Communication Pattern Based Methodology for Performance Analysis of Termination Detection Schemes

Yili Tseng
Department of Computer Information Sciences
Florida A & M University
E-mail: ytseng@ieee.org

Ronald F. DeMara
School of Electrical Eng. & Computer Sci.
University of Central Florida
E-mail: demara@mail.ucf.edu

Abstract

Efficient determination of processing termination at barrier synchronization points can occupy an important role in the overall throughput of parallel and distributed computing systems. Even though relatively efficient termination detection techniques have been proposed for certain environments, no effective performance analysis methodology has been introduced to determine application attributes that favor the use of a particular termination detection technique. This fact has hindered the adoption and development of termination detection schemes. This paper addresses this problem by developing a communication pattern based methodology to improve the precision of the theoretical performance of termination detection techniques in lieu of laborious experiments or potentially subjective benchmarking studies. By measuring message complexity from the idle period respect, it provides a simple and effective way to evaluate existing termination detection techniques or design new termination detection algorithms.

1. Introduction

Parallel processing techniques frequently offer cost-effective ways to boost throughput. As networking technology matures and environments such as the Internet rapidly expand, distributed computing is another effective method to fully utilize the available resources to increase throughput. Efficient barrier synchronization and quiescence detection techniques are essential for optimizing throughput in multiple processor architectures. An ensemble of Processing Elements (PE's) is said to be synchronized, or to have reached a quiescent state [1], upon completion of each interval of concurrent activity. Points at which synchronization occur are referred to as barriers [7][8][9].

The barrier synchronization or termination detection problem is a fundamental issue to both parallel and distributed computing. The design objective of a termination detection technique is to minimize the overhead required to enforce completion of each barrier prior to the resumption of subsequent processing. Its performance affects the overall performance of the parallel and distributed multiprocessor systems profoundly since any idle processor in the system cannot proceed to execute the next procedure before the synchronization has been completed. Even if the processors can be reactivated to process tasks of another application to utilize the processing cycles, overhead is incurred [6]. Moreover, many termination signaling techniques support only one active barrier per PE. An ineffective termination detection scheme will exchange more messages which eventually congest the communication channels and impair the transmission of messages required by the underlying computation. Therefore the termination detection process plays an important role in parallel and distributed computing environments

The termination detection problem in parallel and distributed computing has been extensively researched in the past and many termination detection algorithms have been proposed, both with software-based [1] [8] [9] [10] [11] [13] [17] and hardware-based [12] [14] [15] [16] [6] solutions. As those techniques have evolved the performance analysis still remains an obstacle. Even algorithms claiming optimal performance give only asymptotic estimates and relative comparisons [3][4]. That fact makes new designs hard to evaluate whether practical improvement has been achieved and for which applications are the techniques best-suited. The message exchange patterns of various parallel applications differ significantly among different problems because of their nature. Thus, it is difficult to assemble a benchmarking suite for parallel and distributed applications representative of typical barrier activity because the parallelization approaches vary widely. Hence, a new methodology to evaluate termination detection algorithms under a wide range of parallel and distributed approaches would be useful. With such a methodology, a termination detection technique can be precisely analyzed and fine-tuned before being rigorously put into practical experiments and/or benchmarking tests. In this paper, an communication pattern based methodology is proposed to provide more precise theoretical performance analysis of termination detection techniques.

In Section 2, we introduce details of the difficulties encountered in performance analysis of termination detection techniques through inspection of the typical algorithms of three major categories. In Section 3, an idle period based methodology is proposed with application information to improve the precision of performance analysis. The concluding remarks are made in Section 4.

