A NEW PARALLEL ALGORITHM FOR THE MAXIMAL INDEPENDENT SET PROBLEM*

MARK GOLDBERG†‡ AND THOMAS SPENCER†§

Abstract. A new parallel algorithm for the maximal independent set problem is constructed. It runs in $O(\log^4 n)$ time when implemented on a linear number of EREW-processors. This is the first deterministic algorithm for the maximal independent set problem (MIS) whose running time is polylogarithmic and whose processor-time product is optimal up to a polylogarithmic factor.

Key words. parallel computation, NC, efficient, deterministic, maximal independent set, matching

AMS (MOS) subject classification. 68R10

1. Introduction. When researchers investigate the parallel complexity of a problem, one of the main questions they ask is whether a polylogarithmic running time is achievable on a PRAM containing a polynomial number of processors. If the answer is positive, then the problem and the corresponding algorithm are said to belong to class $\mathcal{NC}$ introduced in [22] (see also [8], [25]). Having constructed an $\mathcal{NC}$-algorithm for a given problem, it is natural to try to improve its computational complexity. The complexity of a parallel algorithm is characterized by the pair (T, P), where $T = T(N)$ is the worst-case running time, $P = P(N)$ is the number of processors used, and N is the size of the input. It has been traditional to consider the product $W = T(N)P(N)$ as a unified measure for different parallel algorithms solving the same problem. W represents the total amount of work that the parallel algorithm does; it is also the running time of the sequential algorithm into which the algorithm can be converted.

Let a sequential algorithm A_s with running time T_s and a parallel algorithm A_p solve the same problem. The *relative efficiency* $E(A_p, A_s)$ of A_p with respect to A_s measures the amount of extra work that A_p does, and it is given by $E = T_s/W_p$. The value of $E(A_p) = E(A_p, A_s)$, where A_s is the fastest known sequential algorithm for the problem, characterizes the efficiency of A_p. Thus, in general, this characteristic of a parallel algorithm depends on the progress in designing a sequential algorithm, but for the problems with a linear sequential algorithm, $E(A_p)$ is absolute. Clearly, the optimal algorithms introduced by Galil in [9] have the maximum relative efficiency of $O(1)$. We should expect that for many problems, achieving the speedup from a polynomial to a polylogarithmic running time is only possible if the efficiency $E(A_p)$ is a function that tends to 0 when $N \to \infty$. Consequently, the algorithms whose relative efficiency is large enough may be called *efficient*. In particular, we call an

* Received by the editors February 9, 1987; accepted for publication (in revised form) April 26, 1988.

† Department of Computer Science, Rensselaer Polytechnic Institute, Troy, New York 12180-3590.

‡ The work of this author was supported in part by the National Science Foundation under grant DCR-8520872.

§ The work of this author was supported in part by the National Science Foundation under grant CCR-8810609.

$\mathcal{NC}$-algorithm *efficient* if it has relative (to the fastest sequential algorithm) efficiency at least $\Omega(1/\log^l N)$, where l is a constant.

In this paper, we present an *efficient deterministic* parallel algorithm for the *Maximal Independent Set* problem (MIS). Recall that a subset I of the vertices of a graph G is *independent* if there are no edges between any two vertices in I. An independent set I is *maximal* if it is not a proper subset of any other independent set. MIS is the problem of constructing a maximal independent set of a given graph.

Karp and Widgerson were the first to prove that MIS is in $\mathcal{NC}$. On graphs with n vertices and m edges, their algorithm [17] runs in $O(\log^4 n)$ time and uses $O(n^3/\log^3 n)$ processors. In [19] Luby constructed a probabilistic algorithm that runs in $O(\log n)$ time when implemented on a linear number of processors under the CRCW PRAM model of computation. In the EREW PRAM model, it runs in $O(\log^2 n)$ time. The deterministic version of the algorithm uses $O(n^2 m)$ processors. A still different probabilistic algorithm for MIS was described by Alon, Babai, and Itai in [4].

