Parallel Distance-k Coloring Algorithms for Numerical Optimization

Assefaw Hadish Gebremedhin* Fredrik Manne Alex Pothen†

Abstract

Matrix partitioning problems that arise in the efficient estimation of sparse Jacobians and Hessians can be modeled using variants of graph coloring problems. In a previous work [7], we argue that *distance-2* and *distance-*$\frac{3}{2}$ graph coloring are robust and flexible formulations of the respective matrix estimation problems. The problem size in large-scale optimization contexts makes the matrix estimation phase an expensive part of the entire computation both in terms of execution time and memory space. Hence, there is a need for both shared- and distributed-memory parallel algorithms for the stated graph coloring problems. In the current work, we present the first practical shared address space parallel algorithms for these problems. The main idea in our algorithms is to randomly *partition* the vertex set equally among the available processors, let each processor *speculatively* color its vertices using information about already colored vertices, detect eventual conflicts in parallel, and finally re-color conflicting vertices sequentially. *Randomization* is also used in the coloring phases to further reduce conflicts. Our PRAM-analysis shows that the algorithms should give almost linear speedup for sparse graphs that are large relative to the number of processors. Experimental results from our OpenMP implementations on a Cray Origin2000 using various large graphs show that the algorithms indeed yield reasonable speedup for modest numbers of processors.

*Department of Informatics, University of Bergen, N-5020 Bergen, Norway. {assefaw, fredrikm}@ii.uib.no

†Computer Science Department, Old Dominion University, Norfolk, VA 23529 USA; CSRI, Sandia National Labs, Albuquerque NM 87185 USA; and ICASE, NASA Langley Research Center, Hampton, VA 23681-2199 USA. pothen@cs.odu.edu

‡This author's research was supported by NSF grant DMS-9807172, DOE ASCI level-2 subcontract B347882 from Lawrence Livermore National Lab; and by DOE SCIDAC grant DE-FC02-01ER25476.

1 Introduction

Numerical optimization algorithms that rely on derivative information often need to compute the Jacobian or Hessian matrix. Since this is an expensive part of the computation, efficient methods for estimating these matrices via finite differences (FD) or automatic differentiation (AD) are needed. It is known that the problem of minimizing the number of function evaluations (or AD passes) required in the computation of these matrices can be formulated as variants of graph coloring problems [1, 2, 3, 9, 13]. The particular coloring problem differs with the optimization context: whether the Jacobian or the Hessian matrix is to be computed; whether a direct or a substitution method is employed; and whether only columns, or only rows, or both columns and rows are to be used to evaluate the matrix elements. In addition, the type of coloring problem depends on the kind of graph used to represent the underlying matrix. In [7], we provide an integrated review of previous works in this area and identify the *distance-2* (D2) graph coloring problem as a unifying, generic, and robust formulation. The D2-coloring problem has also noteworthy applications in other fields such as channel assignment [11] and facility location problems [14].

Large-scale PDE-constrained optimization problems can be solved only with the memory and time resources available on parallel computers. In these problems, the variables defined on a computational mesh are already distributed on the processors, and hence parallel coloring algorithms are needed for computing, for instance, the Jacobian. It turns out that the problems of efficiently computing the Jacobian and Hessian can be formulated as the D2- and D$\frac{3}{2}$-coloring problems, respectively. The latter coloring problem is a relaxed variant of the former, and will be described in Section 2.2. In these formulations, the bipartite graph associated with the rows and columns of the matrix is used for the Jacobian; in the case of the Hessian matrix, the adjacency graph corresponding to the symmetric matrix is used.

In this paper, we present several new deterministic as well as probabilistic parallel algorithms for the D2- and D$\frac{3}{2}$-coloring problems. Our algorithms are practical and effective, well suited for shared address space programming, and have been implemented in C using OpenMP primitives. We report results from experiments conducted on a Cray Origin 2000 using large graphs that arise in finite element methods and in eigenvalue computations.

In the sequel, we introduce the graph problems in Section 2, present the algorithms in Section 3, discuss our experimental results in Section 4, and conclude the paper in Section 5.

2 Background

2.1 Matrix Partition Problems

An essential component of the efficient estimation of a sparse Jacobian or Hessian using FD or AD is the problem of finding a suitable *partition* of the columns and/or the rows of the matrix. The particular partition chosen defines a system of equations from which the matrix entries are determined. A method that utilizes a diagonal system is called a *direct* method, and one that uses a triangular system is called a *substitution* method. A direct method is more restrictive but the computation of matrix entries is straightforward and numerically stable. A substitution method, on the other hand, is less restrictive but it may be subject to approximation difficulties and numerical instability. Moreover, direct methods offer more parallelism than substitution methods. In this paper, we focus on direct methods that use column partitioning.

A partition of the columns of a nonsymmetric matrix A is said to be *consistent* with the direct determination of A if whenever a_{ij} is a non-zero element of A then the group containing column j has no other column with a non-zero in row i [1]. Similarly, a partition of the columns of a symmetric matrix A is called *symmetrically consistent* with the direct determination of A if whenever a_{ij} is a non-zero element of A then *either* (i) the group containing column j has no other column with a non-zero in row i, or (ii) the group containing column i has no other column with a non-zero in row j [2]. From a given (symmetrically) consistent partition $\{C_1, C_2, \ldots, C_\rho\}$ of the columns of A, the nonzero entries can be determined with ρ function evaluations (matrix-vector products).