2. Obstacles of performance analysis

Although lots of barrier synchronization schemes have been proposed, most resort to simulations on sequential computers or small-scale experiments on parallel systems when it comes to the performance analysis because of lack of benchmarking suites. There are three major drawbacks: 1. Parallel systems are costly and unavailable to most researchers. 2. Either simulation or experiment needs more efforts than quantitative analysis. 3. Both experiment and simulation cover only few small-scale cases. Quantitative analysis is still the best way to carry out performance analysis because it is easy to expand to enclose all cases or wrap the cases which the researchers are interested in. Usually, the designer needs only to experiment the cases he is concerned. However, difficulty exists in quantitative analysis of termination detection performance. In this section, three notable termination detection algorithms which belong to three main classes and claim message-optimal performance, namely the CV algorithm [3], the LTD algorithm[4], and the Tiered algorithm [14][6][5], are introduced to reveal their intricate considerations impacting their performance, even they all apply quantitative performance analysis. Their optimality justify the representation of the typical termination detection schemes in their class..

```
Procedure start;
(* performed by the root node when it decides to
   detect termination of the underlying computation *)
begin
        mystate ⇐ DT;
        for each outgoing network link ln do
                begin
                        color ln;
                        Send a warning message on ln
                end
end

Procedure receive_warning;
(* performed when a node p, receives a warning message
from its neighbor q *)
begin
        color the incoming link (q,p);
        if (mystate <> DT) then
                begin
                  mystate ⇐ DT;
                  for each outgoing link ln do
                      begin
                        color ln;
                          Send a warning message on ln
                      end
                  end
                end;
end
```

Figure 1. Procedures used in CV algorithm

2.1. CV algorithm

The procedures used in the CV algorithm [3] are given in Figures 1 and 2. The CV algorithm first organizes all the participating processors as a logical spanning tree of PE's. The algorithm can start after the underlying computation begins. When the CV algorithm starts, it first flushes every link in the spanning tree to take care of messages sent before the algorithm starts. Next, the root node changes its state to DT (detecting termination), sends a warning message to each of its children, and color the link at the same time. In turn, the warning messages are passed through links connected to all its child nodes until all the participating nodes are notified of the detecting termination decision. The CV algorithm maintains a stack in each PE to keep track of sending and received activities on each PE. When a node becomes idle, it examines its stack from the top. For every received entry, it sends the

```
Procedure send_message(q : neighbor);
(* performed when a node p wants to send a message
to its neighbor q *)

begin
        Push TO(q) on the stack;
        send the actual message to node q
end;

Procedure receive_message(y : neighbor);
(* performed when a node x receives a message
from its neighbor y on the link (y,x) that was colored by x*)
begin
        receive message from y on the link (y,x)
        if link (y,x) has been colored by x) then
                push FROM(y) on the stack
end;

Procedure stack_cleanup;
begin
        while (top entry on stack is not of the form ``TO()") do
                begin
                 pop the entry on the top of stack;
                 let the entry be FROM(q);
                 send a remove_entry message to q;
                end
end;

Procedure idle;
(* performed as soon as the node becomes idle *)

begin
        stack_cleanup
end;

Procedure receive_remove_entry(y : neighbor);
(* performed when a node x receives a remove_entry message
from its neighbor y *)

begin
    scan the stack and delete the first entry of the form TO(y);
    if idle then
        stack_cleanup
end;
```

Figure 2. Procedures used in CV algorithm

remove_entry message to the sender and erases the entry from its own stack. It repeats this until it encounters a sending entry. This procedure is defined as *stack_cleanup*. When a node receives a *remove_entry* message, it scans its stack and deletes the sending entry related to this message and repeats the *stack_cleanup* procedure as previously described if it is idle. A node sends a *terminate* message to its parent when four conditions are met: 1. It is idle. 2. Its stack is empty. 3. Each of its incoming links is colored. 4. It has received the *terminate* messages from each of its children. When the root node meets the four requirements to report a *terminate* message, it declares the termination.

