1986

Parallel Independent Set Algorithms for Sparse Graphs

Gregory Shannon

Report Number:

86-634

PARALLEL INDEPENDENT SET ALGORITHMS FOR SPARSE GRAPHS

Gregory Shannon

CSD-TR-634
October 1986

Parallel Independent Set Algorithms for Sparse Graphs

Gregory Shannon[*]

Department of Computer Sciences

Purdue University

West Lafayette, Indiana 47907

October 23, 1986, Version 18

Abstract

In this paper we present algorithms for the independent set problem on n-vertex m-edge graphs using $n + m$ processors and $O(n + m)$ space on the concurrent-read concurrent-write parallel RAM model of computation. If the graph is planar, a maximal independent set with at least $n/14$ vertices can be found in $O(\log^* n \log n)$ time. If Δ is a graph's maximum degree, we show how to find in $O(2^\Delta \log^* n)$ time a maximal independent set with at least $n/(\Delta+1)$ vertices. All previous work on parallel algorithms for the maximal independent set problem has dealt with showing that the problem is in the class NC and has produced algorithms which use at least $n^3/\log^3 n$ processors.

[*]Supported in part by Hewlett-Packard's Faculty Development Program and by the National Science Foundation under grant DCR-8320124.

1 Introduction

Good algorithms for planar graph problems are important since many problems in the computer sciences can be formulated as planar graph problems. Previous work has concentrated on developing efficient planar graph algorithms for sequential models of computation which characterize uniprocessor machines. Multiprocessor machines now feasible are fundamentally different from uniprocessor machines and imply new models of computation, therefore, we need to find good planar graph algorithms for parallel models of computation.

In this paper we discuss parallel algorithms for finding independent subsets of a graph's vertices. A subset of a graph's vertices is *independent* if no pair of vertices in the set are adjacent. An independent set is *maximal* if each of the graph's vertices is independent or adjacent to an independent vertex. An independent set of n-vertex graph is *large* if it has $O(n)$ vertices.

Previous work on finding independent sets in parallel by Karp and Wigderson [KW85] and Luby [Lub85] showed that the maximal independent set problem for general graphs is in the class NC. Their algorithms both use at least $n^3/\log^3 n$ processors for planar graphs. In this paper we utilize the sparsity of planar graphs to create independent set algorithms which use only n processors. That is, we present parallel algorithms which use a number of processors proportional to the size of the input. We know of no past work on n processor algorithms for finding independent sets of planar graphs in polylog time.

Our algorithms are for the parallel RAM (PRAM) model of computation which consists of a common memory and a collection of synchronous processors. Any processor can read from or write to any location in the common memory at any time. When more than one processor writes to the same memory location at the same time, the conflict is resolved arbitrarily. That is, one of the processors arbitrarily succeeds in writing to that location. (This model is also known as the CRCW-arbitrary PRAM, see [SV82,BH85].)

Given an n-vertex m-edge graph, we present $n+m$ processor and $O(n+m)$ space PRAM algorithms which can find:

- A large ($\geq n/14$ vertices) independent set of a planar graph in $O(\log^* n)$ time.
- A large maximal independent set of a planar graph in $O(\log^* n \log n)$ time.
- A maximal independent set of a graph with maximum degree Δ in $O(2^{\Delta} \log^* n)$ time.

2 Independent Set Algorithms

We assume that graphs are represented as adjacency lists [AHU74] in the PRAM's common memory. An independent set of a graph is represented by marking only the independent vertices as independent. It will be clear from the discussion that our parallel algorithms only use $n+m$ processors and $O(n+m)$ space; therefore time complexity will be the only complexity measure discussed. We first solve the problem of finding a large maximum independent set I of an n-vertex planar graph $G = \langle V, E \rangle$.

Let V_6 be G's vertices which are of degree 6 or less. Matula, Shiloach, and Tarjan [MST80] and others demonstrate that V_6 contains at least half of G's vertices. Boyar and Karloff [BK86] record that any maximal independent set of a graph with maximum degree Δ has contains at least $1/(\Delta+1)$ of the graph's vertices. Therefore, any maximal independent set I_6 of $G[V_6]$ contains at least $n/14$ of G's vertices.

Given I_6 we can recurse and find a maximal independent set I' of $G' = G - I_6 - N(I_6)$. $N(I_6)$ is the set of vertices in G adjacent (Neighbors) to some vertex in I_6. The maximal independent set I of G is then $I_6 \cup I'$ and is also large ($\geq n/14$). Since $V_6 \subseteq I_6 \cup N(I_6)$ and $|V_6| \geq n/2$, there are $O(\log n)$ levels of recursion. In our description of how to compute I_6, it will be clear that we need not directly compute G'. Since it will take $O(\log^* n)$ time to compute I_6, our algorithm's time complexity will be $O(\log^* n \log n)$. The linear space and n processor complexities will be obvious.

