Efficient Termination Detection for Loosely Synchronous Applications in Multicomputers

Chengzhong Xu, *Member, IEEE*, and Francis C.M. Lau, *Member, IEEE*

Abstract—We propose a simple algorithm which is based on edge-coloring of system graphs for termination detection of loosely synchronous computations. The proposed algorithm is fully symmetric in that all processors run syntactically identical code and can detect global termination at the same time. Under the 1-port communication model, the algorithm is optimal in terms of termination delay, the difference between the time when a global termination occurs and the time it is detected, in a number of structures—chain, ring of even number of nodes, k-ary n-cube and k-ary n-mesh of low degree, where k is even; and near-optimal for other cases. The optimality analysis is based on results from a related problem, periodic gossiping in 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.

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

1 Introduction

Termination detection is a fundamental operation in parallel and distributed computations. It has been heavily studied in the past for efficient and readily implementable solutions. One important area in which termination detection has a major role to play is data-parallel computations [5]. These computations generally follow the so-called Single-Program-Multiple-Data (SPMD) paradigm in which each processor executes the same program but on a different portion of the problem domain. The computation proceeds in steps which are separated by synchronization points. During each step, the processors operate independently on their own data, and then communicate with their data-dependent, usually directly connected, peers. These processors are loosely synchronized in that a processor needs to synchronize only with its data-dependent peers, and can proceed into the next step once it is sure that it has received/sent all the information it needs to receive/send in this step. For this computational model, even though a processor interacts only with a subset of other processors, global termination detection is still needed to confirm the permanence of idleness because an idle peer may get activated again by other processors. Clearly, because of this elusive nature of processors' idleness, the problem of distributed detection of global termination is a nontrivial one.

A termination detection algorithm in this context should have the following three desirable properties.

1) The algorithm should minimize the *termination delay*—the difference between the time when global termination occurs and the time it is detected;
2) The algorithm should be symmetric in that all processors execute the same syntactically identical code, and detect global termination at the same time;
3) The algorithm should incur as few extra control messages as possible.

The termination delay translates into the message complexity of the algorithm. The symmetry feature makes the algorithm easy to be incorporated in the SPMD paradigm. It also ensures simultaneous termination of all processors, which is most desirable in applications that consist of a sequence of tasks (or phases). Minimizing the termination delay of a phase enables the next phase to be issued earlier, and consequently reduces the overall processing time. Such multiphase computations are frequently found in advanced image and vision processing systems.

Loosely synchronous computations have a number of characteristics that can facilitate the termination detection process. First, processes are statically created and assigned to processors in the SPMD fashion. Second, the computation proceeds in steps; and the predicate for termination is evaluated periodically, instead of at arbitrary times in general models. Third, the data dependency between a pair of processors is usually mutual—a send operation in one processor is matched by a receive operation in another processor; therefore, message passing can be treated as instantaneous. Lastly, processors communicate with each other periodically according to a fixed communication pattern; as such, the state of a processor can be piggy-backed on data messages.

For the termination detection problem as described above, many of the existing algorithms, such as those for asynchronous communication models [1], [13] and for dynamic process models [11] are too general to meet our requirements. For the model that assumes instantaneous communications, there are a number of algorithms based on probing techniques [6], [16], [15], [19]. They use a token to keep track of the number of idle processors, which

• *C. Xu is with the Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202. E-mail: czxu@ece.eng.wayne.edu.*
• *F.C.M. Lau is with the Department of Computer Science, the University of Hong Kong, Hong Kong. E-mail: fcmlau@cs.hku.hk.*

Manuscript received Mar. 22, 1994; revised Feb. 22, 1995.
For information on obtaining reprints of this article, please send e-mail to: transpds@computer.org, and reference IEEECS Log Number D95137.

is passed along a Hamiltonian circuit or a spanning tree. These algorithms are nonsymmetric, and their termination delay is N in the case of Hamiltonian circuit or larger than $2D$ in the case of spanning tree (one for the expanding wave and another for the contracting wave), where N is the number of processors and D is the diameter of the system graph.

