JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 18, 516–523 (1993)

Utilizing Global Simulation Information in Conservative Parallel Simulation on Shared Memory Multiprocessors*,†

JIAJEN M. LIN AND SANTOSH G. ABRAHAM

Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, Michigan 48109-2122

Global simulation information is accessible on shared memory multiprocessors and can improve the efficiency of parallel simulation. However, most existing concurrent simulators do not aggressively exploit this information. In this paper, we propose a *Directly Accessing Information* (DAI) scheme, a conservative simulation scheme that collects useful global simulation information in shared memory systems to reduce non-essential blocking and resolve local deadlocks. A parallel queueing network simulator was constructed on a Sequent Symmetry multiprocessor. Special implementation techniques were used to eliminate locking mechanisms for accessing global simulation information. Also, search pruning techniques were developed to reduce the amount of global simulation information used to unblock Logical Processes (LPs). Experimental results demonstrate that the DAI scheme achieves good speedups and substantially outperforms conventional conservative schemes in a reasonably large problem domain where message densities are larger than approximately 0.75. © 1993 Academic Press, Inc.

1. INTRODUCTION

In conventional parallel discrete event simulation using the conservative approach [3], non-essential blocking can greatly reduce simulation parallelism. Non-essential blocking occurs when the message to be simulated next has arrived but sufficient information from neighboring LPs (logical processes) is not yet available to confirm that no message with an earlier timestamp can arrive. This blocking can possibly lead to a *local deadlock* which consists of a set of blocked LPs linked by a cycle of empty channels. And, local deadlocks may eventually evolve into a *global deadlock* when no LP can be activated, often because all LPs are linked by a cycle of empty channels.

Non-essential blocking can be reduced by exploiting information in the physical system [5, 10] or information contained in the simulation itself. In this paper, we focus on exploiting simulation information to improve performance. This information includes the simulation states of LPs, simulated times of LPs, and timestamps of messages on channels. Simulation information is considered *local information* to a host processor if the corresponding LPs and channels are currently assigned to that processor. The simulation information of all other LPs is considered *global information*.

Conventional parallel simulation algorithms were originally developed for distributed systems. In such systems, even obtaining a small amount of global simulation information through *null messages* [3] or *probe messages* can easily increase network traffic and cause performance degradation. Recently, parallel discrete event simulators have also been developed on shared memory multiprocessors [1, 4, 8, 14, and 15]. In such systems, the overhead and delays for exchanging global simulation information are relatively small. Therefore, global simulation information can be exploited to a much larger extent to unblock an LP and reduce non-essential blocking.

Motivated by the high potential benefit, we propose a *Directly Accessing Information* (DAI) scheme which aggressively exploits global simulation information to reduce non-essential blocking and resolve consequent deadlocks. This scheme includes pruning techniques to eliminate unnecessary search, as well as special implementation techniques to eliminate all locking mechanisms associated with accessing global simulation information. A queueing network simulator was implemented on the Sequent Symmetry multiprocessor to evaluate our DAI scheme. The experimental results demonstrate that the benefit of reducing non-essential blocking can outweigh the cost of searching for global simulation information. We show that global simulation information can be used in an efficient, asynchronous, and highly parallelized way.

The remainder of this paper is organized as follows. Section 2 reviews and compares the related work. Section 3 presents our DAI scheme and Section 4 discusses how to efficiently implement this scheme. Section 5 describes the queueing network simulator and related empirical studies on the Sequent Symmetry. Section 6 concludes this paper.

* We acknowledge the Mathematical and Computer Science Division at the Argonne National Laboratory for providing access to the Sequent multiprocessor. This research was supported in part by NSF Grant NSF-ASC-880890 and ONR Contract N00014-93-1-0163.

† This paper is an improved version of [11], and contains additional experimental results.

0743-7315/93 $5.00

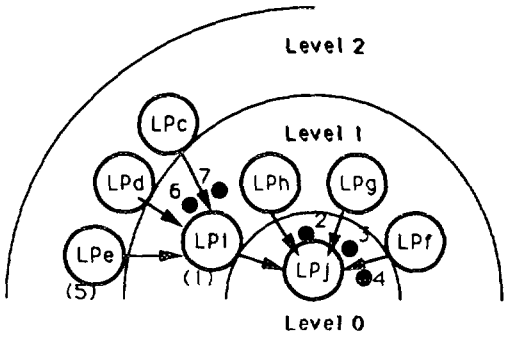

FIG. 1. Effect of using global simulation information.

2. RELATED WORK

Many studies have been performed to reduce non-essential blocking and/or resolve consequent deadlocks [5]. Among these research efforts were several empirical studies conducted on shared memory multiprocessors [1, 4, 12, 14, 15].

To illustrate our idea of aggressively exploiting shared memory, we present an example in Fig. 1, which shows an LP, an LP_j, and some of its blocked ancestors. The simulated time of an LP is shown in each pair of parentheses and the timestamp of a message is shown without parentheses. Simulated times and message timestamps are denoted in the same way in all the other figures in this paper. Three circular arcs are drawn in the figure to mark different levels of global simulation information for LP_j.

A direct implementation of the Chandy–Misra–Bryant (CMB) algorithm with null messages, as in [14], will access only local simulation information (i.e., the information at Level 0). In a simulation scheme that directly accesses the simulated times of neighboring LPs as in [4, 15], some global simulation information at Level 1 (i.e., the simulated times but not message timestamps) is available. Our DAI scheme extends the search for global simulation information to as many levels as necessary. In this example, global simulation information in Level 2 guarantees that LP_i will not process a message with a timestamp less than 5. LP_j can then be unblocked to process the message with timestamp 2 without any delays, even though its parent, LP_i, is still blocked.

