The system consists of only one type of integrated circuit, its structure being irrespective of the transform size, which considerably reduces the cost of implementation.

Despite a large amount of hardware, the system is easily testable and restructurable.

References

[1] D. F. Elliot and K. R. Rao, *Fast Transforms. Algorithms, Analysis, Applications*. New York: Academic, 1982.

[2] J. W. Cooley and J. W. Tukey, "An algorithm for the machine computation of complex Fourier series," *Math. Comput.*, Apr. 1985.

[3] A. Pomerleau, M. Fournier, and H. L. Buijs, "On the design of a real time modular FFT processor," *IEEE Trans. Circuits Syst.*, vol. CAS-23, pp. 630-633, Oct. 1976.

[4] G. Bongiovanni, "Two VLSI structures for the discrete Fourier transform," *IEEE Trans. Comput.*, vol. C-32, pp. 750-754, Aug. 1983.

[5] A. Sawai, *Programmable LSI Digital Signal Processor Development, VLSI Systems and Computations*, H. T. Kung *et al.*, Eds. Rockville, MD: Computer Science Press, 1981.

[6] N. Kanopolous and P. N. Marinos, "On the architecture of a programmable, high performance single chip Fourier transform processor," in *Proc. Euromicro*, 1984, pp. 319-325.

[7] K. Sapiecha and R. Jarocki, "Modular architecture for high performance implementation of FFT algorithm," in *Proc. 13th Int. Symp. Comput. Architecture*, IEEE, Tokyo, 1986, pp. 261-270.

[8] R. G. Bennets, *Design of Testable Logic Circuits*. Reading, MA: Addison-Wesley, 1984.

[9] K. Sapiecha, "Restructuring of modular FFT architecture," in *Proc. XI Int. Conf. Fault Tolerant Syst. Diagnostics*, Suhl, 1988, pp. 47-53.

[10] L. N. Bhuyan and D. P. Agrawal, "Performance analysis of FFT algorithms on multiprocessor systems," *IEEE Trans. Software Eng.*, pp. 512-521, July 1983.

[11] D. P. Agrawal, "Testing and fault tolerance of multistage interconnection networks," *IEEE Comput. Mag.*, Apr. 1982.

[12] C. D. Thompson, "Fourier transforms in VLSI," *IEEE Trans. Comput.*, pp. 1047-1057, Nov. 1983.

[13] G. Bilardi and M. Sarrafzadeh, "Optimal discrete Fourier transform in VLSI," in *VLSI: Algorithms and Architectures*, P. Bertolazzi and F. Luccio, Eds. New York: Elsevier North-Holland, 1985.

[14] J. Vuillemin, "A combinatorial limit to the computing power of VLSI circuits," in *Proc. 21st Symp. Foundations Comput. Sci.*, IEEE Comput. Soc., Oct. 1980, pp. 294-300.

Parallel Graph Algorithms Based Upon Broadcast Communications

Chang-Biau Yang, R. C. T. Lee, and Wen-Tsuen Chen

Abstract—**In this paper, we shall show some common guidelines which we can use to design parallel algorithms under the single-channel broadcast communication model. We use several graph problems as a vehicle. The graph problems we solve include topological ordering, the connected component problem, breadth-first search, and depth-first search. If an ideal conflict resolution scheme is used, all of our algorithms require $O(n)$ time by using n processors. Under such a situation, our algorithms are all optimal. If a realistic conflict resolution is used, our algorithms require $O(n \log n)$ time by using $n/\log n$ processors. For both cases, all of our algorithms achieve optimal speedups.**

Manuscript received August 15, 1987; revised February 12, 1990. This work was supported in part by the National Science Council of the Republic of China under Contract NSC77-0408-E007-04.

C.-B. Yang and W.-T. Chen are with the Institute of Computer Science, National Tsing Hua University, Hsinchu 30043, Taiwan, Republic of China.

R. C. T. Lee is with the National Tsing Hua University, Hsinchu, and the Academia Sinica, Taipei, Taiwan, Republic of China.

IEEE Log Number 9035959.

I. Introduction

In this paper, we shall solve several graph problems based upon the *single-channel broadcast communication model* which was also used to solve some other problems by Dechter and Kleinrock [3] and Levitan and Foster [9], [10]. This model consists of some processors sharing one common channel for communications. Each processor in this model can communicate with others only through this shared channel. Whenever a processor broadcasts messages, any other processor can hear the broadcast messages. If more than one processor wants to broadcast simultaneously, a *broadcast conflict* occurs. When a conflict occurs, a conflict resolution scheme is used to resolve the conflict. This resolution scheme will enable one of the broadcasting processors to broadcast successfully. We shall use Capetanakis' tree algorithm [1] as our conflict resolution scheme since this algorithm was proved to be the best one for the worst cases [3]. In this tree algorithm, the time required for resolving a conflict is $O(\log p)$ if there are p processors incorporated in the broadcasting multiprocessor system.

