Parallel Heuristics for Improved, Balanced Graph Colorings[†]

Robert K. Gjertsen, Jr.
Department of Computer Science
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 PRAM 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 and NSF grant ASC-9501583.*

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 [6] [14] [19] [20] [21] [23] require graph colorings. Determining an optimal coloring for a general graph is known to be an NP-hard problem [8]. Fortunately, effective sequential heuristics [4] [21] have been developed and implemented for graph coloring problems of practical importance where good, but not necessarily optimal, colorings are required.

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 (JP) developed a scalable graph coloring heuristic [12]. 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 JP parallel coloring heuristic has two 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 exploit the fact 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 [12]. In this paper, we combine this approach with a more sophisticated strategy based on ideas from a successful sequential coloring heuristic. Our new heuristic (PLF) has the same expected runtime as the JP heuristic under the PRAM 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 new 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.

We organize the remainder of this paper as follows. 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 $|V|$. A *clique* $C = (V', E')$ is a subgraph 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) in E for any pair of vertices $u, v \in I$. 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 [8]. 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 an ordering 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) [25], 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 an ordering of the vertices in V such that $deg(v_1) \geq deg(v_2) \geq \ldots \geq deg(v_n)$. 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 a maximum 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.
- 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 in V' with the maximum *saturation degree*, where the saturation degree of a vertex v is the number of different colors used by vertices in $\{v_1, v_2, \ldots, v_{i-1}\}$ that are adjacent to v. Intuitively, this heuristic first colors vertices most constrained by previous color choices.

The IDO and the LFO heuristics have a running time proportional to $\sum_{v \in V} deg(v)$. The SDO method is the slowest heuristic and requires $\sum_{v \in V} deg^2(v)$ time. In certain cases the running time of the SDO heuristic can be improved at the expense of space. For example, for a regular graph with all vertices of degree Δ, the heuristic can be run in time and additional space proportional to $\Delta|V|$.

Sequential coloring heuristics have been compared by several authors, including a study by Matula et al. with random graphs [18], 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 [7]. Overall, these studies found that the heuristics perform well on wide variety of test problems. 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 [10] [22] [24]. 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 [12]. With this method the vertices on each processor are classified as either local or global vertices. The set of *local* vertices V^L is defined by $V^L = \{v \mid \Gamma(v) = \Gamma(u)\ \forall (v,u) \in E\}$. We define the set of *global* vertices V^G to be the remaining vertices, that is, $V^G = \{v \mid \exists (v,u) \in E \text{ with } \Gamma(v) \neq \Gamma(u)\}$. The local and global vertices assigned to processor i by Γ are denoted as V_i^L and V_i^G, respectively, and their union denoted by V_i. The approach uses two phases to color the graph:

1. Color the global vertices, that is, 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 vertex 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 $G(V^G) = (V^G, E^G)$. This coloring, restricted to V_i^G, can be 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 [12]. □

The advantage of the method is that once the global separator vertices are colored, the processors can work independently to color their local vertices. 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 [12] colors the global vertices by finding independent sets of vertices, coloring them, and updating neighboring vertices on other processors asynchronously. The heuristic is inspired by a parallel algorithm introduced by Luby to determine a maximal independent set in a graph [17]. The Luby heuristic has a fast expected runtime (logarithmic in the number of vertices) under the PRAM 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 of global synchronizations as its PRAM runtime complexity). The JP heuristic has a slightly faster expected PRAM runtime, but its major advantage is that it 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 PRAM model [11] with each processor assigned a single vertex from the graph. Later we will study the heuristic modified for a parallel, distributed-memory MIMD machine. Recall that the PRAM computational model assumes that processors communicate through a common shared memory. The CREW variant assumes that different processors are able to read data from same location at the same time, but that only

one processor at a time can write data to a memory location. The essential aspect of the PRAM model used in our analysis is that a processor can access data computed on another processor and stored in shared memory in constant time.

We assign each vertex v a unique number $\rho(v)$, which we use to generate a partial ordering of the vertices. We call the function $\rho(v)$ the *ordering function*. Let $u(v)$ be an independent random number, uniformly distributed between 0 and 1. The JP heuristic chooses ordering function $\rho(v) = u(v)$. 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 that I is an independent set of vertices and, therefore, can be colored in parallel. However, 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 (i.e., communicate through shared memory) 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 PRAM model is $EO(\log(n)/\log\log(n))$. 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 PRAM 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. 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

```
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 vertices in V_i^G not waiting for messages}
n-colored = | color-queue |;
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 distributed-memory, JP coloring heuristic for the ith processor*

colorings for graphs derived from finite-difference stencils and finite-element models [12].

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 the 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. Because 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] have 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 ordering 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 PRAM 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 PRAM model is* $EO(\log(n)/\log\log(n))$ *for a bounded degree graph 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.

We use a test suite of fifteen problems. For all problems, the graph characteristics and more detailed experimental results for several values of p can be found in Tables 5 and 6. We give descriptions of the origins of the problems below. The first four problems are *scaled* problems; for each of these problems we are able to generate a specific instance for a given number of processors such that the problem size scales nearly linearly with the number of processors.

- *CRYSTAL* – A finite-element model of a piezoelectric crystal [3]. The domain is a rectangular solid that is regularly discretized by using second-order, hexahedral elements.
- *ELASTIC2D* – The linear elasticity equations for the plane stress problem on a rectangular region with a central hole discretized by linear triangular elements [15].
- *PREFINE* – A square domain discretized by quadrilateral finite elements such that ninth-order elements are used in one corner, linear elements in the other three corners, and a progression of orders in between.

- *SHELL* – A spherical domain with holes discretized by fourth-order triangular shell elements [13].

The next eleven problems are *static* problems. For these problems, the same instance of the problem is solved on different numbers of processors.