Since there is a trivial sequential algorithm that runs in linear time, every efficient algorithm for MIS must use at most a linear number of processors (up to a polylogarithmic factor). A deterministic algorithm that uses a linear number of processors was proposed by Goldberg in [12]; its running time is $O(n^\alpha)$, where $\alpha > \frac{1}{2}$ is arbitrary. Using the *deterministic coin-flipping technique* introduced by Cole and Vishkin in [6], Goldberg, Plotkin, and Shannon [11] developed algorithms for MIS as well as for the vertex-coloring problem, VC, which run in $O(\log^* n)$ time[1] on graphs with bounded maximum degree. Unfortunately, when the degree is allowed to grow, the algorithms become inefficient.

The algorithm we present in this paper runs in $O(\log^4 n)$ time on an EREW PRAM consisting of $O(m+n)$ synchronous processors that share a common memory [8], [22], [25], [26]. Each processor is a standard random access machine [2] capable of doing elementary operations on words of length $O(\log(n+m))$.

We follow the usual graph-theoretic terminology [7]. Our graphs are without loops or parallel edges. The vertices of a graph on n vertices are represented by integers $0, 1, \cdots, n-1$; the edges are given by a list of pairs $\{(i,j)\}$, where $0 \le i < j \le n-1$. Given a set T of vertices of a graph $G = (V, E)$, the neighborhood $N(T)$ is defined as the set of all vertices in V that are adjacent to at least one vertex in T. Thus, a set I is independent if $I \cap N(I) = \emptyset$. A subgraph of a graph G induced on a set A of vertices is denoted by $G[A]$. A matching is a collection of disjoint edges.

A partial coloring ϕ of a graph G is given by a collection of disjoint independent subsets $(C_1, \cdots, C_p)$ of G. We say that the vertices in C_i have color i $(1 \le i \le p)$; thus, the colors are always positive integers. If $V(G) = \cup_{i=1}^p C_i$, then ϕ is called complete. A trivial partial coloring is that for which $p = |V(G)|$; hence, for a trivial partial coloring, every vertex is its own color class. Given a partial coloring $\phi = (C_1, \cdots, C_p)$, we define matrix $D(\phi) = (d_{ij})$ and function $Q(\phi)$ by

$$d_{ij} = |N(C_i) \cap C_j|, \qquad (i, j = 1, \cdots, p);$$

$$Q(\phi) = \max_{1 \le i \le p} (|C_i| + |N(C_i)|).$$

For $h > 0$, we define a graph $B(\phi, h)$, on p vertices, by setting vertices i and j adjacent if and only if both $d_{ij} \ge h$ and $d_{ji} \ge h$ $(0 \le i, j \le p-1)$.

[1] We write $\log n$ for $\log_2 n$ and $\log^* n$ for the minimum i such that the ith iteration of log function applied to n is < 2.

If L is a list of items sorted according to a key function f, then a maximal sublist of L with identical values of the key is called an interval of L. Every sorted list L can be viewed as the concatenation of its intervals.

A pair (r, s) is lexicographically smaller than another pair (r', s') if and only if either $r < r'$, or $r = r'$ and $s < s'$.

2. The algorithm. All parallel algorithms for MIS mentioned above as well as our algorithm have the same top-level description as the very first algorithm developed by Karp and Widgerson in [17].

```
begin
    I := ∅;      A := V(G);
    (* I is an independent set *)
    while A ≠ ∅ do
        begin
        C := FINDSET(A);
        I := I ∪ C;
        A := A − (C ∪ N(C))
    end;
end;
```

It is not hard to prove that an algorithm with such a structure will have a polylogarithmic running time if every application of FINDSET runs in polylogarithmic time and produces independent set C such that $|C \cup N(C)| = \Omega(|A|/\log^s |A|)$ for some fixed $s \geq 0$. Later, we will see that for our version of FINDSET, $s = 1$.

