (a) *and* (b) *Speedup and normalized number of colors of the parallel restricted star coloring algorithm on select application graphs from Table* 6.1. (c) *Number of colors used by sequential versions of the restricted star and the distance-*2 *coloring algorithms.*

TABLE 6.2
*Top: Properties of additional test graphs* [28, 17] *used for experiments on restricted star coloring. Bottom: Results of the parallel restricted star coloring algorithm on the listed graphs.*

| Name | $\|\mathcal{V}\|$ | $\|\mathcal{E}\|$ | Degree | | Number of colors | | | Execution time (s) | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Max | Avg | *d*1 | *r.star* | *d*2 | *d*1 | *r.star* | *d*2 |
| mol1-3 | 131,072 | 5,636,096 | 86 | 86 | 43 | 197 | 226 | 0.10 | 17.4 | 17.1 |
| mol1-4 | 131,072 | 14,680,064 | 224 | 224 | 100 | 528 | 583 | 0.23 | 106.3 | 107.4 |
| apoa1-10 | 92,224 | 17,100,850 | 503 | 371 | 182 | 976 | 1055 | 0.12 | 177.2 | 182.3 |
| apoa1-12 | 92,224 | 28,886,369 | 832 | 626 | 303 | 1688 | 1767 | 0.20 | 507.4 | 522.3 |
| mol1-6 | 131,072 | 50,200,576 | 766 | 766 | 301 | 1891 | 2009 | 0.72 | 1226.7 | 1273.2 |

| Name | Speedup | | | Normalized nr. of colors | | |
|---|---|---|---|---|---|---|
| | $p = 16$ | $p = 32$ | $p = 64$ | $p = 16$ | $p = 32$ | $p = 64$ |
| mol1-3 | 27.1 | 24.5 | 13.0 | 1.11 | 1.17 | 1.21 |
| mol1-4 | 20.0 | 32.5 | 36.3 | 1.17 | 1.24 | 1.31 |
| apoa1-10 | 20.0 | 30.1 | 35.4 | 1.18 | 1.27 | 1.51 |
| apoa1-12 | 18.9 | 24.9 | 25.6 | 1.20 | 1.45 | 1.81 |
| mol1-6 | 15.0 | 23.8 | 32.2 | 1.27 | 1.49 | 1.94 |

To further demonstrate the advantage of exploiting symmetry in Hessian computation via restricted star coloring, we tested the parallel restricted star coloring algorithm on five additional graphs obtained from a molecular dynamics application [28, 17]. The top part of Table 6.2 lists the structural properties of these graphs as well as the number of colors and the execution time in seconds used by the *sequential* greedy distance-1, distance-2, and restricted star coloring algorithms. The bottom part of Table 6.2 shows the speedup and normalized number of colors used by the *parallel* restricted star coloring algorithm on these graphs for various numbers of processors. These experiments were conducted on a platform different from the one used for the graphs listed in Table 6.1; the platform here is a cluster equipped with

Table 6.3

*Top: Structural properties of the bipartite graphs [11] used for partial distance-2 coloring experiments. Bottom: Results of the partial distance-2 coloring algorithm on the listed graphs.*

