COMPUTER SCIENCE PUBLICATION

EFFICIENT DISTRIBUTED TERMINATION DETECTION FOR SYNCHRONOUS COMPUTATIONS IN MULTICOMPUTERS

Cheng-Zhong Xu and Francis C.M. Lau

Technical Report TR-94-04

March 1994

DEPARTMENT OF COMPUTER SCIENCE
FACULTY OF ENGINEERING
UNIVERSITY OF HONG KONG
POKFULAM ROAD
HONG KONG

Efficient Distributed Termination Detection for Synchronous Computations in Multicomputers

Cheng-Zhong Xu
Department of Computer Science
University of Paderborn
Paderborn, Germany
czxu@uni-paderborn.de

Francis C.M. Lau
Department of Computer Science
The University of Hong Kong
Hong Kong
fcmlau@csd.hku.hk

Abstract

We propose a simple algorithm which is based on edge-coloring of directed graphs for termination detection of synchronous computations. The proposed algorithm is fully symmetric in that all processes are syntactically identical and can detect global termination simultaneously. It is optimal in terms of delay for termination detection in a number of structures under the 1-port (one message exchange with at most one neighbor per time step) communication model—chain, ring of even number of nodes, mesh of even number of nodes in every dimension and low degree, and k-ary n-cube of low degree where k is even; and near-optimal for other cases. The proofs for optimality are based on results from an equivalent problem, the gossiping problem for edge-colored graphs. This algorithm has been applied to some practical cases in which the overhead due to its execution is found to be insignificant. (A preliminary version of this paper appears as [54].)

Index Terms—data parallelism, distributed algorithms, interconnection networks, multicomputers, parallel and distributed systems, synchronous computations, termination detection.

*On leave from Department of Computer Science, Shantou University, P.R. China.

1 Introduction

Termination detection is an important operation for parallel and distributed computations. It has been studied in the past as mainly a theoretical problem. But in recent years, even system implementors have found the problem relevant and are constantly looking out for efficient and readily implementable solutions. One important area in which termination detection has a major role to play is data-parallel computations, computations that are based on data parallelism [22] or domain decomposition [17]. Concerning this type of computations, Fox found that a large percentage of them are synchronous or loosely synchronous [16]. And it is often the case that a synchronous or loosely synchronous computation would proceed in phases, and a phase cannot begin until the previous phase has completely terminated [3]. Termination detection is therefore needed at the end of each phase. Examples of such computations can be easily found in areas such as image and signal processing [7], parallel solutions of partial differential equations [1], and time-driven discrete event simulations [40]. It is also applicable to load balancing in multicomputers; in dynamic load balancing based on the relaxation method, a processor (modeled by a process) balances its workload by iteratively exchanging load information with the neighboring processors [9, 53]. In this paper, we give a new solution to the termination detection problem, which is fully distributed, time efficient, and is very easy to implement. We have actually adopted and successfully implemented the solution in a couple of major applications of data parallel computations [55]. Because of the lack of a global state in a distributed environment, distributed termination detection is non-trivial and has been one the most heavily studied problems in distributed computing since its early discussions [12, 18]. A brief survey of the most well known solutions can be found in [33].

The scenario in which termination detection is to operate is as follows. The distributed computation is in terms of a collection of disjoint processes. At any time during the computation, there are *Busy* and *Idle* processes. *Busy* processes (and only *Busy* processes) can send messages to other processes; an *Idle* process would turn *Busy* upon receipt of a message from a *Busy* process; *Busy* processes can become *Idle*. The goal is to devise an efficient distributed algorithm that can detect global termination of the computation by every process, where global termination is the stable state in which all processes are *Idle*. Messages in real life take non-zero time to travel from one process to another, and as such there is the danger of false detection: all processes are found

to be *Idle* and global termination is inferred, but a message is still in transit which can revive an *Idle* process in the next moment. Systems that have this danger are generally referred to as *asynchronous* systems [3]. Termination detection for this kind of system is difficult, and most of the existing approaches are based on the technique of message counting [33, 34, 27]. For *synchronous* systems, false detections do not exist or can be easily avoided. In the most common case, every send operation is matched by a receive operation, and both operations are blocking, which will not complete until the message is actually received and the sender is acknowledged, as in the CSP model [23]. Hence, false detection due to messages in transit is impossible. In fact, we can treat the passing of these messages as being instantaneous or equivalently we treat the channels as FIFO channels. Because of the need to unblock the sender, the synchronous mode is most suitable for computations that require only nearest-neighbor communications in a network. This kind of computations is now quite commonplace, especially when the underlying machine architecture is based on a point-to-point interconnection network.

The solution we propose in this paper is for static synchronous systems. It does not require preprocessing of the system graph (*i.e.*, imposing a subgraph on the original graph for termination detection messages). Many of the existing solutions rely on the construction of a spanning tree or cycle that covers all the vertices (in some cases, all the edges) [5, 6, 12, 13, 18, 19, 20, 35, 37, 38, 44, 50]. The result of such constructions is that the time for the solution would be at least $O(n)$ where n is the number of nodes. The solution by Lozinskii requires that all shortest paths towards a detector node be pre-computed [32]. Cohen and Lehmann [8] and Lài [29] proposed solutions for dynamic systems, systems in which processes might be dynamically created or destroyed. We point out that these solutions would have limited use in data parallel computations as dynamic creation of processes is generally not necessary, not to mention the cost of such a non-trivial operation.

The termination detection problem we set out to solve is represented by an undirected graph G whose vertices are processes (or processors) and whose edges are bidirectional communication links. We denote the diameter of the graph by $\mathcal{D}(G)$ and define "delay" to be the number of steps to detect global termination from the time it happens. Obviously, the lower bound of the delay is $\mathcal{D}(G)$. There exist algorithms whose delay is optimal or can be tuned to become optimal with respect to this lower bound [45, 48, 49]. These three algorithms are based on the same idea of keeping a counter in every process to tell how far the farthest *Busy* node might be. They are all

