June 1991 Report No. STAN-CS-91-1369

Approximating Matchings in Parallel

by

T. Fischer, A.V. Goldberg, S. Plotkin

Department of Computer Science

Stanford University

Stanford, California 94305

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)

2. REPORT DATE
June 1991

3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE
Approximating Matchings in Parallel

5. FUNDING NUMBERS

6. AUTHOR(S)
Ted Fischer, Andrew Goldberg, Serge Plotkin

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Computer Science Department
Stanford University
Stanford, CA 94305

8. PERFORMING ORGANIZATION REPORT NUMBER
STAN-CS-91-1369

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
ONR
Arlington, VA 22217

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

24. DISTRIBUTION / AVAILABILITY STATEMENT
unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

We show that for any constant $k > 0$, a matching with cardinality at least $1 - 1/(k+1)$ times the maximum can be computed in NC.

14. SUBJECT TERMS

15. NUMBER OF PAGES
5

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT

18. SECURITY CLASSIFICATION OF THIS PAGE

19. SECURITY CLASSIFICATION OF ABSTRACT

20. LIMITATION OF ABSTRACT

Approximating Matchings in Parallel

Ted Fischer* Andrew V. Goldberg+ Serge Plotkin‡

June 1991

*Department of Computer Science, Cornell University, Ithaca NY 14853. Research supported by ONR Graduate Fellowship.

‘Department of Computer Science, Stanford University, Stanford CA 94305. Research partially supported by NSF Presidential Young Investigator Grant CCR-8858097 with matching funds from AT&T and DEC, a grant from 3M Corporation, a grant from Mitsubishi Corporation, and ONR Contract N00014-88-K-0166.

‡Department of Computer Science, Stanford University, Stanford CA 94305. Research supported by NSF Research Initiation, Award CCR-900-8226, by U.S. Army Research Office Grant #DAAL-03-91-G-0102, and by ONR Contract N00014-88-K-0166.

Abstract

We show that for any constant $k > 0$, a matching with cardinality at least $1 - \frac{1}{k+1}$ times the maximum can be computed in NC.

1 Introduction

Matching is a fundamental combinatorial problem. (See [10, 15].) Furthermore, the special case of bipartite matching seems to be a important problem of parallel computation. For example, an NC algorithm for bipartite matching would imply NC algorithms for the problems of constructing depth-first search trees in both directed and undirected graphs. (See Aggarwal and Anderson [1] and Aggarwal, Anderson, and Kao [2].)

During the last decade, parallel algorithms for the bipartite matching problem received a lot of attention. The best currently known deterministic algorithms for the problem are due to Goldberg, Plotkin, and Vaidya [6] and Goldberg, Plotkin, Shmoys, and Tardos [5]. These algorithms run in $O^*(n^{2/3})$ and $O^*(m^{1/2})$ time, respectively[1]. (Here n denotes the numebr of nodes and m the number of edges in the input graph.)

Special cases of the bipartite matching problem are known to be in NC. Lev, Pippenger, and Valiant [11] gave an NC algorithm to find a perfect matching in a regular bipartite graph. Miller and Naor [13] gave an NC algorithm to find a perfect matching in a planar bipartite graph (if one exists).

Matching was shown to be in RNC by Karp, Upfal, and Wigderson [9] (see also [14] for a simpler and faster algorithm). However, the general problem is not known to be in NC.

In this paper we consider the problem of approximating maximum matchings in an arbitrary graph. We describe an NC algorithm that, for a constant $k > 0$, finds a matching with cardinality of at least $1 - \frac{1}{k+1}$ times the maximum. Our algorithm runs in $O(\log^3 n)$ time using $O(n^{2k+2})$ processors.

2 Preliminaries

In this section we introduce the notation and the parallel computation model.

Let $G = (V, E)$ be an undirected graph. Define $n = |V|$, $m = |E|$. A set of edges $M \subseteq E$ is a *matching* if no two edges of M share a node. The *cardinality* of the matching is $|M|$. The mat ching problem is to find a matching of maximum cardinality.

Given a matching M, we say that a node v is *matched* if $(v, w) \in M$ for some $w \in V$ and free otherwise. An *augmenting path* is a simple path $P = v_0, v_1, \ldots, v_l$ such that the endpoints v_o and v_l are free, for odd i in $[0 \ldots l]$ we have $(v_i, v_{i+1}) \in M$, and for even i we have $(v_i, v_{i+1}) \in E - M$. We

[1] We say that an algorithm runs in $O^*(f(n))$ time if it runs in $O(f(n) \log^k(n))$ time for some constant k.

define the length of a path to be the number of edges on the path. Note that since the endpoints are free, the above definition implies that the length of an augmenting path must be odd. Given an augmenting path, *we* can *uugment* the matching M by deleting from M the edges on the path that are in M, and adding all of the other edges on the path to M. This results in a matching with one more edge. It is a well known fact that the absence of an augmenting path implies optimality of the current matching.