- *BCSSTK16, BCSSTK17, BCSSTK18, BCSSTK29, BCSSTK33* – Graphs associated with matrices arising from problems in structural engineering, all from the Harwell-Boeing sparse matrix collection [5].
- *DT* – A graph arising from the Delaunay triangulation of a random point set in two dimensions with three coupled unknowns associated with each vertex.
- *ELASTIC3D* – The linear elasticity equations on a six-sided solid domain discretized by quadratic finite elements.
- *FDGRID3D* – A finite-difference discretization of a cube for which a 27-point stencil is used in the middle "slice" and a 7-point stencil is used elsewhere.
- *KALL* – A three-dimensional finite-element mesh from a structural mechanics application using tetrahedral, beam, and plate elements.
- *RAND1* – An undirected random graph where the density is chosen such that each vertex has an expected degree of 25. Specifically, the graph is constructed by considering each possible edge e between any two vertices, and including the edge with probability $25/|V|$.
- *RAND2* – A larger version of *RAND1*.

For both the JP and PLF coloring heuristics, the SDO method was used for the *Seq-color* () routine for the global phase, and the SDO method was used for the local phase. This combination performs at least as well as any other combination of the heuristics tested in [9]. We ran each of the problems on 2^i processors, where $i = 0, \ldots, 6$ processors; in a few cases the problems could not fit on a single processor, and for the CRYSTAL problem we were able to run it on 128 and 256 processors as well. In Table 1, we present a summary showing that, overall, the PLF heuristic yields fewer colors than the JP heuristic.

TABLE 1

Number of times the JP or PLF heuristics yields the fewest colors, $|\sigma|$, for coloring the entire graph and the fewest number of colors, $|\sigma_G|$, for the global edge separator graph. For example, the BCSSTK16 problem was run on 2^i processors for $i = 1, \ldots, 6$ and 5 times PLF gave fewer colors than JP for the entire graph and 1 time PLF and JP gave the same number of colors.

Problem	PLF Best $\lvert\sigma\rvert$	PLF Best $\lvert\sigma_G\rvert$	Tie $\lvert\sigma\rvert$	Tie $\lvert\sigma_G\rvert$	JP Best $\lvert\sigma\rvert$	JP Best $\lvert\sigma_G\rvert$
CRYSTAL	8	8	0	0	0	0
ELASTIC2D	0	1	6	4	0	1
PREFINE	1	2	5	4	0	0
SHELL	4	4	1	0	1	2
BCSSTK16	5	6	1	0	0	0
BCSSTK17	1	0	2	4	3	2
BCSSTK18	0	3	2	2	4	1
BCSSTK29	0	1	2	0	4	5
BCSSTK33	3	3	2	2	1	1
DT	0	0	4	2	2	4
ELASTIC3D	5	5	0	0	1	1
FDGRID3D	1	2	2	1	3	3
KALL	6	6	0	0	0	0
RAND1	6	6	0	0	0	0
RAND2	6	6	0	0	0	0
Total	57	64	27	19	19	20

In Figure 4, we plot the runtimes for each of the fifteen problems as a function of the number of processors used. For the scaled problems, the runtime of the heuristics should remain constant as the number of processors is increased, with the exception of small growth in runtime because (1) for small $|P|$, the ratio of global to local vertices is increasing, resulting in more communication and (2) the running time bound for the heuristic suggests a slight increase in runtime with increasing $|P|$. The runtimes for both heuristics are similar and appear to be scalable for all four problems. Two anomalies are (1) the runtime for one processor on the CRYSTAL problem which we believe is due to cache effects, and (2) the sawtooth graph associated with the PREFINE problem which arises because the problem is slightly different for $|P| = 2^i$ when i is odd than when i is even. Because the problems sizes do not exactly scale linearly with $|P|$, we have adjusted the runtimes according to the problem sizes to accurately show the scaling (e.g., if the problem size assigned to each processor increases by a factor of 1.1 when we go from 2 to 4 processors, the runtime for 4 processors is divided by 1.1). More information on runtimes can be found in Table 6. These results also show that the number of colors required tends to increase with the number of processors used.

For the static problems, the runtime of the heuristics should decrease proportionally to $1/|P|$; that is, the lines representing the runtimes in the log-log plots shown in Figure 4 ideally should have a slope of -1. This goal is not, of course, realistic because the ratio of communication to computation is increasing as $|P|$ increases. However, for all but the two random graphs, both heuristics do well, although they do better on larger problems than on smaller problems. The performance on random graphs is poor primarily because of the large separator size; for random graphs the expected size of these separators is $O(|V|)$ [16]. We see in Table 5 that all vertices are global in the partitioning of these random graphs, even for small $|P|$.

Finally, given the randomized nature of the heuristics, one should examine the variation of the quality of the colorings and the runtimes over different runs with different sequences of pseudorandom numbers. Using independent sequences of pseudorandom numbers, in Table 2 we show that for five of the problems the variation in the number of colors and the runtime is small for ten runs of each problem on 32 processors. Similar results were obtained, but are not presented in this paper, for the balanced coloring heuristics given in the next section.