Thus we have the following two problems of interest. Given the sparsity structure of a nonsymmetric $m \times n$ matrix A, find a consistent partition of the columns of A with the fewest number of groups. We refer to this problem as NONSYMCOLPART. The second problem of our interest, which we call SYMCOLPART, states: given the sparsity structure of a symmetric $n \times n$ matrix A, find a symmetrically consistent partition of the columns of A with the fewest number of groups.

2.2 Graph Problems

In a graph, two distinct vertices are said to be *distance-k neighbors* if the shortest path connecting them consists of *at most* k edges. The number of distance-k neighbors of a vertex u is referred to as the *degree-k* of u and is denoted by $d_k(u)$. A *distance-k ρ-coloring* (or (k, ρ)-coloring for

short) of a graph $G = (V, E)$ is a mapping $\phi : V \rightarrow \{1, 2, \ldots, \rho\}$ such that $\phi(u) \neq \phi(v)$ whenever u and v are distance-k neighbors. We call a mapping $\phi : V \rightarrow \{1, 2, \ldots, \rho\}$ a $(\frac{3}{2}, \rho)$-coloring of a graph $G = (V, E)$ if ϕ is a $(1, \rho)$-coloring of G and every path containing three edges uses at least three colors. Notice that a $(\frac{3}{2}, \rho)$-coloring is a restricted $(1, \rho)$-coloring, and a relaxed $(2, \rho)$-coloring, and hence the name. For instance, consider a path u, v, w, x in a graph. The assignment $2, 1, 2, 3$ to the respective vertices is a valid D1- and D$\frac{3}{2}$-coloring, but not a valid D2-coloring. The *distance-k graph coloring problem* asks for a (k, ρ)-coloring of a graph with the least possible value of ρ.

Let A be an $m \times n$ rectangular matrix with rows $r_1, r_2, \ldots, r_m$ and columns $a_1, a_2, \ldots, a_n$. We define the *bipartite graph* of A as $G_b(A) = (V_1, V_2, E)$ where $V_1 = \{r_1, r_2, \ldots, r_m\}$, $V_2 = \{a_1, a_2, \ldots, a_n\}$, and $(r_i, a_j) \in E$ whenever $a_{ij} \neq 0$, for $1 \leq i \leq m$, $1 \leq j \leq n$. When A is an $n \times n$ symmetric matrix with non-zero diagonal elements, the *adjacency graph* of A is defined to be $G_a(A) = (V, E)$ where $V = \{a_1, a_2, \ldots, a_n\}$ and $(a_i, a_j) \in E$ whenever a_{ij}, $i \neq j$, is a non-zero element of A. Note that the non-zero diagonal elements of A are not explicitly represented by edges in $G_a(A)$. In our work, we rely on the bipartite and adjacency graph representations. However, in the literature, a nonsymmetric matrix A is often represented by its *column intersection graph* $G_c(A)$. In this representation, the columns of A constitute the vertex set, and an edge (a_i, a_j) exists whenever the columns a_i and a_j have non-zero entries at the same row position (i.e., a_i and a_j are not structurally orthogonal). As argued in [7], the bipartite graph representation is more flexible and robust than the 'compressed' column intersection graph.

Coleman and Moré [1] showed that problem NONSYMCOLPART is equivalent to the D1-coloring problem when the matrix is represented by its column intersection graph. We have shown [7] that the same problem is equivalent to a partial D2-coloring when a bipartite graph is used and discussed the relative merits of the two approaches. The word 'partial' reflects the fact that only the vertices corresponding to the columns need to be colored.

McCormick [13] showed that the approximation of a Hessian using a direct method is equivalent to a D2-coloring on the adjacency graph of the matrix. One drawback of McCormick's formulation is that it does not exploit symmetry. Later Coleman and Moré [2] addressed this issue and showed that the resulting problem (SYMCOLPART) is equivalent to the D$\frac{3}{2}$-coloring problem.

3 Parallel Coloring Algorithms

The distance-k coloring problem is NP-hard for any fixed integer $k \geq 1$ [12]. A proof-sketch showing that D$\frac{3}{2}$-coloring is NP-hard is given in [2]. Furthermore, Lexicographically First $\Delta + 1$ Coloring (LFC), the polynomial variant of D1-coloring in which the vertices are given in a predetermined order and the question at each step is to assign the vertex the smallest color not used by any of its neighbors, is P-complete [8]. The practical implication of this is that designing efficient fine-grained parallel algorithm for LFC is hard.

In practice, greedy sequential D1-coloring heuristics are found to be quite effective [1]. In a recent work [6], we have shown effective methods of parallelizing such greedy algorithms in a coarse-grained setting. Here, we extend this work to develop parallel algorithms for the D2- and D$\frac{3}{2}$-coloring problems.

Jones and Plassmann [10] describe a parallel distributed memory D1-coloring algorithm that uses randomization to assign priorities to the vertices, and then colors the vertices in the order determined by the priorities. There is no speculative coloring in their algorithm. It was reported that the algorithm slows down as the number of processors is increased. Finocchi et al. [5] suggest a parallel D1-coloring algorithm organized in several rounds; in each round, currently uncolored vertices are assigned a tentative pseudo-color without consulting their neighbors mapped to other processors; in a conflict resolution step, a maximal independent set of vertices in each color class is assigned these colors as final; the remainder of the vertices are uncolored, and the algorithm moves into the next round. However, they do not give any implementation and we believe that this algorithm incurs too many rounds, each with its synchronization and communication steps, for it to be practical on large graphs.

