Constructing a Maximal Independent Set in Parallel

Mark Goldberg (†), *Thomas Spencer*

Department of Computer Science
Rensselaer Polytechnic Institute

ABSTRACT

The problem of constructing in parallel a maximal independent set of a given graph is considered. A new deterministic *NC*-algorithm implemented in the *EREW PRAM* model is presented. On graphs with n vertices and m edges, it uses $O((n+m)/\log n)$ processors and runs in $O(\log^3 n)$ time. This reduces by a factor of $\log n$ both the running time and the processor count of the previously fastest deterministic algorithm which solves the problem using a linear number of processors.

Key words: parallel computation, *NC*, graph, maximal independent set, deterministic.

1. Introduction

The problem of constructing in parallel a maximal independent set of a given graph, *MIS*, has been investigated in several recent papers. Karp and Wigderson proved in [KW] that the problem is in *NC*. Their algorithm finds a maximal independent set of an n-vertex graph in $O(\log^4 n)$ time and uses $O(n^3/\log^3 n)$ processors. In successive papers, the authors proposed algorithms which either are faster, or use a smaller number of processors. Luby in [L1] and Alon et al in [ABI] presented probabilistic algorithms running in $O(\log^2 n)$ time on a *EREW PRAM* with a linear number of processors. Luby also described a technique for converting probabilistic algorithms into deterministic ones; the technique preserves the running time but requires an increase in the number of processors used to $O(n^2 m)$, where m is the number of edges in the graph. The first deterministic *NC*-algorithm on a linear number of processors (*EREW* model) was constructed in [GS]; its running time is $O(\log^4 n)$. Recently, Luby [L2] proposed a general method for converting randomized parallel algorithms into deterministic ones, which does not require an increase in the number of processors. In the case of *MIS*, the method yields a new algorithm running on a linear number of processors in polylogarithmic time; however, it runs

(†) supported in part by the National Science Foundation under grant DCR-8520872

slower than than that in [GS]. All *NC*-algorithms for *MIS* mentioned above use the following top-level design proposed in [KW]:

> Start with an empty independent set I. Find an independent set I', add it to I, and delete I' and the vertices adjacent to a vertex in I' from the graph. Repeat the previous step until the graph is empty.

Call *FINDSET* the procedure which constructs I'. One can easily prove that an algorithm with such a structure belongs to *NC* if *FINDSET* is designed so that, on any n-vertex graph ($n>0$), it runs in polylogarithmic time and delivers an independent set C such that $|C \cup N(C)| = \Omega(n/\log^s n)$ for some fixed $s \geq 0$; ($N(C)$ is the set of neighbors of C).

In this paper, we present a new deterministic algorithm for *MIS* which improves the running time of the algorithm in [GS] by a factor of $\log n$ and simultaneously reduces, also by a factor of $\log n$, the number of processors used. The algorithm is implemented in the *EREW* model of computation. Thus, the processor-time product of the algorithm in [GS] is improved by $\log^2 n$. These gains are due to the new version of the *FINDSET* procedure. The new procedure finds in $O(\log^2 n)$ time an independent set I such that removing $I \cup N(I)$ reduces the size of the graph by half. Thus, the procedure *FINDSET* needs to be called only $O(\log n)$ times.

To reduce the number of processors used, we modify the definition of the size of the graph so that it takes into account the number of edges deleted. The result of this modification is that the processor-reduction technique of Miller and Reif [MR] can be applied to reduce the number of processors necessary by a factor of $\log n$.

Both the algorithm of [GS] and the one of this paper use the idea of a *partial coloring*. A partial coloring is an assignment of colors to some, not necessarily, all of the vertices such that no two adjacent vertices have the same color. Examples of partial colorings include the trivial coloring where every vertex has a different color, and the empty coloring where no vertex is colored. If a coloring assigns colors to all of the vertices, then it is called *complete*. The set of vertices with the same color is called a *color class*.