TABLE 2

Mean and variance of the number of colors and runtimes (in seconds) for the two heuristics on 32 processors for ten runs using different pseudorandom number sequences

Problem	mean $\|\sigma\|$ JP	std. dev. $\|\sigma\|$ JP	mean $\|\sigma\|$ PLF	std. dev. $\|\sigma\|$ PLF	mean time JP	std. dev. time JP	mean time PLF	std. dev. time PLF
SHELL	73.3	1.06	72.4	0.97	2.23	0.03028	2.25	0.00675
ELASTIC3D	44.4	0.97	40.4	0.52	0.43	0.00789	0.48	0.00316
BCSSTK16	39.0	0.82	34.5	0.71	0.20	0.01729	0.28	0.00483
BCSSTK29	33.4	0.70	35.9	0.32	0.28	0.00422	0.30	0.00000
BCSSTK33	40.8	0.92	39.9	0.74	0.41	0.00568	0.59	0.00632

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

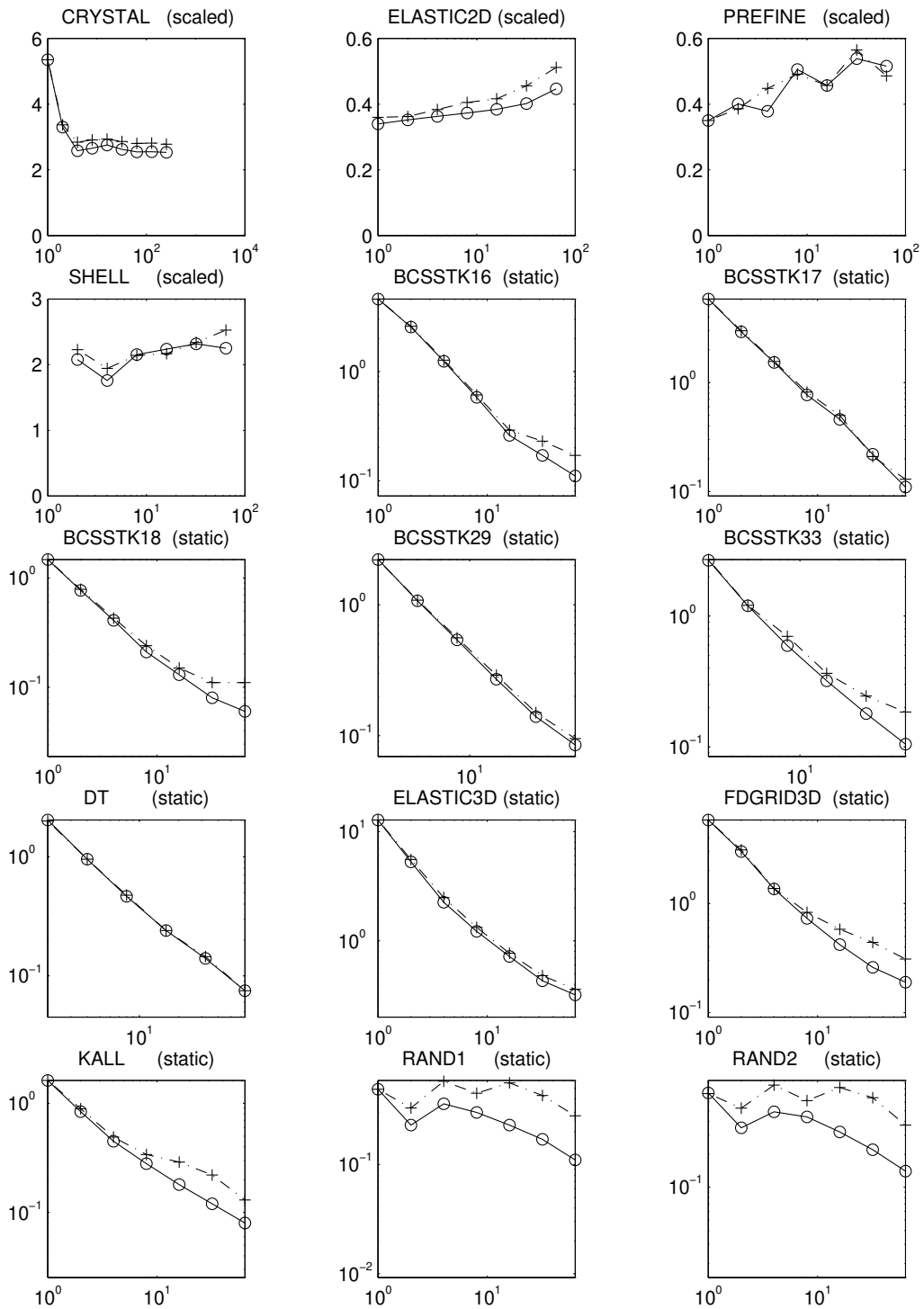

FIG. 4. *Execution times for the JP algorithm (denoted by the solid line with a circle at each data point) and the PLF algorithm (denoted by the dot-dash line with a plus at each data point). The times for the four scaled problems are weighted to take into account any nonexact scaling of the problems sizes. The x-axis gives the number of processors (on a log scale), and the y-axis gives the execution times in seconds (on a log scale for the static problems).*

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 [21] 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

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

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

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

The imbalance of color c is given by

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

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

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

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.

The balancing heuristics that we consider in this paper do not move vertices between processors. Therefore, if the vertex partition Γ does not assign nearly equal weights of vertices to each processor, these heuristics cannot be effective. We can obtain a lower bound on I_σ based on the inherent weight imbalance, $I_{(G,\Gamma)}$, for the graph G with partitioning Γ, which we define as

$$I_{(G,\Gamma)} = \max_{i \in P} \{ w(V_i) - \frac{w(V)}{|P|} \} \ . \tag{4.5}$$

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

Pommerell et al. [21] give algorithms for producing balanced colorings; a parallel variant of their most effective algorithm is summarized in Figure 5. This heuristic colors only the global vertices, ignoring edge dependencies between vertices on the same processor. The algorithm 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 in the parallel execution of this algorithm 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.