Our algorithm (in its generic form) may be viewed as a compromise between these algorithms, where we permit speculative coloring, but limit the number of synchronization steps to two in the whole algorithm. However, our current algorithms rely on the shared address space programming model; we will adapt our algorithms to distributed memory programming models in future work. We are unaware of any previous work on parallel algorithms for D2- and D$\frac{3}{2}$-coloring problems.

3.1 Generic Greedy Parallel Coloring Algorithm (GGPCA)

The steps of our generic parallel coloring algorithm can be summarized as shown below; refer to [6] for a detailed discussion of the D1-coloring case. Let $G = (V, E)$ be the input graph and p be the number of processors.

Phase 0: *Partition*
Randomly partition V into p equal blocks $V_1 \ldots V_p$. Processor P_i is responsible for coloring the vertices in block V_i.
Phase 1: *Pseudo-color*
for $i = 1$ **to** p **do in parallel**
for each $u \in V_i$ **do**
assign the smallest available color to u, paying attention to already colored vertices (both local and non-local).
—*barrier synchronize*—
Phase 2: *Detect conflicts*
for $i = 1$ **to** p **do in parallel**
for each $u \in V_i$ **do**
check whether the color of u is valid. If the colors of u and v are the same for some 'neighbor' v of u then
$L_i = L_i \cup min\{u, v\}$
—*barrier synchronize*—
Phase 3: *Resolve conflicts*
Color the vertices in the conflict list $L = \cup L_i$ sequentially.

In Phase 0, the vertices are randomly partitioned into p equal blocks each of which is assigned to some processor. In Phase 1, the processors color the vertices in their respective blocks in parallel. When two 'neighbor' vertices reside on different processors, the two processors could color both simultaneously, possibly assign them the same value, and cause a conflict. The purpose of Phase 2 is to detect and store any such conflict vertices which are subsequently re-colored sequentially in Phase 3.

3.2 Simple Distance-2 (SD2) Coloring Algorithm

The meaning of 'available' color in GGPCA depends on the required coloring. In the case of a D2-coloring, a vertex is assigned the smallest color not used by any of its distance-2 neighbors. Let Δ denote the maximum degree-1 in the graph $G = (V, E)$. It can easily be verified that, in a D2-coloring, a vertex can always be assigned one of the colors from the set $\{1, 2, \ldots, \Delta^2 + 1\}$. Moreover, since the distance-1 neighbors of a vertex are

distance-2 neighbors with each other, the 2-chromatic number, i.e., the least number of colors required in a D2-coloring, is at least $\Delta+1$. Thus, the greedy approach is an $O(\Delta)$-approximation algorithm. We refer to the variant of GGPCA that applies to D2-coloring as Algorithm SD2.

Note that the sequential time complexity of greedy D2-coloring is $O(\Delta^2|V|)$. The following results show that the number of conflicts discovered in Phase 2 of Algorithm SD2 is often small for sparse graphs, making the algorithm scalable when the number of processors $p = O(\sqrt{\frac{|V|^2}{\Delta|E|}})$. The proofs are straightforward extensions of our proofs for the D1-coloring case given in [6] and hence are omitted here. One only needs to observe that the degree-2 and degree-1 of a vertex u are related by $d_2(u) \leq \Delta d_1(u)$. Let $\overline{\delta} = 2|E|/|V|$ denote the average degree-1 in G.

Lemma 1 *The expected number of conflicts created at the end of Phase 1 of Algorithm SD2 is at most* $\approx \frac{\Delta\overline{\delta}(p-1)}{2}$.

Theorem 2 *On a CREW PRAM, Algorithm SD2 distance-2 colors the input graph consistently in expected time* $O(\Delta^2(\frac{|V|}{p} + \Delta\overline{\delta}p))$ *using at most* $\Delta^2 + 1$ *colors.*

Corollary 3 *When* $p = O(\sqrt{\frac{|V|^2}{\Delta|E|}})$, *the expected runtime of Algorithm SD2 is* $O(\frac{\Delta^2|V|}{p})$.

The number of conflicts predicted by Lemma 1 is an overestimate. The analysis assumes that whenever two distance-2 neighbor vertices are colored simultaneously, they are assigned the same color, thereby resulting in a conflict. However, a more involved probabilistic analysis that takes the distribution of colors used into account may provide a tighter bound. Besides, the actual number of conflicts in an implementation could be significantly reduced by choosing a random color from the allowable set, instead of the smallest one as given in Phase 1 of GGPCA.

3.3 Improved Distance-2 (ID2) Coloring Algorithm

The number of colors used in Algorithm SD2 can be reduced using a 'two-round-coloring' strategy. The underlying idea in the D1-coloring case is due to Culberson [4] and was used in our parallel algorithms for D1-coloring [6]. In Lemma 4 we extend the result to the D2-coloring case; the proof is basically a reproduction of Culberson's proof for the distance-1 coloring case. The greedy sequential algorithm referred to in the lemma is one that

visits the vertices of a graph $G = (V, E)$ in some order (a permutation of $\{1, 2, \ldots, |V|\}$), each time assigning a vertex the *smallest* allowed color. In particular, the first vertex to be visited is assigned color 1.