The time required for running an algorithm under the broadcast communication model is divided into two parts: communication time and computation time. The *communication time* includes the time for broadcasting messages and the time for resolving conflicts. (We assume that the time required for broadcasting one unit of message is a constant.) Before a processor broadcasts or after a processor hears the broadcast messages, it may perform some tasks in itself. The time for performing these tasks is the computation time. For designing an algorithm based upon the broadcast communication model, we hope that the sum of communication time and computation time is minimized.

Some researchers have worked in the area of designing parallel algorithms with broadcasting multiprocessor systems. Levitan and Foster [9], [10] used a broadcasting protocol multiprocessor model to solve the maximum problem, the sorting problem, and the minimum spanning tree problem. Dechter and Kleinrock [3], Ramarao [14], and Marberg and Gafni [11] also proposed sorting algorithms under the broadcast communication model. For complete reviews of algorithms designed under the broadcast communication model, refer to Yang's Ph.D. dissertation [17].

Since not many papers have been written on algorithms based upon the broadcast communication model, we try to find out whether there exists a general guideline to design parallel algorithms under this model. To achieve this goal, we use several graph problems as a vehicle. That is, by examining the parallel algorithms designed to solve these problems, we hope that we can find some common characteristics existing in all of these algorithms.

Parallel algorithms for graph problems have been discussed by many researchers. For a complete review of parallel graph algorithms, see [13]. Yang's Ph.D. dissertation [17] also listed several tables for comparing the results for solving graph problems in the previously published papers.

In this paper, we use the single-channel broadcast communication model to solve the following graph problems: the *topological ordering* of an acyclic digraph, finding *connected components* of an undirected graph, the *breadth-first search* and the *depth-first search* of a connected undirected graph. Suppose that there are n vertices in a given graph and n processors are used. All algorithms we propose to solve these problems require $O(n)$ time under the assumption that an ideal conflict resolution scheme [3], [9], [10] is used. (An ideal conflict resolution scheme resolves a conflict in a constant time.) Under such a situation, our algorithms are all optimal. For practical consideration, we use Capetanakis' tree algorithm as our conflict resolution scheme. Then all these algorithms

can be slightly modified to use only $n/\log n$ processors and the required time is $O(n \log n)$. Under either the ideal case or the practical case, all of our algorithms achieve optimal speedups. We shall only analyze the complexity of the algorithm for the topological ordering problem and omit the analyses for other problems since the analyses are similar.

After presenting all of the parallel algorithms we design, we then show some common characteristics of these algorithms. These common characteristics can be used as a guideline to design parallel algorithms under the single-channel broadcast communication model.

II. Basic Graph Definitions and Notations

A *graph* $G = (V, E)$ consists of a finite nonempty set V of *vertices* and a finite set E of *edges* in which each element is a pair of vertices. If the pair of vertices associated with an edge is ordered, the graph is called a *directed* graph or a *digraph*. Otherwise, it is an *undirected* graph. v_i is said to be *adjacent* to v_j if $(v_i, v_j) \in E$. In a graph $G = (V, E)$, if $(v_1, v_2), (v_2, v_3), \cdots, (v_{k-1}, v_k) \in E$ and $v_i \neq v_j$ for all $i \neq j$, $1 \leq i, j \leq k$, except v_1 and v_k, then we call $(v_1, v_2), (v_2, v_3), \cdots, (v_{k-1}, v_k)$ a *path* from v_1 to v_k. For the case that $v_1 = v_k$, this path is a *cycle*. A digraph is *acyclic* if there is no cycle in the graph. In a digraph $G = (V, E)$, v_i is a *predecessor* of v_j if $(v_i, v_j) \in E$. A graph $G = (V, E)$, where $|V| = n$, is called a *chained* graph if it has exactly $n - 1$ edges and these edges form an acyclic path. In this paper, graphs are assumed to be without *self-loops* and *parallel edges*.

In this paper, without losing generality, we assume that $|V| = n$ and the vertices $v_1, v_2, \cdots, v_n$ are labeled as $1, 2, \cdots, n$, respectively. The *adjacency matrix* to represent $G = (V, E)$ is of size $n \times n$ and the value of the entry (i, j) is 1 if $(i, j) \in E$ and 0 if $(i, j) \notin E$.

