An Improvement on Parallel Computation of a Maximal Matching

Yijie Han

Electronic Data Systems, Inc.
37350 Ecorse Rd.
Romulus, MI 48174

Abstract

We present an improved optimal parallel algorithm with time complexity $O(\log^3 n)$ for computing a maximal matching in a graph. The improvement is made on the recent result of Kelsen which requires $O(\log^4 n)$ time for computing a maximal matching. Our parallel algorithm is designed on the EREW PRAM model.

Keywords: Graph algorithms, maximal matching, optimal algorithms, parallel algorithms.

1 Introduction

A matching of a graph is a subset M of edges of the graph with no two edges in M incident on the same vertex. A matching M is maximal if M is not a proper subset of another matching. A maximal matching can be computed by a simple greedy sequential algorithm in linear time. The problem is not trivial in the parallel case.

In this paper we design a parallel algorithm for computing a maximal matching. The computation model we use is the exclusive read exclusive write (EREW) parallel random access machine (PRAM). On the EREW PRAM the memory is shared among all processors. However, simultaneous read or write of a single memory cell by several processors are prohibited. The parallel maximal matching algorithm we present here is an optimal parallel algorithm. A parallel algorithm is optimal if its time processor product $T_p \cdot p = O(T_1)$, where T_1 is the time complexity of the best known sequential algorithm for the problem. Our algorithm is optimal in the sense that its time processor product is $O(m + n)$, where m is the number of edges of the graph and n is the number of vertices of the graph.

Parallel computation of a maximal matching has been studied by several researchers [2][5][6][7][8]. Israeli and Shiloach showed[5][6] that a maximal matching can be computed with a randomized algorithm in time $O(\log n)$ using $m+n$ processors or with a deterministic algorithm in time $O(\log^3 n)$ using $m+n$ processors. Recent results of Han[2] show that a maximal matching can be computed in $O(\log^{2.5} n)$ time with $O((m+n)/\log^{0.5} n)$ processors or in $O(\log^2 n)$ time with $M(n)$ processors, where $M(n)$ is the number of processors needed to compute a matrix product in $O(\log n)$ time. Currently $M(n)$ is $n^{2.376}$[1]. Very recently Kelsen obtained the first optimal parallel algorithm[7] for computing a maximal matching. Kelsen's algorithm runs in time $O(\log^4 n)$ using $(m+n)/\log^4 n$ processors on the EREW PRAM.

In this paper we show an improvement on Kelsen's parallel algorithm. Our results show that a maximal matching can be computed in $O(\log^3 n)$ time using $(m+n)/\log^3 n$ processors on the EREW PRAM. We achieve the improvement by applying a result of Luby[8] which computes a bipartite graph from a graph efficiently.

2 The Maximal Matching Algorithm

Let $G = (V, E)$ be a graph with vertex set V and edge set E. G is represented by the edge list representation. All edges (u, v) incident on vertex u are initially sorted by the value of v and stored consecutively in memory. Thus an undirected edge (v, w) is represented twice, once in the edge list of vertex v and once in the edge list of vertex w. We also assume that these two representations are double linked to each other. Let M be a matching of G. We use $N_G(M)$ to denote the set $\{e | e \in E$, there is an edge $e' \in M$ such that e and e' incident on the same vertex $\}$. A graph $S = (V', E')$ is a subgraph of G if $V' \subset V$ and $E' \subset E$. A graph is bipartite if the vertices of the graph can be partitioned into two sets A, B such that all edges of the graph are edges of the form (a, b) where $a \in A$ and $b \in B$.

Our parallel maximal matching algorithm is as follows:

Algorithm Matching
$Graph := G$;
$m_G := m$;
for $i := 1$ to $\lceil \log_{12/11} m \rceil$

begin

Step 1: Compute a bipartite graph B. B is a subgraph of $Graph$ and contains $m_B \geq m_G/2$ edges.

Step 2: Compute a matching M of B such that $M \cup N_B(M)$ contains at least $m_B/6$ edges.

