# UNBOUNDED SPEED VARIABILITY IN DISTRIBUTED COMMUNICATION SYSTEMS*

John Reif and Paul Spirakis
Aiken Computation Laboratory
Division of Applied Sciences
Harvard University, Cambridge, Massachusetts 02138

ABSTRACT

This paper concerns the fundamental problem of synchronizing communication between distributed processes whose speeds (steps per real time unit) vary dynamically. Communication must be established in matching pairs, which are mutually willing to communicate. We show how to implement a distributed local scheduler to find these pairs. The only means of synchronization are boolean "flag" variables, each of which can be written by only one process and read by at most one other process.

No global bounds in the speeds of processes are assumed. Processes with speed zero are considered dead. However, when their speed is nonzero then they execute their programs correctly. Dead processes do not harm our algorithms' performance with respect to pairs of other running processes. When the rate of change of the ratio of speeds of neighbour processes (i.e., relative acceleration) is bounded, then any two of these processes will establish communication within a constant number of steps of the slowest process with high likelihood. Thus our implementation has the property of achieving relative real time response. We can use our techniques to solve other problems such as resource allocation and implementation of parallel languages such as CSP and ADA. Note that we do not have any probability assumptions about the system behavior, although our algorithms use the technique of probabilistic choice.

## 1. INTRODUCTION

Recently, [Reif, Spirakis, 1981] showed how to achieve interprocess communication with real time response using probabilistic synchronization techniques, assuming that the speeds of all processes were bounded between fixed nonzero bounds. This lead to (see Appendices I and II of [Reif, Spirakis, 1981]) real time resource allocation algorithms and real time implementation of message passing in CSP.

In this paper we assume *no global bounds on the processors' speeds*. They can vary dynamically from zero to an upper bound which may be different for each processor, and not known by the other processors. We allow a possibly infinite number of processes, so that there may not be a global upper bound on the speeds. Processes may die (have zero speed) but when they have nonzero speed then we assume they execute their programs correctly. We are interested in *direct* interprocess communication (rather than packet switching) which is of the form of *handshake* (rather than buffered), as in Hoare's CSP, [Hoare, 1978]. The essential technique that we utilize is that of *probabilistic choice*. This technique, introduced to synchronization problems by [Rabin, 1980], [Lehmann and Rabin, 1981] and [Francez, Rodeh, 1980], was also utilized in our previous work.

The use of probabilistic choice in the algorithms leads to considerable improvements in the space and time efficiency [Rabin, 1980], [Reif, Spirakis, 1981]; we feel that this may be because of the locality of the decisions and because complex sequences of processes' steps prohibiting communication have very low probability of occurrence. Of course, we assume that each process makes probabilistic choices independent of other processes. We also introduce new *adaptive techniques*: The processes estimate the speeds of neighbour processes and select them to communicate with probabilities depending on the speeds, penalizing the slowest processes. These adaptive techniques do not seem to have ever been utilized in the previous synchronization literature.

This paper proposes a new high level synchronization construct associated with interprocess communication. The construct is implemented very efficiently by the use of boolean flag variables. We do not use any standard high level synchronization construct (such as shared variables with a mutual exclusion mechanism) since these have no known efficient implementations. (There is not even any known bounded time implementation of a mutual exclusion mechanism when processes run on different processors).

*This work was supported in part by the National Science Foundation Grant NSF-MCS79-21024 and the Office of Naval Research Contract N00014-80-C-0674.

If processes are bounded in speed then it is natural to define real time response to be a response to a communication request that uses no more than constant number of units of real time. This measure is inapplicable in our case in which there is no global upper bound and no nonzero lower bound on speeds. Thus we introduce the notion of *relative real time response* which is establishment of communication between *any pair* of neighbouring processes within constant number of local rounds. (A *local round* of neighbour processes, i,j is the minimum time interval which contains at least one step of each process and exactly one step of the slowest of i,j). We achieve this by our probabilistic algorithms with some probability of error which can be made arbitrarily low. We conjecture that it is not possible to achieve relative real time response without use of randomization. The best *deterministic* symmetric algorithms which attempt to form matchings in distributed systems have a relative response depending linearly on the network's diameter.†

The paper is organized as follows: In the next section we define our model for communication. In Section 3 we discuss applications of this model. In Section 4 we give a relative real time implementation of communication in this model. In Section 5 we give correctness properties of our proposed implementation and time analysis.

## 2. THE MODEL VS-DCS (Variable Speed Distributed Communication System)

### 2.1 The Model

We develop here a theoretical model related to, but more general than, the DCS (Distributed Communication System) of [Reif, Spirakis, 1981]. A detailed description of the fundamental issues can be found in [Reif, Spirakis, 1981].

We assume a possibly infinite collection of processes $\pi = \{1,2,\ldots\}$. Events of the system are totally ordered on the real-time line $[0,\infty)$. Each process consists of a fixed set of synchronous parallel subprocesses (i.e., with same speeds). The processes wish at various times to communicate with other processes but have no means of communication except via the communication system. This is implemented by many *poller* subprocesses (five for each target process) which are synchronous with themselves. We assume a fixed connections graph H which is undirected and has the set $\pi$ as its vertex set. An edge $\{i,j\}$ indicates that process i is physically able to communicate with process j (but not necessarily willing to). H is assumed to have finite valence. We also assume for each time t the *willingness digraph* $G_t$ which indicates the willingness of a given process i at a given time t. (We indicate this by the edge $i \xrightarrow[t]{} j$ and say i is a *willing neighbour* of j). Note that $i \xrightarrow[t]{} j$ only if $\{i,j\} \in H$. Let $i \xleftrightarrow[t]{} j$ if $i \xrightarrow[t]{} j$ and $j \xrightarrow[t]{} i$. The edges of the graph $G_t$ are stored distributedly so that the edges departing from a given process are only known to that process. We assume that the out-degree of each vertex of $G_t$ is upper bounded by a fixed constant v.

For each $t \geq 0$ the (possibly infinite) digraph $M_t$ with vertices $\pi$ and directed edges $i \underset{t}{\rightsquigarrow} j$ denotes which processes *open communication* to which other processes at time t. We denote $i \underset{t}{\leftrightsquigarrow} j$ if both $i \underset{t}{\rightsquigarrow} j$ and $j \underset{t}{\rightsquigarrow} i$. Thus $i \underset{t}{\leftrightsquigarrow} j$ denotes i,j *achieve mutual communication* at time t. $M_t$ is the digraph that implementations of distributed synchronization achieve. We assume that

(A1) Two way communication between any two processes $i,j \in \pi$ requires only one step of i and j. (Thus, processes communicate in short "bursts").