Informally, FINDSET works as follows. Starting with a trivial partial coloring ϕ_0 of graph $H = G[A]$, it constructs a sequence of partial colorings ϕ_j $(j \geq 0)$. For each $j \geq 0$, the procedure checks whether

$$(*) \qquad Q(\phi_j) > \frac{c_0 k}{\log k}$$

where $k = |A|$. If $(*)$ holds, FINDSET outputs the color class C' for which $|C'| + |N(C')| > c_0 k/\log k$; otherwise, it constructs a new partial coloring by decoloring some of the vertices and uniting some of the color classes. The construction is done by the procedure REDUCE. The input to REDUCE is a partial coloring ϕ and a matching M in the complement $\bar{B}$ of graph $B(\phi, h)$ (the selection of $h > 0$ is specified later). The matching M supplies a collection of pairs of color classes of ϕ. For each pair (C, C'), either the set $C \cap N(C')$ or the set $N(C) \cap C'$ (whichever is smaller) is decolored and the remaining vertices in $C \cup C'$ are declared to be a new color class. The new color classes obtained in this way, and the old color classes that were not changed, comprise the set of color classes of the new partial coloring.

A procedure MATCH finds a matching in $\bar{B}(\phi, h)$. It will be seen that the running time of the whole algorithm depends on the size of the matching delivered by this procedure as well as on its running time. In the context of our algorithm, each graph B to which MATCH is applied is such that its complement $\bar{B}$ contains a quadratic number $\Theta(|V(B)|^2)$ of edges even if the original graph G is sparse. This is our reason for not using any of the known algorithms for constructing a maximal matching (see [1], [10], [14], [15], [16], [18], [19], [20]). The subroutine MATCH runs in $O(\log n)$ time on an EREW PRAM with $O(n+m)$ processors. For graphs with a dense compliment, MATCH constructs a matching of size $\Omega(|V(B)|)$, which is maximum up to a constant. On the other hand, it is not necessarily maximal.

Our algorithm uses $O(n+m)$ processors; every vertex and every edge of a graph has a processor associated with it; abusing the language, we identify a vertex or an edge with the corresponding processor. Each edge has, for each of its endpoints, a pointer to a record that stores the color of that vertex. If $\Delta(v)$ is the degree of a vertex v, then there are $\Delta(v)$ records containing the information related to v. Thus, each edge can access its endpoints independently. An uncolored vertex has its color set to 0; a vertex that has been deleted by an earlier iteration of the top-level loop has its color set to -1.

A Pascal-like description of FINDSET is as follows:

```
function FINDSET(A);
begin
    k := |A|;      H := G[A];
    φ := trivial coloring of H;
    while Q(φ) < c_0 k/log k do
        begin
            p:= the number of colors of φ_i;
            h := c_1 k/(p log k);
            B := BUILD(φ, h);
            M := MATCH(B);
            φ := REDUCE(φ, h, M)
        end;
    C := a color class with Q(C) ≥ c_0 k/log k;
    FINDSET := C
end;
```

In the description of FINDSET and MATCH we use two constants, c_0 and c_1; their values, which guarantee the necessary performance behavior, are defined in §3.

The function BUILD accepts a partial coloring ϕ and a number $h > 0$ and constructs the auxiliary graph $B(\phi, h)$. It does this by computing the values of the d_{ij} that are nonzero, and then finding out which d_{ij} and d_{ji} are both greater than h.

To calculate the d_{ij}, BUILD first creates a list of records containing, for each edge, its endpoints and their colors written in increasing order. Next, BUILD sorts this list lexicographically by color. Each interval of the resulting list consists of the edges with the endpoints colored by the same pair of colors. Let L_{ij} be the interval containing the edges whose endpoints are colored i and j. To calculate d_{ij}, BUILD sorts L_{ij} by the vertex colored j; the number of intervals of this list is the value of d_{ij}. Similarly, d_{ji} is the number of intervals that result when L_{ij} is sorted by the vertex colored i.

