broadcasting of the first packet in a recursive manner. This is because their algorithms have to perform the broadcasting one by one; that is, it can not be performed in a pipeline fashion. In total, broadcasting the m packets can be completed in $O(mn \log_2 n)$ time. In contrast, because of the message nonredundancy in our algorithm, the source node can start to broadcast the next packet while the first two phases of our algorithm are completed. Since performing the first two phases of our broadcasting algorithm takes $O(\log_2 n)$ time, the m packets can be broadcast in $O(m \log_2 n + n \log_2 n)$ time in a pipeline fashion. Hence our nonredundant broadcasting algorithm takes less time and produces less traffic than the redundant ones for broadcasting a stream of packets.

V. Conclusions

In this paper, we proposed a distributed broadcasting algorithm with time and traffic optimum in star graphs on the one-port communication model. By recursively partitioning a star graph into smaller disjoint substar graphs, our algorithm can broadcast a message to all the other nodes in the given n-star in $O(n \log_2 n)$ time. We also showed the traffic improvement of our algorithm over two other algorithms proposed by [3] and [8]. Besides, our algorithm is more efficient than the above algorithms while broadcasting a stream of packets.

References

[1] S. B. Akers, D. Harel, and B. Krishnamurthy, "The star graph: An attractive alternative to the n-cube," in *Proc. 1987 Int. Conf. Parallel Process.*, Aug. 1987, pp. 393–400.
[2] S. B. Akers and B. Krishnamurthy, "A group-theoretic model for symmetric interconnection networks," *IEEE Trans. Comput.*, vol. 38, pp. 555–565, Apr. 1989.
[3] S. G. Akl, K. Qiu, and I. Stojmenovic, "Fundamental algorithms for the star and pancake interconnection networks with applications to computational geometry," *Networks*, vol. 23, no. 4, pp. 215–225, July 1993.
[4] R. Dechter and L. Kleinrock, "Broadcast communications and distributed algorithms," *IEEE Trans. Comput.*, vol. 35, pp. 210–219, Mar. 1986.
[5] S. M. Hedetniemi, S. T. Hedetniemi, and A. L. Liestman, "A survey of gossiping and broadcasting in communication networks," *Networks*, vol. 18, no. 4, pp. 319–349, 1988.
[6] S. L. Johnsson and C.-T. Ho, "Optimum broadcasting and personalized communication in hypercubes," *IEEE Trans. Comput.*, vol. 38, pp. 1249–1268, Sept. 1989.
[7] X. Lin and L. M. Ni, "Multicast communication in multicomputer networks," *IEEE Trans. Parallel, Distrib. Syst.*, vol. 4, pp. 1105–1117, Oct. 1993.
[8] V. E. Mendia and D. Sarkar, "Optimal broadcasting on the star graph," *IEEE Trans. Parallel, Distrib. Syst.*, vol. 3, pp. 389–396, July 1992.
[9] J. P. Sheu, W. H. Liaw, and T. S. Chen, "A broadcasting algorithm in star graph interconnection networks," in *Proc. 1992 Int. Conf. Parallel, Distribut. Syst.*, Hsinchu, Taiwan, Dec. 1992, pp. 204–210.

Performance Characterization of the Tree Quorum Algorithm

Her-Kun Chang and Shyan-Ming Yuan

***Abstract*—The tree quorum algorithm, which logically organizes the sites in a system to a tree structure, is an efficient and fault-tolerant solution for distributed mutual exclusion. In this paper, the performance characteristics of the tree quorum algorithm is analyzed. A refinement algorithm is proposed to refine a logical tree structure by eliminating nodes or subtrees which do not improve the performance. Thus the refined tree performs better than the original.**

***Index Terms*—Distributed mutual exclusion, tree quorum algorithm, availability, communication cost.**

I. Introduction

A distributed system consists of a set of sites which are loosely coupled by a computer network. One advantage of distributed systems is resource sharing. That is the resources in a distributed system can be shared among the sites in the system. Examples of sharable resources are memory, peripheral, CPU, clock, etc. The sites in a distributed system may issue requests to a shared resource at arbitrary time. When two or more sites intended to access the same resource, a conflict occurs. A mechanism is required to synchronize conflicting requests so that at most one site is allowed to access the resource at any time instant. This problem is known as distributed mutual exclusion [1]–[11]. A survey of various algorithms for mutual exclusion can be found in [6] and a simple taxonomy for distributed mutual exclusion algorithms was reported in [7].

