Parallel Heuristics for Improved, Balanced Graph Colorings†

Robert K. Gjertsen, Jr.
Computer Science Department
University of Illinois, Urbana, IL 61801
(gjertsen@ncsa.uiuc.edu)

Mark T. Jones
Department of Computer Science
University of Tennessee
Knoxville, TN 37996
(jones@cs.utk.edu)

Paul E. Plassmann
Mathematics and Computer Science Division
Argonne National Laboratory, Argonne, IL 60439
(plassman@mcs.anl.gov)

Abstract: The computation of good, balanced graph colorings is an essential part of many algorithms required in scientific and engineering applications. Motivated by an effective sequential heuristic, we introduce a new parallel heuristic, PLF, and show that this heuristic has the same expected runtime under the P-RAM computational model as the scalable coloring heuristic introduced by Jones and Plassmann (JP). We present experimental results performed on the Intel DELTA that demonstrate that this new heuristic consistently generates better colorings and requires only slightly more time than the JP heuristic.

In the second part of the paper we introduce two new parallel color-balancing heuristics, PDR(k) and PLF(k). We show that these heuristics have the desirable property that they do not increase the number of colors used by an initial coloring during the balancing process. We present experimental results that show that these heuristics are very effective in obtaining balanced colorings and, in addition, exhibit scalable performance.

† *This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38. In addition, the second author received support from the 1994-1995 UTK Professional Development Award Program.*

1. Introduction. The graph coloring problem arises in many scientific computing applications. For example, the efficient computation of sparse Jacobian matrices [4] and the parallel solution of sparse triangular linear systems [5] [11] [12] [14] [17] [18] [19] [21] require graph colorings. Determining an optimal coloring for a general graph is known to be an NP-hard problem [7]. Fortunately, effective sequential heuristics [4] [19] have been developed and implemented for graph coloring problems of practical importance.

On serial computers these heuristics are inexpensive relative to the other required computational tasks in most applications. However, if these other tasks are executed on a massively parallel computer, the sequential coloring heuristics may dominate the execution time. In addition, it may not be practical to use a serial implementation of these heuristics because the problems may be too large to fit into the memory available to a single processor. To address this problem, Jones and Plassmann developed a scalable graph coloring heuristic [13]. By scalable we mean that if the size of the subproblem assigned to each processor is kept constant, the running time of the heuristic remains constant (or increases very slowly) as the number of processors is increased.

Although it is effective in many contexts, the Jones and Plassmann (JP) parallel coloring heuristic has two apparent deficiencies. First, the colorings produced by this heuristic often use slightly more colors than colorings computed by the best sequential heuristics. Second, for graphs that have highly variable local structure, the JP heuristic can generate colorings where the number of vertices assigned a particular color varies significantly among processors. This color imbalance can result in a significant load imbalance in subsequent parallel computation based on this coloring. In this paper we introduce new heuristics that address both of these problems while maintaining the scalable performance of the original heuristic.

To solve the first problem, we recall that the JP heuristic achieves a fast parallel running time by coloring a sequence of independent sets based on random numbers assigned to the vertices [13]. In this paper, we combine this approach with a more sophisticated strategy based on ideas from a successful sequential coloring heuristic. This new heuristic (PLF) has the same expected runtime as the JP heuristic under the P-RAM parallel computation model, and we show experimentally that it generates better colorings.

To address the second problem, we introduce two new parallel heuristics, PDR(k) and PLF(k), that dramatically improve the color balance of the JP and PLF heuristics on highly irregular graphs. An important feature of these heuristics is that they use an existing coloring (such as that produced by JP or PLF) and balance that coloring without increasing the total number of colors used. In addition, we show that these heuristics empirically exhibit scalable performance.

In the remainder of this paper, we specify the graph coloring problem and review effective sequential heuristics in §2. In §3 the JP heuristic is reviewed, and we introduce the PLF heuristic. The balanced coloring problem is described in more detail in §4, prior work is discussed, and the new heuristics, PDR(k) and PLF(k), are introduced. Computational experiments performed on the Intel DELTA are described in §3 and §4 to support the claims of improved quality and scalability of the new methods. Finally, we summarize this work in §5.

2. The Graph Coloring Problem and Sequential Heuristics. We begin by first reviewing the necessary graph terminology. Let $G = (V, E)$ be an undirected graph with vertex set V and edge set $E = \{(u, v) \mid u, v \in V\}$. The set of vertices adjacent to $v \in V$ is denoted as $adj(v)$, and the *degree* of a vertex v, defined by $deg(v) = |adj(v)|$, is the number of vertices adjacent to v. We write the maximum degree of the graph G as $\Delta(G) = \max\{deg(v) \mid v \in V\}$. We say that a set of graphs is of *bounded degree* if for each G in this set, $\Delta(G)$ is bounded and independent of $|V|$ (i.e., $\Delta(G)$ is not proportional to the number of vertices of G). A *connected component* of G is a subgraph $G' = (V', E')$ of G such that for all $u, v \in V'$ there exists a path from u to v in G'. A *clique* $C = (V', E')$ is a connected component of G for which there exists an edge in E' between each pair of vertices in V'. An *independent set* of G is a set of vertices I

such that there exists no edge (u, v) with $u, v \in I$. A *coloring* of G is a mapping $\sigma : V \rightarrow K$ that maps each vertex $v \in V$ to a color $K = \{1, 2, \ldots, k\}$ such that $\sigma(v) \neq \sigma(u)$ for each $(u, v) \in E$. Note that the set of vertices assigned a color in K is an independent set in G. We denote the number of colors used in the mapping σ by $|\sigma|$.

The objective of the graph coloring problem is to find a coloring σ of G such that $|\sigma| \leq |\tau|$ for all colorings τ. The smallest number of colors required for coloring G is known as the *chromatic number* of G and is denoted by $\chi(G)$. As previously mentioned, the graph coloring problem is NP-hard for graphs that require three or more colors [7]. There are some well-known bounds on $\chi(G)$ including $\chi(G) \leq \Delta(G) + 1$ and $\chi(G) \geq |C_{\max}|$, where $C_{\max}$ is the largest clique in G.