```
Choose tolerance I_max;
c = 1;
tol = 0;
C_i ← V_i^G;    {Candidate vertices}
U_i ← V_i^G;    {Uncolored vertices}
While ( ⋃_{i=1}^{P} U_i ≠ ∅ ) do
        if ( C_i ≠ ∅ ) then
                Choose v ∈ C_i;
                σ(v) = c;
                U_i ← U_i \ {v};
        endif
        if ( C_i = ∅ on any processor ) then
                if (tol < I_max) then
                        tol = tol + 1;
                else
                        tol = 0;
                        c = c + 1;
                        C_i ← U_i;
                endif
        endif
        C_i ← C_i \ {adj(v) ∪ v};
endwhile
```

FIG. 5. *Parallel version of the balanced coloring heuristic for processor i given by Pommerell et al. [21] with* $w(v) = 1$

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 that the coloring 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

$$\gamma^*(p) = \frac{1}{|\sigma|} \sum_{v \in V_p} w(v) \ , \tag{4.6}$$

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

$$\gamma(c,p) = \sum_{v \in V_p \mid \sigma(v)=c} w(v) \ . \tag{4.7}$$

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

$$\delta(c,p) = \gamma(c,p) - \gamma^*(p) \ . \tag{4.8}$$

We choose to attempt to minimize positive balance deviations on each processor, $\delta(c,p)$, 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 [14], 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, optimizations that minimize balance deviations can be performed 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 balance deviations implies that $I_\sigma = 0$.

The deviance reduction (DR) 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 a positive deviation on a processor. This greedy strategy considers vertices from the color with current largest positive deviance first. 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 [8], which works by first sorting the items by size and trying to pack the largest items first. Motivated by the bin-packing heuristic, the DR heuristic chooses the vertex with the largest weight from the set of vertices of the color under consideration. This vertex is moved to the least-filled bin consistent with colors of its neighbors. In Figure 6, we give a sequential version of a balancing heuristic based on deviance reduction. Note that one can make multiple passes of the DR heuristic to further improve the balance.

Perform initial coloring σ;
$\tilde{\sigma} = \sigma$;
$V(j) = \{v \in V \mid \sigma(v) = j\}$;
$\gamma^* = \sum_{v \in V} w(v)/|\sigma|$; {Ideal bin weight}
$\gamma(j) = \sum_{v \in V(j)} w(v)$ for $j = 1, 2, \ldots, |\sigma|$; {Calculate current bin weights}
$i = 1$;
Choose j such that $\gamma(j) = \max\{\gamma(l),\ l = 1, 2, \ldots, |\sigma|\}$;
While $(\gamma(j) - \gamma^* > \epsilon$ and $i \leq n)$ **do**
    Choose $v_i \in V(j)$ such that $w(v_i) \geq w(u)\ \forall u \in V(j)$;
    $S = \{l \mid l = 1, 2, \ldots, |\sigma|,\ l \neq \tilde{\sigma}(adj(v_i))$,
        and $\gamma(j) > \gamma(l) + w(v_i)\}$; {Find eligible colors}
    if $(S \neq \emptyset)$ **then**
        Choose k such that $\gamma(k) = \min\{\gamma(l) \mid l \in S\}$; {Choose smallest bin}
        $\tilde{\sigma}(v_i) = k$;
        $\gamma(\sigma(v_i)) = \gamma(\sigma(v_i)) - w(v_i)$;
        $\gamma(k) = \gamma(k) + w(v_i)$;
    endif
    $V(j) \leftarrow V(j) \setminus \{v_i\}$;
    $i = i + 1$;
    Choose j such that $\gamma(j) = \max\{\gamma(l) \mid V(l) \neq \emptyset,\ l = 1, 2, \ldots, |\sigma|\}$;
endwhile

FIG. 6. *Sequential version of the deviance reduction (DR) balanced coloring heuristic*

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 6. Assume that the 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. □

We now introduce two parallel heuristics, PLF(k) and PDR(k). An initial coloring is required for both of these methods; we assume that the PLF heuristic is used. These heuristics can make any number of recoloring passes through the vertex set while attempting to balance the coloring. For PLF(k) and PDR(k) we perform k recolorings for balancing; thus, for example, PLF(2) and PDR(2) use two sweeps though the vertex set. These heuristics use the sequential DR heuristic given in Figure 6 to obtain balanced colorings. The framework for the two heuristics is the same as the JP MIMD heuristic given in Figure 3 except that rather than using the routine *Seq-color*() we use the sequential DR heuristic.

The difference between the two heuristics, PLF(k) and PDR(k), is how the global vertices are ordered; this difference is obtained by using different choices for the ordering function $\rho(v)$. For PLF(k) we use the same choice for $\rho(v)$ as we use for the original PLF heuristic, that is, $\rho(v) = u(v) + deg(v)$. Thus, the only difference between one pass through the PLF(k) heuristic and PLF is that instead of coloring vertices for the first time with the routine *Seq-color*(), we recolor the same sets of vertices by calling the sequential DR heuristic.

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 of each vertex at the start of each pass through the recoloring heuristic. Thus, we use the vertex ordering function $\rho(v) = \delta(\sigma(v), p) + u(v)$, where $\delta(\sigma(v), p)$ is the color deviation before any vertices have been recolored on a sweep.[1] By using the local deviation of the color $\sigma(v)$ as the basis for independent sets, vertices assigned colors having the greatest imbalance are given priority over other vertices.

Note that a PRAM 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.

[1] Note that we cannot change the ordering function $\rho(v)$ during a sweep through the vertices (as the values $\delta(\sigma(v), p)$ change) because this would change the values of the constants *n-wait*(v) given in Figure 3.

4.2. Experimental Results for the Balanced Coloring Heuristics. In this section we present experimental results that demonstrate the effectiveness of the two heuristics at minimizing the color imbalance. We use the same test suite described in the preceding section and again perform the experiments on the Intel DELTA. For all of the balancing experiments the vertex degrees are used as the vertex weights, namely, $w(v) = deg(v)$. We examine two issues in these experiments: (1) the quality of the balancing achieved by the two algorithms, and (2) the runtimes of the algorithms.

The results in Table 7, and additional experiments not reported here, indicate that at most two passes of either the PDR or PLF heuristic are sufficient to rebalance the colorings; additional passes do not significantly improve the balancing. In Table 3 we examine the relative performance of the heuristics after two passes on problems from the test suite. We use the total imbalance, I_σ, to compare the heuristics. Note that the performance of the methods is nearly indistinguishable. More detailed results can be found in Table 7.

