Information Processing Letters 22 (1986) 77–80
North-Holland

18 January 1986

A FAST AND SIMPLE RANDOMIZED PARALLEL ALGORITHM FOR MAXIMAL MATCHING

Amos ISRAELI * and A. ITAI **

Department of Computer Science, Technion — Israel Institute of Technology, Haifa 32000, Israel

Communicated by K. Mehlhorn
Received 12 October 1984
Revised April 1985

A parallel randomized algorithm to find a maximal matching is presented. Its expected running time on a CRCW-PRAM with $|E|$ processors in $O(\log|E|)$. The expected time is independent of the structure of the input graph. This improves the best known deterministic algorithm by a factor of $\log^2|E|$.

Keywords: Maximal matching, parallel algorithm, randomized algorithm

1. Introduction

Let $G(V, E)$ be an undirected graph. A set $M \in E$ is a *matching* if no two edges of M have a common vertex. The matching M is *maximal* if it is not properly contained in any other matching. Note that this does not necessarily imply that M has more edges than any other matching.

A maximal matching can be found sequentially by the following greedy algorithm: Start with an empty matching and add any edge which is not adjacent to any edge that is already in M. Unfortunately, it is not clear how to use parallelism to implement this algorithm in less than linear time. The best known deterministic parallel algorithm for maximal matching is given in [2], where $|V|+|E|$ processors are needed to find a maximal matching of a graph $G(V, E)$ in $\log^3|E|$ time. The model of computation is the CRCW-PRAM which allows simultaneous READ/WRITE by more than one processor from/to the same memory cell. Other maximal matching algorithms apear in [3,4].

In this paper we present a very simple randomized algorithm for maximal matching. The expected complexity of the algorithm is $O(\log|E|)$ on the CRCW-PRAM and $|E|$ processors are used. The algorithm is randomized in the sense of [5], i.e., it employs randomly chosen numbers. Its properties and efficiency do not depend on the assumption that the input graph is random. A basic operation of our algorithm is the following *Random Choice Operation* (RCO): Given a nonzero boolean vector x, choose an index i such that $x_i \neq 0$, where all indices with nonzero entries have the same probability to be chosen.

In Section 2 we outline the algorithm. In Section 3 we prove that the expected time complexity of the algorithm is $O(\log|E|r)$ where r is the time needed for each RCO. This leads to a trivial, $O(\log^2|E|)$, implementation of the algorithm on an EREW-PRAM. In Section 4 we show how to perform RCO in $O(1)$ expected time on a CRCW-PRAM, which yields the promised $O(\log|E|)$ complexity. This is a novel feature which we believe to be of an independent interest.

* Present affiliation: Aiken Computing Laboratory, Harvard University, Cambridge, MA 02138, U.S.A.
** Currently on sabbatical leave at the Department of Computer Science, The University of Chicago, IL, U.S.A.

2. An outline of the Algorithm

The maximal matching algorithm is divided into phases. Let Φ_i be the *ith* phase. The input of Φ_i is a graph $G_i \in G$ and the output is a (not necessarily maximal) matching of G_i, M_i. The union of all M_i's is a maximal matching. The input graph for Φ_{i+1}, G_{i+1}, is obtained by removing the edges of M_i and their incident edges from G_i ($G_1 = G$). The algorithm terminates when all the edges of the input graph are removed.

Each phase Φ_i is divided conceptually into two stages.

Algorithm

Stage 1. Find a sparse subgraph of G_i, S_i, with maximal degree less than or equal to 2.

Stage 2. Find a matching of the sparse subgraph.

A more detailed outline of Φ_i is given below.

Stage 1.1 (***Choose an edge***). Each vertex $v \in V$ chooses at random (with equal probability) an adjacent edge and directs it outward.

Comment: Let $R_i(V, E_{R_i})$ be the subgraph of G_i induced by all the chosen edges. The outdegree of every vertex $v \in V$ satisfies $d^{out}_{R_i}(v) = 1$. The maximal indegree of a vertex in R_i is still unbounded.

Stage 1.2 (***Bound indegrees***). Each vertex $v \in V$ such that $d^{in}_{R_i}(v) \geqslant 1$ chooses an incoming edge.

Comment: Let $S_i(V, E_{S_i})$ be the graph induced by all the chosen edges, where the directions imposed in Stage 1.1 are now ignored. Every vertex $v \in V$ satisfies $d_{S_i}(v) \leqslant 2$.

Stage 2.1 (***Find a matching***). Each vertex chooses at random an incident edge of S_i. An edge $e \in E_{S_i}$ belongs to the matching M_i if it was chosen by both its endpoints.

Stage 2.2 (***Cleanup***). Remove from G_i all the edges of M_i with all their incident edges to get G_{i+1}—the input graph for the next phase.

3. The complexity of the Algorithm

A vertex $v \in V$ of a graph $G(V, E)$ is *bad* if the degree of at least $\frac{2}{3}$ of its neighbors is greater than its own degree. A vertex is *good* if it is not bad. An edge $e \in E$ is *bad* if both its endpoints are bad. Otherwise it is *good*. In order to prove that the expected number of phases is logarithmic we first show that the expected number of good edges removed from G_i at the end of Φ_i constitutes a constant fraction of the total number of good edges of G_i.