Numerous fast greedy coloring heuristics appear in the literature; all color a graph by using some criterion to order the vertices. The general procedure followed by these methods is outlined in Figure 1. All the greedy methods choose a vertex color in the same way; they differ in how the

$V' \leftarrow V$;
For $i = 1$ to n **do**
Choose vertex $v_i \in V'$ according to coloring criterion;
Choose the smallest possible color $\sigma(v_i)$ for vertex v_i;
$V' \leftarrow V' \setminus \{v_i\}$;
endfor

FIG. 1. *General sequential coloring heuristic*

vertices to be colored are ordered. Effective, well-known ordering techniques include the largest first ordering (LFO) [23], the incidence degree ordering (IDO) [4], and the saturation degree ordering (SDO) [2]. The vertex ordering for each of these methods is determined at each step i in the above greedy heuristic as follows:

- LFO chooses $v_i \in V'$ such that $deg(v_i) \geq deg(v_j)$ for all $j > i$. At each step the vertex with the maximum degree in the graph $G' = (V', E')$ is chosen. Intuitively, the method first colors the vertices that could produce the highest colors.
- IDO chooses the first vertex v_1 to be the vertex with the maximum degree in G. Subsequently, vertex v_i is chosen as the vertex with the maximal degree in the subgraph induced by $\{v_1, v_2, \ldots, v_{i-1}, v_i\}$. That is, the vertex with the maximum *incidence degree* is chosen at each step, where the incidence degree of a vertex is the number of adjacent colored vertices. At each step the IDO algorithm chooses the vertex maximally constrained by G'.
- SDO chooses the initial vertex v_1 to be the vertex with the maximum degree in G. Subsequently, vertex v_i is chosen as the vertex with the maximum *saturation degree* in V' where the saturation degree of a vertex v is the number of different colored vertices adjacent to v.

The IDO method has a running time proportional to $\sum_{v \in V} deg(v)$, and LFO has a running time of $\sum_{v \in V} deg(v) + |V| \log |V|$. The SDO method is the slowest heuristic and requires $\sum_{v \in V} deg^2(v)$ time; this complexity can be improved to $\sum_{v \in V} deg(v)$, but requires approximately doubling the amount of storage [4].

Sequential coloring heuristics have been compared by several authors, including a study by Matula et al. with random graphs [16], an analysis by Brélaz on general random graphs [2], and work by Coleman and Moré [4] with random graphs and matrices from various well-known test suites such as the Harwell collection [6]. Overall, these studies found SDO to be best, closely followed by IDO. LFO performed well, but was not quite as good as either SDO or IDO. Features of the LFO heuristic are incorporated in the PLF heuristic described in the next section.

3. An Improved Parallel Graph Coloring Heuristic. We begin by reviewing the context of a practical, distributed-memory heuristic for coloring a graph G. We assume that we have a good graph partitioning (an assignment of vertices to partitions) and a good assignment of partitions to processors. We combine these two aspects by assuming that the number of partitions equals the number of processors. Let the set of processors be P, and let the mapping $\Gamma : V \rightarrow P$ represent the assignment of the vertex set V to the set of processors P.

We note that good heuristics exist for determining such partitionings [9] [20] [22]. By "good" we mean that the heuristics are able to assign nearly equal numbers of vertices (or vertex weights) to each processor while minimizing the edges that cross partitions (edges whose vertices are assigned to different processors). Of course, these heuristics are not able to determine an optimal partitioning because this problem is NP-hard; however, they perform well in practice.

Assuming that we possess a good graph partitioning and assignment mapping Γ, we initially focus on a high-level procedure for coloring the graph $G = (V, E)$ in parallel. We use the original approach of Jones and Plassmann [13]. With this method the vertices on each processor are classified as either local or global nodes. The set of *local* nodes V^L is defined by $V^L = \{v \mid \Gamma(v) = \Gamma(u)\ \forall (v,u) \in E\}$. We define the set of *global* nodes V^G to be the remaining vertices, that is, $V^G = \{v \mid \exists (v,u) \text{ with } \Gamma(v) \neq \Gamma(u)\}$. The local and global vertices assigned to processor i by Γ are denoted by V_i^L and V_i^G, respectively, and their union as V_i. The approach uses two phases to color the graph:

1. Color the global vertices (i.e., those vertices that are on the global edge separator). The global edge separator E^G is the set of edges whose vertices are assigned to different processors.
2. Color the local vertices independently on each processor, thereby extending the global coloring to a coloring σ for the entire graph.

This approach produces a valid coloring σ as stated by the following theorem.

THEOREM 3.1. *Let σ_G be a coloring for $G(V^G)$. This coloring, restricted to V_i^G, can be independently extended to a coloring σ_i for the subgraph $G(V_i)$. If we define the function σ by $\sigma(v) = \sigma_i(v)$ when $v \in V_i$, then σ is a coloring for G.*

Proof: See [13]. □

The primary advantage of the method is that once the global separator vertices are colored, the processors can work independently to color their local vertices. An additional benefit is that (V^G, E^G) may be sparser than G and hence may require fewer colors than a heuristic that must consider all of G simultaneously. This two-level approach is used for all the methods described in this paper. With this approach, one must determine how to color the separator vertices in the global phase and how to color the local vertices in the second phase. For the local phase, a good sequential coloring heuristic such as IDO or SDO can be used by each processor; the SDO method generates a better coloring but is more time consuming. The problematic issue of coloring the global vertices is considered next.

3.1. Previous Parallel Coloring Heuristics. The JP heuristic [13] colors the global vertices by finding independent sets of vertices, coloring them, and updating nonlocal neighbors asynchronously. The heuristic is inspired by a parallel algorithm introduced by Luby to determine a maximal independent set in a graph [15]. The Luby heuristic has a fast expected runtime (logarithmic in the number of vertices) under the P-RAM computational model. However, the Luby algorithm has the practical disadvantage that it is an inherently synchronous algorithm, requiring many global synchronizations (for each color, the algorithm requires the same number global synchronizations as its P-RAM runtime complexity). The JP heuristic has a slightly faster expected runtime, but its major advantage is that is an *asynchronous* algorithm. This feature of the heuristic allows for very efficient, scalable implementations on distributed-memory machines.

We first formulate the JP heuristic under the CREW P-RAM model [10] with each processor assigned a single vertex from the graph. Later we will study the heuristic modified for a parallel,

distributed-memory MIMD machine. We assign each vertex v a unique number $\rho(v)$, which we use to generate a partial ordering of the vertices. Let $u(v)$ be an independent random number, uniformly distributed between 0 and 1. The JP heuristic chooses $\rho(v) = u(v)$. Consider, at some point in the heuristic, the subset of uncolored vertices I, where $v \in I$ if and only if $\rho(v) > \rho(w)$ for all uncolored vertices $w \in adj(v)$. Note I is an independent set of vertices and, therefore, can be colored in parallel. We do not have to explicitly construct these independent sets. Instead, for each vertex v we divide $adj(v)$ into two sets, those vertices $w \in adj(v)$ with $\rho(w) > \rho(v)$ and the remaining vertices. We wait for messages from the former set that give the colors these vertices have been assigned; color v the smallest unused color; and send this color to the latter, uncolored set.