A central controller can be used to control mutually exclusive access to a shared resource. All requests intended to the resource are sent to the controller and scheduled by the controller. Using a central controller is simple and easy to implement. However, the controller is vulnerable to site failure. When the controller fails, no access to the resource is allowed, i.e., the entire system is *halted*. It is desirable to reduce the probability that the system is halted by using more than one sites to participate the decision making. For example, majority consensus [11] can be used to achieve mutual exclusion wherein a site is allowed to access the resource if it can get permissions from a majority of all participating sites.

Majority consensus can tolerate at most $N/2$ sites failures, where N is the number of participating sites. On the other hand, the communication overhead is costly, since at least N messages ($N/2$ for request and $N/2$ for reply) are required to be exchanged. Several algorithms try to reduce the communication overhead by imposing logical structures to the sites [1], [4]. The tree quorum algorithm [1], which logically organizes the sites to a tree structure, can reduce the number of messages exchanged to $O(\log N)$ in the best case. In this paper, the performance characteristics of the tree quorum algorithm is analyzed.

The *availability*, which is defined to be the steady-state probability that the system is up (not halted), is usually used to evaluate a distributed algorithm. Another important performance measure for a distributed algorithm is its communication cost. Certainly, the purpose

Manuscript received November 1, 1992; revised December 8, 1993. This work was supported in part by NSC under Grant NSC82-0408-E009-053.

The authors are with Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan 30050.

IEEE Log Number 9411968.

TABLE I
AVAILABILITIES OF THREE LOGICAL TREES

case	$p(X)$	$AVB(L)$	$AVB(R)$	$AVB(X)$
1	0.89	0.75	0.75	0.8963
2	0.90	0.75	0.75	0.9000
3	0.91	0.75	0.75	0.9038

of using more than one sites to participate the decision making is to increase the availability. For a distributed algorithm, however, the communication cost will increase as the number of participating sites increases. In general, increasing availability and reducing cost are two conflicting goals, i.e., a system with higher availability usually has higher cost compared to a system with lower availability.

Can we reduce the communication cost while the availability dose not decrease? This paper shows that the answer is positive for the tree quorum algorithm. We find out that increasing the number of participating sites dose not necessarily increase the availability for the tree quorum algorithm. Giving a logical tree, we can refine the tree structure to improve its performance by eliminating nodes or subtrees that do not increase the availability. An algorithm is proposed to do such refinement. The refinement algorithm can reduce the number of nodes of the tree while the availability of the refined tree is not less than the original.

The remainder of this paper is organized as follows. The tree quorum algorithm is reviewed in the next section. In Section III, a refinement algorithm for binary tree is proposed. Numerical results of the refinement algorithm are shown in Section IV. We extend the algorithm to arbitrary tree structures in Section V and some concluding remarks are given in the final section.

II. TREE QUORUM ALGORITHM

The tree quorum algorithm logically organizes the sites in a system as a tree structure. For a binary tree, a tree quorum (recursively) consists of

1) the root and a tree quorum of the left subtree, or
2) the root and a tree quorum of the right subtree, or
3) a tree quorum of the left subtree and a tree quorum of the right subtree.

It was shown that any pair of tree quorums intersect with each other [1]. Thus mutual exclusion is ensured by requiring that an access request to get permissions from nodes which form a tree quorum.

The availability of a node is the probability that the node is operational at any time instant. For the tree quorum algorithm, the *availability* of a (logical) tree is the probability that at least one tree quorum can be formed from the tree. Thus the availability of a binary tree is the probability that

1) the root is operational and a tree quorum can be formed from the left subtree, or
2) the root is operational and a tree quorum can be formed from the right subtree, or
3) a tree quorum can be formed from the left subtree and a tree quorum can be formed from the right subtree.

Every node X in a binary tree contains the following four fields:

- $\boldsymbol{p(X)}$: the availability of the node X,
- $\boldsymbol{AVB(X)}$: the availability of the subtree rooted at X,
- $\boldsymbol{LEFT(X)}$: the root of the left subtree of X ($LEFT(X) =$ nil if X has no left subtree),
- $\boldsymbol{RIGHT(X)}$: the root of the right subtree of X ($RIGHT(X) =$ nil if X has no right subtree).