For the instantaneous communication model, there is another class of algorithms which are based on counting techniques [8], [16], [17], [18]. These algorithms are based on the idea of keeping a counter in every processor to tell how far the nearest non-idle 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 $D + 1$ [18]. The extra one step is for confirmation of global termination. Our algorithm borrows this idea of using a simple integer counter to keep track of the distance of the nearest busy processor. We find however that these four algorithms rely on a powerful communication primitive which allows a node to broadcast to or exchange information with all of its direct neighbors in one time step. We call these algorithms "broadcast-based" in the rest of this paper. This primitive is possible only with idealized models such as the all-port communication model [10]. The more realistic models are those that are based on 1-port communications, 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 [12]. 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. This is supported by Valiant's rather popular BSP model [20]. In practice, machines like iPSC/2 and iPSC/860 and the Connection Machine 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 overly long. Therefore we consider the 1-port communication model a more reasonable and realistic one than the 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 d steps, where d is the degree of the network. Referring to the D 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 $\tilde{D}+1$ *iteration steps*, where $\tilde{D}$ is called the *color diameter* of the network and an iteration step is equal to $d + 1$ communication steps between processors in the worst case. For most regular topologies such as the mesh and the torus with even nodes in each dimension, an iteration step equals exactly d communication steps, and $\tilde{D} = D/d$, and hence the delay is $D + d$ 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.

The optimality analysis of the termination delay is based on the insight that the lower bound of the termination delay in communication steps should be the minimum time required by every processor to learn the state of every other processor. The latter is exactly the lower bound of the time for *gossiping* [9] in the given system graph. Recently, Liestman and Richards proposed a novel periodic gossiping technique for edge-colored system graphs [12]. Their algorithm bears much resemblance to our termination detection algorithm[1] in terms of the communication pattern. Therefore, some of the analysis techniques for periodic gossiping used in their paper are applicable in our optimality analysis of termination delay.

In addition to being fully distributed and time efficient, our algorithm is totally symmetric. Like those broadcast-based algorithms, the only limitation of our algorithm is that it requires each processor to know the color diameter of the network, which however can be easily calculated once a coloring scheme is chosen. We present in Sections 2 and 3 the computation model, the idea and intuition behind the algorithm, and the basic definitions. Section 4 gives the algorithm and proofs of its major properties. An equivalence between the communication part of our algorithm and the gossiping problem for edge-colored graphs is established in Section 5, based on which we prove the optimality of our algorithm for some popular topologies.

2 Computation Model and Basic Ideas

We consider loosely synchronous computations in multicomputers. The multicomputer is assumed to consist of n autonomous processors connected by a point-to-point communication network. We represent the network by a simple connected graph $G = (V, E)$, where V denotes the set of processors labeled from 1 to N and $E \subseteq V \times V$ is a set of bidirectional edges. Each edge $<i, j> \in E$ corresponds to the bidirectional communication link between processor i and processor j. Let $D(G)$ be the diameter of G. Let $d(i)$ denote the degree of a node p_i in G, and $d(G)$ denote the maximum of the degrees of G's nodes.

A loosely synchronous computation in multicomputer proceeds in calculation-communication steps. Termination condition is evaluated at the end of each step. The scenario in which termination detection is to operate is as follows. At the point of termination evaluation, we distinguish between *Busy* and *Idle* states of a processor. A processor is in *Idle* state when its local computation is finished; *Busy* otherwise. An *Idle* state is one in which the problem state value remains unchanged after an iteration phase. A *Busy* processor would communicate its newly computed result in a data message to its dependent processors and an *Idle* proc-

1. A preliminary version was presented in [21].

essor would wait for data messages from its *Busy* neighbors in the iteration. A *Busy* processor can become *Idle* for the next iteration, and an *Idle* processor may return to *Busy* on receipt of a data message from a *Busy* processor during the communication phase. Global termination occurs when all processors become *Idle*. Apart from these data messages of the underlying computation, control messages are used to pass information about termination around, which can emanate from both *Busy* and *Idle* processors. Processors would not change their states on receipt of control messages.

Normally, the data dependency between a pair of processors in the loosely synchronous computations is mutual. A send operation in one processor is matched by a receive operation in another one. We therefore assume the data exchange to be instantaneous, which simplifies the termination detection problem because there won't be false detection due to messages in transition. Nevertheless, termination detection in the present model is still a nontrivial problem as processors could change from *Idle* to *Busy* when they receive data messages from a *Busy* processor.