2.2. LTD algorithm

The LTD algorithm [4] is an improvement over the CV algorithm. Its algorithm for all PE's is given in Figure 3. The two procedures used in the algorithm are given in Figure 4. Like the CV algorithm, it organizes participating processors as a spanning tree of PE's first and it can start after the underlying computation starts. When the root decides to start the algorithm, it changes to DT (detecting termination) status and sends a *start* message to each of its children. In turn, its children send *start* messages to their own children until all participating processors are notified of the root's decision. Each PE maintains four variables, namely in_i, out_i, $mode_i$, and $parent_i$ where i is the identity of the PE to apply message counting to decide whether all the messages sent by it have been finished. The integer array, $in_i[1..n]$, is used to keep track of the messages which are received from PE$_1$ to PE$_n$ and have not been finished. The number of messages sent by PEj to PEi is stored in $in_i[j]$. The integer, out_i, records the number of unfinished messages sent by PE$_i$ itself. The Boolean variable, $mode_i$, shows the status of the processor (DT or NDT). The pointer which indicates where the most recent major message came from is stored in $parent_i$. A **major** message is the message which is received when the processor is idle and has finished all the messages which it sent to other processors, otherwise the received message is defined as a **minor** message. Whenever a node turns idle, it calls procedures *respond_minor* and *respond_major* to detect termination. What procedure *respond_minor* does is to send one *FINISH(k)* message to each non-parent node which has sent it messages to inform them of the number of messages which it has finished for them, where k is the total number of messages which it has finished for a specific node before turning idle. This is the largest improvement over the CV algorithm. It uses one *FINISH(k)* message instead of k *remove_entry* messages as in the CV algorithm to save *(k-1)* messages. As for the procedure *respond_major* , it checks if all the sent messages have been finished and sends one *FINISH(k)* message to its current parent node if all the messages sent by it are completed. After the root turns idle and finds out that all the messages it sent out are finished, it concludes the termination.

```
A1:(Upon sending a basic message to $P_j$)
        $out_i := out_i + 1$;

A2:(Upon receiving a basic message from $P_j$)
        $in_i[j] := in_i[j]+1$;
        if ($parent_i$ = NULL) ^ ( $i \neq 1$) then  $parent_i := j$;

A3:(Upon deciding to switch to DT mode) /* for $P_1$ */
 or (Upon receiving a START message) /* for $P_i$ , $2 \le i \le n$ */
        $mode_i$ := DT;
        for each child $P_j$ of $P_i$ do
                send a START message to $P_j$;
        end for
        if ( $P_i$ is idle) then
                call respond_minor(i);
                call respond_major(i);
        end for

A4:(Upon receiving a FINISHED(k)) from $P_j$
        $out_i := out_i - k$;
        if ( $mode_i$ = DT) ^ ($P_i$ is idle) then call respond_major(i);

A5:(Upon turning idle)
        if ($mode_i$ = DT) then
                call respond_minor(i);
                call respond_major(i);
        end if;
```

Figure 3. Algorithms for P_i , $1 \le i \le n$, in LTD algorithm

```
Procedure respond_minor(i: integer)
begin
        for each $j \neq parent_i$ with $in_i[j] \neq 0$ do
                send a FINISHED($in_i[j]$) to $P_j$ ;
                $in_i[j]$:= 0;
        end for;
end;

Procedure respond_major(i: integer)
begin
        if ( $out_i$ = 0 ) then
          if ( i = 1) then report termination
          else
                send a FINISHED($in_i[parent_i]$) to $parent_i$ ;
                $in_i[parent_i]$ := 0;
                $parent_i$ := NULL;
           end if;
        end if;
end;
```