Lemma 4 *Let ϕ be a distance-2 coloring of a graph G using α colors, and π a permutation of the vertices such that if $\phi(v_{\pi(i)}) = \phi(v_{\pi(l)}) = c$, then $\phi(v_{\pi(j)}) = c$ for $i < j < l$. Applying the greedy sequential distance-2 coloring algorithm to G where the vertices have been ordered by π will produce a coloring ϕ' using α or fewer colors.*

Proof: The proof is a simple induction showing that the first i color classes[1] listed in the permutation will be colored with i or fewer colors. Clearly, the first color class listed will be colored with color 1. Suppose some element of the ith class requires color $i+1$. This means that it must have a distance-2 neighbor of color i. But by induction the vertices in the 1st to the $(i-1)$th classes used no more than $i-1$ colors. Thus, the conflict has to be with a member of its own color class, but this contradicts the assumption that ϕ is a valid distance-2 coloring. □

The idea in Lemma 4 is that if the greedy coloring algorithm is re-applied on a graph, with the vertices belonging to the same color class (in the original coloring) listed consecutively, then the new coloring obtained is better or at least as good as the original. One ordering (among many) that satisfies this condition, with a good potential for reducing the number of colors used, is to list the vertices consecutively for each color class in the reverse order of the introduction of the color classes. Based on Lemma 4, we modify Algorithm SD2 and introduce an additional parallel coloring phase between Phases 1 and 2. Algorithm ID2 below outlines the resulting 4-phase algorithm.

Phases 0 and 1. Same as Ph. 0 and 1 of GGPCA.
(Let s be the number of colors.)
Phase 2. **for** $k = s$ **downto** 1 **do**
Partition ColorClass(k) into p equal blocks $V'_1, \ldots, V'_p$
for $i = 1$ **to** p **do in parallel**
for each $u \in V'_i$ **do**
assign the smallest available color to vertex u.
—*barrier synchronize*—
Phases 3 and 4. Same as Phases 2 and 3 of GGPCA, respectively.

[1] Vertices of the same color constitute a color class.

In Phase 2, most of the vertices in a color class are at a distance greater than two edges from each other, and the exceptions arise from the conflict vertices colored incorrectly in Phase 1. Since the number of such conflict vertices from Phase 1 is low, the number of conflict vertices at the end of the re-coloring phase will be even lower. Phases 3 and 4 are included to detect and resolve any eventual conflicts not resolved in Phase 2.

3.4 Simple Distance-$\frac{3}{2}$ (SD$\frac{3}{2}$) Coloring Algorithm

Recall that a D$\frac{3}{2}$-coloring is a *relaxed* D2-coloring (see the example in Section 2.2). One way of relaxing the requirement for D2-coloring in GGPCA so as to obtain a valid D$\frac{3}{2}$-coloring is to let two vertices at a distance of *exactly* two edges from each other share a color as long as the vertex in between them is already colored with a (color of) lower value. We refer to the variant of GGPCA that employs this technique to achieve a distance-3/2 coloring as Algorithm SD$\frac{3}{2}$.

Note that both Algorithms ID2 and SD$\frac{3}{2}$ take asymptotically the same time as Algorithm SD2.

3.5 Randomization

The potential scalability of GGPCA depends on the number of conflicts discovered in Phase 2, since these are resolved sequentially in Phase 3. For dense graphs the number of conflicts could be large enough to destroy the scalability of the algorithm. To overcome this problem, we use randomization as a means for reducing the number of conflicts. In the randomized variants of Algorithms SD2, SD$\frac{3}{2}$, and ID2, a vertex is assigned the next available color with probability q, where $0 < q \leq 1$. The first attempt is made with the smallest available color, i.e., the color is chosen with probability q. An attempt is said to be successful if the vertex is assigned a color. If an attempt is not successful, then the next available color is tried with probability q, and so on, until the vertex gets a color. Algorithms SD2, SD$\frac{3}{2}$, and ID2, can be seen as the deterministic variants where $q = 1$. We refer to the randomized versions of the respective algorithms as RSD2, RSD$\frac{3}{2}$, and RID2.

Let u and v be two vertices with the same (infinite) set of allowable colors. If u and v are colored concurrently, it can be shown that the probability that u and v get the same color is $q/(2-q)$. This shows how randomization leads to a reduction in the number of conflicts; the lower the value of q, the lower chance for a conflict to arise. It should however be noted that a 'low' value

Set	Problem	$\|V\|$	$\|E\|$	Δ	δ	$\overline{\delta}$
Set I (FE)	mrng2	1,017,253	2,015,714	4	2	4
	fe144	144,649	1,074,393	26	4	15
	m14b	214,765	1,679,018	40	4	16
Set II (EV)	ev01	10,134	1,318,579	634	93	260
	ev02	19,845	3,353,890	749	78	338

Table 1: Test Graphs: the last three columns list the max., min., and average degree-1, respectively.

of q may result in an increase in the number of colors used. Choosing the right value for q thus becomes a design issue.

4 Experimental Results and Discussion

Our test bed consists of graphs that arise from finite element methods and from eigenvalue computations [6]. Table 1 provides the test graphs' structural information: columns $|V|$ and $|E|$ give the number of vertices and edges of each graph, respectively, and the maximum, minimum, and average degree-1 of each graph is given under columns Δ, δ, and $\overline{\delta}$, respectively.