To find the intervals of a sorted list, every member of the list compares itself with the element on its right and the element on its left. This indicates the elements that are the ends of the intervals. Then, every other member assigns itself to the corresponding interval. This can be done in $O(\log n)$ time using the path-doubling technique of Wyllie [27].

```
function BUILD(φ, h);
begin
    L := a list of the edges with the colors of their endpoints
            listed in increasing order;
    sort L lexicographically by the colors of the endpoints;
    determine the set of intervals of L;
```

```
        for each interval L_ij in parallel do
            (* the subscripts i, j are the corresponding colors *)
            begin
                sort L_ij in increasing order of the endpoint colored by j;
                set d_ij to be the number of intervals of L_ij;
                sort L_ij in order of the endpoint colored by i;
                set d_ji to be the number of intervals of L_ij;
                if d_ij > h and d_ji > h then
                    include (i, j) in E(B(φ, h));
            end;
    end;
```

All sorts are done using Cole's algorithm [5]; thus, BUILD runs in $O(\log n)$ time on an EREW PRAM with a linear number of processors.

The idea of the procedure MATCH is as follows. Let the vertices of B be numbered by $0, 1, \cdots, p-1$, where $p = |V(B)|$, let K_p be the complete graph on $V(B)$, and let $\chi = p$ if p is odd, and $\chi = p-1$ if p is even. It is well known that there is a partition of the edges of K_p into χ matchings $P_0, P_1, \cdots, P_{\chi-1}$, each of size exactly $\lfloor p/2 \rfloor$. We give an explicit construction of such a partition in terms of the function "index" defined below, where the edge (i, j) is assigned to the matching $P_{\text{index}(i,j)}$. If the total number of edges of $\bar{B}$ is quadratic, the set $\bar{M}_t$ of the maximum size contains $\Omega(p)$ edges. For the same t, the size of M_t is a minimum. When such a t is found, we can check every of $\lfloor p/2 \rfloor$ pairs of $\bar{P}_t$ to compute $\bar{M}_t$.

```
    function index(i, j, p);
    begin
        if p is odd then
            index := (i + j) mod p
        else
            if j = p − 1 then
                index := 2i mod (p − 1)
            else
                index := index(i, j, p − 1)
            end;

    function MATCH(B);
    begin
        M := ∅; p := |V(B)|;
        if p is even then χ := p − 1 else χ := p;
        for each edge (i, j) of B in parallel
            compute index(i, j, p);
        for l := 0 to χ − 1 in parallel compute g(l)
            the number of edges (i, j) with index(i, j, p) = l;
        find t such that g(t) is minimized;
        compute M := P_t ∩ B̄;
        remove all but ⌈p(c_1 − c_0)/(2c_1)⌉ edges from M;
    end;
```

Computing the number of the edges with a given index is done by sorting the edges of B according to their indices and then determining the lengths of all intervals. Finding a t for which the corresponding color class P_t contains the fewest number

of edges can be done using Valiant's algorithm [24]. To determine $P_t \cap \bar{B}$, MATCH appends a list of edges in $P_t \cap B$ to the list of the edges in P_t. Then, the list is sorted lexicographically to bring the duplicates next to each other. If a pair (a, b) occurs in the list twice, then both occurrences are removed. The remaining pairs are a list of the edges in $\bar{B} \cap P_t$.

It turns out that the analysis is simpler if the matching returned by MATCH has a known size. We will show in §3 that, for our choice of c_0 and c_1, the size of $\bar{B} \cap P_t$ is at least $p(c_1 - c_0)/(2c_1)$. Thus, after removing a few edges from $\bar{B} \cap P_t$, MATCH returns a matching with exactly $\lceil p(c_1 - c_0)/(2c_1) \rceil$ edges. It is easy to see that every application of MATCH is executed on a linear number of processors in $O(\log n)$ time.

The matching M calculated by MATCH is used by REDUCE to construct a new partial coloring with a smaller number of color classes. In particular, REDUCE does the following three things:

(1) Decides which vertices are to be decolored;
(2) Merges the appropriate color classes; and
(3) Renumbers the color classes.

It is easy to implement REDUCE so that it runs in $O(\log n)$ time on a CRCW PRAM with a linear number of processors, which also yields an implementation on an EREW PRAM running in $O(\log^2 n)$ time. A more elaborate technique is needed to implement REDUCE so that it runs in $O(\log n)$ time on an EREW PRAM.