FINDSET constructs a sequence of partial colorings starting with the trivial coloring. The objective is to find a "big" color class C (the definition of "big" is given later). If such a color class C is found, *FINDSET* deletes $C \cup N(C)$ from the graph; if no color class is "big", some of the color classes are merged. A side-effect of this is that some vertices become uncolored. *FINDSET* halts when all vertices are either deleted or uncolored. It returns the union of all the "big" color classes it found.

The success of such an approach depends on the definition of a "big" color class and on the efficiency of merging. When an algorithm decides to merge, it should do it "fast" and so that "not-too-many" vertices are decolored. The technical means by which this task is accomplished in our implementation is the comparison of two representations of the edge set E of a colored graph. The first one views E as the union of the sets $L(C)$ of edges with an endpoint in the color class C, where C ranges over the set of all color classes. Roughly speaking, $L(C)$ measures the size of the part of the graph that would be deleted if C is added to I. The second representation of E partitions the edges into classes so that the membership of an edge is determined by the colors of both its endpoints. Every class of the second representation measures the size of the graph which would be decolored if merging were performed according to this particular class. The Propositions 1 and 2 establish that if there are no big classes, then there is efficient

merging.

We expect that the consideration of these representations can be helpful in the design of parallel algorithms for other graph problems.

2. Terminology

The definitions of class *NC* and models of computations can be found in [P], [V], [KR]. The graph-theoretic terminology used in this paper is standard [BM]. The degree of a vertex v in a subgraph H is denoted $deg_H(v)$. Given a set $K \subset V(G)$ of vertices, $\sigma_H(K) = \sum_{v \in K}(1+deg_H(v))$ is called the *weight* of K in H. If the graph is understood, the indices in deg_H and $\sigma_H(K)$ are omitted. We use *weight* as the definition of size when we say that *FINDSET* finds an independent set I such that the deletion of $I \cup N(I)$ reduces the size of H by half.

Let $\mathbf{C} = [C_0,...,C_{r-1}]$ be the collection of color classes of a partial coloring ϕ. Then, L_i denotes the set of edges which have one endpoint in C_i, and N_i denotes the set of vertices which have neighbors in C_i. The consideration of L_i's yields the first method of classifying the edges. To understand the second method, we need to define the functions

$$\chi(p) = \begin{cases} p & \text{if } p \text{ is odd}, \\ p-1 & \text{if } p \text{ is even}. \end{cases}$$

$$rev(i,q;p) = (q-i) \bmod \chi(p).$$

$$index\ (i,j;p) = \begin{cases} (i+j) \bmod \chi(p) & \text{if } 0 \le i,j \le \chi(p)-1, \\ 2i \bmod \chi(p) & \text{if } j = \chi(p), \\ 2j \bmod \chi(p) & \text{if } i = \chi(p). \end{cases}$$

One can readily prove that for every q and i $(0 \le i,q \le \chi(p)-1)$, $j = rev(i,q;p)$ is the only integer in the interval $[0,\chi(p)-1]$ with $index(i,j;p) = q$.

Let ϕ be a vertex coloring and $\mathbf{C} = [C_0,...,C_{r-1}]$ be the set of its color classes. For $k = 0,...,\chi-1$ let $\pi_k = \{(C_i,C_j) : index(i,j;r) = k\}$ be a partition of $\mathbf{C}$ into $\lfloor r/2 \rfloor$ pairs and possibly one singleton. These χ partitions of $\mathbf{C}$ are called the regular partitions of $\mathbf{C}$. One can readily check, that for all $i \neq j$, π_i and π_j do not share a common pair. Thus, the collection $\{\pi_0, \ldots, \pi_{\chi-1}\}$ presents a minimal edge coloring of the complete graph whose vertices are $C_0,...,C_{r-1}$.