Our model of parallel computation is the *exclusive-read, exclusive-write parallel random-access machine (ERE W PRAM)* [4]. We assume that the reader is familiar with the algorithm for *parallel list compression* [3] in the context of this model.

3 Algorithm Description

The main idea of the algorithm is to augment along "short" augmenting paths until all augmenting paths are "long". Lemma 4.2 of the next section shows that if a matching does not admit a short augmenting path, then its cardinality is close to the optimum.

The input to our algorithm is a graph $G = (V, E)$ and a positive integer k, and its output is a matching M of G which admits no augmenting paths of length $2k - 1$ or less. The algorithm makes k iterations; at iteration i, it finds a maximal set of node-disjoint augmenting paths of length $2i - 1$ and augments along these paths. We denote the matching maintained by the algorithm by M. Initially $M = \emptyset$.

The i-th iteration works as follows. First, the algorithm constructs a graph $A = (V_A, E_A)$ with nodes in V_A corresponding to augmenting paths of length $2i - 1$. A pair of nodes is connected by an edge if the corresponding paths share a node. Next, the algorithm finds a maximal independent set in A, and augments the current matching in G along the paths corresponding to the chosen nodes.

Observe that a maximal independent set of nodes in A corresponds to a maximal set of augmenting paths of length $2i - 1$ in G; since the nodes are independent, the augmenting paths are disjoint, and no conflict will arise when the augmentations are performed in parallel.

It remains to describe how to construct the graph A. Now, a path is uniquely defined by the ordered sequence of nodes it connects. To generate all paths of length $2i - 1$, we could consider all sequences of 2i nodes, testing the existence of the necessary connecting edges. This would generate $O(n^{2i-1})$ paths. However, we are only interested in those sequences, π, which form augmenting paths. On an augmenting path, every node is matched except for the endpoints of the path.

Rather than considering all sequences of 2i nodes, we need only choose a sequence of $i - 1$ edges from the matching, then choose two unmatched nodes for the endpoints. Let the sequence of edges be $\pi = (v_2, v_3), (v_4, v_5), \ldots, (v_{2i-2}, v_{2i-1})$, with the two endpoints v_1 and v_{2i}. The sequence corresponds to the path with the edges (v_1, v_2), (v_3, v_4), $\ldots (v_{2i-1}, v_{2i})$ added to the ones from the sequence. 0 bserve that this generates all sequences corresponding to augmenting paths of length

$2i$, yet it only generates $O(n^{i+1})$ different sequences.

We construct all sequences π of i $-$ 1 edges from M and two endpoints as described above, and assign i processors to each sequence. We then test the existence of edges (v_{2j-1}, v_{2j}) in E for j = 1 . .i. Since the sequence of edges was selected from M, and the edges being tested all share at least one node with an edge from the sequence, the tested edges cannot be in M. Therefore, if the edges are all in E, the sequence corresponds to an augmenting path. Using i processors per path, the construction takes O(logn) time. Using list compression [3] to eliminate the sequences which do not form augmenting paths, we then construct V_A in $O(\log n)$ time.

To determine for X, Y $\in V_A$ if the edge (X,Y) should be in E_A, we need to check if their corresponding paths share any nodes. Using i processors for each pair of paths, we can test this in O(log n) time.

4 Correctness and Analysis

First we prove that the algorithm is correct. The following lemma of Hopcroft and Karp implies that there are no augmenting paths of length 2i $-$ 1 or less after iteration i.

Lemma 4.1 [8] **If a matching is augmented along a maximal set of shortest augmenting paths, then the shortest augmenting path length increases.**

The next lemma is the heart of the correctness proof of our algorithm. Intuitively, the lemma states that if a matching does not admit short augmenting paths, then its cardinality is close to optimal.

Lemma *4.2* **Suppose a matching** M **does not admit augmenting paths of length** $2k - 1$ **or less. Then** $|M| \geq \frac{k}{k+1}|M^*|$.

Proof: Let M and M^* denote the current and optimum matchings, respectively. Consider the symmetric difference between M and M^*. It contains $|M^*| - |M|$ node-disjoint augmenting paths with respect to M. Since each of these paths contains at least k edges of M, we have $|M^*| - |M| \leq |M|/k$, or

$$|M| \geq \frac{|M^*|}{1 + 1/k} = \frac{k}{k+1}|M^*|.$$

▮

The above two lemmas imply that the algorithm is correct:

Theorem 4.3 The matching M **found by the algorithm satisfies** $|M| \geq \frac{k}{k+1}|M^*|$.