fully distributed, symmetric, and require knowledge of the diameter of the graph, but do not impose a spanning tree or cycle on the graph. In particular, the algorithm by Szymanski *et al.* has a delay of $\mathcal{D}(G)+1$ [49]. The +1 is due to the fact the farthest *Busy* node in a globally terminated state must be at a distance greater than the network diameter away (*i.e.*, $\geq \mathcal{D}(G)+1$). Our algorithm borrows this idea of using a simple integer counter to keep track of the farthest *Busy* node. We find however that the above three algorithms rely on a powerful communication primitive that allows a node to broadcast to or exchange information with all (or a subset) of it direct neighbors in one time step. We call these algorithms "broadcast-based" for the rest of this paper. This primitive is possible only with idealized models such as the n-port communication model [25], or that based on the Multiple Link Availability (MLA) assumption [2]. Krumme *et al.* refer to these models as the H$*$ or F$*$ models [26], where H and F stand for half- and full-duplex respectively, and $*$ stands for parallel communication with all neighbors. The more realistic models are those that are based on 1-port communications, called H1 or F1 by Krumme *et al.*, with which a node can complete at most one communication transaction (half- or full-duplex) along one of its edges in one time step. Such a model is assumed in many instances of recent research in multicomputer algorithms [9, 42, 4, 24, 31] (see also the text by Kumar *et al.* [28]). The understanding is that in a multicomputer, the time to complete a broadcast to or a complete exchange of information with all of a node's neighbors is a function of the number of neighbors (hence the number of individual communications). This is supported by Valiant's rather popular BSP model [52]. In the BSP model, a cost of $hg+s$ is charged to a superstep (like a phase) in which each node sends and is sent at most h messages. Krumme *et al.* has argued for the validity of restrictive H1 and F1 communication models [26]. In practice, machines like iPSC/2 and iPSC/860 [46] and the Connection Machine [22] are basically of the 1-port type. Even though some of the latest designs of message-passing processors have incorporated multiple on-chip communication modules which can operate simultaneously, because of the fixed cost of setting up a communication, the total time of sending h messages, assuming the best possible overlapping in time, is still largely determined by h unless the messages are rather long. Examples of such designs include the MDP chip [11] and the Inmos transputer [36].[1] Therefore

[1]We have tried an experiment on the transputer which is most famous for its built-in parallel communication capabilities. The best time of executing in parallel four exchanges of an integer with a transputer's four direct neighbors (an operation required by the broadcast-based algorithms) we could achieve is 70% of the serialized version.

we consider the 1-port communication model a more reasonable and realistic one than the n- or all-port model. The communication model assumed in this paper is the 1-port (full-duplex) model in which a processor can *exchange* a message with at most one direct neighbor in a time step.

If we adopt the 1-port model, one step in a broadcast-based algorithm must now be counted as δ steps, where δ is the degree of the network. Refering to the $\mathcal{D}(G)$ lower bound, this makes the broadcast-based algorithms far from optimal. Our algorithm, to be presented in the ensuing sections, performs much better with the 1-port model; its delay is $\tilde{\mathcal{D}}(G) + 1$ *iteration steps*, where $\tilde{\mathcal{D}}(G)$ is called the *color diameter* of the network and an iteration step is approximately δ communication steps between nodes. For some regular topologies, an iteration step equals exactly δ communication steps, and $\tilde{\mathcal{D}}(G) = \mathcal{D}(G)/\ \delta$, and hence the delay is $\mathcal{D}(G) + \delta$ communication steps which is optimal for reasonably large networks or networks with a small degree. For other common topologies, it is near optimal. If instead the idealized all-port communication model is assumed, our algorithm is still comparable to the broadcast-based algorithms for most of the popular topologies.

In addition to being fully distributed and time efficient, our algorithm is totally symmetric—*i.e.*, each node executes the same syntactically identical code—which makes it easy to implement. The symmetry also contributes to the property that all the processes detect global termination simultaneously. This is particularly desirable for multi-phase computations in which one phase cannot start until the previous phase is completed (such as the image processing examples discussed in [30]). Like those broadcast-based algorithms, the only limitation of our algorithm is that it requires each node to know the color diameter, $\tilde{\mathcal{D}}(G)$, of the network. It is possible at extra cost to do away with this requirement using strategies that can detect the diameter of the network on the fly, such as those in [49, 41]. Sections 2, 3, and 4 are on the computation model, the idea and intuition behind the algorithm, and the basic definitions respectively. Section 5 presents the algorithm and proofs of its major properties. An equivalence between our algorithm and the gossiping problem for edge-colored graphs is established in Section 6, based on which we prove the optimality of our algorithm for some popular topologies.

2 Basic ideas

The problem's structure is represented by by a simple graph $G = (V, E)$, where V is a set of vertices labeled from 0 to $n-1$ and $E \subseteq V \times V$ is a set of bi-directional edges. Each vertex represents a process (or a processor) and an edge $< i, j > \in E$ a bi-directional communication channel between process i and process j. G has no self-loops, since no process needs to communicate with itself. We let $\mathcal{D}(G)$ to be the diameter of G, which is longest path among the shortest paths for all pairs of source and destination; and $\delta(G)$ to be the degree of G, which is the maximum of the number of the neighboring nodes connected to any node of the graph. Figure 1 shows an example of a simple mesh network with four processors (processes). The termination detection process is patterned after the model of synchronous iterative equations [3]. The iterative equations that correspond to Figure 1 are

$$
\begin{aligned}
X_0^{t+1} &= F_0(X_0^t, X_1^t, X_3^t) \\
X_1^{t+1} &= F_1(X_0^t, X_1^t, X_2^t) \\
X_2^{t+1} &= F_2(X_1^t, X_2^t, X_3^t) \\
X_3^{t+1} &= F_3(X_0^t, X_2^t, X_3^t)
\end{aligned}
$$

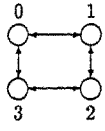

Figure 1: An example of a process structure

At each iteration step, process i performing the partial operator F_i first exchanges the result (X_i) from the last iteration step with its dependent processes and then computes and updates X_i. The superscripts in the equations represent the indices of the iteration steps. For our termination detection algorithm which is symmetric for all processes, all the F_i's are the same.

Suppose the computation being observed will terminate in finite time with a global termination condition being true; then the task is to devise a termination detection algorithm that would allow *every* process of the graph to detect this condition within the shortest time after it becomes true. In distributed memory multicomputers, the global state is distributed: nodes are only aware of their own current states and the current states of other directly connected nodes, and any decision concerning global termination is to be inferred from the conjunction of local termination conditions of all the nodes. Our objective is to minimize the *termination delay*—that is, the difference between the time when global termination occurs and the time when it is detected by the processes.

The broadcast-based algorithms ([45, 48, 49]) are based on a model similar to the above. They use an integer counter S to maintain the control information about termination. $S = 0$ if and only if the node in which this process resides is in the $Busy$ state. During the communication phase of an iteration step, the process exchanges its counter value with those of neighboring processes. Then in the computation phase of the iteration step, each $Idle$ process updates its counter to be $1 + \min\{S, InputS\}$, where $InputS$ is the set of all received counter values. It is easy to see that the counter in a process actually corresponds to the distance between this process and the nearest $Busy$ process, and since the control information of a $Busy$ process can propagate over at most one edge (but reaching one or more neighbors) in an iteration step, a process that just turned $Idle$ (only $Idle$ processes would try to detect global termination) requires at least $\mathcal{D}(G) + 1$ steps to confirm global termination.[2] Hence, the delay for termination detection using the broadcast-based algorithm is equal to $\mathcal{D}(G) + 1$.

Notice that the change of the counter values in an broadcast-based algorithm behaves in a fashion similar to Jacobi relaxation for solving equations: in the communication phase of an iteration, a process updates its counter only after having collected all the counter values of the neighboring processes. This may cause unnecessary delays in propagating individual counter values within the process structure. An alternative is to follow the Gauss-Seidel way of solving equations: during the communication phase of an iteration, a process updates its counter immediately after its every exchange with a neighboring process; this way, counter values may get propagated much faster.