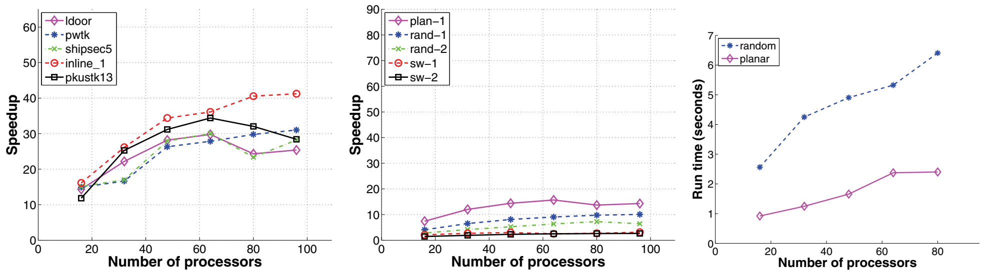
| Name | $\|\mathcal{V}_1\|$ | $\|\mathcal{V}_2\|$ | $\|\mathcal{E}\|$ | Degree in $\mathcal{V}_1$ | | Colors | Execution time (s) |
|---|---|---|---|---|---|---|---|
| | | | | Max | Avg | | |
| lhr71c | 70,304 | 70,304 | 1,528,092 | 63 | 10.9 | 65 | 1.43 |
| stormG2 | 528,185 | 1,377,306 | 3,459,881 | 48 | 1.8 | 48 | 0.85 |
| cont11_l | 1,468,599 | 1,961,394 | 5,382,999 | 5 | 1.6 | 8 | 0.52 |
| cage13 | 445,315 | 445,315 | 7,479,343 | 39 | 8.4 | 118 | 2.27 |
| cage14 | 1,505,785 | 1,505,785 | 27,130,349 | 41 | 9.0 | 136 | 8.86 |

| Name | Speedup | | | Normalized nr. of colors | | |
|---|---|---|---|---|---|---|
| | $p = 16$ | $p = 32$ | $p = 64$ | $p = 16$ | $p = 32$ | $p = 64$ |
| lhr71c | 11.3 | 13.4 | 6.9 | 1.02 | 1.02 | 1.02 |
| stormG2 | 7.4 | 8.6 | 5.3 | 1.00 | 1.00 | 1.00 |
| cont11_l | 4.8 | 7.2 | 6.3 | 1.50 | 1.50 | 1.62 |
| cage13 | 4.6 | 4.9 | 3.7 | 0.97 | 0.96 | 0.96 |
| cage14 | 3.4 | 5.2 | 3.4 | 0.95 | 0.99 | 0.98 |

2.4 GHz Quad-Core AMD Opteron CPUs and 32 GB of memory and is interconnected via an Infiniband network. The results in Table 6.2 show that the parallel star coloring algorithm scales quite well (near linear speedup was observed up to 32 processors, while the speedup thereafter was quite moderate due to the relatively small size of the graphs). The results further show that the algorithm uses a number of colors close to the numbers used by the sequential variant and exploits available symmetry at no or only modest increase in runtime compared to distance-2 coloring.

**6.7. Results on parallel partial distance-2 coloring of bipartite graphs.** We evaluated the performance of the parallel algorithm for partial distance-2 coloring of bipartite graphs using five nonsymmetric matrices obtained from the University of Florida Sparse Matrix Collection [11]. The structural properties of these graphs as well as the sequential coloring results are given in the top part of Table 6.3. The parallel partial distance-2 coloring results obtained using them are presented in the bottom part of Table 6.3. In each test case except for *cont*11_*l*, the number of colors used by the parallel algorithm was observed to be within 2% of that used by the sequential algorithm. For *cont*11_*l*, the number of colors increased from 8 up to 13 when using the parallel algorithm. In general, the speedup observed in the experiments on partial distance-2 coloring of bipartite graphs is poorer than those observed in the distance-2 coloring of general graphs. A partial reason for this lies in the graph partitioner used in the experiments. Specifically, in both the distance-2 coloring of general graphs and the partial distance-2 coloring of bipartite graphs, the same partitioner (MeTiS) was used, whereas MeTiS is designed for general graphs and does not exploit the properties of bipartite graphs. In particular, the partitioner tries only to achieve balance in sizes (weights) of parts by considering total number (or weight) of vertices in each part and ignores the fact that there are two groups (types) of vertices. This may lead to an imbalance among the same group of vertices in different parts, causing the poorer performance.