If a tree consists of only one node, it degenerates to a central controller and the availability of the tree is the availability of itself, i.e., $AVB(X) = p(X)$. For a node X having both left and right subtrees,

$$\begin{aligned} AVB(X) = &\, p(X)AVB(LEFT(X))AVB(RIGHT(X)) \\ &+ p(X)AVB(LEFT(X)) \\ &\cdot (1 - AVB(RIGHT(X))) \\ &+ p(X)(1 - AVB(LEFT(X))) \\ &\cdot AVB(RIGHT(X)) \\ &+ (1 - p(X))AVB(LEFT(X)) \\ &\cdot AVB(RIGHT(X)). \end{aligned} \tag{1}$$

Let $AVB(\text{nil}) = 0$, (1) is valid for any nonleaf node X.

Let $A_l = AVB(LEFT(X))$ and $A_r = AVB(RIGHT(X))$, (1) can be written as following formulas:

$$\begin{aligned} AVB(X) = &\, p(X)(1 - (1 - A_l)(1 - A_r)) \\ &+ (1 - p(X))A_l A_r \\ = &\, A_l(1 - (1 - p(X))(1 - A_r)) \\ &+ (1 - A_l)p(X)A_r \\ = &\, A_r(1 - (1 - p(X))(1 - A_l)) \\ &+ (1 - A_r)p(X)A_l. \end{aligned} \tag{2}$$

III. REFINEMENT ALGORITHM FOR BINARY TREES

Let X be the root of a binary tree attached with left and right subtrees rooted L and R, respectively. We can compute $AVB(X)$ from $p(X), AVB(L), AVB(R)$ by (1).

Table I shows threes examples, which give us the motivation of refinement. In all cases, $AVB(L) = AVB(R) = 0.75$ (the subtrees rooted at L and R consist of one or more nodes). In cases 1, 2 and 3, $p(X) = 0.89$, 0.90 and 0.91, respectively. By (1), $AVB(X) = 0.8963$, 0.9000 and 0.9038, respectively. In case 1, $AVB(X)$ is higher than $p(X), AVB(L)$ and $AVB(R)$. That is, the availability of the whole tree is higher than any of the three parts: root, left subtree and right subtree. In case 2, $AVB(X) = p(X)$, i.e., the availability of the whole tree is equivalent to a single node (the root). The availability of the tree dose not increase but the cost increases. It is worse in case 3 that $AVB(X) < p(X)$. The availability of the whole tree is even lower than a single node (the root). In cases 2 and 3, a larger tree structure does not increase the availability but increase the cost. It is better to eliminate the subtrees to reduce the communication cost since the availabilities are not greater than the original.

Let $\{a_1, a_2, a_3\} = \{p(X), AVB(LEFT(X)), AVB(RIGHT(X))\}$ such that $a_1 \geq a_2 \geq a_3$, by (2),

$$AVB(X) = a_1(1 - (1 - a_2)(1 - a_3)) + (1 - a_1)a_2 a_3. \tag{3}$$

From (3), $AVB(X) > a_1$ if and only if the following condition is satisfied

$$(1 - a_1)a_2 a_3 > a_1(1 - a_2)(1 - a_3) \tag{4}$$

Condition (4) recommends that if $(1-a_1)a_2a_3 \leq a_1(1-a_2)(1-a_3)$, it is better to eliminate the nodes or subtrees corresponding to a_2 and a_3, and replace the tree with the node or subtree corresponding to a_1. The idea can be used to refine a tree to improve its performance. For example, if $a_2 = a_3 = 0.75$, as the cases in Table I, $AVB(X) \leq a_1$ when $a_1 \geq 0.9$.

Fig. 1 is an example of refinement. The original tree and final refined tree are shown in Fig. 1(a) and (c), respectively. Fig. 1(b) shows a partial refinement of the original tree (partially refined tree).