[2]It is possible for processes not situated at the perimeter of the network to detect global termination in less than $\mathcal{D}(G) + 1$ steps by taking advantage of positional information. But this possibility is ruled out in order to maintain the symmetry property.

Let's illustrate this point with the following example.

Consider the iterative computation in Figure 1. For the mesh in the figure, $\mathcal{D}(G) = 2$. Suppose at some time t_0, all processes except process 0 are *Idle* with $S = 1$; then at time $t_0 + 1$, process 0 becomes *Idle* and tries to detect global termination. With the broadcast-based algorithms, two iteration steps are needed for process 0 to detect the state of the farthest process, process 2, and one more step to confirm the stability of its state. Hence, The delay is three iteration steps (which is equal to $\mathcal{D}(G) + 1$). Note that the communication phase within an iteration in this case consists of two communication instances, one with each of a node's neighbors. Let's assume that they must be done in sequence. Now suppose we divide the communication phase of an iteration step into two consecutive sub-phases in the horizontal and vertical dimensions respectively, and a process would update its counter immediately after every exchange along a dimension; then, process 0 can determine the stable state of process 2 in two iteration steps (one iteration step to detect process 2's state and one iteration step to confirm). Thus, the delay for termination detection is only two steps.

The improvement as illustrated in the above example stems from the updating of a process' counter value immediately after an exchange as opposed to updating after having exchanged with all the neighbors. Generally, a process that connects with more than one neighbor can interact with only one of its neighbors at a time in the synchronous communication model, which fits well with the 1-port physical model discussed previously. The interaction order can be determined by edge-coloring of the process graph. The division of the edges into the horizontal and vertical dimensions in the above example is an example of edge-coloring using two colors, one for each dimension. In order to shorten the time needed to complete the communication phase, the edge-coloring algorithm must assign the minimum number of colors to a given graph. In the best case, this number is equal to the degree, $\delta(G)$, of the graph, as in the above example. Our algorithm is therefore based on edge-coloring using the minimum number of colors.

3 Edge-coloring of graph

Given the system graph G, we color the edges of G with the minimum number of colors, denoted by $\mathcal{C}(G)$, such that any two adjoining edges are of different colors if their other ends are different

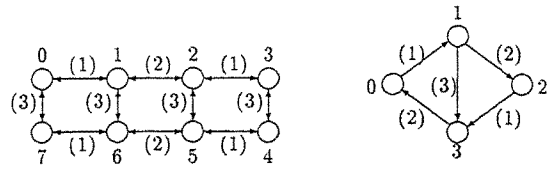

Figure 2: Examples of colored digraph

vertices, and of the same color otherwise [15]. We map the colors to integers from 1 to $C(G)$, and tag each edge of G with a color number that corresponds to the assigned color. For a graph G, its minimum number of colors, $C(G)$, is bounded by the degree of G, $\delta(G)$, as follows.

$$\delta(G) \leq C(G) \leq \delta(G) + 1 \tag{1}$$

The proof of this fundamental theorem can be found in [15], from which the method of edge-coloring using minimum number of colors can also be derived. Figure 3 shows two examples of edge-colored graphs. The numbers in parentheses are the assigned color numbers.

Let $< i, j; c >$ denote an edge $< i, j >$ with color number c. We next introduce some essential concepts that are necessary for the subsequent presentation and analysis of our algorithm which is based on edge-coloring.

Definition 3.1 *Let G be an edge-colored graph. A sequence of edges of the form $< i, i_1; c_1 >, < i_1, i_2; c_2 >, \ldots, < i_l - 1, j; c_l >$ is called a* **color path** *of length l from i to j if all intermediate vertices $i_k (1 \leq k \leq l-1)$ are distinct and $1 \leq c_1 < c_2 < \ldots < c_l \leq C(G)$.*

From this definition, a color path is a path in the graph with increasing color numbers along the edges that make up the path. In Figure 3(a) for example, there are several ways of going from vertex 0 to vertex 4, including this one which comprises two color paths: $<0,1;1>, <1,2;2>$ and then $<2.3;1>, <3,4;3>$.

Definition 3.2 *Let G be an edge-colored graph. The* **color-distance** *from vertex i to j, denoted*

by $\tilde{\mathcal{D}}(i,j)$, is the minimum number of color paths a traversal from i to j would go through. The greatest color-distance $\tilde{\mathcal{D}}(i,j)$ among the color-distances of all the vertex pairs in G is called the **color-diameter** *of G, denoted by $\tilde{\mathcal{D}}(G)$.*

It can be easily seen that the color-distance between vertex 0 and vertex 4 in Figure 3(a) is 2, and the color-diameter of the whole graph is also 2.

Within an iteration step in the termination detection algorithm, to be discussed next, each color is activated once, in the order of increasing color numbers. When a particular color is activated, all edges of this color become active and information would flow through or be exchanged over them. Therefore, information at one end of a color path will propagate to the other end within an iteration step of the algorithm. Note that some nodes may have fewer neighbors than the others, but since communications are synchronous and every color must be considered once in an iteration step, the time complexity of the communication phase is a function of the number of colors, $\mathcal{C}(G)$.

4 The algorithm

The control information for termination detection is abstracted as a local integer counter S maintained in every process. This counter's value changes as the iterative termination detection algorithm proceeds. We serialize the communication phase of an iteration step into a number of consecutive exchange operations, one for each color in $\mathcal{C}(G)$. An exchange operation for color c causes two neighboring processes connected by an edge of color c to exchange their counter values. After each of these little exchanges with a neighbor, a process would update its counter immediately with the value received from the other process if the latter is smaller than this process' current counter value. And at the end of each iteration step, the process sets the counter to 0 if the node is still a *Busy* node (not locally terminated); otherwise, it increments the counter value by 1. When S reaches a predefined value, the process stops with the sure knowledge that the computation has reached global termination.

Our algorithm bears certain resemblance to the broadcast-based algorithm, but the updating of the counter S follows a different protocol as has been explained. The algorithm follows. The variable $InputS$ temporarily stores the neighboring counter value received in the current exchange

operation.

```
1   Algorithm for Process i (0 ≤ i < n):
2       S = 0;
3       while (S < PredefinedValue) {
4           for (c = 1; c ≤ C(G); c++)
5               if (an incident edge colored c exists) {
6                   InputS = Exchange(c);
7                   S = min{S, InputS};
8               }
9           if (LocalTerminated) /* Idle */
10              S = S + 1;
11          else /* Busy */
12              S = 0;
13      }
```

For each color (each iteration of the for-loop), the procedure $Exchange(c)$ is executed which causes the counter values of the this process and a neighboring process connected by the edge colored c to exchange their counter values, where c is the current color (loop index). If the process has no incident edge colored c, it skips this one and goes on to the next color. Note that unlike the broadcast-based algorithms in which the counter is not updated until all the neighboring counter values are received, our algorithm compares the local counter with a neighbor's counter value as soon as the latter comes in through the exchange.

This algorithm is fully symmetric in that it is syntactically identical for each process, and there is no central control. Below, we give an example to illustrate how the counter S gets updated. Then, based on S, we show how $PredefinedValue$ in the algorithm above is determined.