III. Topological Ordering of an Acyclic Digraph

In this section, we shall discuss the topological ordering problem [2], [4], [5], [6], [8], [15] as an example to illustrate a process, which is used in this paper, for solving graph problems under the single-channel broadcast communication model.

In an acyclic digraph $G = (V, E)$, $|V| = n$, for $v_i, v_j \in V$, the problem for finding a topological order is: assign each vertex v_i a unique order number between 1 and n, denoted as ORDER(v_i), such that ORDER(v_j) > ORDER(v_i) if v_i is a predecessor of v_j.

For designing an algorithm under the single-channel broadcast communication model, we should specify the way of storing data, broadcasting rule (indicating which processors can broadcast messages), broadcast messages, computation (in each processor), termination condition (the condition of the algorithm to stop), and the final result (the way to represent the result).

It is assumed that the number of processors is n which is also the number of vertices in the given graph. Each processor stores a vertex. Without losing generality, we assume that processor i stores vertex v_i, which is labeled as i. Additionally, processor i stores the *adjacency vector* of v_i. An adjacency vector is an n-bit vector. For $v_i \in V$, if $(v_j, v_i) \in E$, then the jth bit of the adjacency vector of v_i is 1; otherwise, the jth bit is 0. In fact, the adjacency vector of v_i is the column of the adjacency matrix corresponding to v_i. Processor i can check whether v_j is adjacent to v_i in a constant time by using this adjacency vector.

By the definition of the topological order, there is a straightforward algorithm to obtain an answer [6], [15]. Before numbering a vertex, we first number all of its predecessors. Based upon this idea, our algorithm is as follows. (In the following, the term "broadcast" means "successful broadcast" or "broadcast successfully" unless we state otherwise.)

Algorithm Topological-Order:

1) *The way of storing data*: Mentioned as above.

2) *Broadcasting rule*: A processor can broadcast messages if all the predecessors of the vertex stored in this processor have been broadcast. Each processor broadcasts messages exactly once.

3) *Broadcast messages*: The vertex label.

4) *Computation*: In each processor, mark the vertex which has just been broadcast if it is a predecessor of the vertex stored in this processor.

5) *Termination condition*: All processors have broadcast; in other words, no processor broadcasts.

6) *The final result*: The order of a vertex being broadcast is the order number of this vertex. In other words, if a vertex is in the ith place of the broadcasting sequence, the order number of it is i.

Let us consider the time complexity of the algorithm. We first assume that an ideal conflict resolution scheme [3], [9], [10] is used. This scheme resolves a conflict in a constant time. Clearly, the algorithm requires n iterations if there are n vertices in the given graph. In each iteration, there is one successful broadcast and it requires a constant time for computations in each processor. Therefore, Algorithm Topological-Order solves the topological ordering problem in $O(n)$ time by using n processors.

The time for a sequential algorithm to solve the topological ordering problem for an acyclic digraph with n vertices is at least $\Omega(|E|)$ since the algorithm must check all edges in that graph. In the worst case, $\Omega(|E|) = \Omega(n^2)$. The processor-time product for Algorithm Topological-Order is $O(n^2)$ since n processors are used and the required time is $O(n)$. Therefore, Algorithm Topological-Order achieves an optimal speedup if an ideal conflict resolution scheme is used.

In the following, we shall prove that the lower bound of time complexity for any algorithm to solve the topological ordering problem under the single-channel broadcast communication model is $\Omega(n)$ if the graph has n vertices.

Lemma 1: Let $e_1, e_2, \cdots, e_n$ be the edges of a chained digraph $G = (V, E)$, where $e_1, e_2, \cdots, e_n$ form a path, $|V| = n + 1$ and $|E| = n$. Suppose that these n edges are stored in p processors arbitrarily. The number of edges stored in processor i is n_i, $1 \leq i \leq p$. Let n_j be the maximum among all n_i's, $1 \leq i \leq p$. If $n_j < n/2$, there exists one case such that e_{k-1} and e_k are not stored in the same processor, $2 \leq k \leq n$.

Proof: To prove this lemma, we need only to generate one such case. We arrange these edges into a sequence as $e_1 e_3 e_5 \cdots e_{n-1} e_2 e_4 \cdots e_n$. (Without losing generality, we assume that n is even.) We stored these n edges into the p processors according to the arranged sequence sequentially. For step i, $1 \leq i \leq p$, we store the first n_i edges of the arranged sequence into processor i and then delete these n_i edges from the arranged sequence. After performing this process, we claim if e_k, $2 \leq k \leq n$, is stored in processor i, then e_{k-1} would not be stored in processor i. Two cases should be considered as follows.