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 processor to detect this condition within the shortest time after it becomes true. Our principal objective is to minimize the difference between the time when global termination occurs and the time when it is detected by processors, which is defined as the *termination delay*.

The broadcast-based algorithms 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 processor is in the *Busy* state. During the communication phase of an iteration step, the processor exchanges its counter value with those of neighboring processors. Then in the computation phase of the iteration step, each *Idle* processor 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 processor actually corresponds to the distance between this processor and the nearest *Busy* processor, and since the control information of a *Busy* processor can propagate over at most one edge (but reaching one or more neighbors) in an iteration step, a processor that just turned *Idle* (only *Idle* processors would try to detect global termination) requires at least $D(G) + 1$ steps to confirm global termination. Hence, the delay for termination detection using the broadcast-based algorithm is equal to $D(G) + 1$.

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

Generally, a processor having 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 system graph. The edge-coloring technique allows efficient communication while avoiding the possibility of message collisions and congestion. 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. Our algorithm is therefore based on edge-coloring using the minimum number of colors.

3 Edge-Coloring of the System Graph

Given the system graph G, we color the edges of G with the minimum number of colors, denoted by κ, such that no two adjoining edges are assigned the same color. We index the colors with integers from 1 to κ, and represent the κ-color graph as $G_\kappa = (V, E_\kappa)$, of which E_κ is a set of 3-tuples of the form $(i, j; c)$, $(i, j; c) \in E_\kappa$ if and only if c is the chromatic index of the edge $(i, j) \in E$. It is known that the minimum number of colors κ is strictly bounded by $d(G)$, and $d(G) \le \kappa \le d(G) + 1$ [4]. Fig. 1 shows two 3-color graphs derived from a 2×4 mesh.

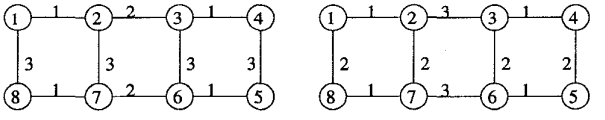


Fig. 1. Examples of two 3-color meshes.

We next introduce some essential concepts that are necessary for the subsequent presentation and analysis of our algorithm based on edge-coloring.

Definition 3.1. *Let G_κ be a κ-color graph of G. A sequence of edges in G_κ of the form*

$$(i = i_0, i_1; c_1), (i_1, i_2; c_2), \ldots, (i_{l-1}, i_l = j; c_l)$$

is called a **color-path** *of length l from i to j if all intermediate vertices $i_s (1 \le s \le l - 1)$ are distinct and $\kappa \ge c_1 > c_2 > \ldots > c_l \ge 1$.*

From this definition, a color-path is a path in the graph with decreasing color numbers along the edges that make up the path. In Fig. 1a for example, there are several ways of going from vertex 1 to vertex 5, including this one which comprises two color paths: (1, 8; 3), (8, 7; 1) and then (7, 6; 2), (6, 5; 1).

Definition 3.2. *In a κ-color graph G_κ, a node j is said to be reachable from node i in one trip if there exists a color path from i to j. The* **color-distance** *from nodes i to j, denoted by $\tilde{D}(i, j)$, is the minimum number of trips required for going from i to j. The greatest color-distance among the color-distances of all the node pairs in G_κ is called the* **color-diameter** *of G_κ, denoted by $\tilde{D}(G_\kappa)$.*

As an example, let us re-examine the 3-color mesh in Fig. 1a. It is clear that node 7 is reachable from 1 in one trip

because of the color path (1, 8; 3), (8, 7; 1), and that node 5 is reachable from node 2 in one trip because of the color path (2, 7; 3), (7, 6; 2), (6, 5; 1). Hence, $\tilde{D}(1, 7) = \tilde{D}(2, 5) = 1$; similarly, we have $\tilde{D}(1, 5) = \tilde{D}(1, 6) = 2$, and thus, $\tilde{D}(G_\kappa) = 2$. Generally, the color-diameter of an edge-colored graph is equivalent to the diameter of an extension of the graph, each edge of which corresponding to a color-path.

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 processors 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.