Lemma 3.1. *The probability that a good vertex of positive degree in* G_i *has a positive degree in* S_i *is greater than or equal to* $1 - e^{-1/3}$.

Proof. Let v be a good vertex of degree $d > 0$ in G_i and neighbors $u_1, \ldots, u_d$. The vertex v has k neighbors, $k \geqslant \lfloor \frac{1}{3}d \rfloor + 1$ such that $d_j = d(u_j) \leqslant d$, $j = 1, \ldots, k$. If any u_j chooses the edge (u_j, v) in Stage 1.1 of the Algorithm, then surely $d_{S_i}(v) > 0$. The probability that u_j chose the edge (u_j, v) in Φ_i is $1/d_j \geqslant 1/d$. The probability that $u_1, \ldots, u_k$ did not choose the edge (v, u_j) is

$$\prod_{j=1}^{k}\left(1 - \frac{1}{d_j}\right) \leqslant \left(1 - \frac{1}{d}\right)^{d/3} = \left[\left(1 - \frac{1}{d}\right)^{d}\right]^{1/3} < e^{-1/3}.$$

Thus, the probability that some edge (v, u_j) is in R_i is greater than $1 - e^{-1/3} > 0$. Note that by ignoring the possibility that the edge chosen in Stage 1.1 by the vertex v is in S_i we only have decreased the probability that $d_{S_i}(v) > 0$. □

A matching M is incident with a vertex v if v is an endpoint of an edge $e \in M$.

Lemma 3.2. *The probability that a vertex of positive degree in* S_i *is incident with* M_i *is greater than or equal to* $\frac{1}{2}$.

Proof. The graph S_i is a collection of cycles and paths. An isolated edge (a path of length 1) is surely in M_i. Hence, the probability that M_i is incident with one of its endpoints is also 1. It is easy to see that the probability that M_i is incident with any other vertex of G_i is $\frac{1}{2}$. □

In order to prove expected logarithmic depth we bound the number of bad edges in a graph.

Lemma 3.3. *At least one-third of the edges of any graph are good.*

Proof. Let G(V, E) be an arbitrary graph. Assume that each edge of G is directed from the endpoint of smaller degree to the endpoint of higher degree. An edge with equal degree endpoints is directed lexicographically. The resulting directed graph is acyclic.

Every bad vertex $v \in V$ satisfies $d^{in}(v) \leqslant \frac{1}{2}d^{out}(v)$. Hence, it is possible to associate with each edge e entering a bad vertex v a pair of edges leaving v such that different edges have disjoint pairs. These edges $s_1(e)$, $s_2(e)$ are called *edge successors.* In other words: for every two bad edges $e_1 \neq e_2$ of G the edges $s_1(e_1)$, $s_2(e_1)$, $s_1(e_2)$, and $s_2(e_2)$ are distinct.

The edges $s_1(e)$ and $s_2(e)$ are *twin-edges* and the edge e is their *parent.* Note that:

(a) Every bad edge has two successors but those two are not necessarily bad edges.

(b) Not all edges leaving a bad vertex have a parent.

A *root-edge* is a bad edge that either has no parent or whose twin is a good edge. *A leaf edge* is a bad edge with (at least one) good edge successor. Since the successors are all distinct, every leaf edge has 'its own' good edge successor. We prove the claim of the lemma by showing that the number of leaf edges is greater than the number of nonleaf bad edges.

Let $r_1, \ldots, r_k$ be the root edges of G. We show how to partition the bad edges of G into k edge disjoint directed acyclic subgraphs of G, $D_1, \ldots, D_k$. The graph D_i starts with the root edge r_i and contains its successors and their successors until reaching leaf edges. The directed graph D_i is acyclic since it is a subgraph of an acyclic orientation of G. With each D_i we associate the good edges whose parent edge is in D_i. Each leaf edge is the parent of (at least) one good edge and different leaf edges have different good edges. We shall prove that at least $\frac{1}{2}|D_i|$ good edges are associated with each D_i. Summing over all the D_i's yields our result.

If the D_i's were full binary trees (i.e., every internal vertex had two children), then the number of nonleaf edges of D_i would have been one less than the number of leaf edges of D_i. Unfortunately, D_i need not be a binary tree since two edges of D_i may enter the same vertex. However, for each D_i we construct a full binary tree T_i whose vertices correspond to edges of D_i. The vertices v_1 and v_2 are the children of v, in T_i, if the corresponding edges e_1 and e_2 are the successors of the edge e, corresponding to v. The number of leaf vertices of T_i is greater than the number of its nonleaf vertices. Thus in D_i the number of leaf edges is greater than the number of nonleaf edges. □

Combining Lemmas 3.1–3.3 we achieve the following theorem.

Theorem 3.4. *The probability that a good edge of* G_i *is removed in* Φ_i *is at least* $\frac{1}{2}(1 - e^{-1/3})$.