Table 2 through 7 provide coloring and timing information of the different algorithms. The number of blocks (processors) is given in column p. Column χ_i gives the number of colors used at the end of Phase i of the corresponding algorithm. The number of conflicts that arise in Phase 1 (or 2) are listed under the column labeled K (or K_2). The time in milliseconds required by the different phases are listed under T_1, T_2, T_3, T_4; column T_{tot} gives the total time used. The last two columns display speedup with respect to two different references: S_{seq} lists the speedup obtained in comparison with the runtime of a pure sequential version, i.e., a version with no conflict detection phase, and S_{par} displays speedup obtained by taking the runtime of a parallel algorithm on one processor as a reference.

Since the deterministic algorithms gave acceptable results on the relatively sparse graphs of Set I, the randomized variants were run only on the graphs from Set II.

The experimental results show that Algorithm SD2 uses many fewer colors than the bound $\Delta^2 + 1$ and that Algorithm ID2 reduces the number of colors by up to 10% compared to SD2. The advantage of exploiting symmetry in Problem SYMCOLPART can be seen by comparing the number of colors used in SD2 and SD$\frac{3}{2}$; the latter can be as much as 37% fewer than the former. In general, the number of colors used in our deterministic algorithms increases only slightly with increasing p. For the randomized

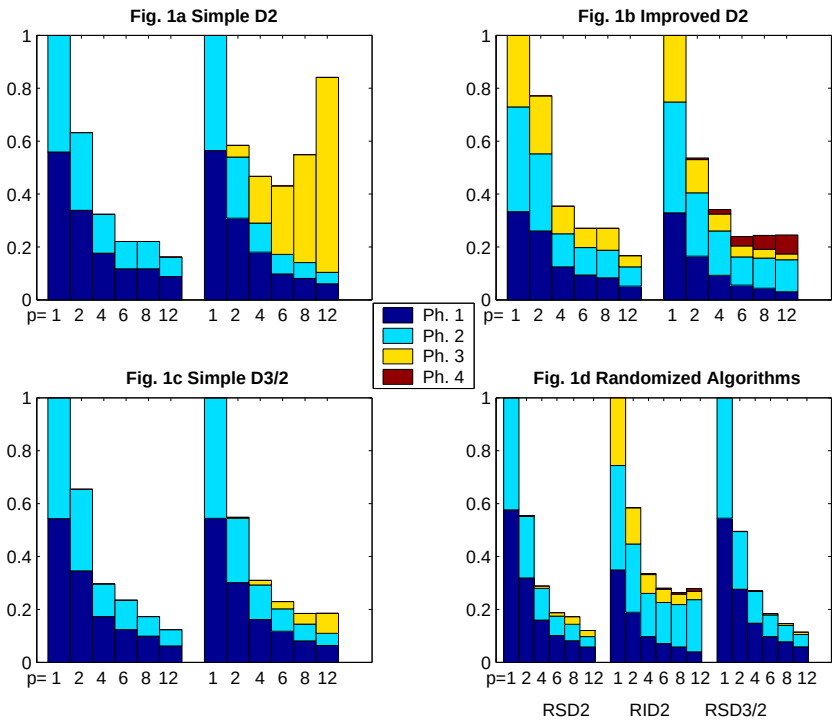

Figure 1: **a-c**: Relative performance of deterministic algorithms on graphs fe144 (left-half on each sub-figure) and ev02 (right-half). Fig. 1d: Relative performance of randomized algorithms on ev02.

algorithms, the number of colors in some cases even decreases as p increases, but we think this is a random phenomenon.

Based on the results displayed in Tables 2 through 7, Fig. 1 shows how the different phases of the algorithms scale as more processors are employed for two representative graphs: fe144 from Set I and ev02 from Set II. For graph fe144, the time used in resolving conflicts sequentially is negligible compared to the overall time. Moreover, it can be seen that Phases 1 and 2 of Algorithms SD2 and SD$\frac{3}{2}$ scale rather well on this graph as the number of processors is increased. However, Phase 2 of Algorithm ID2 does not scale as well. This is due to the existence of many color classes with few vertices which entails extra synchronization and communication overhead in the parallel re-coloring phase.

The picture for the more dense graph ev02 is different: the time elapsed in Phase 3 of Algorithm SD2 is significant and increases as the number of processors is increased. The situation is somewhat better for SD3/2 and ID2. Note that ev02 is about 100 times denser than fe144 (where density $= \frac{|E|}{|V|^2}$), and the results in Tables 2 to 7 and Fig. 1 agree well with the results

in Corollary 3: when the average degree is high, we lose scalability.

Fig. 1d shows how using probabilistic algorithms solves the problem of high number of conflicts for graph ev02. The improvement in scalability comes at the expense of increased number of colors used (see Table 2 to 7). We have experimented using different values for q and found good results when $q = 1/20$ for RSD2, and $q = 1/6$ for RID2 and RSD$\frac{3}{2}$.

It should be noted that the speedups observed in Fig. 1 are all relative to the respective parallel algorithm run with $p = 1$ (see S_{par} in the various tables). A comparison against a sequential version with no conflict detecting and resolving phase would yield less speedup (see S_{seq} in the tables). In particular, relative to a sequential version, the ideal speedup obtained by Algorithms SD2 and ID2 is roughly $\frac{1}{2}p$ and $\frac{2}{3}p$, respectively.