4 The Algorithm

The control information for termination detection is abstracted as a local integer counter S maintained in every processor. 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. An exchange operation for color c causes two neighboring processors connected by an edge of color c to exchange their counter values. After each of these little exchanges with a neighbor, a processor would update its counter immediately with the value received from the other processor if the latter is smaller than this processor' current counter value. And at the end of each iteration step, the processor 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 processor 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 Processor i (1 ≤ i ≤ N):
2       S = 0;
3       while (S < PredefinedValue) {
4           for (c = 1; c ≤ κ; 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 processor and a neighboring processor connected by the edge colored c to exchange their counter values, where c is the current color (loop index). If the processor 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 processor, 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 processor 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 processor 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, $S_i^-(t) \le S_i^c(t)$ for all c. Table 1 gives a partial trace of the counter distribution when we apply the algorithm to the processor structure of Fig. 1a. 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; processor 0 becomes *Idle* at step $t_0 + 2$; processor 1 returns to *Busy* at step $t_0 + 2$ and becomes *Idle* again at $t_0 + 4$; processor 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 processors are *Idle*. All the counter values reach 3 at $t_0 + 7$, which is the (earliest) point at which every processor 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 [8]. First, we derive a recursive expression of S from the above algorithm.

PROPOSITION 1. *For any node i, $1 \le i \le N$,*

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

where $I^+(i) = I(i) \cup \{i\}$, and $I(i)$ is the set of nodes that i can reach in one trip.

PROOF. If $j \in I(i)$, then there exists a color path from i to j. Suppose this color path has the form $(i, i_1; c_1), (i_1, i_2; c_2), \ldots, (i_{l-1}, j; c_l)$, where $c_1 > c_2 > \ldots > c_l$. Then, it is clear that $S_i^{c_1}(t+1) \le S_i^{c_2}(t+1) \le \cdots \le S_{j_1}^{c_l}(t+1) \le S_j(t)$. Since $S_i(t+1) \le S_i^{c_1}(t+1)$, it follows that $S_i(t+1) \le S_j(t)$. In addition, if there exists a processor $j \notin I(i)$ and $j \ne i$, $S_i(t+1)$ will not be influenced by $S_j(t)$. Thus, the proposition is proved. □

TABLE 1
A TRACE OF THE COUNTER DISTRIBUTION OF PROCESSORS IN A 4 × 2 MESH STRUCTURE

Processor	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
$S_i(t_0)$	**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

The numbers in parentheses are the counter values received in the current exchange.

The above proposition reveals that a nonzero counter value of a processor contains more information than just its own current state (which is *Idle*). The historical states of processors from which processor i is reachable may be deduced from S_i. In the most trivial case, if the counter value of a processor i equals 1 at the end of an iteration step, then the processor can infer that there exists at least one *Busy* processor within $\Gamma^+(i)$ 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 some iteration step t, there is a stable processor i with $S_i(t) = a > 0$. Then*

1) *there exists a processor j satisfying $\tilde{D}(i, j) \le a$, $S_j(t - a) = 0$*
2) *for any processor j satisfying $\tilde{D}(i, j) \le a$, $S_j(t - t') > 0$, where $\tilde{D}(i, j) \le t' < a$.*

PROOF. Part 1) can be proved by induction on the integer a. The statement trivially holds for $S_i(t) = a = 1$. Now suppose it holds for $S_i(t) = a = b > 1$. If $S_i(t) = b + 1$, then according to Proposition 1, there exists at least one processor $j \in \Gamma^+(i)$ with counter value $S_j(t - 1) = b$. From the hypothesis, there exists a processor k satisfying $\tilde{D}(j, k) < b$ with the counter value $S_k(t - b - 1) = 0$. Since the processor k also satisfies $\tilde{D}(i, k) < b + 1$, Part 1) is proved. The proof of Part 2) proceeds in a similar way. □

Part 1) of the proposition says that if an *Idle* processor finds its counter value to be $r > 0$ at the end of some iteration step, then there exists a processor which is at most r color paths away from this processor and whose state was *Busy* r iterations ago. That is, the counter value (= 0) of this *Busy* processor got propagated to the *Idle* processor 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* processor in question, then there must not have been any *Busy* processors that were within r color paths from this processor 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 processor i in the κ-color graph G_κ, it detects global termination when $S_i = \tilde{D}(G_\kappa) + 1$.*