(A2) If $i \xrightarrow[t_1]{} j$ and not $i \xrightarrow[t_2]{} j$, $t_2 > t_1$, then $i \underset{t}{\leftrightsquigarrow} j$ for some t such that $t_1 \leq t < t_2$; i.e. processes can withdraw willingness to communicate only after communication has been established.

We wish implementations to be *proper* in the sense that

(a) $i \underset{t}{\rightsquigarrow} j$ only if $i \xleftrightarrow[t]{} j$ (neighbours try to speak only if they are mutually willing to)

(b) $\underset{t}{\leftrightsquigarrow}$ must be a partial matching: If $i \underset{t}{\leftrightsquigarrow} j$ then not $j' \underset{t}{\leftrightsquigarrow} i$ for any $j'$ in $\pi-\{j\}$. (No process is allowed to achieve communication with more than one neighbour at the same time.)

We assume that processes can suddenly die (i.e., have zero speeds) but when they are awake they execute their programs correctly. We furthermore assume that each process has a fixed upper bound on its speed which may be different from the other processes and not known to them.

We define the *relative acceleration bound* $\alpha$ of processes i and j to be the worst case absolute rate of change of the ratio of steps of the two processes per step of any of the two processes. The correctness of our synchronization algorithms does not depend on whether processes are acceleration bounded, however, we assume fixed acceleration bound $\alpha$ in our time complexity analysis (i.e., the relative acceleration of one neighbour with respect to another is bounded by a constant $\alpha$ or can be $-\infty$ if the process dies).

†(Also, [Arjomandi, Fischer, Lynch, 1981] have actually shown that some synchronization problems which are global (in contrast to our problem) cannot be done in real time and require time proportional to the logarithm of the total number of processors in the network. A typical situation where this could occur is the problem of detecting connected components of processes whose speeds are within given bounds, e.g., with nonzero values.)

We assume an "adverse" oracle $\mathcal{A}$ which at time 0 whooses the speeds of all the processes for all times. $\mathcal{A}$ is also able to dynamically change the willingness relation $\xrightarrow[t]{}$ (subject to assumption (A2)) so as to achieve the worst case performance of the implementation of VS-DCS. Note that, in practice, we can assume each process has a *director subprocess* which dynamically changes the willingness relation $\xrightarrow[t]{}$ and at time 0 determines the speed of that process for all times. Thus, in this case, the oracle $\mathcal{A}$ is defined *distributedly* by the director subprocess. It should be noted that the oracle $\mathcal{A}$ is useful to us because it may be explicitly used to define the worst case performance of the system, when communication requests happen at times most difficult for our implementation and speeds vary in the most difficult way. Thus, if we prove that the system has a certain performance for a worst case oracle, then we have upper bounds on the performance of any set of director subprocesses.

The following communication primitives can be implemented by the poller subsystem of each process: (In practice, the director may not get an immediate answer but may proceed to some other instruction and later a time slot for communication will be arranged by the poller subsystem).

> ATTEMPT-COM$_i$(j): indicates that the director of i wishes to communicate with the director of process j.
>
> CANCEL-COM$_i$(j): indicates that the director of i wishes no longer to communicate with j.

The precise semantics of ATTEMPT-COM and CANCEL-COM are given by the relation $\xrightarrow[t]{} \subseteq \pi \times \pi$ (the willingness digraph, defined previously).

Note that assumption (A2) implies that the oracle $\mathcal{A}$ can withdraw willingness to communicate only after communication has been established. Thus, if ATTEMPT-COM$_i$(j) is called at time $t_1$ and CANCEL-COM$_i$(j) is called at time $t_2$ $(t_2 > t_1)$ then communication was established for some $t$ on $[t_1, t_2)$. (In fact, our implementations do not really require this assumption but only require that the willingness to communicate will not be cancelled before some constant number of steps. However, the assumption A2 given here, considerably simplifies our analysis.)

## 2.2 Complexity of VS-DCS

We assume here a global constant $\alpha$. We say $\mathcal{A}$ *is tame for* i, j on time interval $\Delta$ if the pairs $\{(i,j)\} \cup \{(i,k) \mid k$ is a neighbour of $i\} \cup \{(j,k) \mid k$ is a neighbour of $j\}$ are each relative acceleration bounded by $\alpha$ on the time interval $\Delta$.

For every $\varepsilon$ on $(0,1)$ let the $\varepsilon$-*error response* $S(\varepsilon)$ be the minimum integer $> 0$ such that for every pair of neighbours i,j and each time interval $\Delta$ and for every oracle $\mathcal{A}$ which is tame for i,j on $\Delta$, if $i \underset{\Delta}{\leftrightarrow} j$ and the number of steps of the slowest of i,j within $\Delta$ is $\geq S(\varepsilon)$ then there exists a subinterval $\Delta' \subseteq \Delta$ containing at least one step of the slowest of i,j such that $i \underset{\Delta'}{\leftrightsquigarrow} j$ with probability $\geq 1 - \varepsilon$. Intuitively $1 - \varepsilon$ gives a lower bound in the probability of establishing communication in the case process i issues an ATTEMPT-COM(j) at the beginning of $\Delta$, and after $S(\varepsilon)$ steps it calls CANCEL-COM(j). (Note that we presume here that i and j and their neighbours have relative acceleration bound $\alpha$, only during the interval $\Delta$; at other times this acceleration bound may be violated, and furthermore the acceleration bound $\alpha$ need not hold for other processes even during the interval $\Delta$.)

We consider an implementation to be *relative real time* if for all constants $\varepsilon$ on $(0,1)$, the relative $\varepsilon$-error response $S(\varepsilon)$ is independent of any global measure of the willingness digraph $G_t$ (such as $|\pi|$ or any function of it) and only depends on the constant maximum valence $v$ of the vertices of $G_t$, and on the bound $\alpha$ on the relative acceleration). Note that relative real time response *does not imply* that communication is guaranteed within any time interval but instead it is guaranteed within a bounded number of steps of the processes with high likelihood (this is because processes can slow down arbitrarily). In this paper we show how to implement the VS-DCS so that relative real time response is achieved.

## 3. APPLICATIONS

The primitives ATTEMPT-COM, CANCEL-COM are powerful enough to supply real time implementations of synchronization constructs of high level parallel languages such as CSP and ADA.