Case 1: k is odd. If both e_k and e_{k-1} are stored in processor i, then the subsequence $e_k e_{k+2} \cdots e_{n-1} e_2 e_4 \cdots e_{k-1}$ of the arranged sequence would be stored in processor i. The length of this subsequence is $n/2$. This is impossible since $n_i < n/2$. Thus, processor i cannot store both e_k and e_{k-1}.

Case 2: k is even. For this case, it is similar to case 1.

By both these cases, our claim is correct. Therefore, the proof is complete. Q.E.D.

Lemma 2: Under the single-channel broadcast communication model, the lower bound of the time required for an algorithm using p processors, $p \geq 1$, to find a topological order of a chained digraph $G = (V, E)$, where $|V| = n + 1$ and $|E| = n$, is $\Omega(n)$ under the assumption that the edges are preloaded into the processors and each edge is loaded into only one processor.

Proof: If the results need to be reported, the lemma holds since n successful broadcasts are required and only one channel is available. In the following, we assume that the results do not need to be reported.

In order to assign an order number to each vertex, each edge should be referred at least once. If an edge $(u, v) \in E$ is not referred, then the order numbers of u and v cannot be determined. Suppose that n_i edges are preloaded into processor i. Here, $\sum_{i=1}^{p} n_i = n$ since there are n edges in G. Let n_j be the maximum among all n_i's, $1 \leq i \leq p$. We have to consider two cases as follows.

Case 1: $n_j \geq n/2$. For each edge in processor j, either it is referred by processor j or it is broadcast by processor j and referred by other processors. Thus, processor j must read each edge stored in it at least once. The time required for processor j to read its edges is at least $\Omega(n/2) = \Omega(n)$. Therefore, the lemma holds.

Case 2: $n_j < n/2$. Let $v_0, v_1, v_2, \cdots, v_n$ be the $n+1$ vertices of G where $(v_0, v_1), (v_1, v_2), \cdots, (v_{n-1}, v_n)$ form a path. Let (v_{k-1}, v_k) be denoted as e_k for $1 \leq k \leq n$. Without losing generality, n is assumed to be even. Consider how we determine the order numbers of the following $n/2$ vertices: $v_2, v_4, v_6, \cdots, v_n$. By Lemma 1, there exists such a case that for each pair e_{k-1} and e_k, $2 \leq k \leq n$, they would not be stored in the same processor. Suppose that e_k is stored in processor i. In order to determine the order number of v_k, at least one of the following three possible ways must be done. 1) The edge e_{k-1} is transmitted to processor i from other processors. 2) The order number of v_{k-1} is transmitted to processor i from other processors since processor i does not know e_{k-1}. 3) The edge e_k is transmitted to other processors to determine the order number of v_k. At least $n/2$ successful broadcasts will be required if we want to determine the order numbers of $v_2, v_4, v_6, \cdots, v_n$. Thus, the time required for assigning the order numbers of all vertices is at least $\Omega(n)$ since only one channel is available. Q.E.D.

By Lemma 2, under the single-channel broadcast communication model, the lower bound of the time required for an algorithm using p processors, $p \geq 1$, to find a topological order of an acyclic digraph $G = (V, E)$, $|V| = n$, is $\Omega(n)$ since a chained digraph is only a special case of an acyclic digraph. Thus, Algorithm Topological-Order is optimal under the single-channel broadcast communication model if an ideal conflict resolution scheme is used.

Next, we shall consider the realistic case. We use Capetanakis' tree algorithm [1] as our conflict resolution scheme. In Capetanakis' tree algorithm, it takes $O(\log n)$ time to resolve a broadcast conflict if there are n processors involved in the broadcasting multiprocessor system. Therefore, if Capetanakis' tree algorithm is used, Algorithm Topological-Order requires $O(n \log n)$ time when n processors are used. This means that it does not achieve an optimal speedup.

To make the algorithm achieve an optimal speedup, we can slightly modify our algorithm. Instead of n processors used, we use only $n/\log n$ processors. In this modified algorithm, each processor stores $\log n$ vertices with the corresponding adjacency vectors and simulates the processes of $\log n$ processors in Algorithm Topological-Order. The time complexity of this modified algorithm is $O(n \log n)$ since both computation time and communication time are $O(n \log n)$. Under this modification, the algorithm for the topological ordering problem achieves an optimal speedup since any sequential algorithm of this problem requires at least $\Omega(n^2)$ time and the processor-time product of this modified algorithm is $O(n^2)$.