Fix k and let $\pi = \pi_k$. For each $l = 0,...,\lfloor r/2 \rfloor$, define $B_l = (C_{i_l} \cap N(C_{j_l})) \cup (C_{j_l} \cap N(C_{i_l}))$ and $B = \bigcup_{l=0}^{\lfloor r/2 \rfloor - 1} B_l$. The weight $\sigma(\pi)$ of π is then given by $\sigma(\pi) = \sigma(B)$.

3. An Outline of the Algorithm

In this section, we present a description of *FINDSET* and prove that every application of *FINDSET* reduces the weight of the graph by half.

Let G be a graph with n vertices and m edges. *FINDSET* starts by defining an empty independent set I and a trivial vertex coloring on the vertices of G. Then, it proceeds in phases. At every phase, one of the following actions is performed:

<1> A color class C^* is found for which $\sigma(N(C^*)) \geq (n+m)/\log n$. All vertices of C^* are added to I and all vertices from $C^* \cup N(C^*)$ are deleted.

<2> The color classes are partitioned into pairs (C_i, C'_i), with possibly one color class left over. The two color classes of each pair are merged. To make the merged color class independent, the weights of the sets $C_i \cap N(C'_i)$ and $C'_i \cap N(C_i)$ are compared and the vertices of the set with the smaller weight are decolored.

Action <1> is done whenever possible; action <2> is done only when there is no suitable color class C^*. The actions are executed until at most one color class is left. If it is indeed one color class, then independent of its weight the color class is added to I.

Action <2> is done by the procedure *HALF*. It constructs a regular partitioning π of the minimal weight, and merges every pair of color classes matched by π.

The following propositions show that action <2> does not decolor too many vertices.

Proposition 1. Let $\mathbf{C}=[C_0, \ldots, C_{r-1}]$ be the set of color classes of a coloring ϕ and $\{\pi_0,\ldots,\pi_{\chi-1}\}$ be the set of regular partitionings of $\mathbf{C}$. Then

$$\sum_{j=0}^{\chi-1} \sigma(\pi_j) \leq \sum_{i=0}^{r-1} \sigma(N(C_i)). \qquad (*)$$

Proof. The set of partitionings $\{\pi_j\}$ $(j=0,\ldots,\chi-1)$ contains every pair of color classes exactly once. Therefore, for every colored vertex v, its contribution to the left part of the inequality is equal to $\sigma(v)\times$ (the number of color classes that contain vertices adjacent to v). Obviously, the contribution of v to the right part is as big as that.

Proposition 2. Let ϕ be a coloring and $\mathbf{C}=[C_0, \ldots, C_{r-1}]$ be the set of its color classes. If for every $i \geq 0$, $\sigma(N(C_i)) \leq (n+m)/\log n$, then there is a regular partitioning π of $\mathbf{C}$ and a set D of vertices such that

(1) for every pair C', C'' of color classes matched by π, $C' \cup C'' - D$ is an independent set;

(2) $\sigma(D) \leq \dfrac{n+m}{2\log n} \dfrac{r}{\chi(r)}$.

Proof. If for every $i \geq 0$, $\sigma(N(C_i)) \leq (n+m)/\log n$, then it follows from (*) that

$$\sum_{j=0}^{\chi-1} \sigma(\pi_j) \leq \frac{(n+m)}{\log n} r,$$

which in turn implies the existence of a regular partitioning π_k of C with $\sigma(\pi_k) \leq \dfrac{n+m}{\log n} \dfrac{r}{\chi(r)}$.

Let $\{(C_{i_l}, C_{j_l})\}$ be the set of pairs of π_k. For every $l=0,\ldots,\lfloor r/2 \rfloor - 1$, compute

$\sigma(C_{i_l} \cap N(C_{j_l}))$ and $\sigma(C_{j_l} \cap N(C_{i_l}))$ and denote D_l the set having the smallest value of σ. Then, the union $D = \bigcup_l D_l$ satisfies conditions (1) and (2).