Let $S_i^c(t)$ denote the counter value of process i after the exchange (after line 7) over the edge colored c (if there is one) has taken place within iteration step t, $S_i^-(t)$ the counter value of process i at the end of the for loop (after line 8 before line 9), and $S_i(t)$ the counter value at the end of the iteration step t (after line 12). Clearly, $S_i^-(t) \leq S_i^c(t)$ for all c. Table 1 gives a partial trace of the counter distribution when we apply the algorithm to the process structure of Figure 3(a).

The counter distribution is assumed to be $(0,1,1,1,1,1,1,1)$ at t_0, the instant at which our tracing begins; process 0 becomes *Idle* at step t_0+2; process 1 returns to *Busy* at step t_0+2 and becomes *Idle* again at t_0+4; process 2 returns to *Busy* at t_0+4 and becomes *Idle* again at t_0+5, and global termination occurs at this point since all eight processes are *Idle*. All the counter values reach 3 at t_0+7, which is the (earliest) point at which every process is sure of the global termination—the detection delay is three iteration steps. This implies that *PredefinedValue* has been set to 3 in the algorithm. The following analysis on the global termination condition tells us how this number comes about.

The presentation of the analysis is similar to that in [49]. First, we derive a recursive expression of S from the above algorithm.

Proposition 1 *For any process i, $0 \leq i < n$,*

$$S_i(t+1) = \begin{cases} 0 & \text{if process } i \text{ is not terminated} \\ \min_{j \in I^+(i)}(S_j(t)) + 1 & \text{otherwise} \end{cases}$$

where $I^+(i) = I(i) \bigcup \{i\}$, and $I(i)$ is the set of processes that can reach i in one color path.

Proof. If $j \in I(i)$, then there exists a color path from j to i. Suppose this color path has the form $< j, j_1; c_1 >, < j_1, j_2; c_2 >, \ldots, < j_{l-1}, i; c_l >$, where $c_1 < c_2 < \ldots < c_l$. Then, it is clear that $S_j(t) \geq S_{j_1}^{c_1}(t+1) \geq S_{j_2}^{c_2}(t+1) \geq \ldots \geq S_i^{c_l}(t+1)$. Since $S_i^-(t+1) \leq S_i^{c_l}(t+1)$, it follows that $S_i^-(t+1) \leq S_j(t)$. In addition, if there exists a process $k \notin I(i)$ and $k \neq i$, $S_i(t+1)$ will not be influenced by $S_k(t)$. Thus, the proposition is proved. ■

The above proposition reveals that a non-zero counter value of a process contains more information than just its own current state (which is *Idle*). The historical states of processes from which process i is reachable may be deduced from S_i. In the most trivial case, if the counter value of a process i equals 1 at the end of an iteration step, then the process can infer that there exists at least one *Busy* process within $I^+(i)$ at the end of the last iteration step. Generally, if $S_i(t) = r > 0$, then the following properties concerning historical state information can be derived.

Proposition 2 *Suppose at the end of some iteration step t, there is an Idle process i with $S_i(t) = r > 0$. Then*

1. *there exists a process j satisfying $\vec{D}(j,i) \leq r$, $S_j(t-r) = 0$*

2. *for any process* j *satisfying* $\tilde{\mathcal{D}}(j,i) < r$, $S_j(t-t') > 0$, *where* $\tilde{\mathcal{D}}(j,i) \leq t' < r$

Proof. Part 1 can be proved by induction on the integer r. The statement trivially holds for $S_i(t) = r = 1$. Now suppose it holds for $S_i(t) = r = s > 1$. If $S_i(t) = s+1$, then according to Proposition 1, there exists at least one process $j \in I^+(i)$ with counter value $S_j(t-1) = s$. From the hypothesis, there exists a process k satisfying $\tilde{\mathcal{D}}(k,j) \leq s$ with the counter value $S_k((t-1)-s) = S_k(t-(s+1)) = 0$. Since j is within one color path from i, we have $\tilde{\mathcal{D}}(k,i) \leq s+1$; and so Part 1 is proved. The proof of Part 2 proceeds in a similar way. ■

Part 1 of the proposition says that if an *Idle* process finds its counter value to be $r > 0$ at the end of some iteration step, then there exists a process which is at most r color paths away from this process and whose state was *Busy* r iterations ago. That is, the counter value $(= 0)$ of this *Busy* process got propagated to the *Idle* process within r iterations; while on the way, this value got incremented by one at each iteration. Part 2 says that if this value did get propagated all the way to the *Idle* process in question, then there must not have been any *Busy* processes that were within r color paths from this process during the period of the propagation.

On the basis of the properties of S, we are now in the position to establish the condition for global termination in terms of S.

Theorem 4.1 *For any process* i *in a process structure* G, *it detects global termination when* $S_i = \tilde{\mathcal{D}}(G)+1$.

Proof. It suffices to show that when a process satisfies the condition $S_i(t) = \tilde{\mathcal{D}}(G)+1$ at time t, all other processes j, $j \neq i$, have the same counter value, *i.e.*, $S_j(t) = \tilde{\mathcal{D}}(G)+1$. Suppose at time t, $S_i(t) = \tilde{\mathcal{D}}(G)+1$, but there exists a process j such that $S_j(t) = a < \tilde{\mathcal{D}}(G)+1$. According to Part 1 of Proposition 2, there exists a process, say k, with $S_k(t-r) = 0$. On the other hand, since $S_i(t) = \tilde{\mathcal{D}}(G)+1$, all the counter values at time $t-r$ must be positive according to Part 2 of Proposition 2. We conclude from the contradiction that $S_i(t) = \tilde{\mathcal{D}}(G)+1$ if and only if $S_j(t) = \tilde{\mathcal{D}}(G)+1$. Thus, the theorem is proved. ■

In summary, for any given process structure, the global termination condition is dependent on a single value—the structure's color-diameter, $\tilde{\mathcal{D}}(G)$. The processes exit the iteration loop *simultaneously* with the same counter value $\tilde{\mathcal{D}}(G)+1$. Hence, the constant *PredefinedValue* in

the algorithm should be set to $\widetilde{\mathcal{D}}(G)+1$. The delay of the algorithm is $\widetilde{\mathcal{D}}(G)+1$ iteration steps which is measured from the instant the last *Busy* process turns *Idle*.

5 Analysis of the termination delay

We have shown that for a given system structure G, the termination delay of the algorithm is equal to its color diameter $\widetilde{\mathcal{D}}(G)+1$ which is in terms of number of iteration steps. But since the edges of a graph can usually be colored in more than one way, and that different ways of coloring may result in different color diameters, and hence different termination delays. we would like to determine the best coloring scheme for a given structure G so that the minimum termination delay can be achieved. To prove minimality, we need to express termination delay in terms of communication steps. Refering to the termination detection code in the algorithm, we note that one iterative step comprises $\mathcal{C}(G)$ communication steps, $\mathcal{C}(G)$ being the number of colors. Hence if we let d to be the termination delay in communication steps, then $d = (\widetilde{\mathcal{D}}(G)+1) \times \mathcal{C}(G)$.