Algorithm Topological-Order can be extended to solve the critical path problem [5], [6]. For details, refer to the Ph.D. dissertation of Yang [17].

For all other graph algorithms proposed later in this paper, we can use an analysis similar to that used for Algorithm Topological-Order. In addition, all algorithms in this paper have the same feature in terms of time complexity. We shall omit the analysis of time complexity and the "modification" of algorithms in the later sections.

IV. The Connected Component Problem

In this section, we shall discuss the connected component problem [5], [15], [16]. Our connected component problem is the following. For a given undirected graph $G = (V, E)$, we assign a component number to each vertex such that this component number is the smallest vertex label among the vertices in the same connected component.

To use the single-channel broadcast communication model to solve the connected component problem, we again assume that each processor stores a vertex with its adjacency vector. For each processor, it keeps three important parameters. They are p_1, p_2, and p_3. p_1 is the vertex label, p_2 is the previous connected component number which is the connected component number without considering itself, and p_3 is the connected component number with considering itself. Our algorithm is as follows.

Algorithm Connected-Component:

1) Initially, in each processor, $p_1 = p_2 = p_3$. Any processor can broadcast.

2) Whenever a processor broadcasts, the broadcast messages are p_1, p_2, and p_3 of this processor. We call these broadcast parameters b_1, b_2, and b_3 for distinguishing from the parameters in other processors. After b_1, b_2, and b_3 are broadcast, p_1, p_2, and p_3 will be updated in each processor which stores the vertex satisfying one of the following conditions.

i) The vertex is adjacent to the vertex being broadcast.

ii) p_2 of the vertex is equal to b_2.

The updating rules are as follows.

2.a) $p_2 = b_3$.

2.b) $p_3 = b_3$ if $b_3 < p_3$.

3) Only those vertices with at least one neighboring vertex having already been broadcast are allowed to be broadcast in the next time.

4) Each vertex is broadcast exactly once.

5) If no processor can broadcast, then a connected component is identified.

6) After a connected component is entirely found (indicated by an empty broadcasting slot), another connected component is started to be constructed by rules 1)–5) if there exist some vertices which have not been broadcast yet.

Now, we shall prove that Algorithm Connected-Component works correctly.

Lemma 3: After a connected component $C = (V_c, E_c)$ is entirely found by Algorithm Connected-Component, p_3 of each vertex in component C is equal to the smallest number among p_1's of all vertices in component C.

Proof: According to the vertices in component C being broadcast at different times, we partition the vertices into three sets as follows:

$$A_1 = \{v \mid v \in V_c,\ v \text{ has been broadcast}\},$$

$$A_2 = \{v \mid v \in V_c,\ (u, v) \in E_c,\ u \in A_1 \text{ and } v \text{ has not been broadcast yet}\},$$

$$A_3 = \{v \mid v \in V_c,\ v \notin A_1 \text{ and } v \notin A_2\}.$$

Note that the elements of these three sets change when time elapses. We claim that at any time, p_3 of each vertex in A_1 is the smallest number among p_1's of all vertices in A_1. Also, p_2 of each vertex in $A_1 \cup A_2$ is equal to p_3 of any vertex in A_1. We shall prove this claim by induction on the size of A_1.

When $|A_1| = 1$, let $A_1 = \{v_1\}$. The claim is trivially true since for each vertex in component C, $p_1 = p_2 = p_3$ initially, and p_2 of each vertex in A_2 is set to p_3 of v_1 (2.a) of the algorithm).

By hypothesis, assume that the claim is true when $|A_1| \leq k - 1 < |V_c|$. We shall prove that it is true when $|A_1| = k$. Let v_k be the kth vertex added into A_1. Two cases should be considered.

1) Case 1: p_1 of v_k is the smallest one among the p_1's of all vertices in A_1. p_3 of v_k is still equal to p_1 of v_k and it is the smallest one among all p_3's in A_1 since p_3 of v_k is never updated in the previous broadcasts (2.b) of the algorithm). After v_k is broadcast, p_3's of other vertices in A_1 are updated to p_3 of v_k (2.b) of the algorithm) and p_2 of each vertex in $A_1 \cup A_2$ is also updated to p_3 of v_k (2.a) of the algorithm). Therefore, the claim is true when $|A_1| = k$.

2) Case 2: p_1 of v_k is not the smallest one among the p_1's of all vertices in A_1. In this case, p_3 of v_k would have been updated to the smallest p_1 in A_1 in some previous broadcasts (2.b) of the algorithm). Thus, p_3 of v_k is still the smallest one among all p_3's in A_1 (all p_3's in A_1 are equal). In addition, p_2 of each vertex in

$A_1 \cup A_2$ is also updated to p_3 of v_k (2.a) of the algorithm). Therefore, the claim is still true when $|A_1| = k$.