The original tree rooted at A is attached with left and subtrees rooted at B and C, respectively. Node B has children D, E and

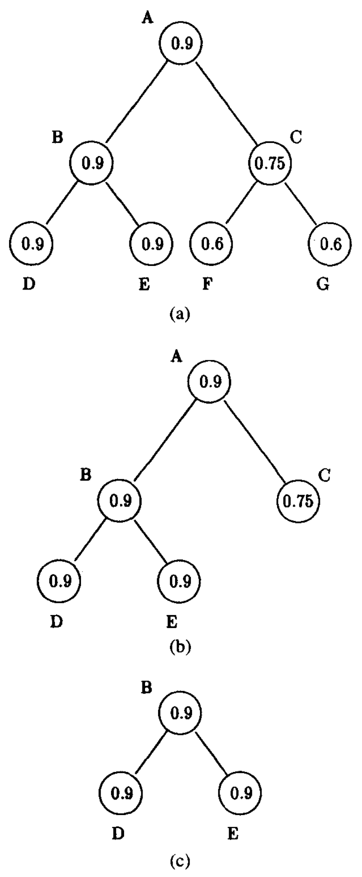

Fig. 1. Refinement example 1.

node C has children F, G. In the original tree, $AVB(B)$ is higher than $p(B), AVB(D)$ and $AVB(E)$. On the other hand, $AVB(C)$ is lower than $p(C)$. Thus it is better to eliminate nodes F and G so that $AVB(C) = p(C)$. This (partial) refinement is shown in Fig. 1(b).

In the partially refined tree, $AVB(A) < AVB(B)$. Thus it is better to replace the tree with the subtree rooted at B, i.e., the subtree tree rooted at B becomes the refined tree.

The number of nodes of the trees in Fig. 1(a)–(c) are 7, 5 and 3, respectively. The availability of the trees in Fig. 1(a)–(c) are 0.9629, 0.9666 and 0.972, respectively. The refined tree has higher availability, less number of nodes and thus lower communication cost than the original.

Fig. 2 is another example of refinement. The tree in Fig. 2(b) is refined (partially) from the tree in Fig. 2(a) and the tree in Fig. 2(c) is refined from the tree in Fig. 2(b). As the example in Fig. 1, the refined tree has higher availability and less number of nodes compared to the original.

The refinement examples in Figs. 1 and 2 recommends that it is not necessary to construct the logical tree using all the sites in the system.

The refinement algorithm for a given binary tree rooted at X is presented in Fig. 3. Giving a logical tree, the refinement algorithm visits the nodes of the tree (recursively) from left to right subtrees and then the root (that is in postordering) and reconstructs it, if necessary, by eliminating the nodes or subtrees that do not increase the availability of the tree.

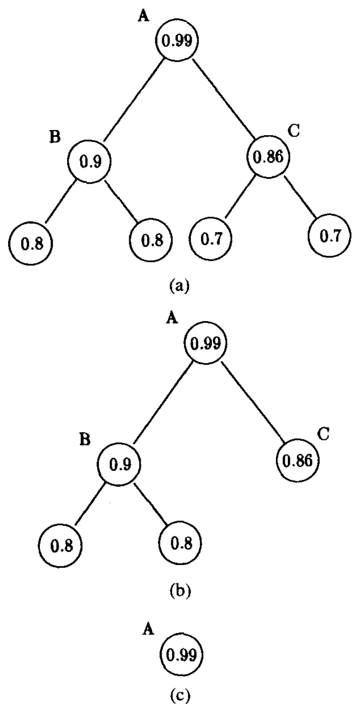

Fig. 2. Refinement example 2.

Lemma 1: $AVB(X)$ is an increasing function of a_1, a_2, a_3.

Proof: Let $\{i_1, i_2, i_3\} = \{1, 2, 3\}$, from (1),

$$\begin{aligned}AVB(X) &= a_{i_1}(a_{i_2}a_{i_3} + a_{i_2}(1 - a_{i_3}) + (1 - a_{i_2})a_{i_3}) \\ &\quad + (1 - a_{i_1})a_{i_2}a_{i_3} \\ &= a_{i_1}(a_{i_2}(1 - a_{i_3}) + (1 - a_{i_2})a_{i_3}) + a_{i_2}a_{i_3}\end{aligned}$$

which is an increasing function of a_{i_1}. Since i_1 can be anyone in $\{1, 2, 3\}$, thus $AVB(X)$ is an increasing function of a_1, a_2, a_3. Q.E.D.

Theorem 1: The availability of a binary tree after refinement by algorithm REFINE is not less than the original.

