A FAST AND SIMPLE RANDOMIZED PARALLEL ALGORITHM FOR MAXIMAL MATCHING

by

A. Israeli and A. Itai

Technical Report #335

September 1984
A FAST AND SIMPLE RANDOMIZED PARALLEL ALGORITHM FOR MAXIMAL MATCHING

A. Israeli and A. Itai

Department of Computer Science
Technion - Israel Institute of Technology
Haifa, 32000, Israel.

ABSTRACT

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

Let \( G(V,E) \) be an undirected graph. A set \( M \subseteq E \) 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 [JS-84] where \( |V| + |E| \) processors are needed to find a maximal matching of a graph \( G(V,E) \) in \( \log |E| \) 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 appear in [Lev-80] and [KW-84].

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 [Ra-76]; 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 \( O(\log |E| \tau) \), where \( \tau \) 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 independent interest.
2. AN OUTLINE OF THE ALGORITHM

The maximal matching algorithm is divided into phases. Let \( \Phi_i \) be the \( i \)-th phase. The input of \( \Phi_i \) is a graph \( G_i \subseteq G \) 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 \( \Phi_{i+1} \), \( G_{i+1} \) 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 \( \Phi_i \) is divided conceptually into two stages:

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 \( \Phi_i \) 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 \( u \in V \) satisfies \( d_{out}(u) = 1 \). The maximal indegree of a vertex in \( R_i \) is still unbounded.

Stage 1.2: (Bound in-degrees.)

Each vertex \( v \in V \) such that \( d_{out}(u) \geq 1 \) chooses at random 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: \( d_{in}(u) \leq 2 \).

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 end-
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 $2/3$ 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 $\Phi_i$ constitutes a constant fraction of the total number of good edges of $G_i$.

**Lemma 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, $k \geq \lfloor d/3 \rfloor + 1$, such that $d_j = d(u_j) \leq d$, $j=1, \ldots, k$. If any $u_j$ chooses the edge $(u_j, v)$ in stage 1, then surely $d_{\Phi_i}(v) > 0$.

The probability that $u_j$ chose the edge $(u_j, v)$ in $\Phi_i$ is $\frac{1}{d_j} \geq \frac{1}{d}$. The probability that $u_1, \ldots, u_k$ did not choose the edge $(v, u_j)$ is:

$$\prod_{j=1}^{k} \left(1 - \frac{1}{d_j}\right) \leq \left(1 - \frac{1}{d}\right)^{\lfloor d/3 \rfloor} = \left(1 - \frac{1}{d}\right)^{1/3} < e^{-1/3}.$$ 

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 by the vertex $v$ is in $S_i$, we have only decreased the probability that $d_{\Phi_i}(v) > 0$.

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

Technion - Computer Science Department - Technical Report CS0335 - 1984
**Lemma 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**

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: \( d^\text{in}(v) \leq 2d^\text{out}(v) \). Hence it is possible to assign two distinct edge successors to every bad edge such that different edges have different successors. In other words: If we denote the two edge successors of the edge \( e \) by \( s_1(e) \) and \( s_2(e) \), then 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. Some bad edges have no parent.
A \textit{root-edge} is a bad edge that either has no parent or whose twin is a good edge. A \textit{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 lemma by showing that the number of leaf edges is greater than the number of non-leaf 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 $\frac{k}{2} |D_i|$ 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 non-leaf 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 1-3 we achieve:

\textbf{Theorem}

The probability that an edge of $G_i$ is removed in $\Phi_i$ is at least $\frac{1}{6}(1 - e^{-1/3})$. 
PROOF

An edge is removed in $\Phi_1$ if it is in $M_4$ or adjacent to an edge of $M_4$. The claim follows since:

(1) At least a third of the edges of $G$ are good. (Lemma 3.)

(2) Every good edge is incident with at least one good vertex and the probability of a good vertex of $G$ to be in $S_4$ is not less than $1-\theta^{-1/3}$ (Lemma 1.)

(3) The probability of $M_4$ to be incident with any vertex of $S_4$ is greater than or equal to $\frac{1}{2}$. (Lemma 2.)

COROLLARY

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

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: Choose a nonzero entry of a nonzero boolean vector $x$ where all nonzero entries have the same probability to be chosen.

Let $x$, $|x| = d$, be a boolean vector. Assume that $x$ is not identically zero. We show how to implement the RCO on the vector $x$ in a CREW-PRAM with $d$ processors in constant expected time and with a small probability of failure. Note that the assumption that $x$ is not identically zero is easily verified by a single concurrent write.
An outline of RCO:

(1) \textit{entry} := \textit{nil}

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

(3) If \( x_{r_i} \neq 0 \) then \( \textit{entry} := x_{r_i} \).

Comment: Here we use concurrent write.

The RCO succeeds if it chooses an \( r \) such that \( x_r = 1 \). It fails if, for all \( i \), \( x_{r_i} = 0 \).

\textbf{Lemma 4}

The probability that the RCO succeeds is \( > 1 - e^{-1} \).

\textbf{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 \( \frac{1}{d} \), and the probability that for all processes \( r_j \neq i \) is \( \left(1 - \frac{1}{d}\right)^d < e^{-1} \). Thus the probability for success is greater than or equal to \( 1 - \left(1 - \frac{1}{d}\right)^d \geq 1 - e^{-1} \).

Note: If the number of nonzero entries is an increasing function of \( d \) then using Chernoff's inequalities we may prove that the probability of success approaches 1 as \( d \to \infty \).

Let \( u \) be a vertex of \( G \) whose degree is \( d \). In stage 1.1 (1.2) of a phase, \( u \) has to choose at random one incident (entering) edge. This can be done by an RCO on the vector of edges incident with \( u \) 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