This asynchronous heuristic is outlined in Figure 2. By enforcing the coloring invariant $\rho(v) > \rho(w)$ for all uncolored $w \in adj(v)$, we obtain a consistent coloring, and there is no need for processor synchronization. The running time of the heuristic for bounded degree graphs under the CREW P-RAM model is $EO(\log(n)/\log\log(n))$. As shown in [13], this time is proportional to the expected maximum length monotonic path in G, where a monotonic path of length t is a path of t vertices $\{v_1, v_2, \ldots, v_t\}$ such that $\rho(v_1) > \rho(v_2) > \ldots > \rho(v_t)$.

```
Choose ρ(v);
n-wait = 0;
send-queue = ∅;
For each w ∈ adj(v) do
    Send ρ(v) to processor responsible for w;
    Receive ρ(w);
    if (ρ(w) > ρ(v)) then
            n-wait = n-wait + 1;
    else
            send-queue ← send-queue ∪ {w};
    endif
endfor
n-recv = 0;
While (n-recv < n-wait) do
    Receive σ(w);
    n-recv = n-recv + 1;
endwhile
σ(v) = smallest available color consistent with the
    previously colored neighbors of v;
For each w ∈ send-queue do
    Send σ(v) to processor responsible for w;
endfor
```

Fig. 2. *Jones/Plassmann (JP) CREW P-RAM asynchronous parallel coloring heuristic*

The parallel distributed-memory MIMD version of this heuristic, given in Figure 3, maintains a group of vertices on each processor. Let the global edge separator E^G be the set of edges (u, v) such that $\Gamma(u) \neq \Gamma(v)$. In Figure 3 *Seq-color* () colors a queue of vertices given a partial coloring σ. The procedure *Pack-and-send* () packs these new colors into messages for the appropriate processors and sends the information to the designated processors. The *Pack-and-send* () routine is designed to overcome the high communication start-up cost, a characteristic of most message-passing architectures. The JP coloring technique is fast and produces good colorings for finite-difference stencils and finite-element models [13].

```
Determine V_i^G, V_i^L;     {Partition vertices}
color-queue = ∅;
For each v ∈ V_i^G do     {Set up queues for separator vertices}
    n-wait(v) = 0;
    send-queue(v) = ∅;
    For each edge (v, w) ∈ E^G do
        Compute ρ(w);
        if (ρ(w) > ρ(v)) then
            n-wait(v) = n-wait(v) + 1;
        else
            send-queue(v) ← send-queue(v) ∪ {w};
        endif
    endfor
    if (n-wait(v) = 0) then
        color-queue ← color-queue ∪ {v};
    endif
endfor
Seq-color ( σ, color-queue );     {Color any vertices in V_i^G not}
n-colored = | color-queue |;       {waiting for messages}
Pack-and-send (σ, color-queue, send-queue );
color-queue = ∅;
While (n-colored < |V_i^G|) do
    Receive msg;
    For each w ∈ msg.vertex-list do
        σ(w) = msg.vertex-color;
        For each v ∈ msg.vertex-adj do
            n-wait(v) = n-wait(v) − 1;
            if (n-wait(v) = 0) then
                color-queue ← color-queue ∪ {v};
            endif
        endfor
    endfor
    Seq-color (σ, color-queue);     {Color subsets of V_i^G once required}
    n-colored = n-colored + | color-queue |;     {messages are received}
    Pack-and-send (σ,color-queue, send-queue);
    color-queue = ∅;
endwhile
Seq-color (σ, V_i^L );     {Color local vertices last}
```

FIG. 3. *The Jones/Plassmann distributed-memory coloring heuristic for the i-th processor*

3.2. An Improved Coloring Heuristic. To improve the resulting colorings, we modify the above method by using the degree of a vertex v in a manner analogous to sequential LFO heuristic. The heuristic is essentially the same as the JP heuristic except that a vertex v is colored first if its degree is larger than that of its uncolored adjacent vertices. If adjacent vertices have the same degree, the random numbers are used to determine the coloring order. Since this heuristic is related to the LFO sequential heuristic, we call the new method the parallel largest first (PLF) heuristic. We note that Allwright et al. [1] has independently determined that this heuristic is effective in numerical calculations involving dynamically triangulated random surfaces.

The PLF heuristic can be implemented by a straightforward modification of the JP heuristic. Let $\rho(v) = u(v) + deg(v)$, and recall that $u(v)$ is between 0 and 1. Thus, replacing the function $\rho()$ in Figure 3 yields an implementation of the PLF heuristic.

The motivation for PLF, as with LFO, is to color the most difficult vertices (those of largest degree) first, where we might be constrained to use the largest colors. Unlike the JP heuristic, PLF considers the local structure of the graph when forming independent sets, producing an improved coloring. However, this strategy may make the execution time of the heuristic more problem dependent.

The following theorem states that PLF has the same CREW P-RAM expected runtime bound as the JP heuristic for graphs whose maximum degree is bounded by some constant Δ.

THEOREM 3.2. *The expected running time of the PLF heuristic under the CREW P-RAM model is* $EO(\log(n)/\log\log(n))$ *for a bounded degree graphs with* n *vertices.*

Proof: Random numbers are used to break ties for inclusion in the independent set among vertices of the same degree. This is equivalent to using the JP heuristic to determine a sequence of independent sets among vertices with the same degree and hence requires at most $EO(\log(n)/\log\log(n))$ time. By assumption, the maximum degree of the graph is bounded by some constant Δ; thus the number of distinct vertex degrees is also bounded by Δ. Hence, the total expected runtime is $EO(\log(n)/\log\log(n))$. □

Although the JP and PLF heuristics have the same expected runtimes, one would expect the actual runtime for PLF to be at least that of the JP heuristic. If there are many distinct vertex degrees, one might expect that the runtime could be much higher. However, the following experimental results show that the PLF heuristic takes only slightly more time than the original heuristic and produces better colorings.

3.3. Comparison of the JP and PLF Heuristics. In this section we experimentally compare the JP and PLF parallel coloring heuristics. The experiments were performed on the Intel DELTA computer, a 16×32 mesh-connected array of Intel i860 processors. Communication is accomplished via message passing on this MIMD architecture.

Two problem types are used to perform these comparisons. The Crystal problem is a set of graphs arising from a finite-element model of a piezoelectric crystal [3]. The domain is a rectangular solid that is regularly discretized and second-order, hexahedral elements are used. The sequence of graphs is chosen such that each graph is twice as large as its predecessor. In this way, the problem size can be scaled with the number of processors such that the number of vertices on an individual processor is kept nearly constant. Details about the sequence of graphs are given in Table 1, where $w(V^G) = \sum_{v \in V^G} w(v)$ gives the total global vertex weight and $w(V^L) = \sum_{v \in V^L} w(v)$ gives the total local vertex weight. Unless otherwise noted, the vertex degrees, $w(v) = deg(v)$, are used for the vertex weights.