Figure 4. Procedures used in LTD algorithm

2.3. Tiered algorithm

The Tiered algorithm [14][6][5] solves the termination detection problem by applying a global invariant that is maintained for all possible execution interleavings. The basic idea of the Tiered algorithm is that a node is designated as the centralized *controller* to maintain the global status of the numbers of the consumed and produced tasks, which is a global invariant. To prevent the false detection, task hierarchy information of the spawning pattern is required along with the consumption count and production count. The process creation message of the underlying computation will carry the designated process

```
Procedure Spawn_A_Task( l : level number)
begin
Increase by one the amount of produced tasks of l-level;
end

Procedure Finish_A_Task( l : level number)
begin
Increase by one the amount of consumed tasks of l-level;
end

Procedure Upon_Idle
begin
Report the amounts of locally consumed and produced tasks
of all levels to controller;
end
```

Operation of the Processing Element in Tiered Algorithm

```
Procedure Receive_A_Report
begin
        Update global ledger with the received amounts
        of local consumed and produced tasks accordingly;
        If( Check_Ledger)
        Then
            declare global termination
        End if;
end

Procedure Check_Ledger
begin
        Check ledger table to determine if consumption a
        nd production counts of every level match;
        If (yes) TRUE
        else   FALSE
end if;
end
```

Operation of the Controller in Tiered Detection Algorithm

Figure 5. Procedures used in Tiered algorithm

level number. The task spawned by a N^{th}-level task is denoted a $(N+1)^{th}$-level task. Every PE needs to update the local consumption count and production count for the related level of the spawning tree of the process whenever it produces or consumes a task. After any participating PE has finished all tasks dispatched to it and becomes idle, it reports all the amounts of the locally consumed and produced tasks at each level of the spawning tree to the controller. The controller first updates the accumulated global records it keeps accordingly. Then the controller determines whether the global consumption count and production count at each level of process-creation nesting are equal. If they are, the controller announces global termination because the amounts of global consumed and produced tasks are equal at each level of the spawning tree. Otherwise, it waits for the next reporting and executes the next round of checking until the global termination is met. The procedures applied by the controller and PE's are listed in Figure 5.

2.4. Difficulty in performance analysis

Quoting from the original papers, the CV algorithm [3] stated that the number of control messages used by it is $T+2L+N-1$ with the order of $\theta(T+L)$. The notation has been unified here to provide consistent comparison among alternate techniques. Let N denote the number of joining PE's while L denotes the number of links connecting them. Let T be the total number of processes created in the barrier. The LTD algorithm as described in their design may require up to $T+2(N-1)$ control messages while fewer than that are necessary on average. [4] However, the authors also indicated that it is virtually impossible to acquire an average-case due to lack of activity patterns for different parallel and distributed applications [4]. Therefore, to what extent improvement can be achieved in practice remains unclear. That fact makes adoption of their design difficult without practical experiments. Even though the Tiered algorithm seems more appealing for many applications than either the CV or LTD algorithms, it is difficult to strongly advocate the advantages by providing message complexity analysis [14][5]. Since only very rough impression of relative performance is available, it cannot effectively help the user choose the right design, let alone matching a specific parallel application with a suitable termination detection algorithm. Apparently, an effective methodology is in need to provide more precise performance analysis. Further research efforts will also benefit from it during development of new termination detection techniques.

3. Idle period based methodology

A communication pattern based methodology [6] is introduced to provide a more effective way to analyze the performance of the termination detection algorithms independent of the need for implementation and execution of benchmarking tests.

3.1. Idle period based approach

When CV algorithm, LTD algorithm, and Tiered algorithm are compared side-by-side, we observe that LTD and Tiered algorithms are processor-centered, contrary to process-centered of the CV algorithm. The process-centered algorithms send out control messages on per-process basis while the processor-centered schemes exchange control messages on per-processor basis, usually after the processor becoming idle like LTD and Tiered algorithms. That offers a clue to take advantage of the number of PE's in action because the control messages are sent only after a processor becomes idle. With closer inspection, we observe that control messages may exceed the number of the participating PE's without regularity in the Tiered algorithm because of its support of processor reactivation. Hence, we can utilize the number of occurrences that PE's become idle instead of the number of joining processors. In other words, any PE needs to report its status after becoming idle in both the LTD and

Tiered algorithms. Thus, the count of idle periods can be used as a basis to analyze message complexity more precisely. In particular, this is how the methodology proposed here is named an idle period based performance approach. If the pattern of subtask dispatching in parallel and distributed computation can be generalized or formulated, the performance analysis of termination detection techniques can be effortlessly performed. However, no research in that respect exists so far. Before that time comes, the number of idle period remains the most effective measure to estimate the performance.