We consider the structures of n-dimensional torus and mesh ($n \geq 1$), and their special cases, the ring, the chain, and the k-ary n-cube. An n-dimensional torus is a variant of the mesh with end-round connections, as illustrated in Figure 3 (the end-round connections of the torus at the back are omitted in the figure for clarity). A k-ary n-cube is a network with n dimensions, k nodes in each dimensions [10, 39]. It is a special case of the n-dimensional torus which allows different numbers of nodes in different dimensions. The hypercube is a special case of both the n-dimensional mesh and the k-ary n-cube. A hypercube is an n-dimensional mesh with the same number of nodes (of 2) in each dimension—that is, a 2-ary n-cube. We limit our scope to these structures because they are the most popular choices of topologies in commercial parallel computers [47, 39, 43, 51].

The termination delay of the algorithm, as we have defined before, is the time period from the occurrence of a global termination till the termination is detected by each processor. Clearly, the lower bound of the termination delay (in communication steps) should be the minimum time required by each processor to learn the state information of every other processors. It is the lower bound of the time for *gossiping* [21] in the given graph.

The gossiping problem has been studied by a number of researchers under various communication models. Under the same serial communication model (1-port, full-duplex, but with no coloring

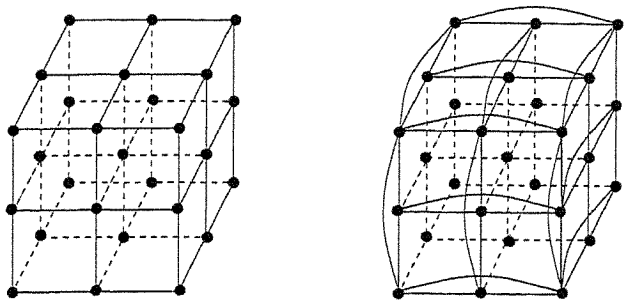

Figure 3: Examples of 3-dimensional mesh and torus (back links omitted)

or whatsoever to schedule the exchanges over the edges[3]) as we are using in this paper, Farley and Proskurowski analyzed the problem for the ring, the mesh, and the torus [14]. Their results, which can serve as the lower bounds of the termination delay, are summarized in Table 5.

Instead of directly working out the appropriate coloring schemes for the structures, measuring their color diameters, and comparing with the above lower bounds, we make use of existing results from a version of the gossiping problem which is equivalent to our termination detection problem to argue about optimality. This version of the gossiping problem, by Liestman and Richards, uses exactly the same edge-coloring technique as we have used here for the chain, the ring, and the mesh [31]. It is easy to see that the two problems—to find a coloring that would yield the shortest color diameter and to find a coloring that would lead to the minimum gossiping time—are equivalent. Therefore, we can use the gossiping results for edge-colored graphs to compare with the above lower bounds. In the following, if the gossiping time in some colored graph is optimal with respect to the corresponding lower bound (Table 5), then the termination delay due to our algorithm using the same coloring scheme is optimal or near optimal. Precisely, if we let g to be the gossiping time, then $d - g < 2 \times C(G)$ communication steps. For graphs with a small degree, this difference should tend to be insignificant. The coloring scheme that is used to arrive at the optimal gossiping time is exactly the optimal coloring scheme we need for our termination detection. Note

[3]Liestman and Richards refer to this as the "standard model" [31].

that "gossiping time" in the following refers to that from using edge-coloring.

Liestman and Richards derived, among their many results, the following minimum gossiping times which are of relevance here. We use $g(G)$ to denote the minimum gossiping time for the structure G, and $g(G^c)$ the gossiping time for G with coloring c.

Theorem 5.1 [31][4]

1. *For a chain of k nodes, P_k, $g(P_k) = 2\lfloor (k-1)/2 \rfloor + 1$.*
2. *For a ring of k nodes (k even), R_k, $g(R_k) = k/2$.*
3. *For a two-dimensional $2 \times k$ mesh $M_{2,k}$, $g(M_{2,k}) = 3\lfloor (k-1)/2 \rfloor + 1$.*

Note that in proving the above, Liestman and Richards gave an actual coloring scheme for each case (using 2 colors for the chain and the ring, and 3 colors for the mesh).

In [31], Liestman and Richards also showed that the minimum gossiping time is no larger than $2 \times \max\{k_1, k_2\}$ for a two-dimensional $k_1 \times k_2$ mesh by using an "alternate" coloring scheme with 4 colors. In the following, we sharpen the bound and show its optimality for the even square mesh. We also present a tight bound of gossiping time for the square torus.

Theorem 5.2 *For a two-dimensional $k_1 \times k_2$ mesh M_{k_1,k_2}, $k_1, k_2 \geq 3$, the minimum gossiping time is bounded as $g_4(M_{k_1,k_2}) \leq 4\lfloor (k-1)/2 \rfloor + 2$, where $k = \max\{k_1, k_2\}$.*

Proof. Suppose in a 4-color situation we color a chain with the sequence $\cdots 1313 \cdots$; then it is easy to verify that the gossiping time of such a chain matches the result of Theorem 5 of [31][5]—*i.e.*, its gossiping time is equal to $4\lfloor (k-1)/2 \rfloor + 1$, where k is the number of nodes in the chain. If instead we color the chain by the sequence $\cdots 2424 \cdots$, then its gossiping time becomes one unit more than the above—*i.e.*, $4\lfloor (k-1)/2 \rfloor + 2$—since the two colors $2, 4$ are exactly one time step behind the colors $1, 3$. Now the two-dimensional mesh in question is basically an assembly of horizontal and vertical chains, and its gossiping time is equal to the maximum of the gossiping times of these chains.

[4] Note the difference in notation, in [31], n is the number of edges (= number of nodes in the ring case) and k is the number of colors; in this section, k is the number of nodes in a dimension and n is the number of dimensions.

[5] $g(P_k) = C(G) \times \lfloor (k-1)/2 \rfloor + 1$, where $C(G)$ is the number of colors.

Therefore, if we always color the longer dimension with $\cdots 1313 \cdots$ and the shorter dimension by $\cdots 2424 \cdots$, then $g_4(M^c_{k_1,k_2}) \leq 4\lfloor (k-1)/2 \rfloor + 2$, where $k = \max\{k_1, k_2\}$ and c denotes the alternate coloring scheme. ■

Figure 4 shows several examples of 4-color meshes using alternate coloring. The corner node with double circles corresponds to the node whose information will take the longest time to propagate to all other nodes. Using dashed ovals, we trace the propagation of information from this node to other nodes against time step numbers in the figure.

Corollary 5.1 *The above gossiping time based on alternate coloring of 4 colors (denoted c) is optimal when the mesh is an even square mesh (i.e., $k_1 = k_2$ is even).*

Proof. For an even square mesh M_k, $g_4(M^c_k) = 4\lfloor (k-1)/2 \rfloor + 2 = 2k - 2$ (refer to the 8×8 mesh in Figure 4) which matches the corresponding lower bound in Table 5. ■

The upper bound of gossiping time for a two-dimensional mesh can be generalized to an n-dimensional $k_1 \times k_2 \times \ldots \times k_n$ mesh, $M_{k_1,k_2,\ldots,k_n}$. We label the edges in the i^{th} dimension of the mesh, $i = 1, 2, \ldots, n$, with alternating i's and $(n+i-1)$'s. Based on this kind of colored meshes, we obtain the following results.