**6.8. Results on parallel distance-1 coloring of $\mathcal{G}^2$.** As discussed in section 2, the distance-2 coloring problem on a graph $\mathcal{G}$ can be solved by constructing and then distance-1 coloring the square graph $\mathcal{G}^2$. This alternative method has the same overall




(a) (b) (c)

FIG. 6.8. (a) *and* (b) *Speedup for distance*-1 *coloring of* $\mathcal{G}^2$ *on the application test graphs and the synthetic test graphs listed in Table* 6.1*, respectively.* (c) *Weak scaling for distance*-1 *coloring of* $\mathcal{G}^2$ *on the synthetic graphs given in Figure* 6.5.

asymptotic time complexity as a greedy distance-2 coloring on the graph $\mathcal{G}$, but the actual runtimes of the two approaches can differ substantially.

We have done experiments on this alternative approach using the graphs listed in Table 6.1. For these graphs, the ratio of the number of edges in $\mathcal{G}^2$ to that in $\mathcal{G}$ is listed in the last column of Table 6.1. As one can see from the table, the storage requirement for $\mathcal{G}^2$ could be up to 7 times larger than what is needed for $\mathcal{G}$ for the application graphs, and up to 59 times larger for the synthetic graphs.

To construct $\mathcal{G}^2$ in parallel, each processor requires the adjacency lists of distance-1 neighbors of its boundary vertices. This requires communication between processors similar to the forbidden color communication in a single round of the proposed parallel distance-2 coloring algorithm. However, instead of a union of colors, adjacency lists are exchanged. Thus the communication cost is higher.

Figure 6.8 shows strong and weak scaling results of the approach based on distance-1 coloring of $\mathcal{G}^2$. The timing for this method includes parallel construction of $\mathcal{G}^2$ and parallel distance-1 coloring of $\mathcal{G}^2$ using the algorithm presented in [6]. While computing speedup, the sequential greedy distance-2 coloring algorithm is used as the reference. Therefore, the results in Figures 6.8(a) and 6.8(b) are directly comparable to those in Figures 6.2(c) and 6.3(c), respectively. The results show that the alternative approach discussed in this section is slower and scales worse than the proposed parallel distance-2 coloring algorithm.

**7. Conclusion.** We have presented efficient distributed-memory parallel algorithms for three closely related coloring problems: distance-2 coloring of general graphs, restricted star coloring of general graphs, and partial distance-2 coloring of bipartite graphs. These problems arise in the efficient computation of sparse Jacobian and Hessian matrices using automatic differentiation or finite differencing. The scalability of the proposed algorithms has been demonstrated on a variety of applications as well as synthetic test graphs. MPI implementations of the algorithms have been made available to the public through Zoltan, a software library for parallel partitioning, load balancing, and data-management services [31].

As the experimental results in section 6.4 showed, the number of rounds required by the parallel distance-2 coloring algorithm in coloring graphs in which a small fraction of the vertices are of relatively very large degree (e.g., small-world graphs) could be relatively high (in the order of a hundred). A separate treatment of such vertices could improve the performance of the proposed algorithm. We plan to explore this in a future work.

## Appendix.

Table A.1

*Results on the parallel distance-2 coloring and restricted star coloring algorithms for the application graphs listed in Table 6.1 using the superstep size $s = 100$. In columns 6 and 10, the number of colors used by the respective sequential greedy coloring algorithm is given in parentheses. Abbreviations: normalized (norm), number (nr), conflicts (cflicts), rounds (rnds), speedup (spdup), number of processors (p).*