TABLE 3

Number of times PLF or PDR yields the smallest total imbalance, I_σ. For example, the BCSSTK16 problem was run on 2^i processors for $i = 1, \ldots, 6$ and 5 times two passes of PLF rebalancing gave a better imbalance and 1 time two passes of PDR gave a better imbalance.

Problem	PLF Best	Tie	PDR Best
CRYSTAL	4	1	3
ELASTIC2D	1	1	4
PREFINE	2	0	4
SHELL	3	0	3
BCSSTK16	5	0	1
BCSSTK17	4	0	2
BCSSTK18	4	0	2
BCSSTK29	2	0	4
BCSSTK33	5	0	1
DT	0	0	6
ELASTIC3D	2	0	4
FDGRID3D	3	1	2
KALL	3	0	3
RAND1	2	0	4
RAND2	4	1	1
Total	44	4	44

Next we examine the quality of the balancings achieved on 16 processors for the PDR algorithm. Since the performance of the algorithms is similar, we chose to examine just one of them in detail. We consider two measures of the balance of a coloring and present bounds of how good those measures can be for a particular problem. If the balancing were optimal, the weight assigned each color on each processor, $\gamma^*(p),\ p = 1, \ldots, |P|$, would be

$$w^* = \frac{w(V)}{|\sigma|\,|P|} \ . \tag{4.9}$$

As $\max I(c)$ is the worst imbalance for any color on any processor, the first measure, M_1, gives the relative size of this maximum imbalance

$$M_1 = \frac{\max I(c)}{w^*} \ . \tag{4.10}$$

Given the imbalance resulting from the partitioning of the vertices, the best that we would expect

to achieve for this measure is

$$\text{Best } M_1 = \frac{I_{(G,\Gamma)}}{w^* \, |\sigma|} \ , \tag{4.11}$$

although this bound is conservative.

To obtain the second measure, imagine that we were to solve a problem where we considered the graph vertices of each color in sequence, and the time the algorithm took for each vertex were equal to the weight of the vertex. Before vertices of the next color could be considered, all processors would have to be finished with their same-colored vertices. The optimal time for this algorithm would be $t_{\text{opt}} = w(V)/|P| + I(G,\Gamma)$, where we assume that we cannot correct the imbalance resulting from the partitioning (this amount is small for the problems we consider, as shown in Figure 5). Given a particular coloring, the time required to solve the problem would be $t_\sigma = w(V)/|P| + I_\sigma$. Thus, the second imbalance measure we consider is defined by

$$M_2 = \frac{t_\sigma}{t_{\text{opt}}} \ . \tag{4.12}$$

The best that one could obtain would be 1.0 for this measure.

In Table 4, we give the initial values for each of these measures prior to balancing on 16 processors. We also show the measures after two passes of the PDR heuristic. For every problem except PREFINE, one of the two measures is quite close to the lower bounds. While we achieved significant reductions for both measures on PREFINE, we note that this problem is inherently difficult and coming close to either of the lower bounds may not be possible. The reason is that the "granularity" of this problem is large; that is, the maximum weight per vertex (maximum degree here) is quite large compared with the average weight in each color.

TABLE 4

Two measures, M_1 and M_2, of the quality of the balancing achieved in the balanced coloring algorithms on sixteen processors. The initial measures are the balance measures prior to balancing and the final measures are after two passes of the PDR balancing algorithm. "Best M_1" gives an approximation for the lower bound on the M_1 measure; the lower bound for M_2 is 1.0. Finally, we contrast the maximum vertex degree with the average weight per color as another measure of the balancing difficulty of a problem.

Name	Init M_1	Final M_1	Best M_1	Init M_2	Final M_2	Δ	Avg. Wgt
CRYSTAL	0.880	0.880	0.215	1.215	1.129	499	1151
ELASTIC2D	0.322	0.006	0.005	1.160	1.000	21	3540
PREFINE	12.960	1.167	0.004	2.927	1.270	288	277
SHELL	0.402	0.149	0.002	1.168	1.041	284	1277
BCSSTK16	0.404	0.098	0.003	1.260	1.056	80	550
BCSSTK17	0.654	0.047	0.005	1.328	1.019	149	787
BCSSTK18	0.906	0.044	0.035	1.270	1.001	48	545
BCSSTK29	0.518	0.024	0.004	1.189	1.011	70	1106
BCSSTK33	0.696	0.055	0.003	1.331	1.026	140	949
DT	0.104	0.005	0.001	1.049	1.003	50	2623
ELASTIC3D	0.520	0.178	0.003	1.271	1.068	419	934
FDGRID3D	2.732	0.006	0.000	1.708	1.004	26	1920
KALL	0.482	0.007	0.002	1.252	1.003	91	1017
RAND1	0.865	0.149	0.115	1.468	1.010	42	235
RAND2	1.140	0.122	0.069	1.469	1.023	45	310

Finally, we look at the runtimes of each of the PLF and PDR balanced coloring heuristics in the previously used graph format. The runtimes for one and two passes of both balanced coloring heuristics are given in Figure 7. Note that the times are, in general, significantly faster

than for the original coloring of the graphs by either the JP or PLF algorithms. The result is a much poorer communication-to-computation ratio than for the other heuristics. Nevertheless, for the scaled problems, the heuristics scale well after $|P| = 8$, when the ratio of global to local vertices has somewhat stabilized. The performance on the static problems does not appear quite as good because of the poor communication to computation ratio. However, for all but the random graphs, we still see significant reductions in runtime as $|P|$ increases. We note that the balancing results and the number of colors for the random graphs are still quite good; it is simply the runtimes that suffer.