PROOF. In order to be certain about global termination when a processor satisfies this condition $S_i(t) = \tilde{D}(G_\kappa) + 1$, it suffices to show that at time t, all other processors p_j, $j \neq i$, have the same counter value, i.e., $S_j(t) = \tilde{D}(G_\kappa) + 1$. Suppose at time t, $S_i(t) = \tilde{D}(G_\kappa) + 1$, but there exists a processor p_j such that $S_j(t) = a < \tilde{D}(G_\kappa) + 1$. According to Part 1) of Proposi-

tion 2, there exists a processor, say p_k, with $S_k(t-a) = 0$. On the other hand, from Part 2) of Proposition 2, all the counter values at time $t-a$ must be positive. We conclude from the contradiction that $S_i(t) = \tilde{D}(G_\kappa)+1$ if and only if $S_j(t) = \tilde{D}(G_\kappa)+1$. Thus, the theorem is proved. □

In summary, for any given processor structure, the global termination condition is dependent on a single value—the structure's color-diameter, $\tilde{D}(G)$. The processors exit the iteration loop *simultaneously* (i.e., at the same iteration step) with the same counter value $\tilde{D}(G)+1$. Hence, the constant *PredefinedValue* in the algorithm should be set to $\tilde{D}(G)+1$. The delay of the algorithm is $\tilde{D}(G)+1$ iteration steps which is measured from the instant the last *Busy* processor turns *Idle*.

5 Optimality Analysis of the Algorithm

We have shown that for a given edge-colored system graph G_κ, the termination delay of the algorithm is equal to its color diameter $\tilde{D}(G_\kappa)+1$ which is in terms of number of iteration steps. Referring to the termination detection code in the algorithm, we note that one iterative step comprises κ communication operations, κ being the number of colors. Hence, the termination delay in communication operations is equal to $(\tilde{D}(G_\kappa)+1)\kappa$. Since the edges of a graph G can usually be colored in more than one way, as illustrated in Fig. 1, 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.

5.1 Lower Bound of Termination Delay

The termination delay of the algorithm, as we have defined before, is the time span 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 every processor to learn the state of every other processor. It is the lower bound of the time for *gossiping* [9] in the given graph.

The gossiping problem has been studied by a number of researchers under various communication models. Under the same 1-port, full-duplex communication model, Farley and Proskurowski analyzed the problem for the ring, the mesh, and the torus [3]. Their results, which can serve as the lower bounds of the termination delay, are summarized in Table 2.

TABLE 2
Lower Bounds of Termination Delay (Gossiping) in Communication Steps

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

5.2 Termination Delay in Meshes and Tori

We now present the optimality analysis of the termination delay of our algorithm in 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. A k-ary n-cube is a network with n dimensions, k nodes in each dimension [2], [14]; 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 [14].

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 [12]. 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 in Table 2, 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 the termination delay is less than $g + 2 \times \kappa$ communication steps. 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 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 [12]. *For a chain of k nodes, C_k, $g(C_k) = 2\lfloor (k-1)/2 \rfloor + 1$; for a ring of k nodes (k even), R_k, $g(R_k) = k/2$; 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 two colors for the chain and the ring, and three colors for the mesh). They 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 $k = \max\{k_1, k_2\}$; it is optimal when $k_1 = k_2$.*

PROOF. Suppose in a 4-color situation we color a chain with the sequence "··· 1313 ···"; then it is easy to verify that the gossiping time of such a chain matches the result of Theorem 5 of [12]—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 "··· 2424 ···," 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. Therefore, if we always color the longer dimension with "··· 1313 ···" and the shorter dimension by "··· 2424 ···," then $g_4(M^c_{k_1,k_2}) \le 4\lfloor(k-1)/2\rfloor+2$, where $k = \max\{k_1, k_2\}$ and c denotes the alternate coloring scheme. When $k_1 = k_2 = k$, $g_4(M^c_{k,k}) = 4\lfloor(k-1)/2\rfloor+2 = 2k-2$ is optimal since it matches the corresponding lower bound in Table 2. □

Fig. 2a shows an example of 4-color mesh using alternate coloring, and a trace of the propagation of information from one node to another.