| | | Distance-2 coloring | | | | Restricted star coloring | | | |
|---|---|---|---|---|---|---|---|---|---|
| $p$ | Graph name | Norm. cflicts | Nr. of rnds | Spdup | Nr. of colors | Norm. cflicts | Nr. of rnds | Spdup | Nr. of colors |
| | nasasrb | 1.21% | 5 | 10.1 | 276 (276) | 1.98% | 4 | 9.8 | 276 (100) |
| | ct20stif | 1.59% | 6 | 8.8 | 210 (210) | 1.32% | 4 | 9.3 | 177 (210) |
| | pwtk | 0.09% | 3 | 13.2 | 180 (180) | 0.66% | 4 | 11.3 | 180 (180) |
| | shipsec8 | 0.87% | 5 | 10.2 | 158 (150) | 0.95% | 4 | 10.8 | 158 (144) |
| | shipsec1 | 0.85% | 4 | 11.4 | 141 (126) | 0.74% | 4 | 11.0 | 126 (126) |
| | shipsec5 | 0.35% | 5 | 11.6 | 156 (140) | 0.56% | 4 | 11.2 | 141 (141) |
| | bmw7st_1 | 0.85% | 12 | 11.5 | 435 (435) | 0.66% | 3 | 11.8 | 432 (408) |
| 16 | bmw3_2 | 0.54% | 4 | 11.2 | 336 (336) | 0.64% | 4 | 12.0 | 336 (204) |
| | inline_1 | 0.36% | 7 | 10.6 | 843 (843) | 0.26% | 4 | 9.8 | 843 (843) |
| | hood | 0.69% | 4 | 13.4 | 108 (103) | 0.67% | 4 | 11.6 | 105 (79) |
| | msdoor | 0.31% | 4 | 14.0 | 112 (105) | 0.29% | 4 | 12.5 | 105 (98) |
| | ldoor | 0.14% | 4 | 14.7 | 119 (112) | 0.14% | 4 | 12.7 | 112 (91) |
| | pkustk10 | 1.28% | 5 | 11.0 | 127 (126) | 0.81% | 4 | 10.1 | 120 (114) |
| | pkustk11 | 1.49% | 5 | 9.4 | 219 (198) | 0.75% | 4 | 8.5 | 183 (174) |
| | pkustk13 | 1.37% | 4 | 9.1 | 302 (303) | 0.95% | 4 | 9.0 | 303 (279) |
| | nasasrb | 6.00% | 24 | 8.9 | 276 (276) | 4.87% | 10 | 13.8 | 189 (100) |
| | ct20stif | 6.21% | 9 | 12.9 | 210 (210) | 4.42% | 5 | 15.7 | 146 (210) |
| | pwtk | 0.66% | 5 | 28.7 | 180 (180) | 0.75% | 4 | 27.7 | 180 (180) |
| | shipsec8 | 2.14% | 4 | 21.4 | 161 (150) | 1.76% | 4 | 20.9 | 140 (144) |
| | shipsec1 | 2.09% | 5 | 22.2 | 134 (126) | 1.79% | 4 | 23.2 | 126 (126) |
| | shipsec5 | 1.06% | 6 | 22.4 | 150 (140) | 1.06% | 4 | 23.1 | 144 (141) |
