A Parallel Graph Coloring Heuristic

Article *in* SIAM Journal on Scientific and Statistical Computing · June 1992

DOI: 10.1137/0914041 · Source: CiteSeer

3 authors, including:

Mark Jones
Virginia Polytechnic Institute and State University
119 PUBLICATIONS **2,452** CITATIONS

SEE PROFILE

Paul E. Plassmann
Virginia Polytechnic Institute and State University
105 PUBLICATIONS **2,012** CITATIONS

SEE PROFILE

ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

A PARALLEL GRAPH COLORING HEURISTIC*

Mark T. Jones and Paul E. Plassmann

Mathematics and Computer Science Division
Preprint MCS-P246-0691
June 1991
(Revised February 1992)

ABSTRACT

The problem of computing good graph colorings arises in many diverse applications, such as in the estimation of sparse Jacobians and in the development of efficient, parallel iterative methods for solving sparse linear systems. In this paper we present an asynchronous graph coloring heuristic well suited to distributed memory parallel computers. We present experimental results obtained on an Intel iPSC/860 which demonstrate that, for graphs arising from finite element applications, the heuristic exhibits scalable performance and generates colorings usually within three or four colors of the best-known linear time sequential heuristics. For bounded degree graphs, we show that the expected running time of the heuristic under the P-RAM computation model is bounded by $EO(\log(n)/\log\log(n))$. This bound is an improvement over the previously known best upper bound for the expected running time of a random heuristic for the graph coloring problem.

Key words: distributed memory computers, graph coloring heuristics, parallel algorithms, random algorithms, sparse matrices

AMS(MOS) subject classifications: 65F10, 65F50, 65Y05, 68Q22, 68R10

* This work was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

1. Introduction. The determination of the chromatic number of a general graph is a well-known NP-hard problem [4]. However, a number of practical problems require the determination of nearly optimal graph colorings. For example, it has been shown [3] that the problem of directly estimating a sparse Jacobian by finite differences with a minimum number of function evaluations is equivalent to a graph coloring problem. Also, it has been shown [10] that the minimum number of parallel steps in the solution of the triangular systems involving incomplete Cholesky factors can be obtained by a matrix reordering derived from the solution of a graph coloring problem. Thus, the development of an effective parallel heuristic is of great practical, as well as theoretical, interest.

In this paper we present an asynchronous graph coloring heuristic well suited to distributed memory parallel computers. Our heuristic consists of two phases: an initial, parallel phase that uses a random heuristic, followed by a local phase that utilizes one of several good sequential coloring heuristics. The initial phase maintains the good expected running time bounds obtained for a Monte Carlo algorithm for determining a maximal independent set [8]. In fact, for bounded degree graphs, we show that an upper bound for the expected running time of our heuristic under the P-RAM computation model is $EO(\log(n)/\log\log(n))$. This bound is an improvement over the previously known best upper bound of $EO(\log(n))$ for a random heuristic.

To illustrate the performance of this heuristic, we present experimental results obtained on an Intel iPSC/860. The results demonstrate that, for graphs arising from finite element applications, the heuristic exhibits scalable performance. In addition, for these problems the heuristic is shown to generate colorings usually using no more than three or four more colors than the best linear time sequential heuristics.

This paper is organized as follows. In §2 we introduce a new Monte Carlo heuristic that eliminates the need for global synchronization of the processors and allows for the more efficient use of data structures. We prove, in §3, that under the P-RAM computational model the expected running time of the asynchronous heuristic affords an improvement over the upper bound for a Monte Carlo algorithm based on finding maximal independent sets. A distributed memory implementation of the asynchronous heuristic is detailed in §4. In §5 we present and discuss experimental results obtained on the iPSC/860. Finally, in §6 we discuss possible avenues for improving the heuristic.

2. An Asynchronous Parallel Graph Coloring Heuristic. In this section we introduce a graph coloring heuristic suitable for asynchronous parallel machines. First we review the graph coloring problem. Consider the symmetric graph $G = (V, E)$ with vertex set V, with $|V| = n$, and edge set E. We say that the function $\sigma : V \to \{1, \ldots, s\}$ is an s-coloring of G, if $\sigma(v) \neq \sigma(w)$ for all edges $(v, w) \in E$. The minimum possible value for s is known as the *chromatic number*

of G, which we denote as $\chi(G)$.

The question as to whether a general graph G is s-colorable is NP-complete [4]. It is known that unless $P = NP$, there does not exist a polynomial approximation scheme for solving the graph coloring problem [4]. In fact, the best polynomial time heuristic known [6] can theoretically guarantee a coloring of only size $c\,(n/\log n)\,\chi(G)$, where c is some constant.

Given these pessimistic theoretical results, it is quite surprising that, for certain classes of graphs, there exist a number of sequential graph coloring heuristics that are very effective in practice. For graphs arising from a number of applications, it has been demonstrated that these heuristics are often able to find colorings that are within one or two of an optimal coloring [3, 7].

$V' \leftarrow V$;
For $i = 1, \ldots, n$ **do**
    Choose a vertex v_i from V';
    $\sigma(v_i)$ = smallest available consistent color;
    $V' \leftarrow V' \setminus \{v_i\}$;
enddo

FIG. 1. *A sequential greedy coloring heuristic*

It is known that an optimal coloring can be obtained with a greedy heuristic if the vertices are visited in the correct order [1]. The basic structure of the greedy heuristic is shown in Figure 1. The only aspect of the sequential heuristic that must be specified is the rule for choosing the vertex v_i. Many strategies for obtaining this vertex ordering have been proposed. One of the most effective heuristics is the saturation degree ordering (SDO) suggested by Brélaz [2]. The SDO vertex ordering is defined as follows. Suppose that vertices $v_1, \ldots, v_{i-1}$ have been chosen and colored. Vertex v_i is chosen to be a vertex adjacent to the maximum number of different colors in the vertex set $\{v_1, \ldots, v_{i-1}\}$. Note that this heuristic can be implemented to run in time proportional to $\sum_{v \in V} deg^2(v)$, where $deg(v)$ is the degree of vertex v.