3.2. Application of idle period based performance assessment

Interpreting an idle period based performance perspective, the number of messages required for the Tiered algorithm can be easily determined as *E*, where *E* is the number of idle periods in a barrier. In other words, there are *E* occasions that a report should be sent to the controller regardless of the process distribution pattern of the barrier. The best case for the LTD algorithm occurs when exactly one message needs to be reported after each PE becomes idle. The (*E-1*) quantity of *FINISHED* messages for all idle periods except the one happening on root node and *(N-1) start* messages makes *E+N-2* control messages required. The worst case degrades the LTD algorithm to the performance of CV algorithm when one *FINISHED* message is necessary for each process in the barrier. Under that circumstance, *T+N-2* control messages are required. For the CV algorithm, *T+2L+N-1* messages are necessary for all occasions. The results are summarized in Table 1 [6]. The exact estimation of message number is given for Tiered algorithm while the lower and upper bounds of message complexity for the LTD algorithm are clearly marked with the communication pattern based methodology. Lacking the idle period based methodology, both the LTD and Tiered algorithms can only claim they have less messages complexity than CV algorithm for most cases without precise estimation of the improvement.

Table 1. Message complexity acquired with idle period based approach

Algorithm	Best case	Worst case
CV algorithm	*T+2L+N-1*	*T+2L+N-1*
LTD algorithm	*E+N-2*	*T+N-2*
Tiered algorithm	*E*	*E*

3.3. Determination of E

The number of idle periods in a barrier could be determined or confined without much difficulty by the characteristics of parallel applications. The following examples demonstrate the mechanisms for two typical categories of parallel applications.

3.3.1. Deterministic applications – parallel matrix multiplication. Assume we want to carry out the multiplication of two *M* by *M* matrices on a parallel system with *N* PE's. There are M^2 inner products to be calculated to assemble the final result. Let us equally dispatch those subtasks to all PE's in the system and evaluate the quantities of messages required by each algorithm. Since the subtasks are equally distributed to each PE, every node will execute them one by one and become idle afterwards. An event is constituted only after a PE becomes idle. Therefore, there are ***N*** idle periods in this barrier, same as the number of nodes in the system. After the quantity of events is determined, the required message number can be easily found. Only *N* messages are issued by the Tiered algorithm. Same spanning trees are assumed for both LTD and CV algorithms to be on the same comparison base. All those subtasks can be directly issued by a root node. Hence, the LTD algorithm needs to report only one *FINISHED*(M^2/N) message for each idle event from each child node to the root node, *(N-1)* messages in total. Counting setup messages, *(2N-2)* messages are necessary for LTD algorithm. As for CV algorithm, one *terminate* messages for each idle period from each child node is required, *(N-1)* totally. However, one message for each subtask is also required to the issuing node. In all, M^2 such messages are required. Finally combined with the setup messages, $M^2+2L+N-1$ messages are required by CV algorithm.