The second problem, the Kall problem, is a single problem unlike the sequence of Crystal problems. The Kall problem is a 3D finite-element mesh from a structural mechanics application. The model includes primarily tetrahedral elements; however, it also includes some beam and plate elements. Table 2 gives details on the characteristics of this graph as it is partitioned over an increasing number of processors.

For both the JP and PLF coloring heuristics, the IDO method is used for the *Seq-color* () routine for the global phase, and the SDO method was used for the local phase. This combination

TABLE 1
Crystal problem specification for different $|P|$

Processors	$\|V\|$	$\|V^G\|$	$w(V^G)$	$\|V^L\|$	$w(V^L)$
1	866	0	0	866	142850
2	1446	337	79220	1109	172430
4	2278	886	206030	1392	208052
8	4918	2225	533245	2693	416133
16	10294	5260	1282125	5034	781237
32	21334	11487	2833756	9847	1553574
64	43606	24712	6136937	18894	2985113
128	88726	51212	12802343	37514	5981147
256	179350	105438	26476597	73912	11803341

TABLE 2
Kall problem specification for various $|P|$

Processors	$\|V\|$	$\|V^G\|$	$w(V^G)$	$\|V^L\|$	$w(V^L)$
1	10556	0	0	10556	162774
2	10556	393	9258	10163	153516
4	10556	977	22992	9579	139782
8	10556	1864	40673	8692	122101
16	10556	2455	52743	8101	110031
32	10556	4604	80865	5952	81909

performs at least as well as any other combination of the heuristics tested in [8]. The number of colors required for the global phase and the total coloring are given for the JP and PLF heuristics for the Crystal and Kall problems in Tables 3 and 4, respectively. Note that the PLF heuristic is consistently superior.

TABLE 3
Colors required for Crystal problem using JP and PLF parallel heuristics

Processors	JP $\|\sigma_G\|$	JP $\|\sigma\|$	PLF $\|\sigma_G\|$	PLF $\|\sigma\|$
1	108	108	108	108
2	110	110	108	108
4	110	110	108	108
8	113	114	108	108
16	115	116	108	113
32	117	118	108	113
64	118	120	109	113
128	118	120	110	114
256	120	121	114	114

The Crystal problem is used to measure the scalability of the PLF heuristic. The problem size is kept proportional to the number of processors. Hence, in order to be scalable the runtime of the heuristics should remain constant as the number of processors is increased. Observe in Figure 4 that the PLF and JP methods essentially do meet this criterion; although there is a slight increase in running time, it is a very slowly growing function of the number of processors. Times for small numbers of processors are omitted because the ratio of global to local vertices and edges does not stabilize until $|P| = 8$, as indicated by Table 1.

The Kall problem is used to directly compare the execution times of PLF and JP and to

TABLE 4
Colors required for Kall using JP and PLF parallel heuristics

Processors	JP $\|\sigma_G\|$	JP $\|\sigma\|$	PLF $\|\sigma_G\|$	PLF $\|\sigma\|$
1	9	9	9	9
2	10	11	8	9
4	11	11	8	9
8	13	13	9	9
16	12	12	9	9
32	13	13	9	10

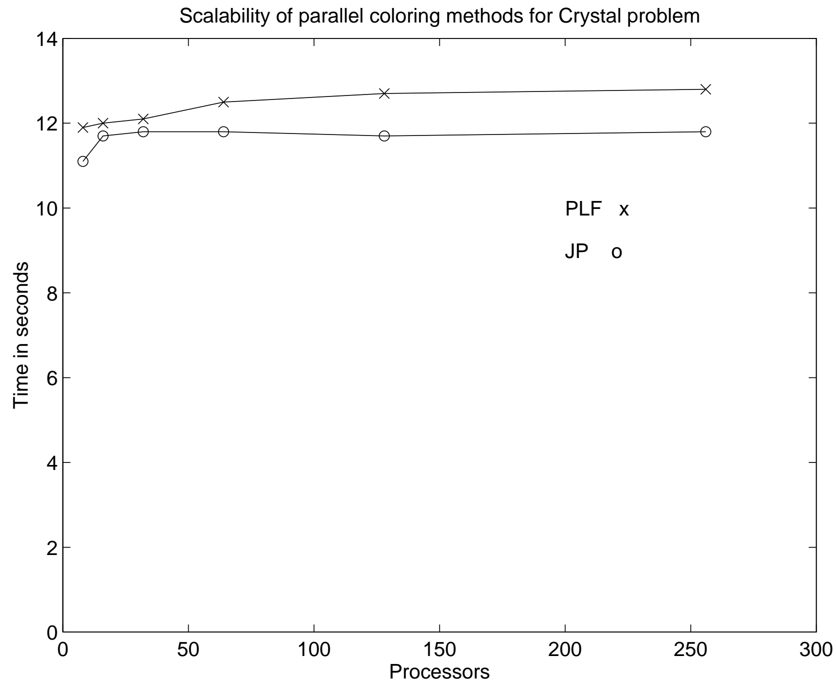

FIG. 4. *Time required by the JP and PLF parallel heuristics for the Crystal problem sequence. Note that the time required is nearly independent of the number of processors, demonstrating the scalability of both heuristics.*

examine their performance for a fixed size problem as $|P|$ increases. Comparing the execution time of the methods in Figure 5, we see that JP and PLF are nearly indistinguishable. The heuristic performs well as $|P|$ increases, with some deterioration as the ratio of communication to computation increases.

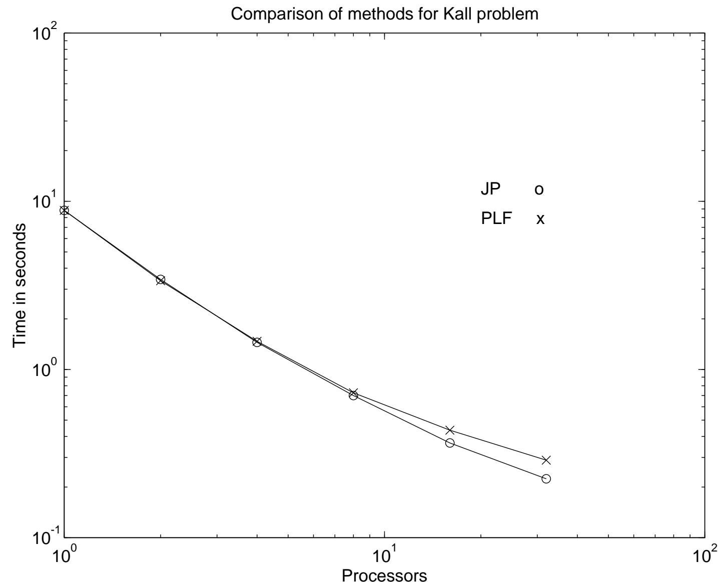

FIG. 5. *Comparison of the execution times of the JP and PLF heuristics as a function of the number of processors used*

4. Balanced Graph Colorings for Irregular Problems. A graph coloring is often used to represent how work is scheduled for execution on a parallel computer. The graph vertices indicate the tasks to be performed, vertices of the same color represent independent work that can be performed concurrently, and the graph edges represent dependencies between the tasks. Processors can simultaneously work on the vertices (tasks) of color i and proceed to vertices colored $i+1$ when all adjacent vertices of color i are completed.