A modification of the SDO heuristic is the *incidence degree ordering* (IDO) introduced by Coleman and Moré in their work [3] on using coloring heuristics to obtain consistent partitions for sparse Jacobian estimation. The IDO vertex ordering has the advantage that its running time is a linear function of the number of edges. To describe the IDO heuristic, we again suppose that vertices $v_1, \ldots, v_{i-1}$ have been chosen. Vertex v_i is chosen to be a vertex whose degree is a maximum in the subgraph of G induced by the vertex set $\{v_1, \ldots, v_{i-1}\} \cup \{v_i\}$. This heuristic can be implemented to run in time proportional to $\sum_{v \in V} deg(v)$, or $O(|E|)$ time.

Unfortunately, these heuristics are essentially breadth-first searches of the graph and, as such, do not represent scalable, parallel heuristics. To do better, one notes that if the vertices v and w are not adjacent in G (i.e., are independent in G), then these vertices can be colored in parallel. Thus, a heuristic for determining an independent set in parallel could be adapted to a parallel coloring heuristic. This observation motivates the parallel coloring heuristic shown in Figure 2.

$V' \leftarrow V$;
While $V' \neq \emptyset$ **do**
    Choose an independent set I from V';
    Color I in parallel;
    $V' \leftarrow V' \setminus I$;
enddo

FIG. 2. *Outline of a parallel coloring heuristic*

The problem of determining an independent set in parallel has been the focus of much theoretical research. An approach that has been successfully analyzed is to determine an independent set, I, based on the following Monte Carlo rule. Here we denote the set of vertices adjacent to vertex v by $adj(v)$.

1. For each vertex $v \in V'$ determine a distinct, random number $\rho(v)$.
2. $v \in I \Leftrightarrow \rho(v) > \rho(w)$, $\forall\, w \in adj(v)$.

In the Monte Carlo algorithm described by Luby [8], this initial independent set is augmented to obtain a maximal independent set. The approach is the following. After the initial independent set is found, the set of vertices adjacent to a vertex in I, the neighbor set $N(I)$, is determined. The union of these two sets is deleted from V', the subgraph induced by this smaller set is constructed, and the Monte Carlo step is used to choose an augmenting independent set. This process is repeated until the candidate vertex set is empty and a maximal independent set (MIS) is obtained. The complete Monte Carlo algorithm suggested by Luby for generating an MIS is shown in Figure 3. In this figure we denote by $G(V')$ the subgraph of G induced by the vertex set V'. Luby shows that an upper bound for the expected time to compute an MIS by this algorithm on a CRCW P-RAM is $EO(\log(n))$. The algorithm can be adapted to a graph coloring heuristic by using it to determine a sequence of distinct maximal independent sets and by coloring each MIS a different color. Thus, this approach will solve the $(\Delta + 1)$ vertex coloring problem, where Δ is the maximum degree of G, in expected time $EO((\Delta + 1)\log(n))$.

A major deficiency of this approach on currently available parallel computers is that each new choice of random numbers in the MIS algorithm requires a global synchronization of the processors. A second problem is that each new choice of

$I \leftarrow \emptyset$;
$V' \leftarrow V$;
$G' \leftarrow G$;
While $G' \neq \emptyset$ **do**
    Choose an independent set I' in G';
    $I \leftarrow I \cup I'$;
    $X \leftarrow I' \cup N(I')$;
    $V' \leftarrow V' \setminus X$;
    $G' \leftarrow G(V')$;
enddo

FIG. 3. *Luby's Monte Carlo algorithm for determining a maximal independent set*

random numbers incurs a great deal of computational overhead, because the data structures associated with the random numbers must be recomputed. In Figure 4 we present an asynchronous heuristic that avoids both of these drawbacks. The heuristic is written assuming that each vertex v is assigned to a different processor and the processors communicate by passing messages.

With the asynchronous heuristic the first drawback (global synchronization) is eliminated by choosing the independent random numbers only at the start of the heuristic. With this modification, the interprocessor communication can proceed asynchronously once these numbers are determined. The second drawback (computational overhead) is alleviated because with this heuristic, once a processor knows the values of the random numbers of the vertices to which it is adjacent, the number of messages it needs to wait for can be computed and stored. Likewise, each processor computes only once the processors to which it needs to send a message once its vertex is colored. Finally, note that this heuristic has more of the "flavor" of the sequential heuristic, since we choose the smallest color consistent with the adjacent vertices previously colored.

One should be concerned that the expected running time of this heuristic is comparable to the expected running time of the heuristic based on the MIS Monte Carlo algorithm. In the next section we show that under the P-RAM computational model, one is able to obtain an improvement over the upper bound for the expected running time of the MIS algorithm.

3. Expected Running Time of the Asynchronous Heuristic. In this section we derive an upper bound for the expected running time of the heuristic presented in Figure 4 under the P-RAM computational model. Most of the practical problems we are concerned with are generated from local, physical models. Thus, the maximum degree of the associated graphs is independent of the size of

```
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};
enddo
n-recv = 0;
While (n-recv < n-wait) do
    Receive σ(w);
    n-recv = n-recv + 1;
enddo
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;
enddo
```

FIG. 4. *An asynchronous parallel coloring heuristic*

the system. It is reasonable, therefore, to consider the model problem of a graph G with n vertices and bounded degree Δ. Since we assume that Δ is bounded, the previous work by Luby [8] shows that a random heuristic for the $(\Delta + 1)$ vertex coloring problem can be accomplished on the P-RAM model in $EO(\log(n))$ time. As discussed above, this heuristic is based on a synchronous random algorithm for determining a sequence of maximal independent sets.