3.3.2. Non-deterministic applications – parallel marker-passing reasoning. Marker-passing is an important mechanism in parallel artificial intelligence applications. As shown in Figure 6, a well-known knowledge base about Clyde the elephant is stored in a semantic network [19][18], of which concept nodes span four PE's in our simple example. The query "What color is Clyde?" can be solved with the following procedures: 1. The node Clyde is marked with M1 marker. 2. The M1 marker is propagated through the IS-A links in parallel

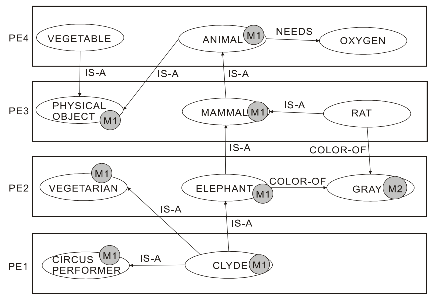

Figure 6. Modified parallel marker-passing example from Fahlman

until all nodes superior to Clyde are marked with M1. 3. Each COLOR-OF link of which incoming end is connected with M1 marker marks the node at its output end with M2 marker. 4. The node marked with M2 reports its color. Detection of termination of the marker propagation is required for step 2. Let us evaluate it with the idle period based methodology. Inspecting Figure 6, there is obviously only one idle period for PE1, PE2, and PE4 because the subtasks are all dispatched to them simultaneously and will be executed continuously. However, the M1 marker is first passed to MAMMAL node on PE3, then propagated to PE4, and passed back to PHYSICAL OBJECT node on PE3 again. In the best case, there is only one idle period for PE3 if M1 marker is passed back before PE3 can finish the accompanying function for MAMMAL node. In the worst case, there are two idle periods for PE3 if the M1 marker is propagated back after the function for MAMMAL node is finished and PE3 reports. For the current example, E is determined as either 3 or 4. The numbers can be substituted into the formulae of Table 1 and a more precise than before estimation is available. More instances should be applied to constitute the average case for a specific problem the designer is looking into. Actually, idle period based methodology is more helpful for non-deterministic applications because of the applications' irregularity and complexity. Helped by the idle period based methodology, all cases, namely the best, the worst, and the average can make a more precise theoretical estimation without rigorously running customized benchmarking programs. Even exact estimation is still unavailable; the difference of required message number among different algorithms is more noticeable because of the introduction of E. Adoption of a suitable algorithm for a specific application will be more achievable.

3.4. Additional aids

Aside from message number, data bits required for termination detection control messages should also be considered for network bandwidth assessment. With message complexity more accurately identified, the bit complexity can be more precisely predicted as well. The results are shown in Table 2 [6]. This can help users adopt the appropriate termination detection algorithm based upon tradeoffs presented by the underlying network. On the other hand, considering the reality that parallelism differs significantly among different parallel applications; with E predictable for specific parallel or distributed applications, both proper hardware and termination detection algorithm can be chosen to gain best performance. For example, best combination of machine and termination detection algorithm can be determined for different parallel applications after taking corresponding values of E, N, L, and T into consideration. Detection delay is another way to measure the efficiency of termination detection techniques. The idle period concept can somewhat help. The detection delay can be measured by the hierarchy the message must traverse after the last PE where the last subtask is executed becomes idle. For Tiered algorithm, exactly one message is sent to the central controller after the last active PE turns idle. The order of detection delay is one. As for CV and LTD algorithms, depending on the structure of the spanning tree and the position where the last subtask resides, the detection delay can range from time to send one message to time to send (N-1) messages.

Table 2 Message bit complexity acquired with idle period based approach

Algorithm	Best Case	Worst Case
CV algorithm	$(T+2L+N-1)(\lceil \lg N \rceil+1)$	$(T+2L+N-1)(\lceil \lg N \rceil+1)$
LTD algorithm	$(E+N-2)(\lceil \lg T \rceil+1)$	$(T+N-2)(\lceil \lg T \rceil+1)$
Tiered algorithm	$2E\lceil \lg T \rceil$	$4T\lceil \lg T \rceil$

4. Conclusion

As parallel and distributed computing offers the potential for increased throughput over uni-processor systems, barrier synchronization plays an important role to the performance of multi-processor systems. Even efficient termination detection techniques that have been proposed lack a useful performance analysis methodology. An communication pattern based methodology on idle periods is proposed to improve the precision of performance analysis of theoretical termination detection techniques. It proves to provide a simple and effective way to help evaluate existing termination detection techniques or design new termination detection algorithms. With the help from the same methodology, suitable parallel and distributed hardware can be determined to match specific applications.