The following proposition will be useful in the applications:

PROPOSITION 3.1. *If the oracle* $\mathcal{A}$ *is tame for processes* i,j *on an interval* $\Delta$ *which includes at least* $\delta_0 X + \alpha X^2/2$ *steps of either* i *or* j, *(where* $\delta_0$ *is the speed ratio of processes* i,j *in the beginning of* $\Delta$*), then* $\Delta$ *includes at least* X *local rounds.*

Proof. Consider the number of local rounds to be the "time" during which a fictitious moving object with initial speed $\delta_0$ and acceleration $\alpha$ moves a distance equal to the maximum number of steps done by either i or j on $\Delta$. □

## 3.1 Real Time Resource Granting Systems with Process Failure

Previously, in [Reif, Spirakis, 1981] we utilized the more restricted DCS system (which does not allow process failures) to implement a real time resource granting system. In this paper, we can cope with sudden process failures (zero speeds). In this case, the process governing a resource will first get an estimate on the speed of a process granted the resource (say, $\delta_0$) and then it will attempt to communicate for $\delta_0 S(\varepsilon) + \alpha S^2(\varepsilon)/2$ of its steps with the process granted the resource. (Note that $\delta_0$ will be 1 if the process governing the resource is the fastest). By Proposition 3.1 the above interval should be enough for the process which has been granted the resource to respond. If that process does not respond, the resource governing process may reclaim the resource. If a resource allocator dies, then other processes can play its role (for details see [Reif, Spirakis, 1981b]).

## 3.2 Relative Real Time Implementation of CSP and ADA's Synchronization Constructs

In a typical stage during execution, the processes comprising a CSP program may be divided into two classes: those busy with local computations and those waiting for a partner to communicate with. A distributed guard scheduler can be implemented by using the poller subprocesses of the relative real time VS-DCS system.

The CSP (Communicating Sequential Processes) was defined by [Hoare, 1978] for concurrent programming. The language has elegant synchronization constructs:

(1) An *output command* of the form i!u where i is a process and u is a value which i receives.

(2) An *input command* of the form i?x where i is a process which sends a value which is assigned to variable x.

CSP also allows *alternative statements* which consist of a sequence of guarded commands of the form $G \to C$ where the *guard* G is a list of booleans followed by at most one input command and C is a command list. We assume here the extension of CSP given in [Bernstein, 1981], which allows G to be a list of booleans followed by at most one input or output command. An alternative statement is executed by nondeterminately choosing a guard which is satisfied (by executing its elements from left to right) and then executing the corresponding command list. If no guard is satisfied, the alternative statement *fails*. CSP also allows a *repetitive statement* allowing repeated execution of an alternative statement until it fails.

Thus, the essential problem in implementing CSP is to synchronize execution of input and output commands.

A CSP implementation is *relative real time* if there exists a positive integer $\ell$ (which is independent of the number of processes n) such that if in some alternative or repetitive statement S some guard G is satisfied continuously for a time interval containing at least $\ell$ steps of both processes associated with the guard and if the worst case oracle $\mathcal{A}$ is tame with respect to those processes during that interval, then the command list associated with some satisfied guard is immediately executed with probability $\geq 1-\varepsilon$ and otherwise a *failure exit* is always made, after at most $\ell$ steps. Hence, we allow an *error probability* $\varepsilon$ for a failure exit even though some guard may be satisfied. However, $\varepsilon$ may be fixed to an arbitrarily small constant on the interval (0,1).

For a process i to execute an output command j!u, process i must execute the communication command ATTEMPT-COM$_i$(j). Also, to execute an input command i?x, process j must execute the communication command ATTEMPT-COM$_j$(i). If successful communication is established between i and j, then during that time process j transmits value u to variable x in process i. Processes i,j then execute CANCEL-COM$_i$(j) and CANCEL-COM$_j$(i), respectively. (Note that if processes i or j happen to die at this point, before cancelling communication, then successful communication cannot be made with them during the time their speed is 0, so it is not essential for the communication request $i \leftrightarrow j$ to be cancelled.)

Let S be an alternative statement with guarded input and output commands, say $G_1,\ldots,G_s$ with $s \leq v$. To execute the statement S, process i first executes the booleans appearing in each guard. Let R be the set of processes appearing in those guards of S all of whose booleans evaluate to true. Process i must then execute ATTEMPT-COM$_i$(j) for each process $j \in R$, for $\ell$ steps of i, where $\ell = \alpha S^2(\varepsilon)/2$. At the first time a communication is established between i and some willing process $j \in R$, Pi must immediately execute CANCEL-COM$_i$(j') for each $j' \in R$ and then execute the command list associated with the now satisfied guard in the statement S. Otherwise, after $\alpha S^2(\varepsilon)/2$ steps of i, process i makes a failure exit from statement S. By Proposition 3.1 and the definition of $S(\varepsilon)$, the probability of an incorrect failure exit is $< \varepsilon$.

Also, in ADA, two-way communication between pairs of tasks is allowed in synchronized time instances called *rendezvous*. An *accept* statement of the form *accept* f(-) appearing in task $T_1$ indicates that $T_1$ is willing to rendezvous at f with any task of similar argument type. The task $T_2$ may execute a *call* statement of the form f(-) indicating that $T_2$ is willing to *rendezvous* with $T_1$ at the accept statement containing f. ADA also allows for *selective accept statements* containing multiple accept statements, one of which must be nondeterministically chosen to execute. (This is similar to the *select* statement of CSP.)

ADA's tasks may be implemented by processes whose speeds vary dynamically. (Processes may even fail for various time intervals.) The key implementation problem is to synchronize task rendezvous within relative real time, in spite of the dynamic speed variations. These processes may be connected within a distributed network whose transmission channels may also have variable speeds or fail. Unreliable transmission channels can be viewed as processes which are connected with the processes of the network via reliable communication channels.

We assume that it is possible to analyze (perhaps by data flow analysis) an ADA program to determine an indirected (possibly infinite) connections graph whose nodes are all the tasks possibly created by the ADA program and edges are the possible task communication pairs. Since an actual implementation will have in its hands at any time only a finite set of processes we assume that only the currently active tasks have an associated implementing process and that a call to ADA's *initiate* statement devotes a currently free process to a given newly created task. An *abort* statement garbage collects the implementing process from the deleted task and places it back to the free list of processes. These implementation techniques were developed by [Denis and Misunas, 1974] for real time implementation of data flow machines.