Before the whole algorithm starts, we make some assumptions. Initially, all vertices are unmarked (free). Using recursive doubling, each edge in a vertex's adjacency list knows exactly where that vertex is represented in the common memory. Using sorting if necessary, we assume that each edge in a vertex's adjacency list knows where in the common memory its opposite edge and therefore other endpoint are.

We now show how to find I_6 of a graph $G = \langle V, E \rangle$ assuming that all vertices are marked either free, independent, or dead. The markings are needed so that all the subgraphs that this algorithm produces can be represented implicitly and accessed in constant time.

We determine which vertices are in V_6 by staging 7 edge elections among edges pointing to free vertices in each free vertex's adjacency list. An edge election is when a set of edges all write their ids to the same location in the common memory and the winning edge is the one whose id is the value that actually gets written to the memory location. If an edge wins an election, then it does not vote in any of the few remaining elections. A vertex is in V_6 if none of its edges pointing to free neighboring vertices vote in the seventh election. An edge is in $G[V_6]$ if its two endpoints are in V_6. All of this is straightforward since concurrent-reads and concurrent-writes are allowed. After constant time, each vertex and edge now knows if it is in $G[V_6]$.

Without loss of generality, we assume that $G[V_6]$ has no isolated vertices. If there are any isolated vertices in $G[V_6]$ they then are in I_6 and can be ignored. A *small tree forest* (STF) of a graph is a subset of the graphs edges such that each connected component of the edges is a rooted tree with height 1. Clearly, there exist 6 edge-disjoint STF's $F_1, \ldots, F_6$ of $G[V_6]$ such that each of $G[V_6]$'s undirected edges is some F_i. Later we show how to construct these 6 STF's in $O(\log^* n)$ time. These forests are represented by assigning a label between 1 and 6 to one of each pair of opposite directed edges in $G[V_6]$.

For now, we assume that we have decomposed $G[V_6]$ into the 6 forests. By computing a 64 coloring of $G[V_6]$ in constant time and then successively adding still independent vertices of the same color to I_6, we can finish finding a maximal independent set of $G[V_6]$ in constant time.

A vertex's color is represented in binary with six bits, $b_5b_4b_3b_2b_1b_0$. If the vertex is a leaf in STF F_i, then bit $b_{i-1} = 1$ and 0 if is a root. A vertex knows if is a leaf a tree in STF F_i if any of its edges in $G[V_6]$ are labeled with i. From color $c = 0$ to 63, mark independent each c colored vertex which is not currently connected by an edge in $G[V_6]$ to some other vertex in $G[V_6]$ already marked independent. Since the vertices of the same color are independent, I_6 is a maximal independent of $G[V_6]$ after all vertices are processed.

In order to recurse and compute I', we need only mark each vertex as dead which is adjacent to an independent vertex. Again, this can be done in constant time by using concurrent reads.

We now show how to decompose $G[V_6]$ into at most 6 STF's $F_1, \ldots, F_6$.

Our approach is to build successive STF's of $G[V_6]$ which are *maximal*. That is, for the forest F_i, any vertex with an edge that was not used by a previous forest is the endpoint of some edge in forest F_i. Since each vertex is of at most degree 6, this approach guarantees that only at most 6 forests are needed or even possible.

Let H be *any* graph with no isolated vertices for which we need to construct a maximal STF F.

We first construct an arbitrary spanning forest F of H in constant time. Each vertex adds to F an edge pointing to a neighbor with a larger id. Now a vertex in H may be isolated in $H[F]$ since it has no neighbor with a larger id and it is not the large neighbor pointed to by any of its neighbors. Therefore, any vertex isolated in $H[F]$ arbitrarily adds to F one of its adjacent edges pointing to a neighbor. F is now a spanning forest of H — there are no isolated vertices or cycles, and each vertex is in F. The trees in F are oriented; a vertex is a root if is larger than all of its neighbors and is pointed to by a neighbor. F takes constant time to construct.

Let us now concentrate on a specific tree T in F. We want to remove edges from T so that each vertex in T is finally in some tree of height one.

In constant time each internal vertex in T marks one of its children's edges pointing to it. M is the set of of all marked edges, and (ignoring isolated vertices) $H[M]$ is a collection of disjoint directed chains. Some leaves of F are not represented in $H[M]$.

A 2-ruling R of a chain C is a subset of C's vertices such that all vertices in R are least two edges apart and each vertex in C but not R is at most two edges away from some vertex in R. Vertices in a 2-ruling are between two and five links away from the closest vertex also in the ruling.

For each chain in $T[M]$ find a 2-ruling which contains that chain's head. R is all the vertices in the 2-rulings of the chains in $T[M]$. Cole and Vishkin [CV86] have shown how to compute a 2-ruling on an n-vertex chain in $O(\log^* n)$ time with n processors and linear space on any of the PRAM models. Since processors are already assigned to the vertices and the sum of the lengths of all chains in F is at most n, clearly at most $O(\log^* n)$ time is needed to find a 2-ruling.