The problem can be generalized to include a weight, $w(v)$, associated with each vertex v, representing the amount of work required by that task. Load-balancing problems occur when some processors have a significantly larger number of vertices (or total weight) of the current color i than other processors have. This color imbalance can cause a processor workload imbalance even if the total weight assigned to each processor is equal.

Such imbalances are more likely to occur when the underlying graph is not homogeneous, or regular, in structure. A finite-difference discretization using a single stencil or a uniform order finite-element model is homogeneous, and the coloring imbalance for such graphs is generally not significant. However, an irregular, or nonhomogeneous, graph can arise when different finite-element types or finite-difference stencils are used to model different portions of a physical domain. The resulting graph contains subgraphs that may have very different edge connection patterns and vertex degrees. If these subgraphs are assigned to different processors, one may expect that the assignment of tasks to colors would not be balanced on different processors. Such situations often occur in complex engineering applications.

On a homogeneous parallel computer, a graph coloring is ideally balanced when the processors each have the same total vertex weight per color. It is assumed throughout this section that the vertices have been partitioned among the processors such that each processor is assigned a nearly

equal weighted sum of vertices. Without such an assignment, a balanced coloring is not possible. In the discussions that follow, we assume the parallel computer is homogeneous; however, the definitions and methods can be extended to a heterogeneous system.

In [19] the imbalance of a coloring is quantified by using the following definitions. The average weight of color c with the processor set P is given by

$$\mu_c = \frac{1}{|P|} \sum_{v \in V | \sigma(v) = c} w(v) \quad . \tag{4.1}$$

The imbalance for color c on processor p, $I(c,p)$, is defined by

$$I(c,p) = \left(\sum_{v \in V_p | \sigma(v) = c} w(v) \right) - \mu_c \quad . \tag{4.2}$$

The imbalance of color c is given by

$$I(c) = \max_{p \in P} \{ I(c,p) \} \quad . \tag{4.3}$$

The total color imbalance, I_σ, is given by the expression

$$I_\sigma = \sum_{c \in \sigma} I(c) \quad . \tag{4.4}$$

A coloring is ideally balanced if $I_\sigma = 0$. The metric $I(c)$ measures the largest amount of imbalance for a color c produced by a processor, and I_σ indicates the total imbalance over all colors used. One could construct other measures; however, we have found this method to be a simple and effective indicator.

Pommerell et al. [19] give algorithms for producing balanced colorings; a parallel variant of their most effective algorithm is summarized in Figure 6. This heuristic colors only the global vertices, ignoring edge dependencies between vertices on the same processor. The algorithm

```
Choose tolerance I_max;
c = 1;
tol = 0;
While uncolored global vertices remain do
        All processors attempt to color a global vertex with color c;
        if all processors are not successful then
              if tol < I_max then
                     tol = tol + 1;
              else
                     tol = 0;
                     c = c + 1;
              endif
        endif
endwhile
```

FIG. 6. *Parallel version of the balanced coloring heuristic given by Pommerell et al. with* $w(v) = 1$

clearly limits $I(c)$ to I_{max}, a user-chosen tolerance, and $I_\sigma \leq |\sigma| I_{max}$. A potential shortcoming of the method is that, in general, the number of colors required to color the graph will increase

for small I_{max}. Note that the algorithms given in [19] were designed to run on a sequential computer and are not well suited for MIMD implementation. In parallel, at most $|P|$ vertices are colored per step, and a global synchronization is required after each step. These synchronization steps result in a poor communication-to-computation ratio.

4.1. Two New Parallel Balancing Heuristics. The two parallel heuristics introduced in this section, PDR(k) and PLF(k), work by improving the balance of an existing coloring without increasing the number of colors. Both heuristics are local optimization techniques that improve the balance by selectively moving vertices from one color to another legal color.

We begin by introducing a measure of the deviation of a coloring from a perfectly balanced coloring; our heuristics will perform local optimizations with respect to this measure. Suppose the coloring that we have already computed is σ and that it uses $|\sigma|$ colors. Given the sum of the weights of all the vertices, ideally we could require that this total weight be equally distributed among all the colors. Thus, we define our goal on processor p, $\gamma^*(p)$, to be

$$\gamma^*(p) = \frac{1}{|\sigma|} \sum_{v \in V \ | \ \Gamma(v)=p} w(v) \quad , \tag{4.5}$$

the ideal weight of the vertices assigned each color. Given a particular coloring σ, we can measure how close to that ideal we are by calculating $\gamma(c,p)$ for each color $c = 1, 2, \ldots, |\sigma|$ as

$$\gamma(c,p) = \sum_{v \in V \ | \ \sigma(v)=c, \ \Gamma(v)=p} w(v) \quad . \tag{4.6}$$

This sum gives the current total weight of all vertices assigned color c. The balance deviation, $\delta(c,p)$, for color c on processor p is defined as

$$\delta(c,p) = \gamma(c,p) - \gamma^*(p) \quad . \tag{4.7}$$

We choose to attempt to minimize the balance deviations, $\delta(c,p)$, instead of the total imbalance, I_σ, for two reasons. First, balancing the weight assigned each color is important to ensure load balancing for many problems. For example, when using the coloring to solve sparse triangular systems [12], we would like the work associated with each color to be equal. The condition $I_\sigma = 0$ by itself does not imply this equality. Second, performing optimizations that minimize balance deviations can be done locally on a processor; this process does not require vertices to be moved between processors. We note that if the total weight assigned to each processor is equal, then finding a coloring with zero total deviation implies that $I_\sigma = 0$. Finally, a lower bound on I_σ using these recoloring strategies is based on the inherent weighted imbalance in the graph G with partitioning Γ, $I_{(G,\Gamma)}$, which is

$$I_{(G,\Gamma)} = \max_{i \in P} \{ w(V_i) - \frac{w(V)}{|P|} \}. \tag{4.8}$$

From this definition it is clear that $I_\sigma \geq I_{(G,\Gamma)}$.