Theorem 5.3 *For an n-dimensional $k_1 \times k_2, \cdots, k_n$ mesh $M_{k_1,k_2,\ldots,k_n}$, $k_1, k_2, \ldots, k_n \geq 3$, the minimum gossiping time is bounded as follows.*

$$g_{2n}(M_{k_1,k_2,\ldots,k_n}) \leq 2n\lfloor (k-1)/2 \rfloor + n,$$

where $k = \max\{k_1, k_2, \ldots, k_n\}$.

Corollary 5.2 *The gossiping time based on alternate coloring for n-dimensional meshes is optimal when all the k_i's are equal to the same even number.*

Next, the torus structure. We consider even tori of which the number of nodes in each dimension is even. Such a torus can be colored with $2n$ colors using the alternate coloring scheme as above, where n is the number of dimensions. Figure 5 shows a 8×8 colored torus and a 6×8 colored torus.

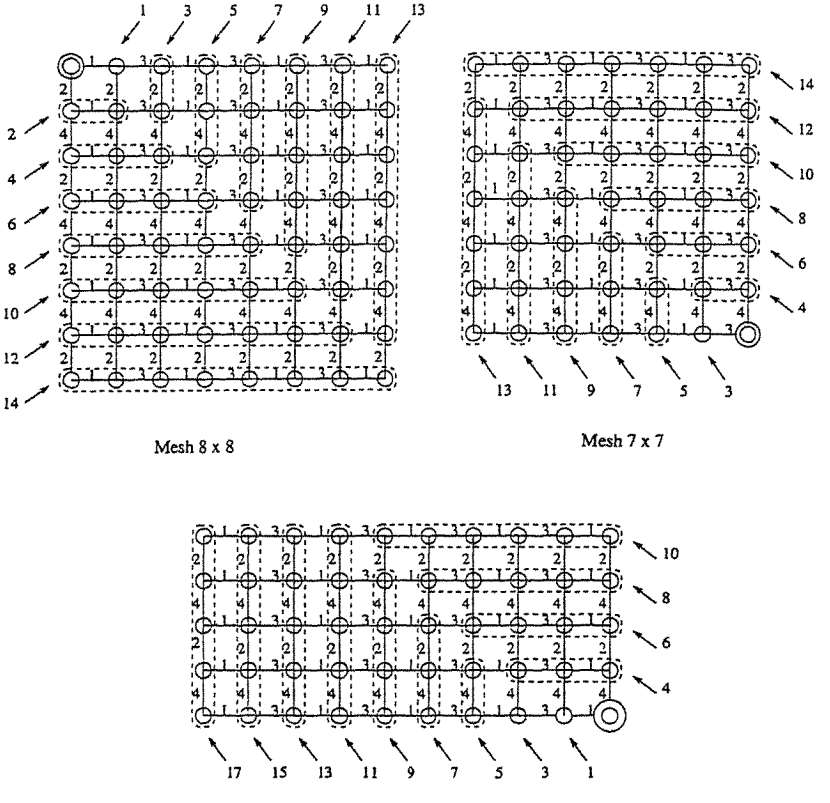

Figure 4: Examples of meshes in 4 colors

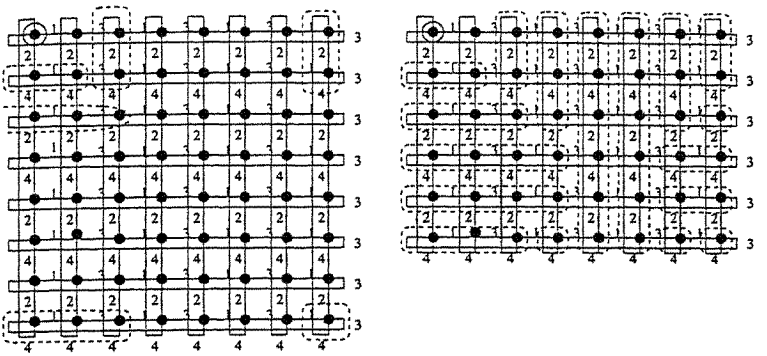

Figure 5: An example of torus in 4 colors

Theorem 5.4 *For a two-dimensional $k_1 \times k_2$ torus T_{k_1,k_2}, $k_1, k_2 > 2$ and are even, the minimum gossiping time is bounded as $g(T_{k_1,k_2}) \leq k$, where $k = \max\{k_1, k_2\}$. It is optimal when $k_1 = k_2$.*

Proof. From Theorem 5.1, a (2-color) even ring completes gossiping in $k/2$ time units, where k is the number of nodes/edges. In our 4-color torus here, which is an assembly of even rings, information advances one step in every two time units; hence, for a ring of k nodes colored with $\cdots 1313 \cdots$, the gossiping time is $2k/2 - 1 = k - 1$, where the -1 is due to the fact that the last step (a 1 or a 3) takes only one time unit; correspondingly, for a ring of k nodes colored with $\cdots 2424 \cdots$, the gossiping time is k. Therefore, if we always color the rings of the longer dimension with $\cdots 1313 \cdots$, we have $g(T^c_{k_1,k_2}) \leq k$, where $k = \max\{k_1, k_2\}$ and c denotes this way of alternate coloring. When $k_1 = k_2$ (i.e., a k-ary 2-cube), $g(T^c_{k_1,k_2}) = k$ is optimal since it matches the lower bound in Table 5. ■

Generalizing we have the following.

Theorem 5.5 *For an n-dimensional $k_1 \times k_2 \times \cdots \times k_n$ torus $T_{k_1,k_2,\ldots,k_n}$, where $k_1, k_2, \ldots, k_n$ are even, the minimum gossiping time is bounded as $g(T_{k_1,k_2,\ldots,k_n}) \leq nk/2$. It is optimal when $k_1 = k_2 = \cdots = k_n$ (i.e., a k-ary n-cube).*

With all the necessary gossiping results in place, we can now comment on the optimality of our termination detection algorithm. By comparing Theorem 5.1 and Table 5, we find that our

termination detection algorithm is optimal for the chain and the even ring ($d - g$ is a constant 2 to 4 time steps). From Corollary 5.2, our termination detection algorithm is near-optimal for even square meshes. And from Theorem 5.5, our termination detection algorithm is near-optimal for the k-ary n-cube, where k is even. All of the above are valid by applying the alternate coloring scheme using the appropriate number of colors, and the near-optimality would tend to optimal if the graph's degree (and hence the number of colors) is small.