Step 3: Remove $M \cup N_{Graph}(M)$ from $Graph$. Remove isolated vertices from $Graph$. Let $Graph :=$ the remaining of $Graph$. Let m_G be the number of edges in $Graph$.

end

It is easy to see that each iteration indexed by i removes a fraction $\frac{1}{12}$ of the edges from the remaining graph. Therefore, after $\lceil \log_{12/11} m \rceil$ iterations a maximal matching is computed. The correctness of the algorithm is obvious. To analyze its time complexity we use the following two theorems. Theorem 1 will be proven in the next section. Theorem 2 is from Kelsen[7].

Theorem 1: A bipartite graph B can be computed from a graph G in time $O(\frac{m+n}{p} + \log^2 n)$ using p processors on the EREW PRAM such that B contains at least half of the edges in G, where m and n are the number of edges and the number of vertices in G.

Theorem 2 [7] : A matching M of a bipartite graph B can be computed in time $O(\frac{m+n}{p} + \log^2 n)$ using p processors on the EREW PRAM such that $M \cup N_B(M)$ contains at least a fraction $1/6$ of the number of edges in B, where m and n are the number of edges and the number of vertices in B.

We now prove the following theorem.

Theorem 3: A maximal matching of a graph can be computed in time $O(\frac{m+n}{p} + \log^3 n)$ using p processors on the EREW PRAM.

Proof: We analyze the time complexity of algorithm Matching. By Theorems 1 and 2 an iteration indexed by i in algorithm Matching takes $O(\frac{m_G}{p} + \log^2 n)$ time with p processors. Each iteration also reduces m_G to $11m_G/12$. Note that isolated vertices are removed. Therefore the number of vertices remaining is at most the twice the number of edges. Therefore the time complexity of algorithm Matching is $O(\sum_{i=1}^{\log_{12/11} m} ((\frac{11}{12})^i \frac{m+n}{p} + \log^2 n))$ $= O(\frac{m+n}{p} + \log^3 n)$. $\Box$

3 Computing a Bipartite Graph

In this section we prove Theorem 1. We adapt Luby's bit pairs benefit algorithm[8]. We assume that readers are somewhat familiar with Luby's work.

We modify Luby's algorithm for the purpose of computing a bipartite graph. Luby's algorithm[8] labels the vertices of a graph with $\{0,1\}$ such that the number of crossing edges (edges incident with both 0 and 1) consists of at least half of the edges in the graph. Thus if we use Luby's algorithm for labeling vertices and then delete all noncrossing edges we obtain a bipartite graph containing at least half of the edges in the original graph.

Luby's labeling algorithm has time complexity $O(\log^2 n)$ using $m+n$ processors. Han and Igarashi[4] and Han[3] improved Luby's result and showed that the labeling can be done in $O(\log n)$ time using $m+n$ processors. Here we show how to adapt Luby's algorithm[8] so that the labeling can be done in time $O(\log^2 n)$ using $(m+n)/\log^2 n$ processors.

Luby's algorithm derandomizes a randomized algorithm. The randomized algorithm[8] is to label each vertex of the graph with a random bit of 0 or 1 with probability 1/2. With this random distribution an edge is a crossing edge with probability 1/2 and the expected number of crossing edges is $m/2$. Therefore there is a sample point in the sample space such that the number of crossing edges under this sample point is $\geq m/2$. The randomized algorithm is derandomized if we find such a sample point.

Luby observed that a sample space which makes the n random variables (for labeling n vertices) pairwise independent is sufficient for the purpose of derandomization. Assuming without loss of generality that n is a power of 2, Luby used the sample space with $2n$ points. Let r be a random variable uniformly distributed on $\{0,1\}^{\log n+1}$. We can write $r = r_{\log n} r_{\log n-1} \cdots r_1 r_0$, where r_i, $0 \leq i \leq \log n$, is a random bit uniformly distributed in $\{0,1\}$. The vertices of the graph is numbered from 0 to $n-1$. Let $v = v_{\log n-1} v_{\log n-2} \cdots v_1 v_0$ be a vertex, where v_i, $0 \leq i < \log n$, is a bit. The random variable labeling vertex v is defined as $r(v) = \oplus_{i=0}^{\log n}(r_i \cdot v_i)$, where $\oplus$ is the exclusive-or operation and $v_{\log n}$ is always set to 1. It can be shown[8] that the random variables obtained for labeling vertices are pairwise independent.