The synchronization facilities of our VS-DCS system provide (by use of the ATTEMPT-COM and

CANCEL-COM primitives) a real time implementation of the *accept* and *call* statements. A version of the *active* statement can be implemented so that deleted tasks and tasks implemented by nontame processes can be detected by their neighbours in real time with some (arbitrarily small) error probability. This can be done in our VS-DCS system by repeatedly attempting communication with neighbouring processes. Finally, the *symmetry* and *locality* of the VS-DCS implementation (due to its probabilistic nature) may help in eliminating the tradeoff between generality of expression and ease of implementation in ADA.

The *probabilistic fairness* guaranteed by the algorithms of the pollers eliminates the danger of bottlenecks which could be created if conventional techniques were used (a new task which centralizes requests and keeps track of busy server tasks is one of the conventional proposed solutions). Most of the problems which VS-DCS could cure are discussed in [Mahjoub, 1981], [Francez, Rodeh, 1980]. A probabilistic solution to some of the discussed problems was given also in [Francez, Rodeh, 1980] but no discussion about real-time properties was done and neither the problem of speed variations and dying processes was addressed.

## 4. RELATIVE REAL TIME IMPLEMENTATION OF VSDCS

### 4.1 Intuitive Description of the Algorithm

We utilize $7v+1$ synchronized parallel processes to implement the poller subprocess of each process $i$. These are the *communicators* $cp_1^i, cp_2^i, \ldots, cp_{2v}^i$, the *speed estimators* $ep_1^i, \ldots, ep_v^i$ and the *judge* subprocesses $jp_0^i, jp_1^i, \ldots, jp_{4v}^i$ of process $i$. Each pair of the communicators $cp_{k'}^i, cp_{k''}^i$ (with $k' \bmod v = k'' \bmod v = k$) is devoted to communication with a specific neighbour (the k-th neighbour). Each estimator is used to continuously update an estimation of the speed of a particular neighbour process. The collection of judges has the task to select under certain conditions one communicator and to give to him the right to open the communication channel of process $i$ to its corresponding neighbour.

We frequently use the technique of *handshake* by which we mean that each subprocess modifies a flag variable observed by the corresponding neighbour subprocess. Process contention between synchronized subprocesses is easy to implement (we can allow each to take a separate step in a small round).

Our algorithm for the k-th communicator subprocess $cp_k^i$ $(1 \leq k \leq 2v)$ of the poller of process $i$ proceeds as follows:

Let $k' = k \bmod v$. At every time $t \geq 0$, $E^i(1), \ldots, E^i(D^i)$ is the list of targets of edges of $G_t$ departing from $i \in \pi$, and $D^i$ is the current number of targets $(D^i \leq v)$. Those variables are dynamically set by the oracle $\mathcal{A}$ and they are the neighbours to which process $i$ is willing to open communication at time $t$. The subprocess $cp_k^i$ deals with the $E^i(k')$ neighbour. If $k \leq v$, then $cp_k^i$ is an *asker* subprocess, else it is a *responder* process. $cp_k^i$ must first handshake with the corresponding subprocess of process $E^i(k')$ to which node $i$ wishes to communicate. We need two handshake subprocesses (ask, respond respectively) per neighbour because of a cetain asymmetry in the handshake (some has first to modify a flag). In particular the asker procedure initiates the handshake and the responder answers to it.

Next we wish to find a time slot in which the two neighbours may communicate. Because there may be contention among other processes $j$ which also wish to communicate with $i$ (and consequently, other askers or responders of node $i$ also will handshake) we must resolve the contention by a fair judge. To do this, we add the process $cp_k^i$ to a $Q^i$ and the collection of judge synchronous subprocesses of poller $i$ takes a random process from this queue and allocates time slots for communication attempts. To ensure that slower neighbours do not utilize any more total time on the average than faster neighbours during communication attempts, we weigh the probabilities of subprocesses to be chosen from the queue by the factor

$$\frac{1/\Delta_{ik}}{\omega(\Delta_i)}$$

where $\Delta_{ij}$ $(j=1,\ldots,v)$ is the current estimation of the steps of process $i$ per step of process $j$, supplied by the estimator $ep_j^i$, and $\omega(\Delta_i) = \Sigma_j 1/\Delta_{ij}$.

The judge subprocesses are organized in a tree (balanced binary tree) of height $\log(2v)+1$. Any time a random process is to be selected from the queue, the judge $jp_0^i$ (the *supreme judge subprocess*) enables the tree of the rest of the judges to conduct a tournament between the waiting processes in the queue and to select a winner with the above stated probability. In that way, the total number of steps needed for a winner to be selected is $O(\log(2v))$. (Note that less efficient ways of using a random number generator to choose one waiting process from the queue could take $O(v)$ steps of process $i$, because of the form of the weight factor in the probabilities.)

The fact that a subprocess is chosen from the queue with the above stated probability, has the effect that each subprocess in the queue attempts to communicate on the average $1/(2v+v^2/\alpha)$ of the total time. (See the analysis for a proof of that.) If a process is chosen by the judges but the communication is not established, the algorithm requires that subprocess to initiate another handshake with its partner (to check if they are still mutually willing to communicate and to synchronize steps). Then, it is again added to the queue to be given another chance to establish communication. This process proceeds until either the director of $i$ withdraws its willingness to communicate with $E^i(k')$ or until it establishes communication.

Note that the time slots for communication attempts, allocated by the supreme judge $jp_0^i$ to each selected communicator, take into account the current speed ratio of the process $i$ and its neighbour corresponding to that communicator, adjusted by a factor related to the worst-case acceleration and the $\log_2 2v$ delay in the process of choosing a winner, to give the opportunity of at least one step overlap in time of process $i$ and its neighbour, if their corresponding channels are both open.

We introduce random waits which help subprocess $cp_k^i$ to eliminate the possibility of schedules set-up by the adverse oracle $\mathcal{A}$ to have always a particular subprocess arrive first in the queue and win the contest. This possibility is eliminated since we have assumed that the oracle sets the speeds at time 0 and cannot affect the random choices done by the processes. Alwo, we assume that the random number generator RANDOM(0,1) of each subprocess yields truly random numbers, uniform on the interval [0,1], and independent of the random numbers generated by any other subprocess.

Note that we trade computation effort (parallelism) in a node to achieve reliable communication. This parallelism is limited because of the bounded valence $v$ of the graph $G_t$. We can alway simulate these synchronous techniques. This will reduce the effective speed of each subprocess by only a factor of $7v+1$.