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 PRAM 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 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, R. Grimes, J. Lewis, and B. Poole, *Sparse matrix test problems*, SIGNUM Newsletter, 17 (1982), p. 22.

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

[7] 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.

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

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

[10] B. Hendrickson and R. Leland, *The Chaco user's guide: Version 2.0*, Sandia Report SAND94–2692, Sandia National Laboratories, Albuquerque, N.M., June 1995.

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

[12] M. T. Jones and P. E. Plassmann, *A parallel graph coloring heuristic*, SIAM Journal on Scientific Computing, 14 (1993), pp. 654–669.

[13] ———, *Computational results for parallel unstructured mesh computations*, Computing Systems in Engineering, 5 (1994), pp. 297–309.

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

[15] ———, *Parallel algorithms for adaptive mesh refinement*, SIAM Journal on Scientific Computing (to appear).

[16] R. J. Lipton, D. J. Rose, and R. E. Tarjan, *Generalized nested dissection*, SIAM Journal on Numerical Analysis, 16 (1979), pp. 346–358.

[17] M. Luby, *A simple parallel algorithm for the maximal independent set problem*, SIAM Journal on Computing, 4 (1986), pp. 1036–1053.
[18] D. Matula, G. Marble, and J. Isaacson, *Graph coloring algorithms*, in Graph Theory and Computing, R. Read, ed., Academic Press, 1972, pp. 104–122.
[19] 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.
[20] J. M. Ortega, *Orderings for conjugate gradient preconditionings*, SIAM Journal on Optimization, 1 (1991), pp. 565–582.
[21] 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.
[22] 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.
[23] R. Schreiber and W.-P. Tang, *Vectorizing the Conjugate Gradient method.* Unpublished manuscript, Department of Computer Science, Stanford University, 1982.
[24] S. Vavasis, *Automatic domain partitioning in three dimensions*, SIAM Journal on Scientific and Statistical Computing, 12 (1991), pp. 950–970.
[25] 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.

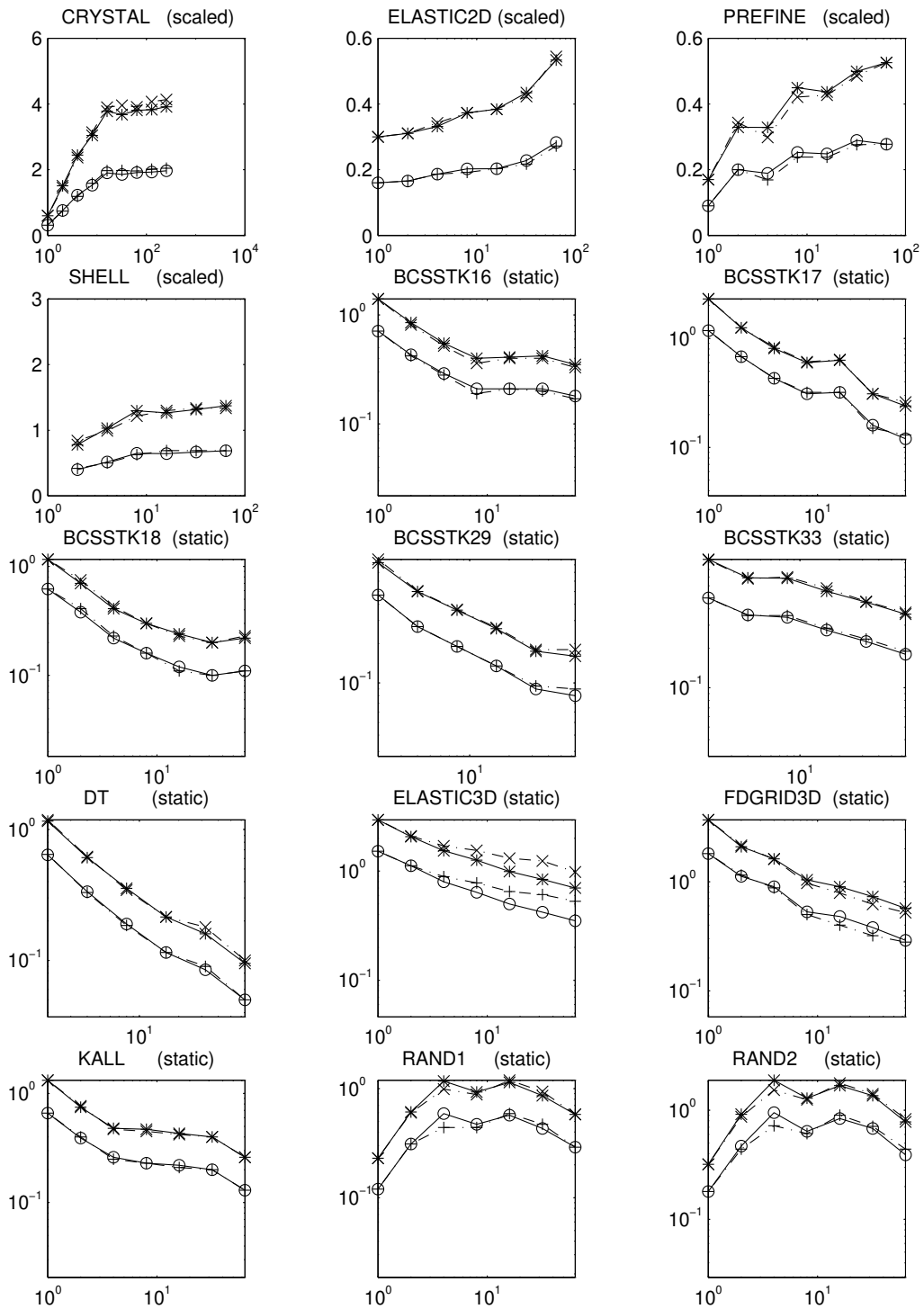

FIG. 7. *Execution times for the PLF balanced coloring algorithm (denoted by the solid lines with a* $\circ$ *denoting one pass and a* $*$ *denoting two passes) and the PDR balanced coloring algorithm (denoted by the dot-dash lines with a* $+$ *denoting one pass and an* $\times$ *denoting two passes). The x-axis represents the number of processors (on a log scale), and the y-axis represents the execution times in seconds. The times for the static problems are on a log scale; the times for scaled problems are weighted to take into account slight differences in the problem size scaling.*

TABLE 5
Graph characteristics ("Local Ratio" denotes the ratio of the number of local vertices to total vertices)

Problem	$\|P\|$	$\|V\|$	$\|E\|$	Local Ratio	Avg. Wgt. Per Proc.	$I_{(G,\Gamma)}$	Min. Deg.	Max. Deg.