Let $B = \sum_{(u,v)\in E} r(u) \oplus r(v)$ denote the number of crossing edges. B is called the benefit function in [8]. The expectation $E[B] = m/2$. There is a sample point on which $B \geq m/2$.

To find such a sample point the random variable r is fixed one bit at a time starting with r_0. Assume $r_0, r_1, ..., r_{i-1}$ are fixed and $r_j = b_j$, $0 \leq j < i$. Now r_i is fixed as follows. For each edge (u, v), compute $B_0(u, v) = E[r(u) \oplus r(v) | r_0 = b_0, r_1 = b_1, ..., r_{i-1} = b_{i-1}, r_i = 0]$ and $B_1(u, v) = E[r(u) \oplus r(v) | r_0 = b_0, r_1 = b_1, ..., r_{i-1} = b_{i-1}, r_i = 1]$. Then compute the benefit function $B_0 = \sum_{(u,v)\in E} B_0(u, v)$ and $B_1 = \sum_{(u,v)\in E} B_1(u, v)$ [8]. If $B_0 > B_1$ then r_i is set to 0 otherwise r_i is set to 1.

Luby observed[8] that if $i \neq \log n$ and $i \neq \max\{j |$ the j-th bit of $u \oplus v = 1\}$, then $B_0(u, v) = B_1(u, v)$. Therefore edge (u, v) does not contribute to the decision of fixing bit r_i. Therefore edges can be grouped into $\log n$ groups. Group i consists of edges (u, v) satisfying $i = \max\{j |$ the j-th bit of $u \oplus v = 1\}$. When we are fixing bit r_i we consider only those edges in group i.

In our situation bit $r_{\log n}$ can be arbitrarily fixed to either 0 or 1 without affecting the partitioning of vertices into two sets. Therefore we need not concern the fixing of bit $r_{\log n}$.

Luby also gives the following method to evaluate $B_0(u, v)$ and $B_1(u, v)$ if edge (u, v) is in group i. At this point, bits $r_0, r_1, ..., r_{i-1}$ have been fixed. For each vertex $v = v_{\log n-1}...v_1 v_0$, compute $v(0) = \oplus_{j=0}^{i-1} v_j \cdot r_j$ and $v(1) = (\oplus_{j=0}^{i-1} v_j \cdot r_j) \oplus v_i$. Then $B_b(u, v) = u(b) \oplus v(b)$, $b = 0, 1$.

Based on the above explanation we now adapt Luby's algorithm. We first present a parallel algorithm which utilizes concurrent read and then show how to remove concurrent read from the algorithm.

Algorithm Labeling

Step 1: For all edges (u, v) do in parallel, compute edge (u, v)'s group number which is $\max\{j |$ the $j-$th bit of $u \oplus v$ is $1\}$. Sort edges by their group numbers.

Step 2: **for** $i := 0$ to $\log n - 1$ **do**

for all edges (u, v) in group i **do in parallel**

begin

Step 2.1: Broadcast $b(0)$ (which is the number containing bits $b_0, b_1, ..., b_{i-1}, 0$) and $b(1)$ (which is the number containing bits $b_0, b_1, ..., b_{i-1}, 1$) to all edges in group i.

Step 2.2: Compute u' which contains from 0-th bit to i-th bit of u and v' which contains from 0-th bit to i-th bit of v.

Step 2.3: Compute $u^0 = u' \wedge b(0)$, $u^1 = u' \wedge b(1)$, $v^0 = v' \wedge b(0)$ and $v^1 = v' \wedge b(1)$, where $\wedge$ is the bit-wise AND operation.

Step 2.4: Compute the parity of the bits in u^0, u^1, v^0, v^1. This is done by a table lookup. That is, we preset a table $T[0..n-1]$ of size n and assign $T[k]$ with the parity of bits in k. The parity of the bits in u^0, u^1, v^0, v^1 can be computed by simply indexing into table T. The results are denoted by $u(0), u(1), v(0), v(1)$. This step uses concurrent read.