## 4.2 The Algorithms of the Poller Subprocesses

In each process $i \in \pi$, we assume synchronous subprocesses

| | |
|---|---|
| askers: | $cp_1^i, cp_2^i, \ldots, cp_v^i$ |
| responders: | $cp_{v+1}^i, cp_{v+2}^i, \ldots, cp_{2v}^i$ |
| estimators: | $ep_1^i, \ldots, ep_v^i$ |
| judges: | $jp_0^i, jp_1^i, \ldots, jp_{4v}^i$ . |

The askers and the responders are the communicators.

In the following algorithm, executed by each of the communicator subprocesses $cp_k^i$, $1 \leq k \leq 2v$, we implement the queue of process $i$ by an array $Q^i(k)$, $1 \leq k \leq 2v$. $Q^i(k)=1$ holds just if $cp_k^i$ waits in the queue. Another array of binary values, $\text{marriage}^i(k)$, $1 \leq k \leq 2v$, is used to indicate which communicator subprocess currently holds process's $i$ channel and attempts communication. When the predicate $\text{marriage}^i(k)=1$ is true, then $cp_k^i$ attempts communication at that time. The algorithms have designed so that at most one of $\text{marriage}^i(j)$, $1 \leq j \leq 2v$, is set at any time. We now present the algorithm for the communicator subprocesses:

```
process cp$_k^i$
   WHILE true DO
         x ← true
      L0: IF $D^i \geq k$ THEN
            BEGIN
            L1: W ← $c_1$·RANDOM(0,1)
                DO $\lfloor W \rfloor$ noops
                IF $k \leq v$ THEN ASK($E^i(k)$) ELSE
                                        RESPOND($E^i(k)$)
                COMMENT: Add k to queue
                $Q^i(k) \leftarrow 1$
                WHILE $\text{marriage}^i(k) = 0$ DO noop
                x ← ESTABLISH-COM($E^i(k)$, $c_2\Delta_{ik}$)
                $\text{marriage}^i(k) \leftarrow 0$
                IF x THEN GOTO L1 ELSE GOTO L0
            END
   OD
```

The constants $c_1$ and $c_2$ are as follows:

$$c_1 = 2(2v+1)c_2$$
$$c_2 = 4(\alpha\beta+1)$$
$$\beta = 6\log(2v) \quad .$$

The speed estimators execute the following algorithm, which continuously does a handshake in order to estimate the speed ratio of the process to which the handshake is attempted and the process of which the speed estimator is a subprocess:

```
   process ep$_k^i$

   DO    FOREVER
         set $F_{ik}$ to 1
         LOOP UNTIL $F_{ki}$ is 1
   A:    set $F_{ik}$ to 0;  s ← CURSTEP
         LOOP UNTIL $F_{ki}$ is 0

   B:    $\Delta_{ik} \leftarrow \frac{\text{CURSTEP}-s}{2}$

   OD
```

Note that $F_{ik}$ is a flag set by $i$, read by $k$. The special register CURSTEP gives the current step of process $i$. We assume that a step consists of an elementary statement of the algorithms; $ep_k^i$'s execution assures that $\Delta_{ik}$ is (within a factor of 2) the actual speed ratio of processes $i$ and $k$, since from step A to step B the fastest of the partners does CURSTEP-s steps and the slowest does 2 steps.

## 4.3 The Algorithms of the Judge Subprocesses

The algorithm of the supreme judge

```
process jp$_0^i$
WHILE true DO
   IF queue $Q^i$ not empty THEN
     BEGIN
      Use the tree of judges to select a random
      element k of the queue $Q^i$ with probability
      $\frac{1/\Delta_{ik}}{\omega(\Delta_i)}$
```

```
    COMMENT: delete k from $Q^i$
    $Q^i(k) \leftarrow 0$
    $\text{marriage}^i(k) \leftarrow 1$
    WHILE $\text{marriage}^i(k) = 1$ DO noop
  END
OD
```

Intuitively, the supreme judge triggers the operation of the tree of judges. In each level, the winners of the previous level are paired up and half of them are selected. The judge subprocesses of each level work synchronously in parallel. Finally, the $jp_0^i$ accepts the choice of the root of the tree of judge subprocesses to be the communicator which is going to attempt communication. The supreme judge removes this winner subprocess from the queue $Q^i$ (by setting $Q^i(k)$ to 0) and allows it to attempt communication (i.e., to use process's $i$ channel) by setting $\text{marriage}^i(k)$ to 1. Note that we can test if $Q^i$ is empty by keeping a counter of the number of elements in $Q^i$.

We now give the algorithms of the judges:

```
process $jp_0^i$
WHILE true DO
  IF $Q^i$ is not empty THEN
    BEGIN
      FOR level = 1,...,log(2v) + 1 DO
          BEGIN
            $L^i \leftarrow$ level; do 6 noops
          END
      $k \leftarrow \text{choice}^i(4v)$
      $Q_k^i \leftarrow 0$
      $\text{marriage}^i(k) \leftarrow 1$
      WHILE $\text{marriage}^i(k) = 1$ DO noop
    END
OD
```

The rest of the judges are organized in a full binary tree of $4v$ nodes. The leaves are the processes $jp_1^i,\dots,jp_{2v}^i$. Each internal node $m \in \{2v+1,\dots,4v\}$ has two children LCHILD(m), RCHILD(m). The root is the process $jp_{4v}^i$. Each $jp_m^i$ has its level stored in MYLEVEL(m).

```
process $jp_m^i$
  IF MYLEVEL(m) = $L^i$ THEN
    BEGIN
      IF $L^i = 1$ THEN
        BEGIN
          $\text{choice}^i(m) \leftarrow m$
          $\text{marriage}^i(m) \leftarrow 0$
          IF $Q^i(m) = 0$ THEN $\text{sum}^i(m) \leftarrow 0$
                         ELSE $\text{sum}^i(m) \leftarrow 1/\Delta_{im}$
        END
      ELSE
        BEGIN
          $r \leftarrow$ RANDOM(0,1)
          $m_1 \leftarrow$ LCHILD(m)
          $m_2 \leftarrow$ RCHILD(m)
          $\text{sum}^i(m) \leftarrow \text{sum}^i(m_1) + \text{sum}^i(m_2)$
          IF $r < \text{sum}^i(m_1)/\text{sum}^i(m)$
             THEN $\text{choice}^i(m) \leftarrow \text{choice}^i(m_1)$
             ELSE $\text{choice}^i(m) \leftarrow \text{choice}^i(m_2)$
        END
    END
```

Note that each judge subprocess which is not a leaf uses a uniform random number generator RANDOM(0,1) to return a random number between 0 and 1 and uses it to select one of the choices of its children with conditional probability $\text{sum}^i(m)/(\text{sum}^i(m_1) + \text{sum}^i(m_2))$ where $m_1$, $m_2$ are the children of $m$.