CRYSTAL	1	886	142850	1.00	142850	0	92	499
CRYSTAL	16	10294	2063362	0.31	128960	27712	92	499
CRYSTAL	64	43606	9122050	0.25	142532	14140	92	499
ELASTIC2D	1	2892	37794	1.00	37794	0	4	21
ELASTIC2D	16	42574	566410	0.92	35400	168	4	21
ELASTIC2D	64	166462	2221250	0.88	34707	321	4	21
PREFINE	1	1024	22260	1.00	22260	0	3	288
PREFINE	16	16384	358596	0.55	22412	81	3	288
PREFINE	64	65536	1436036	0.52	22438	141	3	288
SHELL	2	2225	213505	0.59	106752	142	29	284
SHELL	16	14450	1512250	0.42	94515	190	29	284
SHELL	64	68285	7164221	0.48	111940	170	29	284
BCSSTK16	1	4810	290304	1.00	290304	0	14	80
BCSSTK16	16	4810	290304	0.35	18144	54	14	80
BCSSTK16	64	4810	290304	0.04	4536	54	14	80
BCSSTK17	1	10456	428132	1.00	428132	0	2	149
BCSSTK17	16	10456	428132	0.60	26758	141	2	149
BCSSTK17	64	10456	428132	0.24	6689	93	2	149
BCSSTK18	1	11214	148356	1.00	148356	0	1	48
BCSSTK18	16	11214	148256	0.61	9272	321	1	48
BCSSTK18	64	11214	148256	0.30	2318	215	1	48
BCSSTK29	2	13992	619488	0.97	309744	10	4	70
BCSSTK29	16	13992	619488	0.68	38718	154	4	70
BCSSTK29	64	13992	619488	0.25	9679	108	4	70
BCSSTK33	2	8738	591904	0.89	295952	9	19	140
BCSSTK33	16	8738	591904	0.30	36994	95	19	140
BCSSTK33	64	8738	591904	0.03	9248	81	19	140
DT	2	30000	629550	0.98	314775	18	11	50
DT	16	30000	629550	0.87	39346	20	11	50
DT	64	30000	629550	0.70	9836	28	11	50
ELASTIC3D	1	7908	613026	1.00	613026	0	41	419
ELASTIC3D	16	7908	613026	0.17	38314	113	41	419
ELASTIC3D	64	7908	613026	0.01	9578	181	41	419
FDGRID3D	1	32768	460692	1.00	460692	0	3	26
FDGRID3D	16	32768	460692	0.62	28793	10	3	26
FDGRID3D	64	32768	460692	0.36	7198	17	3	26
KALL	1	10556	162774	1.00	162744	0	3	91
KALL	16	10556	162774	0.49	10173	23	3	91
KALL	64	10556	162774	0.48	2542	28	3	91
RAND1	1	1600	41296	1.00	41296	0	11	42
RAND1	16	1600	41296	0.00	2581	296	11	42
RAND1	64	1600	41296	0.00	645	274	11	42
RAND2	1	2304	59614	1.00	59614	0	10	45
RAND2	16	2304	59614	0.00	3725	258	10	45
RAND2	64	2304	59614	0.00	931	242	10	45

TABLE 6
Runtime and coloring statistics (the number of colors required for the global separator is given in parentheses)

Problem	$\|P\|$	JP Colors	PLF Colors	JP Time	PLF(0) Time	PLF(1) Time	PDR(1) Time	PDR(2) Time
CRYSTAL	1	108 (108)	108 (108)	5.35	5.35	0.31	0.31	0.29
CRYSTAL	16	118 (117)	111 (108)	2.49	2.65	1.72	1.78	1.74
CRYSTAL	64	121 (117)	117 (111)	2.54	2.80	1.91	1.96	1.94
ELASTIC2D	1	9 (9)	9 (9)	0.34	0.36	0.16	0.16	0.14
ELASTIC2D	16	10 (9)	10 (10)	0.36	0.39	0.19	0.19	0.17
ELASTIC2D	64	10 (10)	10 (10)	0.41	0.47	0.26	0.25	0.25
PREFINE	1	81 (81)	81 (81)	0.35	0.35	0.09	0.09	0.08
PREFINE	16	81 (81)	81 (81)	0.46	0.46	0.25	0.24	0.19
PREFINE	64	82 (82)	81 (81)	0.52	0.49	0.28	0.28	0.25
SHELL	2	66 (66)	64 (60)	2.08	2.23	0.40	0.42	0.42
SHELL	16	80 (80)	74 (70)	1.98	1.92	0.57	0.61	0.54
SHELL	64	74 (74)	77 (77)	2.36	2.65	0.72	0.72	0.68
BCSSTK16	1	27 (27)	27 (27)	4.62	4.62	0.71	0.71	0.69
BCSSTK16	16	37 (37)	33 (32)	0.26	0.29	0.21	0.21	0.19
BCSSTK16	64	39 (39)	34 (34)	0.11	0.17	0.18	0.17	0.16
BCSSTK17	1	28 (28)	28 (28)	5.83	5.83	1.18	1.18	1.12
BCSSTK17	16	32 (31)	34 (34)	0.46	0.50	0.32	0.32	0.32
BCSSTK17	64	36 (36)	37 (36)	0.11	0.13	0.12	0.13	0.13
BCSSTK18	1	14 (14)	14 (14)	1.48	1.48	0.62	0.62	0.54