To analyze the running time of our heuristic, we make the following observations. First, for the sake of comparison, we note that this heuristic can also be considered to be synchronous. Second, we assume that each vertex $v \in V$ chooses a unique independent random number $\rho(v)$. Define a *monotonic path* of length t to be a path of t vertices $\{v_1, v_2, \ldots, v_t\}$ in G such that $\rho(v_1) > \rho(v_2) > \ldots > \rho(v_t)$. We have the following lemma.

LEMMA 3.1. *The running time of the P-RAM version of the asynchronous heuristic is proportional to the maximum length monotonic path in G.*

Proof: We assume that the asynchronous heuristic can be made synchronous by including a global synchronization point in the receive and send loops (i.e., all processors that have messages to send, send them; all available messages are

received; and then the processors synchronize). Since each vertex has degree at most Δ, the number of messages to be received and sent by each processor is bounded by this constant. Under the P-RAM computational model we can assume that messages are sent between processors in constant time. Thus, the running time of the heuristic is proportional to the number of these synchronized steps.

It is clear that the number of steps is at least as long as the longest monotonic path in G. If the number of steps were longer, there would exist some step where the random number of a processor was greater than all its neighbors, yet for some reason it did not send its messages. Hence, we have that the running time of the heuristic is proportional to the maximum length monotonic path. □

To analyze the expected running time of the heuristic, we need to construct an upper bound to the probability of the existence of a monotonic path. We construct this bound in two parts, first by finding a bound on the number of paths of length t, and then by determining the probability that a path is monotonic. The following lemma gives a bound on the number of paths of length t in the graph G.

LEMMA 3.2. *The number, $\eta(t)$, of different paths in G of length t is bounded by*

$$\eta(t) \leq n\Delta(\Delta-1)^{t-2} \quad . \tag{3.1}$$

Proof: This bound can be obtained by induction. For $t = 2$, the number of paths is at most $n\Delta$. Now suppose the lemma holds for paths of length $t-1$. Consider some vertex v and all paths of length $t-1$ starting from this vertex. Each of these paths ends at some vertex w. Because an extension of this path cannot return along the edge it used to get to w, there are at most $\Delta - 1$ ways to extend this path. Multiplying the bound for the number of paths of length $t-1$ by $\Delta - 1$ yields the desired result. □

Let X be the random variable equal to the maximum length monotonic path in G. Since the random numbers assigned to the vertices are independent, the probability that a path of length t is monotonic is $(1/t!)$. Let $P\{X = t\}$ be the probability that the maximum length monotonic path is t. This probability is bounded by the probability that there exists a monotonic path of length t. Including the bound given in Lemma 3.2, we find

$$\begin{aligned} P\{X = t\} &\leq \frac{\eta(t)}{t!} \\ &\leq \frac{n\Delta(\Delta-1)^{t-2}}{t!} \quad . \end{aligned} \tag{3.2}$$

THEOREM 3.3. *The expected value of the maximum length monotonic path, EX, is bounded by*

$$EX \leq T + (\Delta-1)^K \tag{3.3}$$

for any K, where T is the minimum integer satisfying

$$T! \geq n\Delta(\Delta-1)^{T-K-1}\exp(\Delta-1)\ . \tag{3.4}$$

Proof: The expected value of X is given by

$$EX = \sum_{t=2}^{n} tP\{X=t\}\ . \tag{3.5}$$

For any integer $T \geq 2$ we have that

$$EX \leq T + \sum_{t=T+1}^{n} tP\{X=t\}\ . \tag{3.6}$$

Thus, we can include the bound on the probability given in equation (3.2) to obtain

$$\begin{aligned} EX &\leq T + \sum_{t=T+1}^{\infty} t\frac{n\Delta(\Delta-1)^{t-2}}{t!} \\ &\leq T + \frac{n\Delta}{(\Delta-1)}\sum_{t=T}^{\infty}\frac{(\Delta-1)^t}{t!} \\ &\leq T + \frac{n\Delta}{(\Delta-1)}\frac{(\Delta-1)^T}{T!}\exp(\Delta-1)\ . \end{aligned} \tag{3.7}$$

If we choose T to be the minimum integer such that

$$T! \geq n\Delta(\Delta-1)^{T-K-1}\exp(\Delta-1)\ , \tag{3.8}$$

we have that the expected maximum length monotonic path is bounded by

$$EX \leq T + (\Delta-1)^K\ . \tag{3.9}$$

□

As a corollary, we are able to achieve a bound in terms of n for the expected running time of this algorithm. To prove this corollary, we require the following short lemma.

LEMMA 3.4. *The inequality*

$$\frac{r!}{s^r} \geq \sqrt{2\pi s}\left(\frac{\Gamma(r/s)}{e^{1/12}\sqrt{2\pi}}\right)^s \tag{3.10}$$

holds for $r \geq s \geq 1$.

Proof: First, we recall the Gamma function identity

$$r! = r\,\Gamma(r) \tag{3.11}$$

and the formula

$$\Gamma(x) = \sqrt{2\pi}\; x^{x-\frac{1}{2}}\; e^{-x}\; e^{\frac{\theta(x)}{12}} \tag{3.12}$$

which holds for $x \geq 1$ and for $0 < \theta(x) < 1$. For $r \geq 1$, we compute the lower bound

$$\Gamma(r) \geq \sqrt{\frac{2\pi}{r}} \left(\frac{r}{e}\right)^r \tag{3.13}$$

by setting $\theta = 0$ in equation 3.12. Thus, we have the bound

$$\frac{r!}{s^r} \geq \sqrt{2\pi r} \left(\frac{r}{se}\right)^r \quad . \tag{3.14}$$