Step 2.5: Compute $B_b(u,v) = u(b) \oplus v(b)$, $b = 0, 1$. Then compute $B_b = \sum_{(u,v) \in \text{group } i} B_b(u,v)$, $b = 0, 1$. If $B_0 > B_1$ then set r_i to 0 otherwise set r_i to 1. Let b_i denotes the value r_i is set to.

end

Step 3: For each vertex $v = v_{\log n-1}...v_1 v_0$, label it with $\oplus_{j=0}^{\log n-1} v_j \cdot b_j$. Again this step can be done as in steps 2.3 and 2.4. That is, first compute $v \wedge b$, where $b = b_{\log n-1}...b_1 b_0$, and then index into a preset table. This step uses concurrent read.

Steps 2.2, 2.3 and 2.4 are designed for the purpose of computing $u(b) = (\oplus_{j=0}^{i-1} u_j \cdot b_j) \oplus u_i \cdot b$ and $v(b) = (\oplus_{j=0}^{i-1} v_j \cdot b_j) \oplus v_i \cdot b$, $b = 0, 1$. Luby's algorithm[8] guarantees that the number of crossing edges is at least the half of the total number of edges in the graph. Because we adapted Luby's algorithm, algorithm Labeling given above will also guarantee the number of crossing edges is no less than $m/2$. Let us analyze the time complexity of algorithm Labeling. Step 1 takes $O(\frac{m}{p} + \log n)$ time with p processors[9] because we are sorting integers in $\{0, 1, ..., \log n\}$. Let m_i be the number of edges in group i. In the i-th iteration of step 2, step 2.1 takes $O(\frac{m_i}{p} + \log n)$ time by using a simple broadcasting procedure. Step 2.2 takes $O(\frac{m_i}{p})$ time. Step 2.3 takes $O(\frac{m_i}{p})$ time. Step 2.4 takes $O(\frac{m_i}{p})$ time. Step 2.5 takes $O(\frac{m_i}{p} + \log n)$ time by a simple prefix sum procedure. Thus step 2 takes $O(\sum_{i=0}^{\log n-1} (\frac{m_i}{p} + \log n)) = O(\frac{m}{p} + \log^2 n)$ time. Step 3 takes $O(\frac{n}{p})$ time. Thus the time

complexity of algorithm Labeling is $O(\frac{m+n}{p} + \log^2 n)$ time using p processors.

We now show how to remove concurrent read from algorithm Labeling to yield an EREW algorithm. The concurrent read is used in step 2.4 and step 3 where we index into a preset table to compute the parity function. The concurrent read can be removed if we assume the parity function is built into each processor as an instruction. Here we also show a scheme to remove concurrent read without assuming that the parity function is built into each processor.

To remove concurrent read from step 2.4 we divide step 2 into 2 stages. Stage 1 executes the first $\log\log n$ iterations in step 2 and stage 2 executes the remaining iterations in step 2. Stage 1 starts with the 0-th iteration and stage 2 starts with the $\log\log n$-th iteration. When we execute step 2.4 in stage 1 (say we are executing i-th iteration), u^0, u^1, v^0, v^1 each contains at most $\log\log n$ bits. Thus we can sort them for all edges in group i. After sorting only 1 representative from the same numbers will index into the preset table and thus concurrent read is eliminated. After this representative computed its parity value it then broadcasts this value to all the same numbers (they are consecutive in the sorted sequence). Sorting and broadcasting can be done in $O(\frac{m_i}{p} + \log n)$ time.