LEMMA 4.1. *It takes* $\beta = 6\log(2v)$ *steps for the tree of judges to select a winner from the queue* $Q^i$. *Furthermore, the probability that the winner is the communicator* $cp_k^i$ *(given that* $Q^i(k) = 1$ *and* $\Delta_{ij}$, $1 \leq j \leq 2v$, *is as given at the beginning of the contest) is*

$$\frac{1/\Delta_{ik}}{\omega(\Delta_i)} .$$

Proof. The judges of each level work synchronously in parallel and each does at most 6 steps, per iteration of their loop. Since the tree has a height of $\log(2v)$, the total number of steps required is $\beta = 6\log(2v)$. The probability that $cp_k^i$ will be selected is the product of the conditional probabilities that $cp_k^i$ will be selected in each node of the path from $k$ to $4v$ (= root). A simple inductive argument on the level of nodes in the tree then proves the lemma. □

## 4.4 Low Level Synchronization Procedures

The following are the low level synchronization procedures used by the poller algorithms:

```
procedure ask$_i$ (target)
  BEGIN
    $Q_{i,\text{target}} \leftarrow 1$;
    WHILE $A_{\text{target},i} = 0$ DO noop
    $Q_{i,\text{target}} \leftarrow 0$;
    WHILE $A_{\text{target},i} = 1$ DO noop
  END
```

Note: The set of the flag $Q_{i,\text{target}}$ means that $i$ asks the target. If the target detects $Q_{i,\text{target}} = 1$ then it answers positively by setting $A_{\text{target},i} = 1$. Both partners reset these flags to 0 at the end of procedures ask and respond.

```
procedure    respond_i (asker)
   BEGIN
      LOOP UNTIL  $Q_{asker,i} = 1$
   BEGIN
      $A_{i,asker} \leftarrow 1$;
      WHILE    $Q_{asker,i} = 1$  DO  noop
      $A_{i,asker} \leftarrow 0$;
   END
END
```

We finally present the code for the procedure ESTABLISH-COM$_i$(target,s). During its execution process i opens its channel to process target. A simple protocol (symmetric handshake) is then attempted to see if the neighbour responded to that communication attempt. If the protocol succeeds then i is sure that process target also opened its channel and communication took place. Else, process i knows that the attempt failed.

```
procedure    ESTABLISH-COM_i (target,s)
   BEGIN
      OPEN CHANNEL_{i,target}; $COM_{i,target} \leftarrow 1$
      $S_0 \leftarrow$ CURSTEP
      $b \leftarrow 1$
      WHILE ($COM_{target,i} = 0$)  or  ($b = 1$)  DO
            IF  CURSTEP-$S_0 \geqslant s$  THEN  $b \leftarrow 0$
      OD
       IF ($COM_{target,i}=1$) AND ($b=1$)
                               THEN success $\leftarrow$ 1
                               ELSE success $\leftarrow$ 0
       $b \leftarrow 0$
       $COM_{i,target} \leftarrow 0$
       CLOSE CHANNEL_{i,target}
       return(success)
   END
```

Note: $COM_{i,target}$ is a flag of node i and $COM_{target,i}$ is a flag of node target. OPEN CHANNEL$_{i,target}$ results in the appearance of i $\underset{t}{\rightsquigarrow}$ target at the time of its execution, and CLOSE CHANNEL$_{i,target}$ sets i $\underset{t}{\not\rightsquigarrow}$ target. Also, s is the maximum number of steps we are allowed to keep the channel open before we fail.

## 5. CORRECTNESS PROPERTIES OF OUR PROPOSED IMPLEMENTATION AND TIME ANALYSIS

LEMMA 5.1. *A matching (with respect to the relation* $\underset{t}{\leftrightsquigarrow}$*) is guaranteed by the implementation.*

Proof. In any time instant, only one of the subprocesses of any poller can have the marriage variable set and its channel open. So, the relation $\underset{t}{\rightsquigarrow}$ is one-one which means that $\underset{t}{\leftrightsquigarrow}$ cannot be more than matching. □

DEFINITION. *A subprocess* $cp^i_k$ *gets the channel when it executes* ESTABLISH-COM$_i$(k).

LEMMA 5.2. *Death of a process does not affect the communication of other processes.*

Proof. Death of process "target" at any time will only cause blocking of only one subprocess ($cp^i_{target}$) per neighbour i of target. This does not disrupt the other subprocesses of the neighbours. □

LEMMA 5.3. *Suppose that* i,j *start to be mutually willing to communicate at some time and continue to be willing for 5 local rounds. Then all four subprocesses* $cp^i_{j_1}$, $cp^i_{j_2}$ *and* $cp^j_{i_1}$, $cp^j_{i_2}$ *(with* $j_1 \bmod v = j_2 \bmod v = j$ *and* $i_1 \bmod v = i_2 \bmod v = i$*) will arrive in the queues of* i *and* j *in a constant number (5) local rounds.*

Proof. Note that at each time the slower of i,j will do only one step in the busy waits of procedures ask or respond. The result follows simply by counting the steps to be executed in each of the procedures. □

Let $\Delta_{i,j}$ be the current estimation (within a factor of two) of the ratio of steps of i per step of j (estimated by i).

DEFINITION. *Let* $\rho_{ij}$ *be the ratio* $1/(\omega(\Delta_i)\Delta_{ij})$.

In the following we assume that the oracle $\mathscr{A}$ is tame with respect to processes i,j in the time interval T they attempt communication.

DEFINITION. *Let* $s_{ij}$ *be the average number of steps that* $cp^i_j$ *does before it is selected to attempt communication, measured from the time it enters the queue.*

LEMMA 5.4. $s_{ij} \leqslant 2vc_2\Delta_{ij}$, *where* $\Delta_{ij}$ *is the most current estimation (i.e., the* $\Delta_{ij}$ *used in the last competition in which* $cp^i_j$ *was the winner), and* $c_2 = 4v(\alpha\beta + 1)$*, as defined previously.*

Proof. Note that the probability to be chosen is bounded above by a geometric density $g_{ij}(x) = (1-\rho_{ij})^x \rho_{ij}$ where x is the number of selections done before $cp^i_j$ is selected. The mean value of this number is $\Sigma^{\infty}_{x=0}\, g_{ij}(x)\cdot x \leqslant 1/\rho_{ij}$. Each time $cp^i_j$ is not chosen, it waits in the queue $Q^i$ for an average number of steps bounded above by the worst case value of

$$\sum_{k=1}^{2v} c_2\Delta_{ik}\rho_{ik} \leqslant \frac{2v\ c_2}{\omega(\Delta_i)}$$

So,

$$s_{ij} \leqslant \frac{1}{\rho_{ij}} \cdot \frac{2v\ c_2}{\omega(\Delta_i)} = 2vc_2\Delta_{ij} \ .$$ □

DEFINITION. *A process* $cp_k^i$ *is in the queue if* $Q^i(k) = 1$.