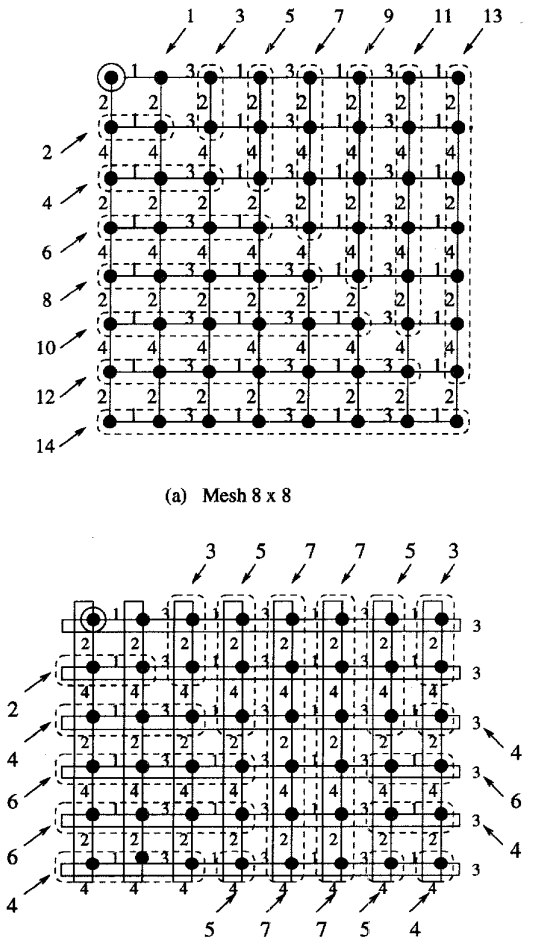
The upper bound of gossiping time for a two-dimensional mesh can be generalized to an n-dimensional $k_1 \times k_2 \times \ldots \times k_n$ mesh, $M_{k_1,k_2,\ldots,k_n}$. We label the edges in the ith dimension of the mesh, $i = 1, 2, \ldots, n$, with alternating i's and $(n+i-1)$s. Based on this 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 $M_{k_1,k_2,\ldots,k_n}$, $k_1, k_2, \ldots, k_n \ge 3$, the minimum gossiping time is bounded as $g(M_{k_1,k_2,\ldots,k_n}) \le 2n\lfloor(k-1)/2\rfloor+n$, where $k = \max\{k_1, k_2, \ldots, k_n\}$; it is optimal when all the k_is 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. Fig. 2b shows a 6×8 colored torus using alternate coloring, and a trace of information propagation from one node to another.

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 $g(T_{k_1,k_2}) \le k$, where $k = \max\{k_1, k_2\}$; it is optimal when $k_1 = k_2$.*

PROOF. From Theorem 5.1, a 2-color ring with even k nodes completes gossiping in $k/2$ time units. 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 ··· 1313 ···, 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 ··· 2424 ···, the gossiping time is k. Therefore, if we always color the rings of the longer dimension with ··· 1313 ···, we have $g(T^c_{k_1,k_2}) \le k$, 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 2. □



(a) Mesh 8 x 8

(b) Torus 6 x 8

Fig. 2. Examples of a 4-color mesh and a 4-color torus 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.

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 $g(T_{k_1,k_2,\ldots,k_n}) \le nk/2$; 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 2, we

find that our termination detection algorithm is optimal for the chain and the even ring ($d - g$ is a constant two to four time steps). From Theorems 5.3 and 5.5, our termination detection algorithm is near-optimal for even square meshes, and 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 in terms of termination delay 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, full-duplex communication model. It is near-optimal for the other cases. The time optimality also implies message optimality because the termination delay determines the number of extra communication operations needed after a global termination state has emerged. Moreover, the edge-coloring technique used in the algorithm allows efficient communication while avoiding the possibility of message collisions and congestions.

From Table 2, 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. 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 implemented our termination detection algorithm in two major applications of iterative data-parallel computations (WaTor simulation and parallel image thinning) [22], [23] and found the overhead due to the execution of this algorithm to be minimal. In [22], we also experimented with periodic distributed remapping which helped to achieve better efficiency of the computations. The remapping mechanism, which is based on some nearest-neighbor load balancing algorithms [23], [24], had to incorporate our termination detection algorithm in implementation.

References

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

[2] W.J. Dally, "Performance Analysis of k-ary n-cube Interconnection Networks," *IEEE Trans. Computers*, vol. 39, no. 6, pp. 775–785, June 1990.