6 Concluding remarks

We have shown that our simple termination detection algorithm based on edge-coloring is time-optimal for the chain, the even ring, the even square mesh of low degree, and the k-ary n-cube (k even) of low degree under the 1-port (single exchange with a neighbor per node per time step) communication model. It is near-optimal for the other cases. From Table 5, we note that the numbers of time steps for the optimal cases under the 1-port model is actually equal to the respective absolute lower bounds for information dissemination in these structures regardless of the communication model (*e.g.*, a k-chain will need $k - 1$ time steps to complete a gossip even under the most powerful communication model). Therefore, if we use the all-port communication model, our algorithm performs as well as the broadcast-based algorithm or any other algorithm in these structures. We have implemented our termination detection algorithm in two major applications of iterative data-parallel computations (WaTor simulation and parallel image thinning) [55] and found the overhead due to the execution of this algorithm to be minimal.

References

[1] D.C. Arney and J.E. Flaherty, "An Adaptive Mesh-Moving and Local Refinement Method for Time-Dependent Partial Differential Equations", *ACM Tran. on Mathematical Software*, Vol. 16, No. 1, March 1990, 48-71.

[2] D.P. Bertsekas, C. Özveren, G.D. Stamoulis, P. Tseng, and J.N. Tsitsiklis, "Optimal Communication Algorithms for Hypercubes", *J. of Parallel and Distributed Computing*, Vol. 11, 1991, 263-275.

[3] D.P. Bertsekas and J.N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, Prentice-Hall, 1989

[4] S.N. Bhatt, G. Pucci, A. Ranade, and A.L. Rosenberg, "Scattering and Gathering Messages in Networks of Processors", *Proc. of the 1992 Brown/MIT Conference*, 1992, 318-332.

[5] S. Chandrasekaran and S. Venkatesan, "A Message-Optimal Algorithm for Distributed Termination Detection", *J. Parallel and Distributed Computing*, Vol. 8, 1990, 245-252.

[6] K.M. Chandy and J. Misra, "A Paradigm for Detecting Quiescent Properties in Distributed Computations", in *Logics and Models of Concurrent Systems*, K. Apt (ed.), Springer-Verlag, 1985, 325-341.

[7] A.N. Choudhary and R. Ponnusamy, "Run-time Data Decomposition for Parallel Implementation of Image Processing and Computer Vision Tasks", *Concurrency: Practice and Experience*, Vol. 4, No. 4, June 1992, 313-334.

[8] S. Cohen and D. Lehmann, "Dynamic Systems and Their Distributed Termination", *Proceedings of ACM Symposium on Principles of Distributed Computing*, 1982, 29-33.

[9] G. Cybenko, "Load Balancing for Distributed Memory Multiprocessors", *J. of Parallel and Distributed Computing*, Vol. 7, 1989, 279-301.

[10] W.J. Dally, "Performance Analysis of k-ary n-cube Interconnection Networks", *IEEE Trans. on Computers*, Vol. 39, No. 6, June 1990, 775-785.

[11] W.J. Dally, J.S. Fiske, J.S. Keen, R.A. Lethin, M.D. Noakes, P.R. Nuth, R.E. Davison, and G.A. Fyler, "The Message-Driven Processor: A Multicomputer Processing Node with Efficient Mechanisms", *IEEE Micro*, Vol. 12, No. 2, April 1992, 23-39.

[12] E.W. Dijkstra and C.S. Sholten, "Termination Detection for Diffusing Computations", *Information Processing Letters*, Vol. 11, No. 1, August 1980. 1-4.

[13] E.W. Dijkstra, W.H.J. Feijen, and A.J.M. van Gasteren, "Derivation of a Termination Detection Algorithm for Distributed Computations", *Information Processing Letters*, Vol. 16, No. 5, 1983, 217-219.

[14] A.M. Farley and A. Proskurowski, "Gossiping in Grid Graphs", *J. of Combinatorics, Information and System Sciences*, Vol. 5, No. 2, 1980, 161-172.

[15] S. Fiorini and R.J. Wilson, "Edge-coloring of Graphs", in *Selected Topics in Graph Theory*, L.W. Beineke and R.J. Wilson, eds., Academic Press, 1978.

[16] G.C. Fox, "Applications of Parallel Supercomputers: Scientific Results and Computer Science Lessons", in *Natural and Artificial Parallel Computation*, MIT Press, 1990, 47-90.

[17] G.C. Fox, M.A. Johnson, G.A. Lyzenga, S.W. Otto, J.K. Salmon, and D.W. Walker, *Solving Problems on Concurrent Processors*, Vol. 1, Prentice-Hall, 1988.

[18] N. Francez, "Distributed Termination", *ACM Tran. on Programming Languages and Systems*, Vol. 2, January 1980, 42-45.

[19] N. Francez and M. Rodeh, "Achieving Distributed Termination without Freezing", *IEEE Trans. on Software Engineering*, Vol. SE-8, No. 3, 1982, 287-292.

[20] C. Hazari and H. Zedan, "A Distributed Algorithm for Distributed Termination", *Information Processing Letters*, Vol. 24, 1987, 293-297.

[21] S.M. Hedetniemi, S.T. Hedetniemi, and A.L. Liestman, "A Survey of Gossiping and Broadcasting in Communication Networks", *Networks*, Vol. 18, 1988, 319-349.

[22] W.D. Hillis, *The Connection Machine*, MIT Press, 1985.

[23] C.A.R. Hoare, "Communicating Sequential Processes", *CACM*, Vol. 21, August 1978, 666-677.

[24] J. JáJá and K.W. Ryu, "Load Balancing and Routing on the Hypercube and Related Networks", *J. of Parallel and Distributed Computing*, Vol. 14, 1992, 431-435.

[25] S.L. Johnsson and C.-T. Ho, "Optimum Broadcasting and Personalized Communication in Hypercubes", *IEEE Tran. on Computers*, Vol. 38, No. 9, September 1989, 1249-1268.

[26] D.W. Krumme, G. Cybenko, and K.N. Venkataraman, "Gossiping in Minimal Time", *SIAM J. on Computing*, Vol. 21, No. 1, February 1992, 111-139.

[27] D. Kumar, "Development of a Class of Distributed Termination Detection Algorithms", *IEEE Tran. on Knowledge and Data Engineering*, Vol. 4, No. 2, April 1992, 145-155.

[28] V. Kumar, *Introduction to Parallel Computing*, Benjamin Cummings, 1994.

[29] T.-H. Lai, "Termination Detection for Dynamic Distributed Systems with Non-First-In-First-Out Communications", *J. Parallel and Distributed Computing*, Vol. 3, 1986, 577-599.

[30] S.-Y. Lee and J.K. Aggarwal, "A Mapping Strategy for Parallel Processing", *IEEE Tran. on Computers*, Vol. C-36, No. 4, April 1987, 433-442.

[31] A. Liestman and D. Richards, "Network Communication in Edge-Colored Graphs: Gossiping", *IEEE Tran. Parallel and Distributed Systems*, Vol. 4, No. 4, April 1993, 438-445.

[32] Lozinskii, E.L., "A Remark on Distributed Termination", *Proceedings of 5th International Conference on Distributed Computing Systems*, 1985, 416-419.