The deviance reduction heuristic works by moving vertices from one color j with positive deviation to another legal color k with a lower deviation when this exchange will reduce the total deviation. If one imagines a bin associated with each color, the color-balancing problem is similar to the bin-packing problem, with the added constraint that a vertex cannot be placed in the same bin as an adjacent vertex. One of the best theoretical bounds for the bin-packing problem is obtained by the "first fit decreasing" heuristic [7], which works by first sorting the items by size and trying to pack the largest items first. We use a greedy strategy, working first with the color with the largest positive deviance. Based on the bin-packing heuristic results, for a specific color, the vertex with the largest weight is chosen. This vertex is moved to the least-filled bin consistent with colors of its neighbors. In Figure 7, we give a sequential version of

```
Perform initial coloring σ;
σ̃ = σ;
V(j) = {v ∈ V | σ(v) = j};
γ* = Σ_{v∈V} w(v)/|σ|;     {Ideal bin weight}
γ(j) = Σ_{v∈V | σ(v)=j} w(v) for j = 1, 2, ..., |σ|;     {Calculate current bin weights}
i = 1;
Choose j such that γ(j) = max{γ(l)};
While γ(j) − γ* > ε and i ≤ n do
        Choose v_i ∈ V(j) such that w(v_i) ≥ w(u) ∀u ∈ V(j);
        S = {l | l ≠ σ̃(adj(v_i)) and γ(j) > γ(l) + w(v_i)};     {Find eligible colors}
        if S ≠ ∅ then     {Move vertex?}
              Choose k such that γ(k) = min{S};     {Choose smallest bin}
              σ̃(v_i) = k;
              γ(σ(v_i)) = γ(σ(v_i)) − w(v_i);
              γ(k) = γ(k) + w(v_i);
        endif
        V(j) ← V(j) \ {v_i};
        i = i + 1
        Choose j such that γ(j) = max{γ(l)};
endwhile
```

FIG. 7. *Sequential version of the deviance reduction (DR) balanced coloring heuristic*

a balancing heuristic based on deviance reduction (DR). Note that one can make multiple passes of the DR heuristic to further improve the balance.

A major advantage of the DR heuristic is that it improves the balancing of an existing coloring without increasing the number of required colors. Thus, the best available coloring heuristic can be used to obtain the initial coloring—using as few colors as possible—and a better balanced coloring can be obtained using that number of colors. The following theorem shows that the number of colors is not increased by the DR heuristic.

THEOREM 4.1. *Let σ be an initial coloring of G. The DR heuristic computes a new coloring $\tilde{\sigma}$, with $|\sigma| \geq |\tilde{\sigma}|$.*

Proof: Because a vertex can be recolored only with an existing color, it is clear that the number of colors cannot increase. □

It is also important that the DR heuristic have a fast runtime. Consider a graph with n vertices, maximum degree Δ, and assume that the vertex weights are the degrees of the vertices. Under these assumptions, the following theorem shows that the sequential heuristic has a linear runtime for bounded Δ.

THEOREM 4.2. *Consider the DR heuristic given in Figure 7. Assume that graph on which the heuristic is used has maximum degree Δ. Let n be the number of vertices in the graph, and assume that we use vertex weights defined by $w(v) = deg(v)$. Then the running time of the sequential DR heuristic is bounded by $O(n\Delta)$.*

Proof: The computation of the ideal bin weight γ^* and the current bin weights $\gamma(j)$ requires time proportional to the number of vertices, or $O(n)$ time.

Recall that any greedy coloring requires no more than $\Delta + 1$ colors. Hence we can assume that $|\sigma| \leq \Delta + 1$. The number of bins is equal to the number of colors; thus, the vertices can be sorted by color in $O(n)$ time. In addition, because we use the vertex degrees as weights, vertices of each color can be sorted by weight in $O(n)$ time.

Using these sorted arrays, we can select the vertex v_i in constant time at each iteration through the while loop. To compute the set S, we need to look at the color of each adjacent vertex. By definition, there can be no more than Δ adjacent vertices. To choose the smallest bin requires no more time than the maximum number of colors, or $O(\Delta)$ time. Finally, selecting the bin with maximum $\gamma(j)$ requires at most $O(\Delta)$ time. All the other steps in the while loop require constant time.

The number of times through the while loop is the number of vertices, n. Hence, the entire heuristic requires $O(n\Delta)$ time. □

```
Given an initial parallel coloring σ;    {|σ| is the maximum color among the processors}
σ̃ = σ;
γ(j,i) = Σ_{v∈V_i | σ(v)=j} w(v) for j = 1,2,...,|σ|;
γ*(i) = Σ_{v∈V_i} w(v)/|σ|;    {Ideal bin weight for this processor}
δ(j,i) = γ(j,i) − γ*(i) for j = 1,2,...,|σ|;
ρ(v) = δ(σ(v),i) + u(v) for all v ∈ V_i^G;
Set up n-wait, send-queue and color-queue according to new ρ;
DR-Seq-color ( σ̃, color-queue );    {Color any vertices in V_i^G not}
n-colored = | color-queue |;         {waiting for messages}
Pack-and-send (σ̃, color-queue, send-queue );
color-queue = ∅;
While (n-colored < |V_i^G|) do
    Receive msg;
    For each w ∈ msg.vertex-list do
        σ̃(w) = msg.vertex-color;
        For each v ∈ msg.vertex-adj do
            n-wait(v) = n-wait(v) − 1;
            if (n-wait(v) = 0) then
                color-queue ← color-queue ∪ {v};
            endif
        endfor
    endfor
    DR-Seq-color (σ̃, color-queue);    {Color subsets of V_i^G once required}
    n-colored = n-colored + | color-queue |;    {messages are received}
    Pack-and-send (σ̃,color-queue, send-queue);
    color-queue = ∅;
endwhile
DR-Seq-color (σ̃, V_i^L );    {Color local vertices last}
```

FIG. 8. *Parallel deviance reduction (PDR) coloring heuristic for the i-th processor*

We now introduce two parallel heuristics, PLF(k) and PDR(k), that use the sequential DR heuristic to obtain balanced colorings. An initial coloring is required for both of these methods; we assume that the PLF heuristic is used. The PLF(k) heuristic performs k recolorings for balancing. The recoloring heuristic is the same as the JP MIMD heuristic given in Figure 3 with the same ordering function as with PLF, $\rho(v) = u(v) + deg(v)$, and the sequential DR heuristic used for *Seq-color* ().