This lemma is true since the claim is true. Q.E.D.

By Lemma 3, it can be easily shown that Algorithm Connected-Component works correctly.

V. Breadth-First Search

In this section, we shall propose an algorithm for finding the order of a breadth-first search of a connected undirected graph [6], [15]. Let $G = (V, E)$ be a connected unidirected graph, where $|V| = n$, and v, v_1, v_2, v_3, and $v_4 \in V$. For given a root (starting) vertex $r \in V$, let $\text{dis}(v)$ denote the length of the shortest path connecting r and v. The problem for us is: for each vertex v, assign the order number of a breadth-first search, which is a unique number between 1 and n, denoted as $\text{BFS}(v)$, to it according to the following rule.

1) If $\text{dis}(v_1) < \text{dis}(v_2)$, then $\text{BFS}(v_1) < \text{BFS}(v_2)$.

2) If $\text{dis}(v_1) = \text{dis}(v_2)$ and there exists a vertex v_3 adjacent to v_1 such that $\text{BFS}(v_3) < \text{BFS}(v_4)$ for each v_4 adjacent to v_2, then $\text{BFS}(v_1) < \text{BFS}(v_2)$.

Note that a breadth-first search defines a breadth-first spanning tree. Our basic idea is to divide the vertices on the same level of the tree into several groups. If two vertices are of the same level and they share the same father vertex, they belong to the same group. For each vertex, we label it with a pair of numbers which consist of a level number and a group number, respectively. Our algorithm is as follows.

Algorithm Breadth-First-Search:

1) Initially, the starting vertex is labeled with (1, 1) and it is the only vertex which can be broadcast,

2) After a vertex v is broadcast, each vertex u adjacent to v which has not been labeled will label itself a level number and a group number according to the following rules:

2.a) If the level number of v is i, then the level number of u is $i + 1$.

2.b) If v is the jth vertex being broadcast among all vertices with the same level number, the group number of u is j.

3) After a vertex v is broadcast, the next one to be broadcast is selected by the following rules:

3.a) Case 1: There exists another vertex w with the same group number on the same level and this vertex has not been broadcast yet. Vertex w can be broadcast in the next time.

3.b) Case 2: The vertices with the same group number on the same level are exhausted. However, there exists another group of vertices on the same level which have not been broadcast yet. This case is indicated by an empty broadcasting slot. The vertices within the next group on the same level can be broadcast in the next time.

3.c) Case 3: The vertices of the level containing v are exhausted. This case can be detected by using a variable to count the number of groups on a level. For this case, the vertices within the first group of the next level can be broadcast in the next time.

4) The breadth-first search order in this graph is the order of the vertices being broadcast.

In the above algorithm, the level number for vertex v is actually $\text{dis}(v) + 1$. In rule 2), when a vertex v is broadcast, it is possible that all its neighbors have been broadcast. Thus, some groups may contain no vertex.

VI. Depth-First Search

In this section, we shall propose an algorithm for the depth-first search (DFS for short) in a connected undirected graph [5], [6], [15], [16]. Our problem for the depth-first search is to assign an order number to each vertex such that these assigned numbers form a legal visiting sequence of a certain depth-first search.

In our algorithm, each processor stores two important parameters and we use these two parameters to guide the searching. They are denoted as p_1 and p_2. p_1 is the order counter. It is increased by one when a vertex is broadcast or say that a vertex is visited. It is initialized by one. At any time, all processors have the same value of p_1. p_2 is the visiting control number. An unvisited vertex whose p_1 and p_2 are equal is allowed to be broadcast in the next time. Our algorithm is as follows.

Algorithm Depth-First-Search:

1) Initially, both p_1 and p_2 of the starting vertex are 1 and for all other vertices, $p_1 = 1$ and $p_2 = 0$. The starting vertex is the only one which can be broadcast for the first time.

2) After a vertex v is broadcast, all processors increase their p_1's by one. Furthermore, if a vertex has not been visited yet and it is adjacent to v, its p_2 is set to be equal to p_1.

3) If there is an empty broadcasting slot, which corresponds to a backtracking, p_2 of each unvisited vertex is increased by one and all p_1's remain unchanged.

4) At any time, all of the processors storing unvisited vertices satisfying $p_1 = p_2$ can broadcast.

5) The order of the vertices being broadcast is the DFS order for this graph.