Intuitively, each vertex of color l needs to know which color, if any, is matched to l by M. We use a routine called BROADCAST to deliver this information. Specifically, BROADCAST is given a list L of ordered pairs of the form (l_i, m_i), where l_i is a color, m_i is a "message," and each color appears on the list at most once. The task of BROADCAST is to label each vertex of color l_i with the message m_i. For this purpose, BROADCAST first creates a new list L' with one record for each colored vertex. Each record is of the form (l_v, v), where l_v is the color of vertex v. Then, it sorts the concatenation of L' and L by color, that is by first coordinate; if two pairs from L and L', respectively, have the color, the pair from L is declared to be smaller. Next, BROADCAST uses the standard path-doubling technique to give m_i to each element of the list with color l_i. Finally, each element from L' that received a message labels its vertex with the message. Clearly, BROADCAST runs in $O(\log n)$ time and uses $O(n + m)$ processors.

In the context of REDUCE, the procedure BROADCAST is used to decide, for each edge $(i, j) \in M$, whether to decolor vertices with color i or with color j. Recall that the vertices of color j are decolored if and only if $d_{ij} \leq d_{ji}$. Both values d_{ij} and d_{ji} can either be obtained from BUILD, or REDUCE can calculate them itself. Having obtained these values, REDUCE orients each edge (i, j) of M so that $d_{ij} \leq d_{ji}$; hence the color of the vertices to be decolored is listed second. Then, REDUCE calls BROADCAST(M) to tell which vertices need to change color. If a vertex v receives a color l as a message, it checks to see if it has a neighbor of color l. If it does, it decolors itself; otherwise it changes its color to l.

Finally, REDUCE renumbers the surviving color classes by consecutive integers, starting with zero. This is necessary for the next application of MATCH to be done correctly. To do this, it sorts all the vertices by color and removes duplicates. The result is a list of all the colors in use. This list is then numbered, and the position of each color in the list is broadcast. Each colored vertex then changes its color to the color it receives.

A Pascal-like description of REDUCE is as follows:

```
function REDUCE(φ, h, M);
begin
    orient each edge (i, j) of M so that d_ij ≤ d_ji;
    BROADCAST(M);
    for each vertex v that received a color, l(v), in parallel do
        if v is adjacent to a vertex of color l(v)
            then decolor v
        else change the color of v to l(v);
    sort the vertices by their (new) colors;
    number the colors in use;
    make L, a list (l, n(l)), where n(l) is the number of l;
    BROADCAST(L);
    each colored vertex changes its color to the message it received;
end;
```

3. Analysis. Our goal is to show that using $O(n+m)$ processors and in $O(\log^2 n)$ time FINDSET constructs an independent set C such that $|C \cup N(C)| > c_0 k/\log k$, where $|A| = k$ and c_0 is a constant. Once this is established, it is easy to see that FINDSET is called $O(\log^2 n)$ times and that the running time of the algorithm is $O(\log^4 n)$.

First, let us estimate the size of the set that MATCH returns. Recall that FINDSET calls MATCH on the graph $B(\phi, h)$. Let ϕ_i be the value of ϕ at the beginning of the ith iteration of the body of the *while* loop in FINDSET, and let p_i be the number of color classes in ϕ_i. If the body of the loop is executed then

$$Q(\phi_i) < \frac{c_0 k}{\log k}.$$

Let Δ_i be the maximum degree of a vertex in $B(\phi_i, h_i)$, where $h_i = c_1 k/(p_i \log k)$. If the degree, in B, of a vertex corresponding to a color class C is Δ_i, then $N(C)$ must contain at least $\Delta_i h_i$ vertices. On the other hand, $|N(C)| < c_0 k/\log k$. Therefore,

$$\Delta_i \frac{c_1 k}{p_i \log k} < \frac{c_0 k}{\log k}, \qquad \text{and} \qquad \Delta_i < \frac{c_0}{c_1} p_i.$$