We set $\theta = 1$ in equation 3.12 to obtain an upper bound for the Gamma function and assume that $r \geq s \geq 1$. We let $r' = r/s$ and find

$$\begin{aligned}
\sqrt{2\pi r} \left(\frac{r}{se}\right)^r &= \sqrt{2\pi r} \left(\left(\frac{r'}{e}\right)^{r'}\right)^s \\
&= \sqrt{2\pi r} \left(e^{\frac{1}{12}} \sqrt{\frac{2\pi}{r'}} \left(\frac{r'}{e}\right)^{r'} e^{-\frac{1}{12}} \sqrt{\frac{r'}{2\pi}}\right)^s \\
&\geq \sqrt{2\pi r} \left(\Gamma(r') e^{-\frac{1}{12}} \sqrt{\frac{r'}{2\pi}}\right)^s \\
&\geq \sqrt{2\pi s} \left(\Gamma(r') \frac{e^{-\frac{1}{12}}}{\sqrt{2\pi}}\right)^s \quad .
\end{aligned} \tag{3.15}$$

Combining equations 3.14 and 3.15, we obtain the desired bound

$$\frac{r!}{s^r} \geq \sqrt{2\pi s} \left(\frac{\Gamma(r/s)}{e^{\frac{1}{12}}\sqrt{2\pi}}\right)^s \quad . \quad \square \tag{3.16}$$

The Gamma function is not monotonic for $x \geq 1$. However, it is monotonic for slightly larger x, for example, $x \geq 3/2$. To avoid this lack of uniqueness, we define the function $\Gamma^+(y) = \max\{\, x \,|\, \Gamma(x) = y \,\}$, which is well defined for $y \geq 1$. We now prove the following corollary to Theorem 3.3.

COROLLARY 3.5. *For $\Delta \geq 2$, the expected number of steps, EX, for the random heuristic is bounded by*

$$EX \leq (\Delta - 1)\,\Gamma^{+}\left(\sqrt{2\pi}\,e^{\frac{13}{12}}\left(\frac{n\Delta}{\sqrt{2\pi}\,(\Delta-1)^{\frac{3}{2}}}\right)^{\frac{1}{\Delta-1}}\right) + 2\ . \tag{3.17}$$

Proof: Choosing $K = 0$ in Theorem 3.3 and subtracting one from T in equation 3.4, we have the following inequality:

$$\frac{(T-1)!}{(\Delta-1)^{T-1}} \leq \frac{n\Delta}{(\Delta-1)}\exp(\Delta-1)\ . \tag{3.18}$$

We assume that $(T-1) \geq (\Delta-1) \geq 1$, and by Lemma 3.4 we have

$$\sqrt{2\pi(\Delta-1)}\left(\frac{e^{-\frac{1}{12}}}{\sqrt{2\pi}}\Gamma\left(\frac{T-1}{\Delta-1}\right)\right)^{\Delta-1} \leq \frac{(T-1)!}{(\Delta-1)^{T-1}}\ . \tag{3.19}$$

Combining the bounds in equations 3.18 and 3.19, we obtain

$$\Gamma\left(\frac{T-1}{\Delta-1}\right) \leq \sqrt{2\pi}e^{\frac{13}{12}}\left(\frac{n\Delta}{\sqrt{2\pi}\,(\Delta-1)^{\frac{3}{2}}}\right)^{\frac{1}{\Delta-1}}\ . \tag{3.20}$$

Using the asymptotic inverse to the Gamma function defined above, we obtain the bound

$$T-1 \ \leq\ (\Delta-1)\,\Gamma^{+}\left(\sqrt{2\pi}e^{\frac{13}{12}}\left(\frac{n\Delta}{\sqrt{2\pi}\,(\Delta-1)^{\frac{3}{2}}}\right)^{\frac{1}{\Delta-1}}\right)\ . \tag{3.21}$$

Because we have chosen $K = 0$, by Theorem 3.3 we have that $EX \leq T + 1$. Thus, consistent with our original assumption that $T \geq \Delta$, we have the desired bound for EX,

$$EX \leq (\Delta-1)\,\Gamma^{+}\left(\sqrt{2\pi}e^{13/12}\left(\frac{n\Delta}{\sqrt{2\pi}\,(\Delta-1)^{\frac{3}{2}}}\right)^{\frac{1}{\Delta-1}}\right) + 2\ . \quad \square \tag{3.22}$$

By using the lower bound for the Gamma function obtained by choosing $\theta(x) = 0$ in equation 3.12 we note that, for fixed Δ, this bound is asymptotically $EO(\log(n)/\log\log(n))$. This bound is an improvement over the $EO(\log(n))$ upper bound obtained by Luby [8].

The bound given in Theorem 3.3 yields a surprisingly close fit to what we have observed in practice. In Figure 5 we compare the bound for EX with our experimental results for regular graphs of degree 4. In the plot the open circles are

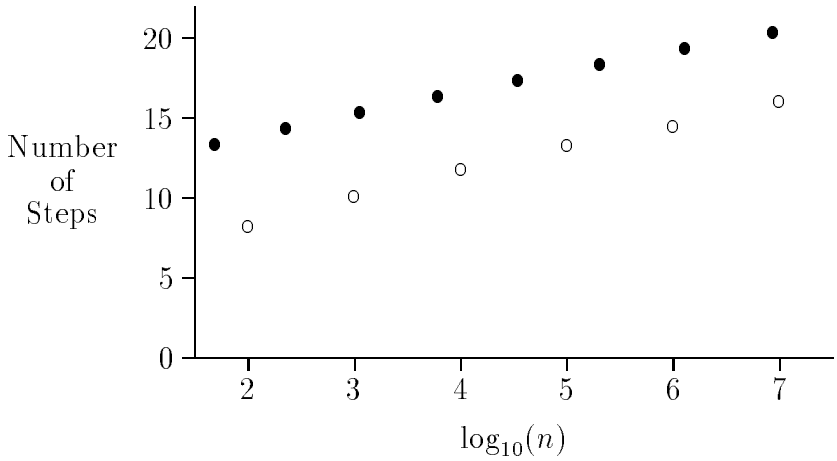

FIG. 5. *Bound obtained by choosing* $K = -1$ *for the expected value* ($\bullet$) *versus experimental average* ($\circ$) *for regular,* $\Delta = 4$*, graphs*

the observed average number of steps for the asynchronous heuristic as a function of the base 10 logarithm of n. The closed circles are obtained from the bound given in Theorem 3.3 where we have chosen $K = -1$. The points are obtained by choosing a value for T and then solving equation 3.4 for the largest n that satisfies the inequality.