Our algorithms in general did not scale well beyond around 16 processors. We believe this is due to, among other things, the relatively high cost associated with non-local physical memory accesses. It would be interesting to see how this affects the behavior of the algorithms on different parallel platforms.

5 Conclusion

We have presented several simple and effective parallel approximation algorithms as well as results from OpenMP-implementations for the D2- and D$\frac{3}{2}$-coloring problems. The number of colors produced by the algorithms in the case where $p = 1$ is generally good as it is typically off from the lower bound $\Delta + 1$ of SD2 by a factor much less than the approximation ratio Δ. As more processors are employed, the algorithms provide reasonable speedup while maintaining the quality of the solution. In general, our deterministic algorithms seem to be suitable for sparse graphs and the probabilistic variants for more dense graphs. We believe the functionality provided by our algorithms is useful for many large-scale optimization codes, where parallel speedups while desirable, are not paramount, as long as running times for coloring are low relative to the other steps in the optimization computations. The three sources of parallelism in our algorithms – partitioning, speculation, and randomization – can be exploited in developing distributed parallel algorithms, but the algorithms would most likely differ significantly from the shared memory variants presented here.

References

[1] T. F. Coleman and J. J. Moré. Estimation of sparse Jacobian matrices and graph coloring problems. *SIAM J. Numer. Anal.*, 20(1):187–209, February 1983.

[2] T. F. Coleman and J. J. Moré. Estimation of sparse Hessian matrices and graph coloring problems. *Math. Program.*, 28:243–270, 1984.

[3] T. F. Coleman and A. Verma. The efficient computation of sparse Jacobian matrices using automatic differentiation. *SIAM J. Sci. Comput.*, 19(4):1210–1233, July 1998.

[4] J. C. Culberson. Iterated greedy graph coloring and the difficulty landscape. Technical Report TR 92-07, Department of Computing Science, The University of Alberta, Edmonton, Alberta, Canada, June 1992.

[5] I. Finocchi, A. Panconesi, and R. Silvestri. Experimental analysis of simple, distributed vertex coloring algorithms. In *Proceedings of the Thirteenth ACM-SIAM Symposium on Discrete Algorithms (SODA 02)*, San Francisco, CA, 2002.

[6] A. H. Gebremedhin and F. Manne. Scalable parallel graph coloring algorithms. *Concurrency: Pract. Exper.*, 12:1131–1146, 2000.

[7] A. H. Gebremedhin, F. Manne, and A. Pothen. Graph coloring in optimization revisited. Technical Report 226, University of Bergen, Dept. of Informatics, Norway, January 2002. Available at: http://www.ii.uib.no/publikasjoner/texrap/.

[8] R. Greenlaw, H.J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press, New York, 1995.

[9] A.K.M S. Hossain and T. Steihaug. Computing a sparse Jacobian matrix by rows and columns. *Optimization Methods and Software*, 10:33–48, 1998.

[10] M. T. Jones and P. E. Plassmann. A parallel graph coloring heuristic. *SIAM J. Sci. Comput.*, 14(3):654–669, May 1993.

[11] S. O. Krumke, M. V. Marathe, and S. S. Ravi. Approximation algorithms for channel assignment in radio networks. In *Dial M for Mobility, 2nd International Workshop on Discrete Algorithms and Methods for*

Mobile Computing and Communications, Dallas, Texas, September 30 – October 1 1998.

[12] Y. Lin and S. S. Skiena. Algorithms for square roots of graphs. *SIAM J. Disc. Math.*, 8:99–118, 1995.

[13] S. T. McCormick. Optimal approximation of sparse Hessians and its equivalence to a graph coloring problem. *Math. Program.*, 26:153–171, 1983.

[14] V. V. Vazirani. *Approximation Algorithms*. Springer, 2001. Chapter 5.

Problem	p	χ_1	χ_3	K	T_1	T_2	T_3	T_{tot}	S_{par}	S_{seq}
mrng2	1	12	12	0	11.5	11	0	22.5	1	0.5
mrng2	2	12	12	1	6.5	6.4	0	12.9	1.7	0.9
mrng2	4	12	12	9	4.2	4	0	8.3	2.7	1.4
mrng2	6	12	12	9	2.8	2.8	0	5.7	4	2
mrng2	8	12	12	9	2.5	2.5	0	5	4.5	2.3
mrng2	12	13	13	16	1.5	1.5	0	3	7.5	3.8
fe144	1	41	41	0	3.8	3	0	6.8	1	0.6
fe144	2	40	40	3	2.3	2.0	0	4.3	1.6	0.9
fe144	4	41	41	13	1.2	1	0	2.2	3.1	1.7
fe144	6	41	41	25	0.8	0.7	0	1.5	4.5	2.5
fe144	8	43	43	20	0.8	0.7	0	1.5	4.5	2.5
fe144	12	42	43	110	0.6	0.5	0	1.1	6.2	3.5
m14b	1	42	42	0	5.6	4.8	0	10.4	1	0.5
m14b	2	43	43	0	3.5	3.1	0	6.6	1.6	0.9
m14b	4	44	44	24	1.8	1.6	0	3.4	3.1	1.7
m14b	6	43	43	16	1.7	1.4	0	3.1	3.4	1.8
m14b	8	44	46	28	1.2	1	0	2.2	4.7	2.6
m14b	12	43	44	70	0.9	0.8	0	1.8	5.8	3.1
ev01	1	3393	3393	0	38.2	28.9	0	67.1	1	0.6
ev01	2	3298	3497	1514	18.7	14.1	7.6	40.4	1.7	0.9
ev01	4	3150	3598	4555	10.3	7	23.4	40.8	1.7	0.9
ev01	6	3215	3725	7786	7.2	5.3	41.3	54	1.2	0.7
ev01	8	2902	3755	10113	5.7	3.9	52	61.6	1.1	0.6
ev01	12	2922	3900	17872	3.9	3	95	102	0.7	0.4
ev02	1	4260	4260	0	144	111	0	255	1	0.6
ev02	2	4212	4353	1012	78.8	58.9	11.2	148.9	1.7	1
ev02	4	4184	4633	4050	46	28	45	119	2.1	1.2
ev02	6	4172	4660	6206	25	18.8	66	110	2.3	1.3
ev02	8	4152	4870	8873	20.6	15.4	104	140	1.8	1
ev02	12	4036	5168	16394	15.5	11	188	214.7	1.2	0.7