This implies that the degree of every vertex of $\bar{B}(\phi_i, h_i)$, is at least $p_i(c_1 - c_0)/c_1$, yielding

$$|E(\bar{B}(\phi_i, h_i))| \geq \frac{c_1 - c_0}{2c_1} p_i^2.$$

Recall that MATCH divides the edges of $\bar{B}(\phi_i, h_i)$ into at most p classes and finds $P_t \cap \bar{B}$, the class with the most edges. Thus,

$$|P_t \cap \bar{B}| \geq \frac{c_1 - c_0}{2c_1} p_i,$$

and MATCH can discard edges from this set to return a matching M_i with

$$|M_i| = \frac{c_1 - c_0}{2c_1} p_i.$$

When REDUCE creates ϕ_{i+1}, it decolors at most

$$|M_i|\frac{c_1 k}{p_i \log k} = \frac{(c_1 - c_0)k}{2\log k}$$

vertices and reduces the number of color classes to ap_i, where $a = (c_1 + c_0)/(2c_1)$. The initial partial coloring ϕ_0 has k color classes with a total of k vertices. Thus the *while* loop in FINDSET will be executed at most $-\log k/\log a$ times. If none of the partial colorings ϕ_i with more than one color class satisfies

$$Q(\phi_i) \geq \frac{c_0 k}{\log k},$$

then FINDSET decolors at most

$$\frac{\log k}{-\log a}\frac{(c_1 - c_0)k}{2\log k}$$

vertices while reducing the number of color classes to one. Thus, the size of the only color class of the last coloring is at least

$$Q_0 = k - \frac{c_1 - c_0}{2}\frac{k}{\log k}\frac{\log k}{(-\log a)} = k - \frac{c_1 - c_0}{2}\frac{k}{(-\log a)}.$$

If we choose c_0 and c_1 such that

$$-\log a = \log\left(\frac{2c_1}{c_1 + c_0}\right) > (1+\epsilon)\frac{c_1 - c_0}{2c_1(1 - c_0)},$$

for some fixed $\epsilon > 0$, then

$$Q_0 > \frac{\epsilon}{1+\epsilon}k > \frac{c_0 k}{\log k},$$

for sufficiently large k. The criterion will be satisfied, for example, if $c_1 = 1$ and $c_0 = 1/3$.

Since every application of the *while* loop is executed in $O(\log n)$ time and the number of times the loop is iterated is $O(\log n)$, we have that the running time of FINDSET is $O(\log^2 n)$. It implies that the running time of the whole algorithm is $O(\log^4 n)$.

4. Open problems. This work addresses several interesting unresolved questions, including the following:

(1) The processor-time product for our algorithm is $O((n+m)\log^4 n)$. We would like to reduce the total amount of work that our algorithm does by reducing the number of processors it requires while not increasing its running time. We note that the technique developed by Miller and Reif in [20] does not seem to apply to this algorithm.

(2) There is a trivial sequential algorithm that colors a given graph G in at most $\Delta + 1$ colors, where Δ is the maximal degree of a vertex in G. Using the standard reduction of VC to MIS, we get an $\mathcal{NC}$-algorithm for $\Delta + 1$-coloring which is run on $O(n\Delta^2 + m\Delta)$ processors. However, no efficient $\mathcal{NC}$-algorithm for $\Delta + 1$-coloring is known.

(3) In [23] Túran proved that every graph with n vertices and m edges contains an independent set of size $\geq n^2/(2m+n)$. Such a set can be constructed by a linear sequential algorithm [13]. Can it be constructed by an $\mathcal{NC}$-algorithm using a linear number of processors? So far, the best approximation is achieved by an algorithm COLOR from [12]. It produces a coloring such that the size of at least one color class is $n/2$ if $m \leq n/4$, and $n^2/(32m)$ otherwise. The algorithm uses a linear number of processors and runs in $O(\log^3 n)$ time.

Acknowledgment. We are grateful to both referees for their helpful comments on the first version of this paper.

REFERENCES

[1] A. Aggarwal and R. Anderson, *A random NC-algorithm for depth first search,* in Proc. 19th Annual ACM Symposium on Theory of Computing, 1987, pp. 325–334.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison–Wesley, 1974.