As a final note, we emphasize that, although the heuristics described above have a random component, their behavior in practice is essentially deterministic. In the above analysis, note that the probability that there exists a monotonic path of length greater than t asymptotically decays faster than exponentially. Thus, the bounds on the expected running time hold with very high probability. In addition, Luby [8] gives a prescription for converting his Monte Carlo MIS algorithm into a deterministic algorithm with the same running time. Hence, these heuristics are fundamentally different from those based on simulated annealing. Although the simulated annealing algorithms can be shown to ultimately obtain optimal solutions, running time bounds comparable to those above do not exist.

4. A Medium Grain Heuristic for Distributed Memory Computers. Our primary interest is the development of a heuristic suitable for distributed memory computers. In this section we describe how the asynchronous Monte Carlo heuristic presented in the preceding section can be combined with the heuristics that have been successful on sequential machines. Using this approach, in the next section we experimentally demonstrate that for certain classes of problems the performance is scalable.

Consider a distributed memory computer with p processors. We assume that the vertices of the graph $G = (V, E)$ are partitioned across these processors by the sets $\{V_1, \ldots, V_p\}$. Let the function $\pi : V \to \{1, \ldots, p\}$ return the number of the partition, or processor, to which each vertex is assigned. We define the *edge*

separator E^S to be the set of edges $E^S \subseteq E$ where the edge $(v,w) \in E^S \Leftrightarrow \pi(v) \neq \pi(w)$. In addition, we define the set of *global vertices* to be the vertex set V^S, where a vertex is in this set if and only if the vertex is an endpoint for some edge in E^S. Let the set of *local vertices*, V^L, be the set $V \setminus V^S$. Finally, denote by V_i^S and V_i^L the vertex sets $V^S \cap V_i$ and $V^L \cap V_i$.

The following theorem shows that it is possible to decompose the asynchronous heuristic into two parts, the first part to color the global vertices, and the second part to color the local vertices. We show that the vertex labeling obtained by piecing together these colorings is a coloring for G. In this theorem we denote the subgraph of G induced by the vertex set V_i by $G(V_i)$.

THEOREM 4.1. *Let σ_S be a coloring for $G(V^S)$. This coloring, restricted to V_i^S, can be independently extended to a coloring σ_i for the subgraph $G(V_i)$. If we define the function σ by $\sigma(v) = \sigma_i(v)$ where $v \in V_i$, then σ is a coloring for G.*

Proof: Consider the vertices V_i on processor i. We assume that vertices V_i^S are consistently colored when the random heuristic colors $G(V^S)$. Thus, only the vertices V_i^L remain to be colored on this processor. By definition, the vertices V_i^L can be connected only to vertices in V_i. Because V_i^S has been colored, the vertices V_i^L may be colored independently from any other vertices in V. By the same observation, we note that if the coloring chosen for each V_i^L is consistent for $G(V_i)$, then these colorings combine to form a consistent coloring for the entire graph. □

From Theorem 4.1, we observe that the parallel graph coloring problem can be accomplished in two phases:

1. Color $G(V^S)$ using the asynchronous Monte Carlo heuristic.
2. On processor i, color $G(V_i^L)$ given $\sigma_S(V_i^S)$ using a sequential heuristic.

A subtle point is that we need the Monte Carlo algorithm to generate independent sets in the graph $G_S = (V^S, E^S)$, not the graph $G(V^S)$. Note that G_S is a sparser graph than $G(V^S)$, since G_S does not contain edges (v,w) where $v,w \in V^S$ but $\pi(v) = \pi(w)$. Such edges are included in $G(V^S)$. We use the notation $\Delta_S = \Delta(G_S)$ and $n_S = |V^S|$. Thus, we have $\Delta_S \leq \Delta(G(V^S))$, and the bounds detailed in Theorem 3.3 depend on the values of Δ_S and n_S.

In Figure 6 we outline the complete distributed heuristic to be executed by the i-th processor. The heuristic calls two procedures: *Seq-color* and *Pack-and-send*. Given a partial coloring of vertices stored in the array σ and a list *queue* of local vertices to be colored, *Seq-color*(σ, *queue*) colors these vertices with a sequential heuristic (such as the IDO heuristic discussed earlier). The procedure *Pack-and-send* sends the vertices in *queue* and their colors σ to nonlocal, adjacent vertices on other processors with lower random numbers. For vertex v this set is stored in the array *send-queue*(v) which is initialized at the beginning of the heuristic.

The ability to pack vertex information into messages allows for the optimization of interprocessor communication. For example, messages sent between pro-

```
Determine V_i^S, V_i^L;      {Partition vertices}
color-queue = ∅;
For each v ∈ V_i^S do    {Set up queues for separator vertices}
    n-wait(v) = 0;
    send-queue(v) = ∅;
    For each edge (v, w) ∈ E^S 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};
    enddo
    if (n-wait(v) = 0) then
        color-queue ← color-queue ∪ {v};
enddo
Seq-color ( σ, color-queue );    {Color any vertices in V_i^S not}
n-colored = | color-queue |;        {waiting for messages}
Pack-and-send (σ, color-queue, send-queue );
color-queue = ∅;
While (n-colored < |V_i^S|) 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};
        enddo
    enddo
    Seq-color (σ, color-queue);    {Color subsets of V_i^S once required}
    n-colored = n-colored + | color-queue |;    {messages are received}
    Pack-and-send (σ,color-queue, send-queue);
    color-queue = ∅;
enddo
Seq-color (σ, V_i^L );      {Color local vertices last}
```

FIG. 6. *A distributed memory parallel coloring heuristic for the i-th processor*

cessors can be packed to overcome the high message startup cost on machines like the Intel iPSC/860. The data structure *msg* that a processor receives contains a packed list of vertices, their colors, and the vertices assigned to the receiving processor to which they are adjacent. As before, the number of nonlocal vertices that must be colored before vertex v is computed at the beginning of the heuristic and stored in $n\text{-}wait(v)$.