BCSSTK18	16	16 (13)	17 (13)	0.13	0.15	0.12	0.11	0.12
BCSSTK18	64	15 (15)	15 (14)	0.06	0.11	0.11	0.11	0.12
BCSSTK29	2	27 (25)	29 (29)	4.44	4.42	0.94	0.94	0.82
BCSSTK29	16	34 (30)	35 (35)	0.54	0.58	0.27	0.27	0.26
BCSSTK29	64	34 (33)	37 (34)	0.17	0.19	0.16	0.18	0.18
BCSSTK33	2	33 (33)	30 (30)	5.33	5.41	0.97	0.97	0.92
BCSSTK33	16	40 (40)	39 (38)	0.64	0.73	0.55	0.57	0.58
BCSSTK33	64	41 (41)	41 (41)	0.21	0.37	0.36	0.38	0.37
DT	2	15 (13)	15 (13)	4.07	4.00	1.28	1.29	1.08
DT	16	15 (13)	15 (15)	0.48	0.48	0.23	0.23	0.20
DT	64	16 (16)	16 (16)	0.15	0.15	0.10	0.10	0.10
ELASTIC3D	1	39 (39)	39 (39)	12.80	12.80	1.52	1.52	1.43
ELASTIC3D	16	44 (43)	41 (41)	0.72	0.77	0.50	0.65	0.66
ELASTIC3D	64	45 (45)	41 (41)	0.32	0.36	0.35	0.53	0.45
FDGRID3D	1	13 (13)	13 (13)	5.82	5.82	1.81	1.81	1.88
FDGRID3D	16	14 (14)	15 (15)	0.42	0.58	0.48	0.40	0.39
FDGRID3D	64	16 (16)	16 (16)	0.19	0.31	0.29	0.28	0.24
KALL	1	9 (9)	9 (9)	1.62	1.62	0.66	0.66	0.66
KALL	16	13 (13)	10 (9)	0.18	0.29	0.22	0.21	0.21
KALL	64	14 (14)	10 (10)	0.08	0.13	0.13	0.13	0.13
RAND1	1	10 (10)	10 (10)	0.49	0.49	0.12	0.12	0.11
RAND1	16	13 (13)	11 (11)	0.23	0.56	0.57	0.59	0.60
RAND1	64	13 (13)	12 (12)	0.11	0.28	0.29	0.29	0.29
RAND2	1	10 (10)	10 (10)	0.73	0.72	0.18	0.18	0.14
RAND2	16	13 (13)	12 (12)	0.32	0.82	0.84	0.90	0.87
RAND2	64	13 (13)	12 (12)	0.14	0.37	0.39	0.44	0.39

TABLE 7
Total imbalance, I_σ, results

Problem	$\|P\|$	PLF(0) I_σ	PLF(1) I_σ	PLF(2) I_σ	PDR(0) I_σ	PDR(1) I_σ	PDR(2) I_σ
CRYSTAL	2	14683	11684	11683	14683	11684	11683
CRYSTAL	16	63054	52703	51917	61457	48288	47963
CRYSTAL	64	71702	39087	35964	70201	45478	43514
ELASTIC2D	2	537	51	19	537	29	17
ELASTIC2D	16	5998	187	182	5851	189	178
ELASTIC2D	64	12655	366	342	11900	393	342
PREFINE	2	11175	355	261	11175	304	233
PREFINE	16	43362	6160	6157	43432	6156	6151
PREFINE	64	47899	9237	9222	47801	8909	8903
SHELL	2	3368	633	648	3368	788	612
SHELL	16	16242	3858	3640	16059	4312	4061
SHELL	64	23606	5225	4763	23400	4229	3888
BCSSTK16	2	2778	769	627	2778	729	708
BCSSTK16	16	4999	1079	944	4780	1235	1069
BCSSTK16	64	4451	2496	2396	4436	2472	2425
BCSSTK17	2	9049	184	195	9049	187	182
BCSSTK17	16	8947	728	597	8972	791	655
BCSSTK17	64	5037	1261	1076	4982	1246	1097
BCSSTK18	2	2660	360	359	2660	361	362
BCSSTK18	16	2766	357	327	2907	378	330
BCSSTK18	64	1744	250	223	1745	273	222
BCSSTK29	2	5430	136	130	5430	150	122
BCSSTK29	16	7395	759	672	7501	736	584
BCSSTK29	64	5704	1306	1123	5765	1529	1159
BCSSTK33	2	5478	1250	1132	5478	740	709
BCSSTK33	16	12147	1186	993	12367	2215	1065
BCSSTK33	64	8352	2687	1879	8086	2875	2153
DT	2	1781	45	45	1781	46	37
DT	16	2036	130	123	1943	128	122
DT	64	1364	198	162	1328	184	158
ELASTIC3D	2	7522	374	351	7522	310	307
ELASTIC3D	16	10416	2648	2179	10523	2672	2740
ELASTIC3D	64	9085	6435	5877	9225	5606	5236
FDGRID3D	2	5707	24	15	5707	19	15
FDGRID3D	16	19686	191	110	20399	202	124
FDGRID3D	64	6674	329	213	7038	267	185
KALL	2	2007	19	13	2007	82	19
KALL	16	2611	67	61	2595	88	54
KALL	64	1218	147	86	1255	138	83
RAND1	2	1285	66	56	1285	184	53
RAND1	16	1643	432	348	1643	513	324
RAND1	64	1120	539	401	1120	574	410
RAND2	2	1899	73	71	1899	169	92
RAND2	16	2123	431	346	2125	525	349
RAND2	64	1300	509	375	1314	516	401