DEFINITION. *Let* $\lambda = 2v\left(1 + \frac{v}{2\alpha}\right)$.

THEOREM 5.1. *Each subprocess expects to get the channel* $\geq 1/\lambda$ *of the time.*

Proof. Let $S_{Q,k}^i$ be the average number of steps $cp_k^i$ attempts to communicate by executing ESTABLISH-COM. Then, by Lemma 4.5,

$$S_{Q,k}^i = \left(\frac{1/\Delta_{ik}}{\omega(\Delta_i)}\right) c_2\Delta_{ik} = c_2/\omega(\Delta_i)$$

where $\Delta_{ik}$ is the estimation used in the last competition in which the judges selected $cp_k^i$ to be the winning process.

Let $T_{Q,k}^i$ be the subinterval of a time interval $T$ in which $cp_k^i$ gets the channel. Let

$$\mu = \text{mean}\left(\frac{\text{length of } T_{Q,k}^i}{\text{length of } T}\right) .$$

In the worst case, where process $i$ is the fastest and all neighbours slow down with the same worst case acceleration $\alpha$, $S_{Q,k}^i$ is the same for all subprocesses in the queue. The worst case contention happens when all $2v$ subprocesses are in the queue. Thus, in the worst case

$$\mu \geq \frac{S_{Q,k}^i}{\sum_{k=1}^{2v} (S_{Q,k}^i+\beta)} = \frac{1}{2v\left(1 + \frac{\beta\omega(\Delta_i)}{c_2}\right)}$$

(since the length of $T$ is the sum of time in channel for each process plus the time of competition for each process). But

$$\omega(\Delta_i) = \sum_{k=1}^{2v} 1/\Delta_{ik}$$

gets its maximum of $2v$ when $\Delta_{ik} = 1$ for $k = 1,\dots,2v$. So,

$$\frac{\beta\omega(\Delta_i)}{c_2} \leq \frac{2v\beta}{c_2} = \frac{2v\beta}{4(\alpha\beta+1)} \leq \frac{v}{2\alpha} .$$

So,

$$\mu \geq \frac{1}{2v\left(1 + \frac{v}{2\alpha}\right)} = \frac{1}{\lambda} . \qquad \square$$

Note that the above theorem justifies the use of the estimate $(1/\Delta_{ik})/\omega(\Delta_i)$ as the probability to select the subprocess $cp_k^i$ from the queue.

In the following we assume $1 \leq i,\ k' \leq v$ and $k = k' + v$. Thus $cp_k^i$, is the asker and $cp_i^k$ is the responder.

LEMMA 5.5. *The relative position of the time intervals during which the channels of two neighbour processes are open, is a uniform random position and is not affected by the oracle* $\mathcal{A}$.

Proof. The oracle sets the speeds of processes at time 0 and cannot affect their probabilistic choices. Also, each subprocess $cp_k^i$ of a process $i$ suffers a random wait before each return to the queue of process $i$. Their random waits are uniformly distributed in the interval whose length is the mean number of local rounds to attempt communication (as we shall see in Theorem 5.3). $\square$

LEMMA 5.6. *The probability of instantaneous overlap of open channels of subprocesses* $cp_k^i$ *and* $cp_i^{k'}$ *is* $\geq (1/\lambda)^2$.

Proof. By Theorem 5.1 and Lemma 5.5. $\square$

DEFINITION. *Let an overlap between processes* $i,k'$ *be the interval in which channels of* $i$ *and* $k'$ *are simultaneously open.*

DEFINITION. *Let success in communication between* $i$ *and* $k'$ *be an overlap of open channels of* $i$ *and* $k'$ *for at least one step of both processes* $i,k'$.

DEFINITION. *A phase of subprocess* $cp_k^i$ *is a random wait, a handshake with* $cp_i^k$, *a wait in queue and a communication attempt.*

DEFINITION. *Let* $\gamma_{min} = 1/2(1/\lambda)^2$.

THEOREM 5.2. *The probability of success in communication in a phase of subprocess* $cp_k^i$ *is* $\geq \gamma_{min}$.

Proof. When the subprocess $cp_k^i$ opens its channel, the number of steps done from the time of the estimation of $\Delta_{ik}$ used in the selection process of the judges, is $\beta = 6\log(2v)$ and hence, since $\Delta_{ik} \geq 1$, the new speed ratio can be $\Delta_{ik} + \alpha\beta \leq (\alpha\beta+1)\Delta_{ik}$ in the worst case. The worst case is when process $i$ is the fastest and process $k'$ slows down continuously with the maximum acceleration, so that process $i$ does more and more steps per step of $k'$. In this case, a communication attempt of $c_2\Delta_{ik}$ $_{k'}$time slots where $c_2 = 4(\alpha\beta+1)$ guarantees that $p_i^i$ will do at least 2 steps during the time $p_k^i$ has its channel open. Note the random relative position of these steps with respect to $p_k^i$'s steps (due to random waits in the poller subprocesses' algorithms.) Thus, given that there is an overlap, the probability is at least $1/2$ that the length of the overlap is at least 1 step.

Hence, by Lemma 5.6, the probability that there is an overlap and its length is $\geq 1$ step of both processes is

$$\geq \frac{1}{2}\cdot\left(\frac{1}{\lambda}\right)^2 . \qquad \square$$

Note that the above theorem justifies the selection of the constant $c_2 = 4(\alpha\beta+1)$ in the communicators' algorithm.

Let $\mathscr{C}$ be the class of oracles $\mathscr{A}$ for which the out-valence of each node of $G_t$ is v for all t. This class of oracles creates the maximum contention and gives the worst relative response time.

DEFINITION. *Let* $q_{ik}(h/\mathscr{A})$ *be the probability that it takes exactly* h *phases for subprocess* $cp_k^i$ *to communicate with* $cp_i^{k'}$.

DEFINITION. *Let* $\gamma_{max} = \frac{1}{(2v)^2}$ .