One last optimization to note is that if the pseudo-random number generator used to determine $\rho(v)$ depends only on the vertex number, then these values do not need to be sent between processors. Instead, each processor can determine these values locally, and the overhead involved with this interprocessor communication can be avoided. This optimization is included in the initialization section of Figure 6.

5. Experimental Results. We have implemented the heuristic described in Figure 6 in the C programming language on a 64-node Intel iPSC/860. In this section we present results obtained with this implementation. One of our main objectives is to demonstrate the scalability of this heuristic consistent with the definition given in [5]. Thus, we would like to show that, for a fixed number of vertices per processor, the running time of the heuristic is only a slowly increasing function of the number of processors used.[1]

To achieve this objective, we have chosen test problems whose size can be easily scaled and are also representative of problems encountered in applications. The problems we consider are generated from finite element models of structures and from finite difference schemes for two and three dimensional regular domains.

We consider two sets of structures problems; both are modeled by using three-dimensional, hexagonal linear elements, where the nonzero structure of the resulting assembled sparse system is used as a test matrix. The problems in Problem Set I are obtained from a model of a long rectangular beam of varying lengths, seven by seven finite elements thick, with the degrees of freedom constrained at both ends. The problems in the Problem Set II are generated from a model of a multistoried building of varying heights with constraints applied by elimination of the bottom layer of vertices.

For these two problem sets the vertex to processor assignment was made by assigning to each processor contiguously numbered blocks of columns based on an initial numbering. These blocks consist of n/p columns, where n is the order of the matrix and p is the number of processors. The initial numbering of the columns is chosen such that nearby nodes in the finite element models are generally close in number. Thus, this matrix partition scheme roughly corresponded to a physical partition.

[1] In our case, the running time will increase with problem size according to the *slowly* growing function given in Theorem 3.3.

TABLE 1
Problem Set I

Problem	n	m	Δ	χ_{IDO}	χ_{SDO}
CUBE1	1,701	46,623	72	21	18
CUBE2	3,888	113,304	72	20	18
CUBE4	8,262	246,666	72	20	19
CUBE8	17,010	513,390	72	21	19
CUBE16	34,506	1,046,838	72	21	19
CUBE32	69,498	2,113,734	72	21	19
CUBE64	139,482	4,247,526	72	21	19

TABLE 2
Problem Set II

Problem	n	m	Δ	χ_{IDO}	χ_{SDO}
SKY1	6,270	145,554	75	19	19
SKY2	12,540	298,751	76	21	20
SKY4	25,080	605,145	76	21	21
SKY8	50,160	1,217,933	76	21	21
SKY16	100,320	2,443,509	76	21	21
SKY32	200,640	4,894,661	76	22	22

In Table 1 and Table 2 we show the sizes of the problems contained in these two sets. The number of vertices in the graphs is listed in column labeled n, the number of edges is listed under m, and the maximum degree of the graph is shown under Δ. The number of colors used by a *sequential* implementation of the incidence degree ordering (IDO) heuristic and the saturation degree ordering (SDO) heuristic is given in the columns labeled by χ_{IDO} and χ_{SDO}.

We also consider two sets of problems arising from standard finite differencing schemes for regular domains. In Table 3 we show the sizes of the test problems generated for the nine point stencil on a square, two dimensional domain. For these problems the domain is partioned into subsquares of equal size, resulting in equal numbers of vertices being assigned to each processor. To keep the aspect ratio of the subsquares the same as the problem is scaled, we change the size of the problems by a factor of four as the problem is scaled. In Table 4 we show the sizes of the test problems generated for the twenty-seven point stencil on a cubic, three dimensional domain. Again, equal numbers of vertices are assigned to each processor because the domain is partioned into subcubes of equal size. For these problems the problem size increases by a factor of eight as the problem is scaled.

Scaling results obtained for Problem Sets I through IV are shown in Table 5 to Table 8. For the results presented in these four tables, the partitioning ensures

TABLE 3
Problem Set III

Problem	n	m	Δ	χ_{IDO}	χ_{SDO}
9PT1	2,500	19,404	8	5	4
9PT4	10,000	78,804	8	5	4
9PT16	40,000	317,604	8	5	4
9PT64	160,000	1,275,204	8	5	4

TABLE 4
Problem Set IV

Problem	n	m	Δ	χ_{IDO}	χ_{SDO}
27PT1	2,197	48,456	26	12	11
27PT8	17,576	421,400	26	12	12
27PT64	140,608	3,511,656	26	13	12

that the average number of vertices per processor, $< n >$, is essentially constant. The number of processors used is listed in the column labeled p. The number of vertices and maximum degree of G_S are given under n_S and Δ_S, respectively. The maximum time in seconds used by a processor in coloring G_S is given under T_S. T_L is the maximum time in seconds used by a processor to solve its local coloring problem. The average number of messages sent by the processors is listed under $< N_{\text{msg}} >$. Also shown are $\tilde{\chi}_S$, the number of colors used in coloring G_S, and $\tilde{\chi}$, the number of colors used to color the entire graph. For these results the IDO heuristic is used to solve the local coloring problems.