[3] A.M. Farley and A. Proskurowski, "Gossiping in Grid Graphs," *J. Combinatorics, Information, and System Sciences*, vol. 5, no. 2, pp. 161–172, 1980.

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

[5] 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. Englewood Cliffs, N.J.: Prentice Hall, 1988.

[6] N. Francez, "Distributed Termination," *ACM Trans. Programming Languages and Systems*, vol. 2, pp. 42–45, Jan. 1980.

[7] N. Francez and M. Rodeh, "Achieving Distributed Termination without Freezing," *IEEE Trans. Software Engineering*, vol. 8, no. 3, pp. 287–292, 1982.

[8] C. Hazari and H. Zedan, "A Distributed Algorithm for Distributed Termination," *Information Processing Letters*, vol. 24, pp. 293–297, 1987.

[9] S.M. Hedetniemi, S.T. Hedetniemi, and A.L. Liestman, "A Survey of Gossiping and Broadcasting in Communication Networks," *Networks*, vol. 18, pp. 319–349, 1988.

[10] S.L. Johnsson and C.-T. Ho, "Optimum Broadcasting and Personalized Communication in Hypercubes," *IEEE Trans. Computers*, vol. 38, no. 9, pp. 1,249–1,268, Sept. 1989.

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

[12] A. Liestman and D. Richards, "Network Communication in Edge-Colored Graphs: Gossiping," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, no. 4, pp. 438–445, Apr. 1993.

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

[14] L.M. Ni and P.K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *Computer*, vol. 26, no. 2, pp. 62–76, Feb. 1993.

[15] S.P. Rana, "A Distributed Solution of the Distribution Termination Problem," *Information Processing Letters*, vol. 17, pp. 43–46, 1983.

[16] S. Rönn and H. Saikkonen, "Distributed Termination Detection with Counters," *Information Processing Letters*, vol. 34, pp. 223–227, 1990.

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

[18] B. Szymanski, Y. Shi, and S. Prywes, "Synchronized Distributed Termination," *IEEE Trans. Software Engineering*, vol. 11, no. 10, pp. 1,136–1,140, Oct. 1985.

[19] R.W. Topor, "Termination Detection for Distributed Computations," *Information Processing Letters*, vol. 18, no. 1, pp. 33–36, 1984.

[20] L.G. Valiant, "A Bridging Model for Parallel Computation," *Comm. ACM*, vol. 33, no. 8, pp. 103–111, Aug. 1990.

[21] C.-Z. Xu and F.C.M. Lau, "Distributed Termination Detection of Loosely Synchronized Computations," *Proc. Fourth IEEE Symp. Parallel and Distributed Processing*, pp. 196–203, Dec. 1992.

[22] C.-Z. Xu and F.C.M. Lau, "Decentralized Remapping of Data Parallel Computations with the Generalized Dimension Exchange Method," *Proc. 1994 Scalable High Performance Computing Conf.*, pp. 414–421, May 1994.

[23] C.-Z. Xu and F.C.M. Lau, "The Generalized Dimension Exchange Method for Load Balancing in k-ary n-cubes and Variants," *J. Parallel and Distributed Computing*, vol. 24, no. 1, pp. 72–85, Jan. 1995.

[24] C.-Z. Xu, B. Monien, R. Lüling, and F.C.M. Lau, "Nearest-Neighbor Algorithms for Load-balancing in Parallel Computers," *Concurrency: Practice and Experience*, vol. 7, no. 7, pp. 707-736, Oct. 1995.

Chengzhong Xu (M'95) received the BS and MS degrees from Nanjing University, China, and the PhD degree from the University of Hong Kong in 1986, 1989, and 1993, respectively, all of them in computer science. From 1994 to 1995, he was a guest professor at the University of Paderborn, Germany. He is currently a visiting assistant professor in electrical and computer engineering at Wayne State University, Detroit, Michigan. His research interests are primarily in the design and analysis of algorithms for mapping and load-balancing problems in parallel computers, and in the development of parallel programming environments for combinatorial optimizations and data-parallel applications.

Francis C.M. Lau (S'78-M'87) has a BSc degree from Acadia University, Canada, and MMath and PhD degrees from the University of Waterloo, Canada. He joined the Department of Computer Science at the University of Hong Kong in 1987 where he is now a senior lecturer. His research interests are in parallel and distributed computing, object-oriented programming, and operating systems.