| | bmw7st_1 | 2.44% | 12 | 20.2 | 435 (435) | 1.95% | 4 | 23.7 | 435 (408) |
| 48 | bmw3_2 | 1.51% | 12 | 24.3 | 336 (336) | 1.17% | 4 | 27.3 | 336 (204) |
| | inline_1 | 1.31% | 14 | 22.2 | 843 (843) | 0.63% | 4 | 24.2 | 843 (843) |
| | hood | 1.67% | 5 | 30.8 | 113 (103) | 1.73% | 5 | 27.8 | 112 (79) |
| | msdoor | 0.85% | 5 | 29.0 | 116 (105) | 0.89% | 4 | 28.9 | 108 (98) |
| | ldoor | 0.45% | 4 | 36.5 | 117 (112) | 0.46% | 4 | 33.5 | 111 (91) |
| | pkustk10 | 3.78% | 5 | 17.4 | 133 (126) | 3.02% | 5 | 21.1 | 123 (114) |
| | pkustk11 | 3.84% | 5 | 16.8 | 209 (198) | 3.02% | 4 | 18.4 | 182 (174) |
| | pkustk13 | 4.53% | 6 | 16.1 | 306 (303) | 2.19% | 5 | 17.8 | 282 (279) |
| | nasasrb | 10.17% | 23 | 6.7 | 276 (276) | 7.90% | 9 | 12.2 | 131 (100) |
| | ct20stif | 14.00% | 15 | 9.2 | 210 (210) | 11.26% | 6 | 15.8 | 156 (210) |
| | pwtk | 2.02% | 4 | 43.6 | 180 (180) | 2.19% | 5 | 38.4 | 180 (180) |
| | shipsec8 | 4.43% | 5 | 28.4 | 166 (150) | 4.53% | 5 | 28.1 | 138 (144) |
| | shipsec1 | 3.60% | 6 | 29.6 | 141 (126) | 2.64% | 5 | 31.7 | 133 (126) |
| | shipsec5 | 2.33% | 4 | 34.5 | 154 (140) | 2.40% | 4 | 35.0 | 132 (141) |
| | bmw7st_1 | 5.59% | 18 | 16.5 | 435 (435) | 4.08% | 5 | 28.8 | 399 (408) |
| 96 | bmw3_2 | 3.10% | 11 | 25.8 | 350 (336) | 2.44% | 6 | 33.0 | 330 (204) |
| | inline_1 | 2.22% | 15 | 35.3 | 843 (843) | 1.46% | 5 | 34.1 | 789 (843) |
| | hood | 2.65% | 6 | 39.4 | 115 (103) | 2.84% | 4 | 39.4 | 105 (79) |
| | msdoor | 1.65% | 6 | 46.1 | 118 (105) | 1.65% | 4 | 46.2 | 130 (98) |
| | ldoor | 0.78% | 4 | 60.6 | 121 (112) | 0.79% | 4 | 58.6 | 111 (91) |
| | pkustk10 | 8.32% | 8 | 22.2 | 130 (126) | 6.19% | 5 | 24.9 | 118 (114) |
| | pkustk11 | 8.55% | 6 | 20.7 | 198 (198) | 6.44% | 5 | 24.4 | 189 (174) |
| | pkustk13 | 10.18% | 8 | 21.2 | 314 (303) | 4.90% | 6 | 24.6 | 275 (279) |