Table 2: Performance of Algorithm SD2.

Problem	p	χ_1	χ_2	χ_4	K_1	K_2	T_1	T_2	T_3	T_4	T_{tot}	S_{par}	S_{seq}
mrng2	1	12	11	11	0	0	9.6	11	8.9	0	29.6	1	0.7
mrng2	2	12	11	11	1	0	4.8	5.7	4.7	0	15.3	1.9	1.3
mrng2	4	13	10	10	0	0	3.6	5.4	2.8	0	11.8	2.5	1.7
mrng2	6	12	10	10	6	0	2.8	3.8	2.4	0	9	3.3	2.3
mrng2	8	12	11	11	9	0	2	2.7	2	0	6.7	4.4	3.1
mrng2	12	12	10	10	20	0	1.5	2	1.5	0	5	6	4.1
fe144	1	41	37	37	0	0	3.2	3.8	2.6	0	9.6	1	0.7
fe144	2	41	38	38	0	0	2.5	2.8	2.1	0	7.4	1.3	1
fe144	4	41	37	37	18	0	1.2	1.2	1	0	3.4	2.8	2.1
fe144	6	41	38	38	39	0	0.9	1	0.7	0	2.6	3.7	2.7
fe144	8	41	37	37	55	0	0.8	1	0.8	0	2.6	3.7	2.7
fe144	12	41	38	38	78	0	0.5	0.7	0.4	0	1.6	6	4.4
m14b	1	42	41	41	0	0	6	8.8	5	0	19.8	1	0.7
m14b	2	43	41	41	0	0	3.9	4.9	3.4	0	12.3	1.6	1.2
m14b	4	43	41	41	26	0	2	2.9	1.7	0	6.6	3	2.2
m14b	6	43	41	41	17	0	1.6	1.5	1	0	4.1	4.8	3.6
m14b	8	44	41	41	31	0	1.2	1.3	1	0	3.6	5.5	4.1
m14b	12	44	41	41	45	0	0.9	1	0.7	0	2.7	7.3	5.5
ev01	1	3393	3148	3148	0	0	38.5	44.5	28.5	0	111.5	1	0.7
ev01	2	3301	3132	3290	1687	316	18.4	24.7	13.4	1.8	58.4	1.9	1.4
ev01	4	3159	3091	3263	4171	863	10	19	7.3	4.6	41	2.7	2
ev01	6	3220	3116	3371	7913	1475	7.8	17.5	5.4	8.5	39.2	2.8	2.1
ev01	8	2927	3041	3468	10932	2675	5.3	15.3	4	14.3	39	2.9	2.1
ev01	12	2906	3071	3481	18264	3040	3.8	17	2.8	16.4	40.6	2.7	2
ev02	1	4260	4016	4016	0	0	150	191	115	0	456	1	0.7
ev02	2	4210	4024	4085	1019	207	75.2	109.2	57.5	2.7	244.7	1.9	1.4
ev02	4	4194	4023	4226	4154	644	42.3	76.5	29.2	7.6	155.7	2.9	2.7
ev02	6	4186	4031	4247	6196	1354	25.4	48.6	19	16	109	4.2	3.1
ev02	8	4138	4013	4349	10732	2007	20	52	15	24	111	4.1	3.1
ev02	12	4030	4008	4455	18392	2819	14	55.2	10	32.7	112.5	4.1	3

Table 3: Performance of Algorithm ID2.