[33] F. Mattern, "Algorithms for Distributed Termination Detection", *Distributed Computing*, Vol. 2, 1987, 161-175.

[34] F. Mattern, "Asynchronous Distributed Termination: Parallel and Symmetric Solutions with Echo Algorithms", *Algorithmica*, May 1990, 325-340.

[35] F. Mattern, "Distributed Control Algorithms (Selected Topics)", in *Parallel Computing on Distributed Memory Multiprocessors*, F. Özgüner and F. Erçal, eds., Springer-Verlag, 1991, 167-185.

[36] D. May, "Occam and the Transputer", in *Developments in Concurrency and Communication*, C.A.R. Hoare, ed., Addison-Wesley, 1990, 65-106.

[37] J. Misra and K.M. Chandy, "Termination Detection of Diffusing Computations in Communicating Sequential Processes", *ACM Trans. on Programming Languages and Systems*, Vol. 4, No. 1, 1982, 37-43.

[38] J. Misra, "Detecting Termination of Distributed Computations Using Markers", *Proceedings of 2nd ACM Symposium on Principles of Distributed Computing*, 1983, 290-294.

[39] L.M. Ni and P.K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks", *IEEE Computer*, Vol. 26, February 1993, 62-76.

[40] D.M. Nicol and J.H. Saltz, "Dynamic Remapping of Parallel Computations with Varying Resource Demands", *IEEE Tran. on Computers*, Vol. 37, No. 9, September 1988, 1073-1087.

[41] D. Peleg, "Time-Optimal Leader Election in General Networks", *J. of Parallel and Distributed Computing*, Vol. 8, 1990, 96-99.

[42] C.G. Plaxton, "Load Balancing, Selection and Sorting on the Hypercube", *Proc. of Symp. on Parallel Algorithms and Architectures*, 1989, 64-73.

[43] G. Ramanathan and J. Oren, "Survey of Commercial Parallel Machines", *ACM Computer Architecture News*, Vol. 21, No. 3, June 1993, 13-33.

[44] S.P. Rana, "A Distributed Solution of the Distribution Termination Problem", *Information Processing Letters*, Vol. 17, 1983, 43-46.

[45] S. Rönn and H. Saikkonen, "Distributed Termination Detection with Counters", *Information Processing Letters*, Vol. 34, 1990, 223-227.

[46] S.R. Seidel, M.-H. Lee, and S. Fotedar, "Concurrent Bidirectional Communication on the Intel iPSC/860 and iPSC/2", *Proc. of 6th Distributed Memory Computing Conference*, 1991. 283-286.

[47] C.L. Seitz, "Multicomputers", in *Developments in Concurrency and Communication*, C.A.R. Hoare, ed., Addison-Wesley, 1990, 131-200.

[48] S. Skyum and O. Eriksen, "Symmetric Distributed Termination", in *The Book of L*, G. Rozenberg and A. Salomaa, eds., Springer-Verlag, 1986, 427-430.

[49] B. Szymanski, Y. Shi, and S. Prywes, "Synchronized Distributed Termination", *IEEE Tran. on Software Engineering*, Vol. SE-11, No. 10, October 1985, 1136-1140.

[50] R.W. Topor, "Termination Detection for Distributed Computations", *Information Processing Letters*, Vol. 18, No. 1, 1984, 33-36.

[51] A. Trew and G. Wilson, *Past, Present and Parallel: A Survey of Available Parallel Computer Systems*, Springer-Verlag, 1991.

[52] L.G. Valiant, "A Bridging Model for Parallel Computation", *CACM*, Vol. 33, No. 8, August 1990, 103-111.

[53] C.-Z. Xu and F.C.M. Lau, "Analysis of the Generalized Dimension Exchange Method for Dynamic Load Balancing", *J. of Parallel and Distributed Computing*, Vol. 16, No. 4, December 1992, 385-393.

[54] C.-Z. Xu and F.C.M. Lau, "Distributed Termination Detection of Loosely Synchronized Computations", *Proceedings of 4th IEEE Symposium on Parallel and Distributed Processing*, December 1992, 196-203.

[55] C.-Z. Xu and F.C.M. Lau, "Decentralized Remapping of Data Parallel Computations with the Generalized Dimension Exchange Method", *Proc. of 1994 Scalable High Performance Computing Conference*, Knoxville, Tennessee, May 1994 (to appear).

Process	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
$S_i(t_0)$	**0**	1	1	1	1	1	1	1
$S_i^1(t_0+1)$	0(1)	0(0)	1(1)	1(1)	1(1)	1(1)	1(1)	1(1)
$S_i^2(t_0+1)$	0	0(1)	0(0)	1	1	1(1)	1(1)	1
$S_i^3(t_0+1)$	0(1)	0(1)	0(1)	1(1)	1(1)	0(0)	0(0)	0(0)
$S_i(t_0+1)$	**0**	1	1	2	2	1	1	1
$S_i^1(t_0+2)$	0(1)	0(0)	1(2)	1(1)	1(1)	1(2)	1(1)	1(1)
$S_i^2(t_0+2)$	0	0(2)	0(0)	1	1	1(1)	1(1)	1
$S_i^3(t_0+2)$	0(1)	0(1)	0(1)	1(1)	1(1)	0(0)	0(0)	0(0)
$S_i(t_0+2)$	1	**0**	1	2	2	1	1	1
$S_i^1(t_0+3)$	0(0)	0(0)	1(2)	1(1)	1(1)	1(2)	1(1)	1(1)
$S_i^2(t_0+3)$	0	0(1)	0(0)	1	1	1(1)	1(1)	1
$S_i^3(t_0+3)$	0(1)	0(1)	0(1)	1(1)	1(1)	0(0)	0(0)	0(0)
$S_i(t_0+3)$	1	**0**	1	2	2	1	1	1
$S_i^1(t_0+4)$	0(0)	0(1)	1(2)	1(1)	1(1)	1(2)	1(1)	1(1)
$S_i^2(t_0+4)$	0	0(1)	0(0)	1	1	1(1)	1(1)	1
$S_i^3(t_0+4)$	0(1)	0(1)	0(1)	1(1)	1(1)	0(0)	0(0)	0(0)
$S_i(t_0+4)$	1	1	**0**	2	2	1	1	1
$S_i(t_0+5)$	2	1	1	1	1	1	1	2
$S_i(t_0+6)$	2	2	2	2	2	2	2	2
$S_i(t_0+7)$	3	3	3	3	3	3	3	3

Table 1: A trace of the counter distribution of processes in a 4×2 mesh structure. The numbers in parentheses are the counter values received in the current exchange.

Chain of size k	$k-1$ if k is even; k otherwise
Ring of size k	$k/2$ if k is even; $\lceil k/2 \rceil + 1$ otherwise
Mesh of size $k_1 \times k_2$	$k_1 + k_2 - 2$ if $k_1 \neq 3$ or $k_2 \neq 3$
Torus of size $k_1 \times k_2$	$\lfloor k_1/2 \rfloor + \lfloor k_2/2 \rfloor$ if k_1 and k_2 are even

Table 2: Lower bounds of gossiping time (termination delay)