**Acknowledgment.** We thank the anonymous referees for their valuable comments, which helped us improve the presentation of this paper.

REFERENCES

[1] G. Agnarsson, R. Greenlaw, and M. M. Halldórsson, *On powers of chordal graphs and their colorings*, Congr. Numer., 100 (2000), pp. 41–65.
[2] G. Agnarsson and M. M. Halldórsson, *Coloring powers of planar graphs*, SIAM J. Discrete Math., 16 (2003), pp. 651–662.
[3] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster, *A fully asynchronous multifrontal solver using distributed dynamic scheduling*, SIAM J. Matrix Anal. Appl., 23 (2001), pp. 15–41.
[4] R. H. Bisseling, *Parallel Scientific Computation: A Structured Approach Using BSP and MPI*, Oxford University Press, London, 2004.
[5] D. Bozdağ, U. Catalyurek, A. H. Gebremedhin, F. Manne, E. G. Boman, and F. Özgüner, *A parallel distance-2 graph coloring algorithm for distributed memory computers*, in Proceedings of HPCC 2005, Lecture Notes in Comput. Sci. 3726, Springer, New York, 2005, pp. 796–806.
[6] D. Bozdağ, A. H. Gebremedhin, F. Manne, E. G. Boman, and U. V. Catalyurek, *A framework for scalable greedy coloring on distributed-memory parallel computers*, J. Parallel Distrib. Comput., 68 (2008), pp. 515–535.
[7] U. V. Catalyurek and C. Aykanat, *Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication*, IEEE Trans. Parallel Distrib. System, 10 (1999), pp. 673–693.
[8] T. F. Coleman and J. J. Moré, *Estimation of sparse Jacobian matrices and graph coloring problems*, SIAM J. Numer. Anal., 20 (1983), pp. 187–209.
[9] T. F. Coleman and J. J. Moré, *Estimation of sparse Hessian matrices and graph coloring problems*, Math. Program., 28 (1984), pp. 243–270.
[10] A. R. Curtis, M. J. D. Powell, and J. K. Reid, *On the estimation of sparse Jacobian matrices*, J. Inst. Math. Appl., 13 (1974), pp. 117–119.
[11] T. Davis and Y. Hu, maintainers, *University of Florida Sparse Matrix Collection*, available at http://www.cise.ufl.edu/research/sparse/matrices/.
[12] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, and C. Vaughan, *Zoltan data management services for parallel dynamic applications*, Comput. Sci. Engrg., 4 (2002), pp. 90–97.
[13] A. H. Gebremedhin, A. Pothen, A. Tarafdar, and A. Walther, *Efficient computation of sparse Hessians using coloring and automatic differentiation*, INFORMS J. Comput., 21 (2009), pp. 209–223.
[14] A. H. Gebremedhin and F. Manne, *Scalable parallel graph coloring algorithms*, Concurrency: Practice and Experience, 12 (2000), pp. 1131–1146.
[15] A. H. Gebremedhin, F. Manne, and A. Pothen, *Parallel distance-k coloring algorithms for numerical optimization*, in Proceedings of Euro-Par 2002, Lecture Notes in Comput. Sci. 2400, Springer, New York, 2002, pp. 912–921.
[16] A. H. Gebremedhin, F. Manne, and A. Pothen, *What color is your Jacobian? Graph coloring for computing derivatives*, SIAM Rev., 47 (2005), pp. 629–705.
[17] A. H. Gebremedhin, A. Tarafdar, F. Manne, and A. Pothen, *New acyclic and star coloring algorithms with application to computing Hessians*, SIAM J. Sci. Comput., 29 (2007), pp. 1042–1072.
[18] D. Hysom and A. Pothen, *A scalable parallel algorithm for incomplete factor preconditioning*, SIAM J. Sci. Comput., 22 (2001), pp. 2194–2215.
[19] M. T. Jones and P. E. Plassmann, *Scalable iterative solution of sparse linear systems*, Parallel Comput., 20 (1994), pp. 753–773.
[20] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM J. Sci. Comput., 20 (1998), pp. 359–392.
[21] S. O. Krumke, M. V. Marathe, and S. S. Ravi, *Models and approximation algorithms for channel assignment in radio networks*, Wireless Networks, 7 (2001), pp. 575–584.
[22] V. S. A. Kumar, M. V. Marathe, S. Parthasarathy, and A. Srinivasan, *End-to-end packet-scheduling in wireless ad-hoc networks*, in Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, LA, 2004, SIAM, Philadelphia, pp. 1014–1023.
[23] S. T. McCormick, *Optimal approximation of sparse Hessians and its equivalence to a graph coloring problem*, Math. Program., 26 (1983), pp. 153–171.

[24] C. A. Morgenstern and H. D. Shapiro, *Heuristics for rapidly four-coloring large planar graphs*, Algorithmica, 6 (1991), pp. 869–891.
[25] M. E. J. Newman, *The structure and function of complex networks*, SIAM Rev., 45 (2003), pp. 167–256.
[26] M. J. D. Powell and Ph. L. Toint, *On the estimation of sparse Hessian matrices*, SIAM J. Numer. Anal., 16 (1979), pp. 1060–1074.
[27] Y. Saad, *ILUM: A multi-elimination ILU preconditioner for general sparse matrices*, SIAM J. Sci. Comput., 17 (1996), pp. 830–847.
[28] M. M. Strout and P. D. Hovland, *Metrics and models for reordering transformations*, in Proceedings of the Second ACM SIGPLAN Workshop on Memory System Performance, 2004, ACM, New York, pp. 23–34.
[29] *The Second DIMACS Implementation Challenge: Maximum Clique, Graph Coloring, and Satisfiability*, http://dimacs.rutgers.edu/Challenges.
[30] Georgia Tech College of Computing, *GTgraph: A suite of synthetic graph generators*, available at http://www-static.cc.gatech.edu/~kamesh/GTgraph/.
[31] The Zoltan Team, *Zoltan: Parallel partitioning, load balancing and data-management services*, software package available online at http://www.cs.sandia.gov/Zoltan/.