Note that although we used the incidence degree heuristic to solve the local coloring problem, for G_L, for the results presented in Tables 5 through 8, one could also employ the more expensive saturation degree heuristic. Recall that the SDO heuristic requires the colors used to color adjacent vertices to compute the saturation degree of each vertex. This information has already been communicated to each processor prior to the coloring of G_L; therefore, the SDO heuristic can be used to color G_L without necessitating any additional interprocessor communication. In Table 9 through Table 12 we present the results for the parallel heuristic modified in this manner.

The results shown in Table 5 through Table 12 demonstrate the scalable performance of the heuristic: for a fixed number of nonzeros per processor, the time required by the global and local phases is essentially constant [5]. Note that as the size of G_S increases, the average number of messages sent per processor gradually increases. By maintaining a reasonable average message size, the high communication overhead on the iPSC/860 can be partially amortized. Also, note that by using the SDO heuristic to solve the local coloring problem, a slight improvement

TABLE 5
Parallel coloring results for Problem Set I, IDO used to solve local problem

Problem	p	$<n>$	n_S	Δ_S	T_S	T_L	$<N_{\text{msg}}>$	$\tilde{\chi}_S$	$\tilde{\chi}$
CUBE1	1	1701	0	0	0.000	0.330	0.0	0	21
CUBE2	2	1944	486	27	0.076	0.368	6.5	14	21
CUBE4	4	2091	2,136	66	0.273	0.379	20.2	24	26
CUBE8	8	2126	5,436	66	0.541	0.376	28.6	24	25
CUBE16	16	2157	11,975	69	0.531	0.375	36.1	25	26
CUBE32	32	2172	25,004	70	0.488	0.378	37.8	27	27
CUBE64	64	2179	51,031	70	0.588	0.381	39.0	26	26

TABLE 6
Parallel coloring results for Problem Set II, IDO used to solve local problem

Problem	p	$<n>$	n_S	Δ_S	T_S	T_L	$<N_{\text{msg}}>$	$\tilde{\chi}_S$	$\tilde{\chi}$
SKY1	2	3,135	1,518	59	0.226	0.417	31.5	21	23
SKY2	4	3,135	3,624	65	0.408	0.408	118.2	23	24
SKY4	8	3,135	7,836	65	0.572	0.420	152.2	22	24
SKY8	16	3,135	16,260	65	0.584	0.412	166.8	23	25
SKY16	32	3,135	33,108	65	0.582	0.410	174.1	24	25
SKY32	64	3,135	66,804	65	0.583	0.408	177.4	25	26

TABLE 7
Parallel coloring results for Problem Set III, IDO used to solve local problem

Problem	p	$<n>$	n_S	Δ_S	T_S	T_L	$<N_{\text{msg}}>$	$\tilde{\chi}_S$	$\tilde{\chi}$
9PT1	1	2,500	0	0	0.000	0.084	0.0	0	5
9PT4	4	2,500	396	5	0.015	0.089	3.2	4	7
9PT16	16	2,500	2,364	5	0.017	0.090	5.1	5	7
9PT64	64	2,500	11,004	5	0.028	0.089	5.1	5	7

TABLE 8
Parallel coloring results for Problem Set IV, IDO used to solve local problem

Problem	p	$<n>$	n_S	Δ_S	T_S	T_L	$<N_{\text{msg}}>$	$\tilde{\chi}_S$	$\tilde{\chi}$
27PT1	1	2,197	0	0	0.000	0.183	0.0	0	12
27PT8	8	2,197	3,752	19	0.124	0.179	82.4	14	15
27PT64	64	2,197	43,272	19	0.270	0.176	179.9	15	17

TABLE 9
Parallel coloring results for Problem Set I, SDO used to solve local problem

Problem	p	$<n>$	n_S	Δ_S	T_S	T_L	$<N_{\text{msg}}>$	$\tilde{\chi}_S$	$\tilde{\chi}$
CUBE1	1	1701	0	0	0.000	6.167	0.0	0	18
CUBE2	2	1944	486	27	0.077	7.630	6.5	14	18
CUBE4	4	2091	2,136	66	0.274	7.675	20.2	24	25
CUBE8	8	2126	5,436	66	0.550	7.722	28.6	24	25
CUBE16	16	2157	11,975	69	0.536	7.577	36.1	25	25
CUBE32	32	2172	25,004	70	0.498	7.771	37.8	27	27
CUBE64	64	2179	51,031	70	0.593	7.732	39.0	26	26

TABLE 10
Parallel coloring results for Problem Set II, SDO used to solve local problem

Problem	p	$<n>$	n_S	Δ_S	T_S	T_L	$<N_{\text{msg}}>$	$\tilde{\chi}_S$	$\tilde{\chi}$
SKY1	2	3,135	1,518	59	0.233	6.361	31.5	21	22
SKY2	4	3,135	3,624	65	0.415	6.130	118.2	23	24
SKY4	8	3,135	7,836	65	0.578	6.357	152.4	22	23
SKY8	16	3,135	16,260	65	0.584	6.097	166.9	23	24
SKY16	32	3,135	33,108	65	0.585	6.141	174.2	24	25
SKY32	64	3,135	66,804	65	0.588	6.018	177.4	24	25

TABLE 11
Parallel coloring results for Problem Set III, SDO used to solve local problem

Problem	p	$<n>$	n_S	Δ_S	T_S	T_L	$<N_{\text{msg}}>$	$\tilde{\chi}_S$	$\tilde{\chi}$
9PT1	1	2,500	0	0	0.000	0.364	0.0	0	4
9PT4	4	2,500	396	5	0.015	0.407	3.2	4	6
9PT16	16	2,500	2,364	5	0.025	0.405	5.1	5	7
9PT64	64	2,500	11,004	5	0.028	0.403	5.6	5	7

TABLE 12
Parallel coloring results for Problem Set IV, SDO used to solve local problem