5. References

[1] F. Mattern, "Algorithms for Distributed Termination Detection," Distributed Computing No. 2, pp. 161-175, 1987

[2] F. Mattern, "Global quiescence detection based on credit distribution and discovery," *Information Processing Letters*, Vol.30, No.4, pp.195-200, Feb., 1989

[3] S. Chandrasekaran and S. Venkatesan, "A message-optimal algorithm for distributed termination detection," *Journal of Parallel and Distributed Computing*, Vol.8, No.3, pp.245-252, March, 1990.

[4] T.-H. Lai, Y.-C. Tseng, X. Dong, "A more efficient message-optimal algorithm for distributed termination

detection," Technical Research Report, OSU-CISRC-1/92-TR-2, Ohio State University, 1992.

[5] K. Drake, "Time and Space Efficient Multiprocessor Synchronization and Quiescence Detection," M.S. Thesis, University of Central Florida, 1995

[6] Y. Tseng, "High Performance Termination Detection Techniques Supporting Multithreaded Execution," Ph.D. Dissertation, University of Central Florida, 2000

[7] N.S. Arenstorf and H.F. Jordan, "Comparing Barrier Algorithms," *Parallel Computing*, 12, pp.157-170, 1989

[8] E.D. Brooks III, "The Butterfly Barrier," *International Journal of Parallel Programming*, Vol.15 No.14, pp.295-307, 1986

[9] D. Hensgen, R. Kinkel, and U. Manber, "Two Algorithms for Barrier Synchronization," International Journal of Parallel Programming, Vol.17 No.1, pp.1-17, 1988

[10] K.H. Cheng and Q. Wang, "A Simultaneous Access Design for Idle Processor Reactivation and the Detection of the Termination of a Parallel Activity," *Journal of Parallel and Distributed Computing*, Vol. 17, pp.370-373, 1993

[11] H. Xu, P.K. McKinley, and L.M. Ni, "Efficient Implementation of Barrier Synchronization in Wormhole-Routed Hypercube Multicomputers," *Journal of Parallel and Distributed Computing*, Vol. 15, pp.172-184, 1992

[12] S. Shang and K. Hwang, "Distributed Hardwired Barrier Synchronization for Scalable Multiprocessor Clusters," *IEEE Transactions on Parallel and Distributed Systems*, Vol.6, No.6, pp.591-605, June 1995.

[13] J.-S. Yang and C.-T. King, "Designing tree-based barrier synchronization on 2D mesh networks," *IEEE Transactions on Parallel and Distributed Systems*, Vol.9, No.6, pp.526-533, June 1998.

[14] R. DeMara, B. Motlagh, C. Lin and S. Kuo, "Barrier synchronization techniques for distributed process creation," *8th International Parallel Processing Symposium Proceedings*, pp.597-603, April 1994.

[15] H.G. Dietz, R. Hoare, and T. Mattox, "A fine-grain parallel architecture based on barrier synchronization," *Proceedings of the Int'l Conf. on Parallel Processing*, pp.247-250, Aug. 1996

[16] R. Hoare et al., "Bitwise aggregate networks," *8th IEEE Symposium on Parallel and Distributed Proceedings*, Oct. 1996

[17] T.H. Lai and L.-F. Wu, "An *(N-1)*-resilient algorithm for distributed termination detection," IEEE Transactions on Parallel and Distributed Systems, Vol.6, No.1, pp.63-78, Jan. 1995.

[18] G.L. Almasi and A. Gottlieb, "Highly parallel computing," 2nd Edition, Benjamin/cummings Publishing Company, Inc., 1994

[19] S.E. Hahlman, "NETL: A system for representing and using real-world knowledge," MIT press, 1979