Proof: For any node X in the binary tree, let $X' = REFINE(X)$. We show, by induction, that $AVB(X') \geq AVB(X)$,

1) *Induction Basis*: If X is a leaf node, $AVB(X') = AVB(X) = p(X)$.
2) *Induction Hypothesis*: If X is a nonleaf node attached with left and right subtrees rooted at L and R, respectively. Let $L' = REFINE(L)$ and $R' = REFINE(R)$. The hypothesis of the induction is that $AVB(L') \geq AVB(L)$ and $AVB(R') \geq AVB(R)$.
3) *Induction Step*: Note a node will not be refined till its both subtrees have been refined. Let $AVB(X)$ be the availability of the subtree rooted at X attached with L and R (i.e. before L and R have been refined). Let $AVB(\hat{X})$ be the availability of the subtree rooted at X attached with L' and R' (i.e., after L and R have been refined). According to Lemma 1 and the hypothesis of the induction, $AVB(\hat{X}) \geq AVB(X)$.

Let a_1 be the maximum in $\{p(X), AVB(L'), AVB(R')\}$. If $AVB(\hat{X}) > a_1, X$ is returned with L' and R' still attached to X; otherwise, the node or subtree corresponding to a_1 is returned

TABLE II
DATA OF COMPARISON

case	availabilities of nodes
A	0.92, 0.90, 0.90, 0.87, 0.82, 0.74, 0.73, 0.67, 0.66, 0.64, 0.62, 0.59, 0.58, 0.57, 0.57
B	0.97, 0.97, 0.94, 0.92, 0.91, 0.87, 0.72, 0.69, 0.66, 0.63, 0.62, 0.61, 0.59, 0.51, 0.51
C	0.97, 0.93, 0.90, 0.82, 0.70, 0.68, 0.64, 0.63, 0.59, 0.59, 0.56, 0.55, 0.54, 0.53, 0.52
D	0.95, 0.94, 0.93, 0.81, 0.73, 0.71, 0.69, 0.66, 0.65, 0.63, 0.56, 0.55, 0.55, 0.54, 0.52
E	0.99, 0.92, 0.86, 0.79, 0.77, 0.74, 0.73, 0.71, 0.69, 0.66, 0.65, 0.65, 0.57, 0.52, 0.52
F	0.86, 0.85, 0.82, 0.77, 0.76, 0.73, 0.67, 0.67, 0.60, 0.56, 0.56, 0.56, 0.51, 0.51, 0.51
G	0.75, 0.74, 0.72, 0.70, 0.70, 0.67, 0.65, 0.65, 0.64, 0.62, 0.60, 0.58, 0.56, 0.54, 0.52
H	0.98, 0.85, 0.79, 0.74, 0.74, 0.73, 0.71, 0.71, 0.71, 0.68, 0.68, 0.67, 0.66, 0.59, 0.53
I	0.76, 0.75, 0.72, 0.67, 0.66, 0.63, 0.57, 0.57, 0.50, 0.46, 0.46, 0.43, 0.41, 0.41, 0.40
J	0.65, 0.64, 0.62, 0.60, 0.60, 0.57, 0.55, 0.55, 0.54, 0.52, 0.50, 0.48, 0.46, 0.44, 0.42

TABLE III
COMPARISON FOR THE ORIGINAL AND REFINED TREES

case	original availability	refined availability	refined number of nodes	availability improvement	node saving
A	0.9755	0.9839	5	0.9 %	47 %
B	0.9922	0.9974	7	0.5 %	33 %
C	0.9714	0.9883	3	1.7 %	80 %
D	0.9757	0.9897	3	1.4 %	80 %
E	0.9873	0.9901	11	0.3 %	27 %
F	0.9274	0.9487	7	2.3 %	33 %
G	0.8630	0.8721	11	1.1 %	27 %
H	0.9807	0.9824	13	0.2 %	13 %
I	0.7591	0.8466	5	11.5 %	47 %
J	0.6660	0.7126	5	7.0 %	47 %

```
Algorithm REFINE(X)
begin
    if $X = nil$ then return($X$)
    $L = REFINE(LEFT(X))$
    $R = REFINE(RIGHT(X))$
    $max = X$
    $AVB(X) = p(X)$
    if $L \neq nil$
        $A_l = AVB(L)$
        if $A_l > AVB(max)$ then $max = L$
    else
        $A_l = 0$
    end_if
    if $R \neq nil$
        $A_r = AVB(R)$
        if $A_r > AVB(max)$ then $max = R$
    else
        $A_r = 0$
    end_if
    $av = p(X)(1-(1-A_l)(1-A_r)) + (1-p(X))A_l A_R$
    if $av > AVB(max)$
        $AVB(X) = av$
        return($X$)
    else
        $LEFT(X) = nil$
        $RIGHT(X) = nil$
        return($max$)
    end_if
end (REFINE)
```