Problem	p	χ_1	χ_3	K	T_1	T_2	T_3	T_{tot}	S_{par}	S_{seq}
mrng2	1	10	10	0	9.3	9	0	18.3	1	0.5
mrng2	2	10	10	1	5.2	4.8	0	10	1.8	0.9
mrng2	4	10	10	6	3.3	2.7	0	6	3	1.6
mrng2	6	10	10	4	2.5	2.5	0	5	3.7	1.9
mrng2	8	11	11	12	2.3	2.3	0	4.6	4	2
mrng2	12	11	11	10	1.7	1.7	0	3.4	5.4	2.7
fe144	1	35	35	0	4.4	3.7	0	8.1	1	0.5
fe144	2	36	36	0	2.8	2.5	0	5.3	1.5	0.8
fe144	4	35	35	6	1.4	1	0	2.4	3.3	1.8
fe144	6	36	36	15	1	0.9	0	1.9	4.3	2.3
fe144	8	35	35	11	0.8	0.6	0	1.5	5.4	2.9
fe144	12	36	36	39	0.5	0.5	0	1	8.1	4.4
m14b	1	34	34	0	5.4	4.6	0	10	1	0.5
m14b	2	37	37	0	2.9	2.6	0	5.5	1.8	1
m14b	4	37	37	3	1.9	1.6	0	3.5	2.9	1.5
m14b	6	39	39	7	1.6	1.3	0	2.9	3.5	1.9
m14b	8	37	37	10	1.2	1.1	0	2.3	4.3	2.3
m14b	12	38	38	16	1	0.8	0	1.8	5.6	3
ev01	1	2148	2148	0	35.5	29	0	64.5	1	0.6
ev01	2	1969	2012	135	19.2	15.4	0.7	35.3	1.8	1
ev01	4	1915	2070	366	9.5	7.6	1.9	19	3.4	1.9
ev01	6	2429	2618	538	7.7	6	2.8	16.5	3.9	2.2
ev01	8	1822	2286	1031	5.6	4.3	6.1	16	4	2.2
ev01	12	2103	2687	1424	4	3	7.5	14.5	4.4	2.5
ev02	1	2697	2697	0	134.4	112.6	0	247	1	0.5
ev02	2	2626	2653	83	74.4	60.1	1	135.5	1.8	1
ev02	4	2590	2736	340	40	32.2	4.4	76.5	3.2	1.8
ev02	6	2584	2818	595	29	21	6.7	56.7	4.4	2.4
ev02	8	2626	2967	960	20	15.6	10	45.5	5.4	3
ev02	12	2536	3049	1569	15.7	11.4	18.6	45.7	5.4	3

Table 4: Performance of Algorithm SD$\frac{3}{2}$.

Problem	p	χ_1	χ_3	K	T_1	T_2	T_3	T_{tot}	S_{par}	S_{seq}
ev01	1	4077	4077	0	38	28	0	66	1	0.6
ev01	2	4084	4088	59	20	15	0.3	35.2	1.9	1.1
ev01	4	4043	4051	139	10	7.8	0.8	19	3.5	2
ev01	6	4132	4144	572	7	5.5	3.5	16	4.1	2.4
ev01	8	3998	4015	388	6	4.3	2	12	5.5	3.2
ev01	12	4019	4042	587	4	2.8	3.2	10	6.6	3.8
ev02	1	5564	5564	0	148.3	109	0	257.3	1	0.6
ev02	2	5581	5584	43	82	60	0.6	143	1.8	1
ev02	4	5518	5525	187	41	31	2.3	75	3.4	2
ev02	6	5553	5562	273	26	19	3.3	49	5.2	3
ev02	8	5576	5583	558	21	16	7.4	44	5.8	3.4
ev02	12	5514	5536	522	15	10	6	31	8.3	4.8

Table 5: Performance of Algorithm RSD2, $q = 1/20$.

Problem	p	χ_1	χ_2	χ_4	K_1	K_2	T_1	T_2	T_3	T_4	T_{tot}	S_{par}	S_{seq}
ev01	1	3719	3169	3169	0	0	36.6	42.5	27	0	106.1	1	0.7
ev01	2	3706	3161	3173	171	21	19	27	15	0.1	61	1.7	1.3
ev01	4	3648	3183	3217	483	152	10	21	7.5	0.8	40	2.7	2
ev01	6	3736	3205	3326	998	193	8	21	5.3	1	35	3	2.3
ev01	8	3601	3167	3236	1272	342	5.7	19	4	2	30	3.5	2.6
ev01	12	3628	3195	3291	1857	447	3.7	18.7	2.6	2.2	27	3.9	2.9
ev02	1	4919	4024	4024	0	0	132.7	150	97.3	0	380	1	0.7
ev02	2	4871	4027	4066	208	43	72	98	52	0.5	223	1.7	1.3
ev02	4	4860	4025	4082	526	127	37	62	27	1.5	128	3	2.2
ev02	6	4865	4040	4084	787	138	27	59	19	1.7	108	3.5	2.6
ev02	8	4869	4032	4100	1577	207	22	61	15	2.3	101	3.8	2.8
ev02	12	4839	4031	4104	1953	320	15	75	12	4	106	3.6	2.7

Table 6: Performance of Algorithm RID2, $q = 1/6$.

Problem	p	χ_1	χ_3	K	T_1	T_2	T_3	T_{tot}	S_{par}	S_{seq}
ev01	1	2392	2392	0	37	30	0	67	1	0.6
ev01	2	2242	2242	23	18	15	0.1	33	2	1.1
ev01	4	2189	2190	46	9.7	7.8	0.3	18	3.7	2
ev01	6	2582	2583	55	7.6	5.4	0.3	13	5.1	2.8
ev01	8	2170	2225	138	5.8	4.8	0.7	11	6.1	3.4
ev01	12	2496	2528	191	3.8	3	1.3	8	8.4	4.6
ev02	1	3142	3142	0	140	117	0	257	1	0.5
ev02	2	3049	3049	18	71	56	0.2	127	2	1.1
ev02	4	3076	3080	51	38	31	0.6	69	3.7	2
ev02	6	3018	3060	123	25	21	1.4	48	5.3	2.9
ev02	8	3106	3125	155	20	16	1.7	37	6.9	3.8
ev02	12	2995	3081	214	15	12	2.4	29	8.8	4.8

Table 7: Performance of Algorithm RSD$\frac{3}{2}$, $q = 1/6$.