In T, remove any edge which connects a vertex and one its children which is in R but not a leaf of T. In constant time we have then split T into a forest of constant height trees. Each tree now has a height of at most 5 and at least 1.

Again in constant time we can handle each tree as a special case and remove a few more edges to finally produce a break down of T into a maximal forest of small trees. Therefore F has been broken up into a maximal forest of height 1 trees.

Though we have not given the PRAM implementation details of how to find either the STF decomposition or a maximal STF, an implementation within the given complexity bounds is straightforward given the PRAM model and graph representation that we use. Therefore, in $O(\log^* n)$ time we can produce a maximal STF of *any* graph and decompose $G[V_6]$ into 6 STF's.

We have now shown how to compute I_6 in $O(\log^* n)$ time and a maximal independent set of G in $O(\log^* n \log n)$ time. Our discussion of how to compute I_6 immediately implies that we can find an independent set of G with at least $n/14$ vertices in $O(\log^* n)$ time.

Given how we find I_6 with a 2^6 coloring of $G[V_6]$ based on the the STF's $F_1, \dots, F_6$, it easy to see how to 2^Δ color any graph with maximum degree Δ. Clearly only $O(\Delta \log^* n)$ time is needed. Therefore, we can find a *maximal* independent set of the graph in $O(2^\Delta \log^* n)$ time. The set would have at least $n/(\Delta + 1)$ vertices as discussed earlier in the section.

We can find an independent set with at least $n/2^\Delta$ vertices of an arbitrary graph with maximum degree Δ in $O(\Delta \log^* n)$ time if we slightly modify the coloring scheme. Given a tree in STF F_i of the graph's STF decomposition, the leaves' color bit b_{i-1} is a 1 if one of the leaves has bits b_0 to b_{i-2} set to 1. Otherwise, that bit is set to 0 for all those leaves. The b_{i-1} color bit of the root of that tree is the opposite of its leaves. Therefore, at least $n/2^{\Delta}$ vertices have color 63 and are independent.

3 Conclusion

In this paper we have shown how to find independent sets on the strong PRAM model of parallel computation. For an n-vertex graph, all of our algorithms for planar graphs use only n processors and linear space. Our algorithms for arbitrary sparse graphs use processors and space proportional to the number of edges in the graph.

There are other weaker PRAM models of computation upon which we can implement our algorithms. Borodin and Hoptcroft [BH85] describe these additional PRAM models: CRCW-common, CREW, and EREW. Note that Cole and Vishkin's 2-ruling algorithm works the same even on the weakest PRAM model, EREW. On CRCW-common PRAM any $\log^* n$ factor can be replaced with a $\log \log n$ factor since we can hold the elections using min operations as described by Shiloach and Vishkin [SV81]. On both the CREW and EREW PRAMs, the $\log^* n$ factor is replaced with a $\log n$ factor in order to hold the elections. Cole and Vishkin point out that their 2-ruling algorithm can also use $n/\log n$ processors and $O(\log n)$ time. Also, Cole [Col86] has shown that ranking on linked lists can be done with $n/\log n$ processors in $O(\log n)$ time on any PRAM model. Therefore, only $n/\log n$ processors on the CREW and EREW PRAMs are needed for the planar graph problems when we are replacing the $\log^* n$ factor with $\log n$ factor.

Maximal independent set algorithms for planar graphs have direct applications to the vertex and edge coloring of planar graphs. In [Sha86] we show that both of these problems have n processor, polylog time, and linear space algorithms on the PRAM model.

References

[AHU74] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer*

Algorithms. Addison-Wesley, 1974.

[BH85] A. Borodin and J. E. Hoptcroft. Routing, merging, and sorting on parallel models of computation. *Journal of Computer and System Sciences*, 30:130–145, 1985.

[BK86] J. Boyar and H. Karloff. Coloring planar graphs in prallel. July 1986. Unpublished Manuscript.

[CV86] R. Cole and U. Vishkin. Deterministic coin tossing and accelerating cascades: micro and macro techniques for designing parallel algorithms. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, pages 206–219, 1986.

[KW85] R. M. Karp and A. Wigderson. A fast parallel algorithm for the maximal independent set problem. *Journal of the ACM*, 32(4):762–773, October 1985.

[Lub85] M. Luby. A simple parallel algorithm for the maximal independent set problem. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, pages 1–10, 1985.

[MST80] D. Matula, Y. Shiloach, and R. Tarjan. *Two Linear-time Algorithms for Five-coloring a Planar Graph.* Technical Report STAN-CS-80-830, Department of Computer Science, Stanford University, Palo Alto, California, November 1980.

[Sha86] G. Shannon. Coloring planar graphs in parallel. 1986. Unpublished manuscript.

[SV81] Y. Shiloach and U. Vishkin. Finding the maximum, merging, and sorting in a parallel compuation model. *Journal of Algorithms*, 2:88–102, 1981.

[SV82] Y. Shiloach and U. Vishkin. An $O(\log n)$ parallel connectivity algorithm. *Journal of Algorithms*, 3:57–67, 1982.