Problem	p	$<n>$	n_S	Δ_S	T_S	T_L	$<N_{\text{msg}}>$	$\tilde{\chi}_S$	$\tilde{\chi}$
27PT1	1	2,197	0	0	0.000	1.666	0.0	0	11
27PT8	8	2,197	3,752	19	0.126	1.540	82.4	14	15
27PT64	64	2,197	43,272	19	0.275	1.618	179.3	15	16

TABLE 13
Parallel coloring results for Problem Set I, $p = 32$

Problem	$<n>$	n_S	Δ_S	T_S	T_L	$<N_{\text{msg}}>$	$\tilde{\chi}_S$	$\tilde{\chi}$
CUBE1	53	1,701	71	0.061	0.003	6.4	24	26
CUBE2	122	3,888	70	0.127	0.009	16.1	25	27
CUBE4	258	8,217	70	0.202	0.020	32.8	25	28
CUBE8	532	15,561	70	0.309	0.056	56.5	26	29
CUBE16	1,078	28,842	69	0.385	0.138	82.5	28	30
CUBE32	2,172	25,004	70	0.488	0.378	37.8	27	27
CUBE64	4,359	25,166	70	0.573	0.853	38.7	25	27

TABLE 14
Parallel coloring results for Problem Set II, $p = 32$

Problem	$<n>$	n_S	Δ_S	T_S	T_L	$<N_{\text{msg}}>$	$\tilde{\chi}_S$	$\tilde{\chi}$
SKY1	196	6,270	64	0.128	0.011	23.0	23	25
SKY2	392	11,313	65	0.212	0.043	45.7	25	26
SKY4	784	18,012	65	0.396	0.081	74.3	24	25
SKY8	1,568	22,716	65	0.432	0.199	104.4	24	25
SKY16	3,135	33,108	65	0.582	0.410	174.1	24	25
SKY32	6,270	28,272	24	0.491	0.988	106.3	21	24

in the total number of colors can be obtained. However, the saturation degree ordering heuristic is significantly more expensive than the incidence degree ordering heuristic in solving the local coloring problem.

In Table 13 and Table 14 we fix the number of processors at 32 and examine the effect on the performance of the heuristic by varying the number of nonzeros per processor. For these results we use the IDO heuristic to solve the local coloring problem.

Overall, the number of colors required is relatively constant, even though the percentage of the vertices in G_S varies dramatically. To some extent this effect can be explained by noting that even though the relative size of G_S is increasing, the local structure of the separators is essentially the same, since the separators arise from physical partitions of a regular domain. In Table 14, when the relative size of G_S does became small enough to allow Δ_S to decrease, the number of colors used to color G_S, $\tilde{\chi}_S$, decreased. Finally, we note the good performance of the heuristic, both in terms of the number of colors used and execution time, as the size of the local problems becomes quite small.

6. Concluding Remarks. We have presented a new parallel graph coloring heuristic well suited to distributed memory computers. Experimental results demonstrate that this heuristic is scalable and that it produces colorings usually

requiring no more than three or four more colors than the best-known linear time sequential heuristics. We have also shown that under the P-RAM computational model, this heuristic has a expected run time bounded by $EO(\log(n)/\log\log(n))$.

This parallel heuristic takes full advantage of locality in the generation of the graph. For example, if the graph is generated by the assembly of a structures model or obtained from the spatial decomposition of a physical model, the asynchronous random phase of the heuristic can efficiently color the global separator. After the separator is colored, the remaining problem decomposes into independent local coloring problems. The only constraint on these local colorings is that they be consistent with the coloring determined for the separator. Thus, any sequential heuristic can be used to solve each of these local coloring problems simultaneously.

For many problems a physical partition can be used to generate a good vertex to processor assignment. When the determination of a partition is not straightforward, a partitioning heuristic would have to be used. For example, recent advances in the automatic partitioning of three dimensional domains [11] or in spectral dissection methods [9] could be employed. We note that a partitioning that maintains locality is advantageous, although not essential, to the performance of the parallel heuristic. The heuristic requires only that the number of vertices assigned per processor allow for good load balancing.

An interesting observation is that even if the coloring obtained for the separator uses more colors than a good sequential heuristic, the separator subgraph is usually sparser than the entire graph. Thus, when coloring the denser local subgraphs, some of the difference between the parallel and sequential heuristics in the number of colors used for the separator subgraph can be offset by the use of a good sequential heuristic to color the remaining local subgraphs.

Finally, motivated by the following observation, we note a possible avenue for improving the heuristic. When one observes the distribution of colors produced by the heuristic, one often sees very few vertices using the highest colors. For example, when coloring the graph 27PT8 on 8 processors, the results in Table 12 show that 15 colors were required by the parallel algorithm, but only 12 by the sequential algorithm. However, the number of vertices using the colors 15, 14, and 13 were 2, 12, and 70, respectively. An interesting topic for further research might be the introduction of a postprocessing step that would attempt to recolor these few vertices with lower color values, and thus decrease the total number of colors used.

Acknowledgment. We thank Tim Kiemel for help in the probabilistic analysis of the expected maximum length monotone path, and John Gilbert for bringing the paper by Michael Luby to our attention. We also thank the referees for detailed and helpful comments.

REFERENCES

[1] B. Bollobás, *Graph Theory*, Springer-Verlag, New York, 1979.

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

[3] 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.

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

[5] J. L. Gustafson, G. R. Montry, and R. E. Benner, *Development of parallel methods for a 1024-processor hypercube*, SIAM Journal on Scientific and Statistical Computing, 9 (1988), pp. 609–638.

[6] D. S. Johnson, *Worst case behavior of graph coloring algorithms*, in Proceedings 5th Southeastern Conference on Combinatorics, Graph Theory, and Computing, Utilitas Mathematica Publishing, Winnipeg, 1974, pp. 513–527.

[7] M. T. Jones and P. E. Plassmann, *Scalable iterative solution of sparse linear systems*, Preprint MCS-P277-1191, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., 1991.

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

[9] 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.

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

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