Theorem. *FINDSET* executes at most $O(\log n)$ actions. If n' and m', respectively, are the number of vertices and edges of the graph G' induced on the set of decolored vertices, then

$$n' + m' \leq (\frac{1}{2} + o(1))(n+m).$$

Proof. If action <1> is applied, then the total number of edges and vertices deleted is at least $(n+m)/\log n$; thus these actions are executed at most $\log n$ times. Obviously, the number of times actions of the second type are applied is also at most $\lceil \log n \rceil$.

Let D^i be the set of vertices that are decolored by the ith application of an action of type 2 and let $A = \cup D^i$. Then,

$$n' + m' \leq \sigma(A) \leq \sum_i \sigma(D^i),$$

$$\sigma(D^i) \leq \frac{n+m}{2\log n} \frac{r_i}{\chi(r_i)},$$

and

$$\sigma(A) \leq \sum_{i=0}^{\lceil \log n \rceil} \frac{n+m}{2\log n} \frac{r_i}{\chi(r_i)} \leq \frac{n+m}{2\log n} \sum_{i=0}^{\lceil \log n \rceil} \frac{r_i}{\chi(r_i)},$$

where r_i is the number of color classes just before the ith application of the type 2 action.

To evaluate $\sum_{i=0}^{\lceil \log n \rceil} \frac{r_i}{\chi(r_i)}$, note that $\chi_i < r_i$ for even r_i's only, and for such r_i's, every application of an action of type 2 reduces the number of color classes by half; if $r_i \geq 3$ is odd, $r_{i+1} \leq (r_i/2)+1$. Thus,

$$\sum_{i=0}^{\lceil \log n \rceil} \frac{r_i}{\chi(r_i)} \leq \sum_{i=1}^{\lceil \log n \rceil} \frac{2^i}{2^i - 1} \leq \lceil \log n \rceil + 2,$$

which implies the theorem.

Corollary. Any application of *FINDSET* yields an independent set I, such that

$$|I \cup N(I)| \geq (1/2 - o(1))(n+m).$$

In the next section, we will show that *FINDSET* can be implemented to run in $O(\log^2 n)$ time. This will imply that the running time of the algorithm is $O(\log^3 n)$.

4. Implementation of the algorithm

We will first see how to implement *FINDSET* to run in $O(\log^2 n)$ time on $n+m$ processors. Obviously, for every application of *FINDSET*, action of either type is executed at most $O(\log n)$ times. Thus, we should show that the execution of each action requires only $O(\log n)$ time.

A graph G is represented by a list $L=L(G)$ of its vertices and edges. Each edge is in this list twice, once in each direction. Thus, the length of L is $n+2m$. For each entry of L, there is a processor attached to it. The processors are numbered by $1,...,n+2m$; the first n of them represent the vertices. In addition to its endpoints, each edge knows the colors (if any) of the endpoints and the location of itself given in the other direction. Almost all of the operations are done by sorting this list based on some key. >From [AKS, C] we know that it is possible to sort m records in $O(\log m)$ time on m processors.

To delete a color class C, each edge checks the colors of its endpoints to see if one has the same color as C. If one does, the edge deletes itself. Then *FINDSET* sorts the list of edges to remove the deleted edges.

To decide whether to do action <1> or action <2>, *FINDSET* needs to calculate $\sigma(C_i)$, for each C_i in ϕ. It can calculate the degree of each vertex by sorting the list of edges by their first vertex. It can then sort the list of vertices by color, and add up the degrees of the vertices.

When *HALF* performs action <2>, it finds the regular partition π that minimizes $\sigma(\pi)$. For this purpose, it needs to know the degrees of each vertex. It can calculate them itself, or it can inherit them from the previous step. To calculate $\sigma(\pi)$, *HALF* calculates, for each edge with two colored endpoints, $index(i,j;r)$, where i and j are the colors of the endpoints. It then sorts the edge list by index, breaking ties by first vertex. The different first vertices that occur with a given index q are the vertices in D^q for the partition π_q. *HALF* then sums the degrees of these vertices to calculate the weights $\sigma(\pi_t)$ $(t=0,..,\chi(r)-1)$ and selects the partition with the minimum weight.