Proof. An edge is removed in Φ_i if it is in M_i or adjacent to an edge of M_i. The claim follows since:

(1) Every good edge is incident with at least one good vertex and the probability of a good vertex of G_i to be in S_i is not less than $1 - e^{1/3}$ (Lemma 3.1).

(2) The probability of M_i to be incident with any vertex of S_i is greater than or equal to $\frac{1}{2}$ (Lemma 3.2). □

Using the fact that at least one-third of the edges of any graph are good (Lemma 3.3), it is easy to see that the expected number of edges removed in Φ_i is at least $\frac{1}{6}(1 - e^{-1/3})$. Thus, we get the following corollary.

Corollary 3.5. *The expected number of phases of the algorithm is* $O(\log|E|)$.

Proof. The proof follows since the expected number of edges removed in Φ_i is at least $\frac{1}{6}(1 - e^{-1/3})$. □

The only implementation problem posed by the Algorithm is the equal probability choice in Stages 1.1 and 1.2. If we assign a processor for each edge, we can perform this operation in $O(\log|E|)$ time with no concurrent write (or read) needed. Thus,

we get a complexity of $O(\log^2|E|)$ expected time on an EREW-PRAM with $|E|$ processors.

4. Concurrent random choice

The Random Choice Operation (RCO) was defined as follows: Choose a nonzero entry of a nonzero boolean vector x where all nonzero entries have the same probability to be chosen.

Let $\acute{x}$, $|x| = d$, be a boolean vector. Assume that x is not identically zero. The RCO *succeeds* if it chooses an r such that $x_r = 1$. It *fails* if, for all i, $x_{r_i} = 0$. We show how to implement the RCO on the vector x in a CRCW-PRAM with d processors in constant expected time and with a small probability of failure.

An outline of RCO: Initially, the register called *entry* has value 0, on termination it will either remain 0 or contain a random index i such that $x_i \neq 0$.

(1) Each processor p_i chooses at random a number r_i, $1 \leqslant r_i \leqslant d$.

(2) If $x_{r_i} \neq 0$, then *entry* $:= r_i$.

Comment: Here is the only place where concurrent writes are used.

The CRCW-PRAM is the Parallel Random Access Machine that allows concurrent write to the same memory cell by more than one processor. In this case, there is a need to specify the mechanism by which the value in the cell, to which more than one processor wrote, is determined. Define the *Priority-PRAM* as the CRCW-PRAM in which the value after concurrent write is the value written by the processor with lowest index. It is easy to see that on the Priority-PRAM a successful RCO, as outlined above, indeed results in a random choice.

Lemma 4.1. *The probability that the RCO succeeds is* $> 1 - e^{-1}$.

Proof. The probability for success is minimal when there is only a single i such that $x_i = 1$. In this case, the probability that $r_j = i$ is $1/d$, and the probability that, for all processes, $r_j \neq i$ is $(1 - 1/d)^d < e^{-1}$. Thus, the probability for success is greater than or equal to $1 - (1 - 1/d)^d > 1 - e^{-1}$. □

Remark 4.2. If the number of nonzero entries is f, then probability of success is $1 - e^{-f}$. Thus, it approaches 1 if f is an increasing function of d.

Remark 4.3. The model of a Priority-write PRAM was introduced for definiteness and Lemma 4.1 depends on this assumption. If, for example, the value written depended on the value attempted to be written instead of the index of the processors, the above procedure would not choose a random nonzero value. However, (2) can be replaced by:

(2a) If $x_{r_i} \neq 0$, then *entry* $:= i$,

(2b) If *entry* $= i$, then *entry* $:= r_i$.

In this version, concurrent writes may occur only in step (2a) and Lemma 4.1 still holds. Moreover, a Priority-write may be simulated in constant time by a CRCW-PRAM in which all concurrent writes attempt to write the same number [1] (for our Algorithm the number of processors remains $O(|E|)$).

Let v be a vertex of G whose degree is d. In Stage 1.1 (1.2) of a phase, v has to choose at random one incident (entering) edge. This can be done by an RCO on the vector of edges incident with v where a zero entry stands for an already removed edge. The Algorithm works even if some vertices do not choose any edge. Thus, the only effect of failures is an increase of the expected running time by a constant factor.

Also, the Algorithm, as presented, may run forever (with probability zero). This, however, may be amended without affecting the expected running time.

References

[1] A. Itai, Arithmetic with concurrent writes, in: 4th ACM Symp. on Principles of Distributed Computing, Minacki, Ontario, 1985, to appear.

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

[3] R.M. Karp and A. Wigderson, A fast parallel algorithm for the maximal independent set problem, Proc. 16th Ann. ACM Symp. on Theory of Computing (1984) 266–272.

[4] G. Lev, Size bounds and parallel algorithms for networks, Tech. Rept. CST-8-80, Dept. of Computer Science, Univ. of Edinburgh, 1980.

[5] M.O. Rabin, Probabilistic algorithms, in: J.F. Traub, ed., Symp. on Algorithms and Complexity, New Directions (Academic Press, New York, 1976) 21–39.