Fig. 3. Refinement algorithm.

and the others are deleted. Let $X' = REFINE(X)$, then $AVB(X') \geq AVB(\hat{X}) \geq AVB(X)$. Q.E.D.

IV. NUMERICAL RESULTS

In this section, we show several numerical results of the refinement algorithm. The comparison is based on full binary trees of depth 4. We choose full binary trees since the definition is well known. Otherwise, $n-1$ links or an $n \times n$ matrix is required to describe an arbitrary tree of n nodes. We choose depth 4 because depth 3 is too small and depth 5 is too large for demonstration.

The availabilities of nodes, shown in Table II, are generated randomly. The values are sequentially assigned to the nodes, starting with the root (level 1), then nodes on level 2 and so on. Nodes on any level are assigned from left to right.

We compare the refined tree with the original tree for the availability and the number of nodes of the trees. The results are shown in Table III. In each case, 'node saving' shows the cost saved compared to the original. Note that, in cases I and J, there are nodes having availabilities <0.5 and the availability improvements are more significant than the other cases. The reason is that if $a_3 \leq 0.5$, condition (4) is always not satisfied. It is shown that the refinement algorithm can significantly reduce the cost as well as increase the availability. Thus the refined tree performs better than the original.

V. AN EXTENSION FOR ARBITRARY TREES

In this section, an extended refinement algorithm is briefly described for arbitrary trees. For an arbitrary tree, a tree quorum (recursively) consists of

1) the root and a tree quorum of any subtree, or
2) tree quorums of all subtrees.

For a nonleaf node X having subtrees rooted at $C_1, C_2, \cdots, C_m$, the availability of the subtree rooted at X can be computed by the following formula:

$$\begin{aligned} AVB(X) = & p(X)(1-(1-AVB(C_1))(1-AVB(C_2))\cdots \\ & (1-AVB(C_m))) \\ & + (1-p(X))AVB(C_1)AVB(C_2)\cdots \\ & AVB(C_m) \end{aligned} \quad (5)$$

Let a_1 be the maximum in $\{p(X), AVB(C_1), AVB(C_2),\cdots, AVB(C_m)\}$, the refinement algorithm can be extended in such a way that if $AVB(X) \leq a_1$ then the node or subtree corresponding to a_1 is returned (and the others are deleted); otherwise X is returned with all subtrees still attached to X. An extended version of Theorem 1 for arbitrary trees can be proved in a similar way.

VI. Conclusions

A central controller can be used to solve distributed mutual exclusion. The drawback is that it is vulnerable to site failure. When the controller fails, the entire system halts. Using more than one sites to participate the decision making may reduce the probability that the system is halted. However, the communication cost usually increases as the number of participating sites increases. In general, increasing availability and reducing cost are two conflicting goals, i.e., a system with higher availability usually has higher communication cost compared to a system with lower availability. Can we reduce the communication cost while the availability dose not decrease? We find out that the answer is positive for the tree quorum algorithm.

In this paper, a refinement algorithm is proposed to refine a logical tree structure by eliminating the nodes or subtrees that do not increase the availability. Giving a tree, the refinement algorithm can reduce the number of nodes of the tree while the availability of the refined tree is not less than the original. Reducing the number of nodes can reduce the communication cost and the cost of maintaining the tree. Thus, the refined tree is preferred because it can significantly save the cost of communication and maintenance.

Acknowledgment

The authors would like to thank the anonymous referees for their helpful comments, which contributed to the quality of this paper significantly.

References

[1] D. Agrawal and A. El Abbadi, "An efficient and fault-tolerant solution for distributed mutual exclusion," *ACM Trans. Comput. Syst.*, vol. 9, no. 1, pp. 1–20, 1991.