Let q be the index of the selected partition π. At this stage, *HALF* is considering the list F of edges whose index is q. For every pair (C_{i_l},C_{j_l}) of π, the vertices of one of the color classes will be either decolored or recolored when C_{i_l} and C_{j_l} are merged; we call this color class *eligible*. To find eligible classes, *HALF* computes $\sigma_{i_l}=\sigma(C_{i_l}\cap N(C_{j_l}))$ for every $l=0,...,\lfloor r/2\rfloor-1$, and sets class C_{i_l} eligible iff $\sigma_{i_j}\le\sigma_{j_l}$. Once *HALF* has identified the eligible color classes, it looks at each edge and decolors those vertices v belonging to the eligible color class C_t that are connected to a vertex with color $rev(t,q;r)$. Finally, the pairs of color classes determined by q are merged into single classes by recoloring, for every pair, all the vertices in the color class with the larger color, i.e. each vertex v with original color $c(v)$ changes its color to $rev(c(v),q;r)$ iff $rev(c(v),q;r)\le c(v)$.

In general, the new color classes do not necessarily have consecutive numbers. To fix this, the vertices are sorted by color, so that the set of colors in use can be determined. Next, *HALF* renumbers the colors and gives each vertex its new color. To update the colors of the first vertex stored with the edges, the list of edges is sorted by first vertex. Then, the pointers to the other representation of the edges are followed to update the color of the second vertex.

Each execution of the main loop of *FINDSET* requires between one and three sorts

and $O(\log n)$ time spent doing other miscellaneous work. Thus, *FINDSET* requires only $O(\log^2 n)$ time in all. The number of times *FINDSET* is called is $O(\log n)$ implying that the running time of the algorithm is $O(\log^3 n)$.

So far, we have assumed that $m+n$ processors are available. However, using the processor reduction technique of Miller and Reif [MR], the algorithm can be executed on $(m+n)/\log n$ processors without increasing the running time by more than a constant. For an arbitrary $k>1$, if there are only $(m+n)/k$ processors, then each real processor can simulate k virtual processors in the algorithm. Since a call to *FINDSET* halves the value $n+m$ of the graph, the number of virtual processors that every real processor simulates decreases by a factor of 2 after each call to *FINDSET*. Thus, there is a constant $C>0$, such that for every $i=1,2,...,$ the ith call of *FINDSET* is executed in $\leq C\,2^{-i}\log^3 n$ time. This increases the running time of the algorithm by a factor 2.

There is no problem allocating virtual processors to real processors. Each virtual processor is responsible for a single item in list L of vertices and edges in the graph. It is only necessary to reallocate virtual processors after each call to *FINDSET*. The new representation of $G-(I \cup N(I))$ can easily be calculated by sorting. The reallocation can be even done after each execution of the loop in *FINDSET*. While this will speed up the algorithm for a typical graph, the worst-case graphs do not get smaller fast enough to improve the asymptotic performance of the algorithm.

Note that the same technique can be applied to reduce by a factor of $\log n$ the number of processors used by the probabilistic algorithms from [L1] and [ABI].

5. Conclusion

An important property of a parallel algorithm is the total work W it does. Obviously, $W \leq P \times T$, where P is the number of the processors used by the algorithm and T is its running time. Thus, our deterministic algorithm for *MIS* does $O((m+n)\log^2 n)$ work which is a factor of $\log^2 n$ more than the work done by the obvious sequential algorithm. It would be nice to find an algorithm that does less work. One approach would be to improve the sorting algorithm that our algorithm uses. This might be possible since the keys of all the sorts are small integers. In fact, Reif has an algorithm that sorts n small integers while doing only $O(n)$ work on a randomized concurrent-read, concurrent-write, PRAM [R]. However, if randomization is introduced in the model, then the algorithms from [L1] and [ABI] are preferable. Thus, real improvement would be achieved if better deterministic sorting algorithms were used. Obviously, they would be of interest for other reasons as well. Another approach is to find a way to sort less often. Both approaches appear to be very difficult.