[3] M. Ajtai, J. Komlós, E. Szemerédi, *An $O(n \log n)$ sorting network,* Combinatorica, 3 (1983), pp. 1–19.

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

[5] R. Cole, *Parallel merge sort,* in Proc. 27th Annual IEEE Symposium on Foundations of Computer Science, 1986, pp. 511–516.

[6] R. Cole and U. Vishkin, *Deterministic coin tossing and accelerating cascades: micro and macro techniques for designing parallel algorithms,* in Proc. 18th Annual ACM Symposium on Theory of Computing, 1986, pp. 206–219.

[7] G. Chartrand and L. Lesniak, *Graphs & Digraphs*, Wadsworth, 1986.

[8] S. A. Cook, *Taxonomy of problems with fast parallel algorithms,* Inform. and Control, 64 (1985), pp. 2–22.

[9] Z. Galil, *Optimal parallel algorithms for string matching,* in Proc. 16th Annual ACM Symposium on Theory of Computing, 1984, pp. 240–248.

[10] Z. Galil and V. Pan, *Improved processor bound for algebraic and combinatorial problems in* RNC, in Proc. 26th Annual IEEE Symposium on Foundations of Computer Science, 1985, pp. 490–495.

[11] A. Goldberg, S. Plotkin, and G. Shannon, *Parallel Symmetry–Breaking in Sparse Graphs,* in Proc. 19th Annual ACM Symposium on Theory of Computing, 1987, pp. 315–324.

[12] M. Goldberg, *Parallel algorithms for three graph problems,* Congr. Numer., 54 (1986), pp. 111–121.

[13] M. Goldberg, S. Lath, and J. Roberts, *Heuristics for the graph bisection problem,* Tech. Report TR 86-8, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY.

[14] A. Israeli and A. Itai, *A fast and simple randomized parallel algorithm for maximal matching,* Computer Science Department, Technion, Haifa, Israel, 1984.

[15] A. Israeli and Y. Shiloach, *An improved parallel algorithm for maximal matching,* Inform. Process. Lett., 22 (1986), pp. 57–60.

[16] R. M. Karp, E. Upfal, and A. Widgerson, *Constructing a perfect matching is in random $\mathcal{NC}$,* in Proc. 17th Annual ACM Symposium on Theory of Computing, 1985, pp. 22–32.

[17] R. M. Karp and A. Widgerson, *A fast parallel algorithm for the maximal independent set problem,* Proc. 16th Annual ACM Symposium on Theory of Computing, 1984, pp. 266–272.

[18] G. Lev, N. Pippenger, and L. Valiant, *A fast parallel algorithm for routing in permutation networks,* IEEE Trans. on Comp., 30 (1981), pp. 93–100.

[19] M. Luby, *A simple parallel algorithm for the maximal independent set problem,* SIAM J. Comput., 15 (1986), pp. 1036–1053.

[20] G. L. Miller and J. H. Reif, *Parallel tree contraction and its applications,* in Proc. 26th Annual IEEE Symposium on Foundations of Computer Science, 1985, pp. 478–489.

[21] K. Mulmuley, U. V. Vazirani, V. V. Vazirani, *Matching is as easy as matrix inversion,* in Proc. 19th Annual ACM Symposium on Theory of Computing, 1987, pp. 345–354.

[22] N. Pippenger, *On Simultaneous Resource Bounds,* in Proc. 20th Annual IEEE Symposium on Foundations of Computer Science, 1979, pp. 307–311.

[23] P. TÚRAN, *On the theory of graphs,* Colloq. Math., 3 (1954), pp. 19–30.

[24] L. G. VALIANT, *Parallelism in comparison problems,* SIAM J. Comput., 4 (1975), pp. 348–355.

[25] ———, *Parallel computation,* in Proc. 7th IBM Symposium on Mathematical Foundations of Computer Science, 1982.

[26] U. VISHKIN, *Synchronous parallel computation—a survey,* Preprint, Courant Institute, New York University, NY, 1983.

[27] J. C. WYLLIE, *The complexity of parallel computations,* Ph. D. thesis, Department of Computer Science, Cornell University, Ithaca, NY, 1979.