[2] H. Garcia-Molina and D. Barbara, "How to assign votes in a distributed system," *J. ACM*, vol. 32, no. 4, pp. 841–860, 1985.

[3] T. Ibaraki and T. Kameda, "A theory of coteries: Mutual exclusion in distributed systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, pp. 779–794, 1993.

[4] M. Maekawa, "A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems," *ACM Trans. Comput. Syst.*, vol. 3, no. 2, pp. 145–159, 1985.

[5] M. L. Nelisen and M. Mizuno, "Coterie join algorithm," *IEEE Trans. Parallel Distrib. Syst.*, vol. 3, pp. 582–590, 1992.

[6] M. Raynal, *Algorithms for Mutual Exclusion.* Cambridge, MA: M.I.T., 1986.

[7] ——, "A simple taxonomy for distributed mutual exclusion algorithms," *Oper. Syst. Rev.*, vol. 25, no. 2, pp. 47–50, 1991.

[8] G. Ricart and A. K. Agrawala, "An optimal algorithm for mutual exclusion in computer networks," *Commun. ACM*, vol. 5, no. 3, pp. 284–299, 1987.

[9] B. A. Sanders, "The information structure of distributed mutual exclusion algorithms," *ACM Trans. Comput. Syst.*, vol. 24, no. 1, pp. 9–17, 1981.

[10] I. Suzuki and T. Kasami, "A distributed mutual exclusion algorithm," *ACM Trans. Comput. Syst.*, vol. 5, no. 3, pp. 284–299, 1987.

[11] R. H. Thomas, "A majority consensus approach to concurrency control for multiple copy databases," *ACM Trans. Database Syst.*, vol. 4, no. 2, pp. 180–209, 1979.

Integer Programming for Array Subscript Analysis

Jaspal Subhlok and Ken Kennedy

***Abstract*—We present a new method to determine whether a convex region contains any integer points. The method is designed for array subscript analysis in parallel programs. The general problem is whether a system of linear equalities and inequalities has an integer solution. A set of known techniques is used to transform the problem to that of finding whether a convex region contains any integer points. The main result of the paper is a set of new search procedures that identify an integer solution in a convex region, or prove that no integer solutions exist. They are based on the geometrical properties of convex regions that are not empty, but also do not contain any integer points. The results contribute to exact and efficient dependence and synchronization analysis of parallel programs.**

***Index Terms*—Subscript analysis, dependence testing, integer programming, parallelizing compilers, parallel program analysis, synchronization analysis.**

I. Introduction

Several mathematical problems that arise in the development of parallelizing compilers can be transformed to integer programming problems. Determining whether a data dependence exists between two array references can be transformed to the problem of determining whether an integer solution to a set of linear equalities and inequalities exists [1], [13]. Similarly, in an event variable synchronization model, determining whether the synchronization present in a program is sufficient to protect a dependence can be transformed to the problem of determining whether a system of linear equations has a non-negative integer solution [3]. Both these problems are instances of integer programming, which is known to be NP-complete.

Solving Diophantine equations and Fourier-Motzkin elimination are well known techniques that have been used by researchers to solve integer programming problems in parallelizing compilers. But they are not sufficient to solve the problem exactly; in the worst case it is necessary to search a convex region for integer points, which is the subject of this paper. We also illustrate how the various methods are combined to form an efficient solution procedure. For details on the context in which the results of this paper are applicable, the reader is referred to [9], [10].

Our approach is oriented towards integer programming problems that are solved in a parallelizing compiler, where the number of equations and variables is tied to the loop nesting depth and the number of subscripts in array references, both of them typically small integers. The small size of problem instances makes it possible to develop an efficient solution procedure even though the general problem is NP-complete. The main results in the paper are for problems in two dimensions. Since a two dimensional problem is obtained from the simplest problem that is not directly solved by other methods, it is likely to occur most often, and we believe

Manuscript received December 21, 1993; revised July 11, 1994. This research was sponsored in part by the Advanced Research Projects Agency/CSTO monitored by SPAWAR under Contract N00039-93-C-0152, and in part by the National Science Foundation under Grant CCR-912008.

J. Subhlok is with the School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213 USA.

K. Kennedy is with the Department of Computer Science, Rice University, Houston, TX 77251 USA.

IEEE Log Number 9411970.