Next we analyse time and processor requirements of iteration i of the algorithm.

Lemma 4.4 On an EREW PRAM, iteration i of the algorithm runs in $O(\log^3 n)$ time using $O(in^{2i+2})$ processors.

Proof: First we consider the construction of V_A. The number of sequences π generated in the construction of A is $O(n^{i+1})$. Since we assign i processors to each sequence, π, we use $O(in^{i+1})$ processors in the construction.

To construct E_A, we assign i processors to each pair of nodes in V_A. Since the number of nodes in V_A is $O(n^{i+1})$, we can implement this task with $O(in^{2i+2})$ processors. Note also that $|E_A| = O(n^{2i+2})$.

As shown in the previous section, A can easily be constructed in O(log n) time.

The next step of the algorithm finds a maximal independent set in A. Using $O(n^{2i+2})$ processors (linear in the size of A), this can be done in $O(\log^3 n)$ time using the algorithm of Goldberg and Spencer [7]. (Luby's algorithm [12] can also be used, but its deterministic version runs in $O(\log^4 n)$ time.)

The final step of every iteration is the augmentation. It is easy to see that this step can be completed in constant time using no additional processors. ∎

Remark: The processor bound of the above lemma can be improved slightly by balancing the first step of the algorithm (construction of A) with the second step (maximal independent set computation). We can decrease the number of processors used to construct A by a factor of $\log^2$ n. The resulting implementation still runs in $O(\log^3 n)$ time, but the processor requirement is reduced by a factor of $\log^2$ n.

Theorem 4.5 On an EREW PRAM, the algorithm runs in $O(k \log^3 n)$ time using $O(kn^{2k+2})$ processors.

Proof: Immediate from Lemma 4.4. ∎

Corollary 4.6 If k is a constant, the algorithm runs in $O(\log^3 n)$ time using a polynomial number of processors.

References

[1] A. Aggarwal and R. J. Anderson. A Random NC Algorithm for Depth First Search. In *Proc. 19th Annual ACM Symposium on Theory of Computing,* pages 325–334, 1987.

[2] A. Aggarwal, R. J. Anderson, and M.-Y. Kao. Parallel Depth-First Search in General Directed Graphs. *In Proc. 21st Annual ACM Symposium on Theory of Computing,* pages 297–308, 1989.

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

[4] S. Fortune and J. Wyllie. Parallelism in Random Access Machines. *In Proc. 10th Annual ACM Symposium on Theory of Computing,* pages 114–118, 1978.

[5] A. V. Goldberg, S. A. Plotkin, D. Shmoys, and É. Tardos. Interior-Point Methods in Parallel Computation. Technical Report STAN-CS-89-1259, Stanford University, 1989.

[6] A. V. Goldberg, S. A. Plotkin, and P. M. Vaidya. Sublinear-Time Parallel Algorithms for Matching and Related Problems. *In Proc. 29th IEEE Annual Symposium on Foundations of Computer Science,* pages 174–185, 1988.

[7] M. Goldberg and T. Spencer. A New Parallel Algorithm for the Maximal Independent Set Problem. *In Proc. 26th IEEE Annual Symposium on Foundations of Computer Science,* pages 161–165, 1987.

[8] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ Algorithm for Maximum Matching in Bipartite Graphs. *SIAM J. Comput.,* 2:225–231, 1973.

[9] R. M. Karp, E. Upfal, and A. Wigderson. Constructing a Maximum Matching is in Random *NC. Combinatorica,* 6:35–48, 1986.

[10] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Reinhart, and Winston, New York, NY., 1976.

[11] G. F. Lev, N. Pippenger, and L. G. Valiant. A Fast Parallel Algorithm for Routing in Permutation Networks. *IEEE Trans. on Comput.,* C-30:93-100,1981.

[12] M. Luby. Removing Randomness in Parallel Computation without a Processor Penalty. In *Proc. 29th IEEE Annual Symposium on Foundations of Computer Science,* pages 162-173, 1988.

[13] G. L. Miller and J. Naor. Flow in Planar Graphs with Multiple Sources and Sinks. In *Proc. 30th IEEE Annual Symposium on Foundations of Computer Science, 1989.*

[14] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as Easy as Matrix Inversion. *Combinatorica,* pages 105–131, 1987.

[15] R. E. Tarjan. *Data Structures and Network Algorithms.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.