LEMMA 5.7. *For any oracle* $\mathscr{A}$*,*

$$q_{ik}(h/\mathscr{A}) \leq (1-\gamma_{min})^{h-1} .$$

*For oracles* $\mathscr{A} \in \mathscr{C}$*,*

$$q_{ik}(h/\mathscr{A}) \leq (1-\gamma_{min})^{h-1} \gamma_{max} .$$

Proof. It suffices to observe that the process of $cp_k^i$ be answered by $cp_i^{k'}$ is a geometric stochastic process with success probability bounded by $[\gamma_{min}, 1]$. For oracles in class $\mathscr{C}$, it is bounded within $[\gamma_{min}, \gamma_{max}]$ due to the contention of all 2v processes in any communication attempt.

By using the above lemma and known expressions for the mean and the tail of a geometric we get

LEMMA 5.8. *For oracles in* $\mathscr{C}$

$$\text{mean}(h) \leq \frac{\gamma_{max}}{(\gamma_{min})^2} .$$

LEMMA 5.9. *For oracles in* $\mathscr{C}$

$$\forall \varepsilon, \quad 0 < \varepsilon < 1, \quad \text{Prob}\{h > h_{max}(\varepsilon)\} \leq \varepsilon$$

*where*

$$h_{max}(\varepsilon) = \frac{\log(\gamma_{min}\, \varepsilon) - \log \gamma_{max}}{\log(1-\gamma_{min})} .$$

Note that, by Lemma 5.1 and Theorem 5.1 in the worst case relation of speeds of processes i,k, the total length of a phase of subprocess $p_k^i$ is the number of local rounds in the random wait plus the number of local rounds up to the end of the communication attempt. This sum is $c_1 = 2(2v+1)c_2$.

This justifies the use of the constant $c_1$ in our Algorithms for the poller subprocesses.

THEOREM 5.3. *For oracles in class* $\mathscr{C}$ *(and hence for the worst case of any "adverse" oracle* $\mathscr{A}$*), the mean number of local rounds to achieve communication is*

$$\leq c_1 \cdot \frac{\gamma_{max}}{\gamma_{min}^2} = 4c_1\lambda^2\left(1 + \frac{v}{2\alpha}\right) = O(v^6\alpha \log v)$$

*and the* $\varepsilon$*-error response of the presented implementation of* VS-DCS *is*

$$S(\varepsilon) \leq c_1\, h_{max}(\varepsilon)$$
$$= O\left(v^5\alpha \log v \log \frac{v}{\varepsilon}\right)$$

*for large* v.

Proof. By previous remark and the fact that

$$h_{max}(\varepsilon) = \frac{\log\left(\frac{\gamma_{min}}{\gamma_{max}}\varepsilon\right)}{\log\left(1 - \frac{1}{\lambda^2}\right)} \to \lambda^2\log\left(\varepsilon^{-1}\left(1 + \frac{v}{2\alpha}\right)\right)$$

$$= \left(2v + \frac{v^2}{\alpha}\right)^2 \log\left(\varepsilon^{-1}\left(1 + \frac{v}{2\alpha}\right)\right)$$

for large v. □

## 6. CONCLUSION

Since we have assumed global parameters α and v to be constant, by Theorem 5.3 our VS-DCS system has relative real time response. Our restrictions on processors rates are much less than in our previous paper [Reif, Spirakis, 1981]. Furthermore, our algorithms seem much more modular and simple in design, although we have utilized new adaptive techniques to deal with arbitrary speed variability.

## REFERENCES

Angluin, D., "Local and Global Properties in Networks of Processors," *12th Annual Symposium on Theory of Computing*, Los Angeles, California, April 1980, pp. 82-93.

Arjomandi, E., M. Fischer, and N. Lynch, "A Difference in Efficiency between Synchronous and Asynchronous Systems," *13th Annual Symposium on Theory of Computing*, April 1981.

Bernstein, A.J., "Output Guards and Nondeterminism in Communicating Sequential Processes," *ACM Trans. on Prog. Lang. and Systems*, Vol. 2, No. 2, April 1980, pp. 234-238.

Dennis, J.B. and D.P. Misunas, "A Preliminary Architecture for a Basic Data-Flow Processor," *Proc. of the 2nd Annual Symposium on Computer Architecture, ACM, IEEE*, 1974, pp. 126-132.

Francez, N. and Rodeh, "A Distributed Data Type Implemented by a Probabilistic Communication Scheme," *21st Annual Symposium on Foundations of Computer Science*, Syracuse, New York, Oct. 1980, pp. 373-379.

Hoare, C.A.R., "Communicating Sequential Processes," *Com. of ACM*, Vol. 21, No. 8, Aug. 1978, pp. 666-677.

Lehmann, D. and M. Rabin, "On the Advantages of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers' Problem," to appear in *8th ACM Symposium on Principles of Program Languages*, Jan. 1981.

Lipton, R. and F.G. Sayward, "Response Time of Parallel Programs," Research Report #108, Dept. of Computer Science, Yale Univ., June 1977.

Lynch, N.A., "Fast Allocation of Nearby Resources in a Distributed System," *12th Annual Symposium in Theory of Computing*, Los Angeles, California, April 1980, pp. 70-81.

Mahjoub, A., "Some Comments on ADA as a Real-time Programming Language," to appear.

Rabin, M., "N-Process Synchronization by a 4 $\log_2 N$-valued Shared Variable," *21st Annual Symposium on Foundations of Computer Science*, Syracuse, New York, Oct. 1980, pp. 407-410.

Rabin, M., "The Choice Coordination Problem," Mem. No. UCB/ERL M80/38, Electronics Research Lab., Univ. of California, Berkeley, Aug. 1980.

Reif, J.H. and Spirakis, P., "Distributed Algorithms for Synchronizing Interprocess Communication Within Real Time," *13the Annual ACM Symposium on Theory of Computation*, Wisconsin, 1981, pp. 133-145.

Reif, J.H. and Spirakis, P., "A Real Time Resource Granting System," to appear. [Also appearing in preliminary form as Appendix II of "Distributed Algorithms..." referenced above.]

Schwartz, J., "Distributed Synchronization of Communicating Sequential Processes," DAI Research Report No. 56, Univ. of Edinburgh, 1980.

Tonag, S., "Deadlock and Livelock-Free Packet Switching Networks," *12th Annual Symposium on Theory of Computing*, Los Angeles, California, April 1980, pp. 82-93.

Valiant, L.G., "A Scheme for Fast Parallel Communication," Technical Report, Computer Science Dept., Edinburgh, Scotland, July 1980.