To construct the PDR(k) heuristic, we also use the the sequential DR heuristic for *Seq-color* (). However, rather than using the vertex degree in $\rho(v)$, we use the local color deviations at the start of each recoloring. Thus, we choose $\rho(v) = \delta(\sigma(v), p) + u(v)$, where $\delta(\sigma(v), p)$ is the color deviation before any vertices have been recolored on the k-th iteration. By using

the local deviation of the color $\sigma(v)$ as the basis for independent sets, the processors with the colors of greatest imbalance, having the largest quantities $I(c,p)$, are given priority over other processor/color pairs. The PDR(k) heuristic is given in detail in Figure 8.

Note that a P-RAM analysis that assumes only one vertex per processor does not make sense for these balancing heuristics. Instead, we experimentally show that the algorithms have a scalable runtime similar to that of JP and PLF.

4.2. Experimental Results for the Balanced Coloring Heuristics. In this subsection we present experimental results that demonstrate the effectiveness of the two heuristics at minimizing the color imbalance. We use two test problems, the Kall problem that was described earlier and the FDgrid problem. The FDgrid problem is a nonhomogeneous problem, specifically designed to test balanced coloring methods. The problem is obtained from the discretization of a square domain, using a 27-point stencil in the middle portion and a 7-point stencil at the east and west ends. Table 5 describes the specifics of the FDgrid problem for the processor sets used. For all of the balancing experiments the vertex degrees are used as the vertex weights, i.e., $w(v) = deg(v)$. The experiments are again performed on the Intel DELTA.

TABLE 5
Description of the FDgrid problem for different $|P|$

Processors	$\|V\|$	$\|V^G\|$	$w(V^G)$	$\|V^L\|$	$w(V^L)$
1	32768	0	0	32768	460692
2	32768	1297	21588	31471	439104
4	32768	3527	68286	29241	392406
8	32768	5351	109130	27417	351562
16	32768	7164	139217	25604	321475
32	32768	9758	193565	23010	267127
64	32768	12540	239332	20228	221360

The first set of experiments examines how the imbalance is affected by varying the parameter k, the number of recolorings, for the heuristics PLF(k) and PDR(k). We measure the total color imbalance, I_σ, as defined earlier in Equation 4.4. Tables 6 and 7 show the experimental results for $k = 0, 1, 2, 3, 4$ with the FDgrid problem and $|P| = 64$. It appears that the most benefit is received by using two recolorings, although one recoloring significantly reduces the imbalance. Additional experiments that we performed support this conclusion, but we do not report those results here. We also note that although PLF(0) and PDR(0) are the same heuristic, the order in which messages are received can vary. This fact accounts for differences in the results for PLF(0) and PDR(0) in the following tables.

TABLE 6
Total imbalance I_σ *produced by PLF(k) and PDR(k) for* $|P| = 64$ *on the FDgrid problem*

k	PLF(k)	PDR(k)
0	6739	6690
1	394	291
2	188	166
3	175	160
4	175	160

The next experiment examines the reduction in the coloring imbalance of the heuristics as the number of processors, $|P|$, varies. In Tables 8, 9, and 10, one sees that applying PLF(1) and PDR(1) significantly reduces the imbalance for the FDgrid and Kall problems. An additional improvement is obtained with one more balancing iteration as used by PLF(2) and PDR(2). In

Table 7

Maximum color imbalance $\max\{I(c)\}$ *produced by PLF(k) and PDR(k) with* $|P| = 64$ *for the FDgrid problem*

k	PLF(k)	PDR(k)
0	1435	1431
1	77	33
2	17	17
3	17	17
4	17	17

Table 8

Total imbalance I_σ *produced by PLF(k) and PDR(k) on the FDgrid problem*

Processors	$\sum_{c\in\sigma}\mu_c$	$I_{(G,\Gamma)}$	PLF(0)	PDR(0)	PLF(1)	PDR(1)	PLF(2)	PDR(2)
2	230344	0	5414	5414	24	29	15	20
4	115169	2	58109	58109	93	66	53	50
8	57582	17	18207	17827	143	146	68	61
16	28787	13	20526	20596	319	227	134	109
32	14390	11	11824	12115	326	233	167	149
64	7191	17	6739	6690	394	291	188	166

Table 9

Maximum color imbalance $\max\{I(c)\}$ *for PLF(k) and PDR(k) on the FDgrid problem*

Processors	$\sum_{c\in\sigma}\mu_c$	PLF(0)	PDR(0)	PLF(1)	PDR(1)	PLF(2)	PDR(2)
2	230344	1841	1841	8	6	3	3
4	115169	17506	17506	31	10	6	7
8	57582	4482	4003	35	30	12	10
16	28787	5609	5593	79	55	14	11
32	14390	2898	3067	69	43	15	17
64	7191	1435	1431	77	33	17	17

Table 10

Total imbalance I_σ *produced by PLF(k) and PDR(k) on the Kall problem.*

Processors	$\sum_{c\in\sigma}\mu_c$	$I_{(G,\Gamma)}$	PLF(0)	PDR(0)	PLF(1)	PDR(1)	PLF(2)	PDR(2)
2	81384	8	1663	1663	31	21	17	12
4	40689	22	4276	4276	70	97	27	29
8	20343	10	2513	2196	59	48	37	39
16	10169	30	2336	2373	60	79	51	51
32	5082	34	1448	1476	109	123	66	78

Tables 8 and 10 we show the lower bound $I_{(G,\Gamma)}$ as defined in Equation 4.8. Recall that this lower bound results from the imbalance inherent to the vertex partitioning; note that the two heuristics are able to obtain a total imbalance relatively close to this lower bound.

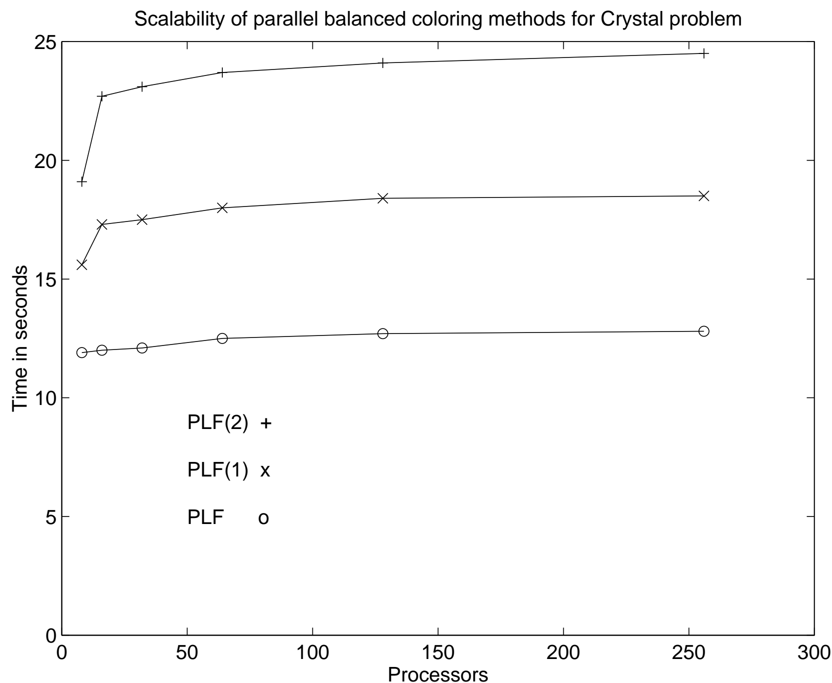

FIG. 9. *Scalability of parallel balanced coloring heuristic PLF(k) on Crystal problem*