A more promising way to improve the algorithm would be to reduce its running time without increasing the work that it does very much (if at all). It may be possible to reduce the running time of the algorithm by executing different calls to *FINDSET* in parallel. There are several ways to do that; the difficulty seems to be in developing a better analytical technique for estimating the running time.

The dual representation of the edge set may also be useful for other problems, edge-coloring and vertex-coloring being among the first candidates. Another potentially fruitful application of this representation is the problem of constructing an independent set of a *guaranteed* size. In [G], an algorithm was described which runs in $O(\log^3 n)$ time on

$O(n+m)$ processors and constructs an independent set with $\geq n^2/32m$ $(m \geq n/2)$ vertices. Conceivably, an appropriate change in the definition of the *weight* of a set will convert our algorithm into one which builds an independent set containing $\geq n^2/(2m+n)$ vertices. This is guaranteed by Túran's theorem [T] which also states that this bound is best possible in terms of n and m.

Acknowledgment. We gratefully thank the three referees for their helpful comments on the first version of the paper.

References

[AKS] Ajtai, M., J. Komlos, E. Szemeredi, An $O(n \log n)$ sorting network, *Combinatorica*, 3 (1983), pp. 1-19.

[ABI] Alon, N., L. Babai, A. Itai, A fast and simple randomized parallel algorithm for the maximal independent set problem, *J. of Algorithms,* **7**(1986), pp. 567-583.

[C] Cole, R., Parallel merge sort, *Proc. 27th Annual Symp. on Foundation of Computer Science,* (1986), pp. 511-516.

[BM] Bondy, J.A., U.S.R. Murty, *Graph Theory with Applications*, North Holland, New York, 1980.

[G] Goldberg, M., Parallel algorithms for three graph problems, *CONGRESSUS NUMERANTIUM,* vol. 54, (1986), pp. 111-121.

[GS] Goldberg, M., T. Spencer, A New Parallel Algorithm for the Maximal Independent Set Problem, SIAM J. Comp., 18 (1989), pp. 419 - 427.

[KR] Karp, R.M., V. Ramachandran, Handbook of Theoretical Computer Science, *Preprint*, University of California, Berkeley, (1987).

[KW] Karp, R.M., A. Wigderson, A Fast parallel algorithm for the maximal independent set problem, *Proc. 16th ACM Symp. on Theory of Computing,* 1984, pp. 266-272.

[L1] Luby, M., A Simple parallel algorithm for the maximal independent set problem, *SIAM J. Comp.*, 15 (1986), pp. 1036-1053.

[L2] Luby, M., Removing Randomness in Parallel Computation Without a Processor Penalty, Proc. 29th Annual Symp. on Foundation of Computer Science, (1988), pp. 162-173.

[MR] Miller, G.L., J.H. Reif, Parallel Tree Contraction and its Applications, *Proc. 26th Annual Symp. on Foundation of Computer Science,* (1985), pp. 478-489.

[P] Pippenger N., On Simultaneous Resource Bounds, *Proc. 20th Annual Symp. on Foundation of Computer Science* (1979), pp. 307-311.

[R] Reif, John H., An Optimal Parallel Algorithm for Integer Sorting, *Proc. 26th Annual Symp. on Foundation of Computer Science,* (1985), pp. 496-504.

[T] Túran, P., On the theory of graphs, *Colloq. Math.* 3 (1954), pp. 19-30.

[V] Vishkin, U., Synchronous parallel computation - a survey, *Preprint*, Courant Institute, New York University (1983).