The depth-first search can be applied to solve many other problems, such as finding cut points [6], [15], bridges [15], and biconnected components [6], [15] in a connected undirected graph. Yang also discussed these problems in his Ph.D. dissertation [17]. Besides, Yang applied the depth-first search to solve the lowest common ancestor problem and the fundamental cycle problem.

VII. A General Guideline for Designing Parallel Algorithms Under the Broadcast Communication Model

We have presented several parallel graph algorithms under the broadcast communication model. After examining these algorithms, we note that they all have the following characteristics.

1) Input data must be stored in an appropriate manner initially in processors.

2) An initial condition should be set. Then the processors satisfying the condition begin to broadcast.

3) Each message broadcast successfully can be heard by all processors. Some computation is then performed in all, or some, processors. The result of the computation determines which processors are able to broadcast next.

4) Each processor broadcasts the same kind of messages.

5) There is a termination condition which is indicated by the fact that no processor broadcasts any more or a predetermined number of broadcasts is exceeded.

Certainly, the broadcast communication model is different from other parallel computation models, such as the shared memory model and the interconnected multiprocessor system [7], [12]. What is the major factor which ensures the successful design of a parallel algorithm under the broadcast communication model? For the shared memory model, we have to maximize the parallelism and we do not have to be concerned with the communication problem because this model has many (or infinite) communication channels. On the other hand, when we design a parallel algorithm under the mesh-connected, or cube-connected, multiprocessor system, we must pay close attention to the data routing problem. Now, for our broadcast communication model, since only one channel is available, the communication scheme should be very carefully designed so that this sole channel is fully utilized. In other words, at any instance, the message broadcast on the channel should be useful for all processors. We may even claim that the success or failure of designing a parallel algorithm under the broadcast communication model critically depends upon whether we can design broadcast messages which will be useful for almost all processors.

For a graph, it is possible that there is an edge between every pair of vertices. That is, in the worst case, each vertex has some kind of relationship with all other vertices. In each parallel algorithm presented in this paper, one vertex and some data related to this vertex are broadcast each time. Since this message may be related to all vertices, the broadcast message is relevant to all processors. Thus, this message can be fully utilized. This is why all of the graph

problems mentioned in this paper can be easily solved under this single-channel broadcast communication model.

VIII. Concluding Remarks

In this paper, we designed several parallel graph algorithms under the single-channel broadcast communication model to investigate some common guidelines to design such kinds of algorithms. The basic idea for our algorithms in this paper is as follows. We store input data into the processors evenly. For a particular problem, some broadcasting rules are set. When the status of a processor satisfies the broadcasting rules at some time, it can broadcast. After a message is broadcast, all other processors can hear the message and use it to change their status. Again, if the status of some processor satisfies the required broadcasting conditions, it can broadcast. As long as such a property can be extracted from a problem, it is very easy to design an algorithm under the single-channel broadcast communication model.

We pointed out in Section VII that the success or failure of designing a parallel algorithm under the broadcast communication model depends critically upon whether the message broadcast each time is relevant to all of the processors or not. If the message is relevant to all processors, then it is fully utilized. We cannot have an efficient parallel algorithm under the broadcast communication model if we cannot achieve this.

Recently, some researchers designed algorithms under the multichannel model [11], [17]. To design algorithms based upon the multichannel broadcast communication model is a direction for future research.

References

[1] J. I. Capetanakis, "Tree algorithms for packet broadcast channels," *IEEE Trans. Inform. Theory*, vol. IT-25, no. 5, pp. 505-515, Sept. 1979.

[2] P. Chaudhuri and R. K. Ghosh, "Parallel algorithms for analyzing activity networks," *BIT*, vol. 26, pp. 418-429, 1986.

[3] R. Dechter and L. Kleinrock, "Broadcast communications and distributed algorithms," *IEEE Trans. Comput.*, vol. C-35, no. 3, pp. 210-219, Mar. 1986.

[4] E. Dekel, D. Nassimi, and S. Sahni, "Parallel matrix and graph algorithms," *SIAM J. Comput.*, vol. 10, no. 4, pp. 657-675, Nov. 1981.

[5] N. Deo, *Graph Theory with Applications to Engineering and Computer Science*. Englewood Cliffs, NJ: Prentice-Hall, 1974.

[6] S. Even, *Graph Algorithms*. Potomac, MD: Computer Science Press, 1979.

[7] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*. New York: McGraw-Hill, 1984.

[8] L. Kucera, "Parallel computation and conflicts in memory access," *Inform. Processing. Lett.*, vol. 14, no. 2, pp. 93-96, Apr. 1982.

[9] S. Levitan, "Algorithms for broadcast protocol multiprocessor," in *Proc. 3rd Int. Conf. Distributed Comput. Syst.*, 1982, pp. 666-671.