Finally, in Figure 9 we illustrate the scalability of the PLF(k) heuristic for $k = 0, 1, 2$ on the Crystal problem. The results of PDR(k) are omitted because they are essentially the same as those given by the PLF(k) heuristic. PLF(0) previously was shown to be scalable, and it evident from the graph that PLF(k) is empirically scalable. The execution time of the heuristic appears to be a slowly increasing function of the number of processors and problem size.

5. Conclusions. The two objectives of our study were to devise new scalable, parallel coloring heuristics that (1) require fewer colors than existing methods, and (2) minimize coloring imbalance while using no more colors than the best parallel coloring method.

The first objective has been achieved by introducing a new heuristic, PLF, that relies on using vertex degrees for independent sets instead of solely random numbers as employed by the original Jones/Plassmann (JP) heuristic. PLF was shown to have the same expected runtime as the JP heuristic under the CREW P-RAM execution model. For our suite of test problems, PLF consistently required fewer colors than JP, and required only slightly more execution time.

To achieve the second objective, balanced colorings, we introduced the the PLF(k) and PDR(k) heuristics. Given an initial coloring, these heuristics perform one or more recolorings that strive to reduce the color deviance by using heuristics based on those successfully used in bin-packing problems. The PLF(k) and PDR(k) heuristics guarantee that the number of colors used by the initial coloring does not increase in the recoloring, while significantly reducing the color imbalance among the processors. Both the PLF(k) and PDR(k) heuristics were empirically shown to be scalable.

REFERENCES

[1] J. Allwright, R. Bordawekar, P. Coddington, K. Dincer, and C. Martin, *A comparison of parallel graph coloring algorithms*, Tech. Rep. SCCS-666, Northeast Parallel Architectures Center, Syracuse University, 1995.

[2] D. Brélaz, *New methods to color the vertices of a graph*, Comm. ACM, 22 (1979), pp. 251–256.

[3] T. Canfield, M. Jones, P. Plassmann, and M. Tang, *Thermal effects on the frequency response of piezoelectric crystals*, in New Methods in Transient Analysis, PVP-Vol. 246 and AMD-Vol. 143, New York, 1992, ASME, pp. 103–108.

[4] T. F. Coleman and J. J. Moré, *Estimation of sparse Jacobian matrices and graph coloring problems*, SIAM Journal on Numerical Analysis, 20 (1983), pp. 187–209.

[5] I. S. Duff and G. A. Meurant, *The effect of ordering on preconditioned conjugate gradients*, BIT, 29 (1989), pp. 635–657.

[6] I. S. Duff and J. K. Reid, *Performance evaluation of codes for sparse matrix problems*, in Performance Evaluation of Numerical Software, L. Fosdick, ed., North-Holland, Amsterdam, 1979, pp. 121–135.

[7] M. R. Garey and D. S. Johnson, *Computers and Intractability*, W. H. Freeman, New York, 1979.

[8] R. K. Gjertsen Jr., *Parallel graph coloring heuristics*, Master's thesis, University of Illinois at Urbana-Champaign, 1994.

[9] B. Hendrickson and R. Leland, *A multilevel algorithm for partitioning graphs*, Tech. Rep. SAND93-1301, Sandia National Laboratories, Applied Mathematical Sciences, Albuquerque, NM, October 1993. Draft.

[10] J. JáJá, *An Introduction to Parallel Algorithms*, Addison-Wesley Publishing Company, 1992.

[11] M. T. Jones and P. E. Plassmann, *The effect of many-color orderings on the convergence of iterative methods*, in Proceedings of the Copper Mountain Conference on Iterative Methods, SIAM LA-SIG, 1992.

[12] ———, *The efficient parallel iterative solution of large sparse linear systems*, in Graph Theory and Sparse Matrix Computation, A. George, J. Gilbert, and J. W. Liu, eds., vol. 56 of The IMA Volumes in Mathematics and Its Applications, Springer-Verlag, 1993, pp. 229–245.

[13] ———, *A parallel graph coloring heuristic*, SIAM Journal on Scientific Computing, 14 (1993), pp. 654–669.

[14] ———, *Scalable iterative solution of sparse linear systems*, Parallel Computing, 20 (1994), pp. 753–773.

[15] M. Luby, *A simple parallel algorithm for the maximal independent set problem*, SIAM Journal on Computing, 4 (1986), pp. 1036–1053.

[16] D. Matula, G. Marble, and J. Isaacson, *Graph coloring algorithms*, in Graph Theory and Computing, R. Read, ed., Academic Press, 1972, pp. 104–122.

[17] R. G. Melhem and V. S. Ramarao, *Multicolor reorderings of sparse matrices resulting from irregular grids*, ACM Transactions on Mathematical Software, 14 (1988), pp. 117–138.

[18] J. M. Ortega, *Orderings for conjugate gradient preconditionings*, SIAM Journal on Optimization, 1 (1991), pp. 565–582.

[19] C. Pommerell, M. Annaratone, and W. Fichtner, *A set of new mapping and coloring heuristics for distributed-memory parallel processors*, SIAM Journal on Scientific and Statistical Computing, 13 (1992), pp. 194–226.

[20] A. Pothen, H. Simon, and K.-P. Liou, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM Journal on Matrix Analysis, 11 (1990), pp. 430–452.

[21] R. Schreiber and W.-P. Tang, *Vectorizing the Conjugate Gradient method*. Unpublished manuscript, Department of Computer Science, Stanford University, 1982.

[22] S. Vavasis, *Automatic domain partitioning in three dimensions*, SIAM Journal on Scientific and Statistical Computing, 12 (1991), pp. 950–970.

[23] D. Welsh and M. Powell, *An upper bound for the chromatic number of a graph and its application to timetabling problems*, Comput. J., 10 (1967), pp. 85–86.