After finishing stage 1 and before executing stage 2 we re-number all vertices. Vertex $v = v_{\log n-1} v_{\log n-2} ... v_1 v_0$ is re-numbered to $v' = v_{\log n-1} v_{\log n-2} ... v_{\log\log n} b$, where $b = \oplus_{j=0}^{\log\log n-1} v_j \cdot b_j$ and b_j, $0 \leq j < \log\log n$, are the fixed bits for r_j, $0 \leq j < \log\log n$. Two different vertices may be re-numbered with the same number. The re-numbering uses $2n/\log n$ numbers. Note that $v(0), v(1)$ in i-th iteration in stage 2 can be obtained by the formulae $v(0) = (\oplus_{j=\log\log n}^{i-1} v_j \cdot b_j) \oplus b$ and $v(1) = (\oplus_{j=\log\log n}^{i-1} v_j \cdot b_j) \oplus b_i \oplus b$. In order to compute $v(0), v(1)$ for all vertices, we use $2n/\log n$ variables containing the numbers $\{0, 1, ..., 2n/\log n - 1\}$ used to re-number the vertices. We also use $2n/\log n$ variables $z_0, z_1, ..., z_{2n/\log n-1}$ containing the current parity function values for the $2n/\log n$ numbers. Initially $z_{v'}$ contains b. At the end of i-th iteration in stage 2, $z_{v'}$ is updated to $z_{v'} := z_{v'} \oplus v_i \cdot b_i$. During the i-th iteration in stage 2 the values of $z_{v'} \oplus v_i \cdot 0$ and $z_{v'} \oplus v_i \cdot 1$ are broadcast to all edges with endpoints re-numbered with v'. For vertex v, $z_{v'} \oplus v_i \cdot 0$ is $v(0)$ and $z_{v'} \oplus v_i \cdot 1$ is $v(1)$. In each iteration in stage 2 updating $z_0, z_1, ..., z_{2n/\log n-1}$ takes $O(\frac{n}{p \log n})$ time. Because edges are initially sorted and represented by the edge lists in the

input, we can sort the edges in step 1 of algorithm Labeling such that all edges in group i with an end point re-numbered with v' are consecutive in the sorted array. Therefore broadcasting $z_{v'} \oplus v_i \cdot 0$ and $z_{v'} \oplus v_i \cdot 1$ to all edges with end points re-numbered with v' can be done in $O(\frac{m_i}{p} + \log n)$ time in the i-th iteration. Thus the i-th iteration in stage 2 takes $O(\frac{m_i}{p} + \frac{n}{p \log n} + \log n)$ time.

The concurrent read in step 3 is removed in a similar way. We first compute parity for the least significant $\log \log n$ bits. This is done using 1 iteration. We then re-number the vertices with $2n/\log n$ numbers. For each such number $v = v_{\log n - 1} ... v_{\log \log n}$, we compute $\oplus_{j=\log \log n}^{\log n - 1} v_j \cdot b_j$ in $\log n - \log \log n$ iterations. We then broadcast the results to all vertices. The time complexity is $O(\frac{n}{p} + \log n)$.

After these changes the Labeling algorithm we obtained is an EREW algorithm. We have thus proved Theorem 1.

Acknowledgment

The author would like to thank anonymous referees for careful reviewing and making constructive suggestions about the manuscript.

References

[1] D. Coppersmith, S. Winograd. Matrix multiplication via arithmetic progressions. *Proc. 19th Ann. ACM Symp. on Theory of Computing*, 1-6(1987).

[2] Y. Han. A fast derandomization scheme and its applications. Tech. Rep. No. 180-90, Dept. of Computer Science, University of Kentucky, Lexington, Kentucky, to appear on *SIAM J. on Computing*.

[3] Y. Han. A parallel algorithm for the PROFIT/COST problem. *Proc. 1991 Int. Conf. on Parallel Processing*, Vol. 3, 103-112(Aug. 1991).

[4] Y. Han and Y. Igarashi. Derandomization by exploiting redundancy and mutual independence. *Proc. Int. Symp. SIGAL'90*, Tokyo, Japan, LNCS 450, 328-337(1990).

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

[6] A. Israeli and Y. Shiloach. A fast and simple parallel algorithm for maximal matching. *Inform. Process. Lett.* **22** (1986) 77-80.

[7] P. Kelsen. An optimal parallel algorithm for maximal matching. *Inform. Process. Lett.* **52** (1994) 223-228.

[8] M. Luby. Removing randomness in parallel computation without a processor penalty. *J. Computer and System Sciences* **47**, 250-286(1993).

[9] R.A. Wagner, Y. Han. Parallel algorithms for bucket sorting and the data dependent prefix problem. *Proc. 1986 Int. Conf. on Parallel Processing*, St. Charles, Illinois, pp. 924-930.