[10] S. P. Levitan and C. C. Foster, "Finding an extremum in a network," in *Proc. 1982 Int. Symp. Comput. Architecture*, Austin, 1982, pp. 321-325.

[11] J. M. Marberg and E. Gafni, "Sorting and selection in multi-channel broadcast networks," in *Proc. 1985 Int. Conf. Parallel Processing*, 1985, pp. 846-850.

[12] M. J. Quinn, *Designing Efficient Algorithms for Parallel Computers*. New York: McGraw-Hill, 1987.

[13] M. J. Quinn and N. Deo, "Parallel graph algorithms," *Comput. Surveys*, vol. 16, no. 3, pp. 319-348, Sept. 1984.

[14] K. V. S. Ramarao, "Distributed sorting on local area networks," *IEEE Trans. Comput.*, vol. 37, no. 2, pp. 239-243, Feb. 1988.

[15] E. M. Reingold, J. Nievergelt, and N. Deo, *Combinatorial Algorithms: Theory and Practice*. Englewood Cliffs, NJ: Prentice-Hall, 1977.

[16] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM J. Comput.*, vol. 1, no. 2, pp. 146-160, June 1972.

[17] C. B. Yang, "Parallel algorithms based upon broadcasting communications," Ph.D. dissertation, Institute of Computer and Decision Sciences, National Tsing Hua University, Hsinchu, Taiwan, R.O.C., June 1988.

Sequential Fault Occurrence and Reconfiguration in System Level Diagnosis

Arun K. Somani

Abstract—**In classical system-level diagnosis model, a complex multiprocessor system is characterized to be uniquely diagnosable under the presence of any arbitrary fault set of size up to t. Fault occurrence, however, is usually a sequential process in real life systems, i.e., multiple faults occur one after another. Any faulty location is immediately diagnosed and the system is reconfigured before possibly any further fault occurs. This paper considers such systems which are designed under the assumption of sequential fault occurrences and reconfiguration and characterizes their test interconnection assignment for unique diagnosability. This paper develops a theorem for sequential k/t-diagnosability where the system is allowed to have up to t faults but not more than k of them occur at a time. For most practical cases, k has a value of 1. The t-diagnosability theorem is then a special case of this theorem for $k = t$. The results of this theorem are more useful to design practical systems where the system is reconfigured after every fault is detected and located, and they do not have to satisfy the constraints $n > 2*t$.**

Index Terms—**Diagnosable systems, reconfiguration, sequential fault diagnosis, symmetric invalidation model, system-level diagnosis.**

I. Introduction

In the classical framework for system-level diagnosis model [1], a diagnosable system S consisting of n units is represented as a directed graph $D = (U, E)$. Each unit is represented as a node in the set $U = \{u_1, u_2, \cdots, u_n\}$. Each unit is tested by a subset of other units in the system. A *test-link*, denoted by t_{ij}, means that the unit u_j is tested by the unit u_i and a test result a_{ij} is produced. The complete collection of tests E in S is called the *test connection assignment*. Each unit $u_i \in U$ is assigned a subset $\Gamma(u_i) = \{u_j \mid t_{ij} \in E\}$ to test and is tested by a set of unit $\Gamma^{-1}(u_i) = \{u_j \mid t_{ji} \in E\}$ in order to perform the task of fault diagnosis. The set of units tested by a set $U_k \subseteq U$ is given by $\Gamma(U_k) = \bigcup_{u_i \in U_k} \Gamma(u_i) - U_k$.

Under the symmetrical invalidation model [1] for the interpretation of test outcomes the test results produced by a fault-free tester unit are always assumed to be reliable. On the other hand, faulty testers may produce arbitrary test results. The possible test outcomes under this model are depicted in Fig. 1. Various other models for the interpretation of the test outcomes have been proposed including the asymmetrical invalidation model [3]. The collection of all such test results, called a *syndrome*, is decoded to identify the faulty units.

A fault set F is a set of units which are faulty. A set of syndromes $S(F)$, corresponding to a fault set F, contains those syndromes which are producible by a system in the presence of the fault set F. A fault set is said to be *uniquely diagnosable* in a system with respect to an allowable family of fault sets $\mathcal{F}$ if any syndrome $s_i \in S(F)$ is not producible by the system in the presence

Manuscript received December 1, 1987; revised October 20, 1988.

The author is with the Fault Tolerant Computing Laboratory, Department of Electrical Engineering, University of Washington, FT-10, Seattle, WA 98195.

IEEE Log Number 9035960.