3. THE DAI SCHEME

In this section, we first describe the simulation model. Then, we define and use search trees to obtain lower bounds on the timestamps of future incoming messages for LPs. Using these lower bounds, our DAI scheme can reduce non-essential blocking sufficiently to avoid global deadlocks. Finally, we present out simulation algorithm.

3.1. Simulation Model

The simulation model consists of a set of LPs interacting with each other by transmitting messages with timestamps through directed channels with infinite-sized buffers. Each message has a *send-timestamp* and a *receive-timestamp*. They represent the initiation and reception times of the corresponding interaction between physical components. Along a particular channel, messages are sent in increasing order of their send-timestamps and the message transmission delays are such that they are also received in the same order. The *message acceptance horizon* of an LP is a computed lower bound on the receive-timestamps of future incoming messages. An LP computes its message acceptance horizon and selects a message with the lowest receive-timestamp if it is less than or equal to the horizon. A selected message is processed to completion nonpreemptively. After processing a message, an LP puts outgoing messages on the corresponding channels, advances its simulated time, and then removes the processed message from its input channel. Note that, with shared memory multiprocessors, channels are implemented in shared memory. Therefore, assuming a memory model that is as strong as processor consistency [6], an explicit acknowledgment is not required after depositing outgoing messages in shared memory.

For simplicity, we assume that each LP always has incoming channels from its parent LPs and no LP can be its own parent. (LP_i is a parent of LP_j if LP_i can send messages directly to LP_j.) No generality is lost by this assumption because extra LPs and channels can be created, if necessary, to satisfy these assumptions. The following notation is used in this paper:

- (LP_i, LP_j): channel from LP_i to LP_j
- $\mathcal{S}_j(t)$: state of LP_j
- $\mathcal{T}_j(t)$: simulated time of LP_j
- $\mathcal{M}_{ij}(t)$: receive-timestamp of the earliest message on non-empty (LP_i, LP_j)
- $\mathcal{H}_j(t)$: message acceptance horizon of LP_j
- $\mathcal{B}_j(t)$: lower bound on the send-timestamp of any future messages sent by LP_j

The state $\mathcal{S}_j(t)$ of LP_j is *active* if LP_j is processing a message, *idle* if LP_j is not active and has no available messages, *ready* if LP_j is not active and has at least one non-empty incoming channel with $\mathcal{M}_{ij}(t) \leq \mathcal{H}_j(t)$, and *blocked* otherwise.

3.2. Search Trees

To compute a message acceptance horizon for an LP, we first derive lower bounds on the send-timestamps of any future messages sent by its parents. The derivation of

the bounds uses *search trees*, which are defined as follows.

DEFINITION 1. A search tree of LP_r is (V, A), where V is a set of vertices and A is a set of arcs. The tree is constructed using a stack and two attributes, denoted *lp* and *lb*, in each vertex. Initially, $V = \{v_r\}$ where $v_r \cdot lp$ is LP_r, $A = \{\}$, the stack is empty, and the tree is constructed by a call to search_tree(LP_r, v_r).

```
search_tree(LP_j, v_y) {
    if (LP_j is in the stack) { v_y · lb(t) = ∞; return; }
    if (LP_j is active) { v_y · lb(t) = T_j(t); return; }
    push LP_j into the stack;
    for (each empty (LP_i, LP_j)) {
      create v_x; v_x · l_p = LP_i; V = V ∪ {v_x};
      create {(v_x, v_y)}; A = A ∪ {(v_x, v_y)};
      search_tree(LP_i, v_x); }
    pop LP_j out of the stack;
    v_y · lb(t) = max(T_j(t), min(min_i{M_ij(t)}, min_x{v_x · lb(t)})), where
    (LP_i, LP_j) is not empty, and (v_x · lp, LP_j) is empty; return; }
```

THEOREM 1. $\mathcal{B}_r(t) = v_r \cdot lb(t)$, *where* v_r *is the root of* LP_r*'s search tree.*

COROLLARY 1. $\mathcal{H}_j(t) = \min_i\{\mathcal{B}_i(t)\}$, *where* LP_i *is a parent of* LP_j.

Theorem 1 and Corollary 1 show how the message acceptance horizon, $\mathcal{H}_j(t)$, of LP_j can be determined using the search tree. The proofs of these results and others are not included because of space constraints but are presented in [9].

3.3. Deadlock Resolution

The following theorem assumes that lower bound computation is done instantaneously at each LP with the most current simulation information of all other LPs.

THEOREM 2. *There will be no global deadlocks in simulating a non-deadlocking physical system if Corollary 1 is applied to all blocked* LPs *for computing their message acceptance horizons.*

Figure 2a shows a local deadlock consisting of three blocked LPs linked by a cycle of empty channels. The implementations of the CMB algorithm with null messages [4, 14, 15] cannot unblock any of the LPs when there is no lookahead. The implementations of the CMB algorithm with global deadlock resolution [4, 14, 15] do not try to unblock any of these LPs until all other LPs have reached the end of the simulation or have formed other local deadlocks.

Our DAI scheme resolves local deadlocks and thereby avoids global deadlocks. Figure 2b shows a part of the search tree of LP_j. The numbers associated with the empty channels represent lower bounds on the send-timestamps of future messages sent along these channels. As the figure shows, a lower bound of 6 for LP_j is obtained. Thus, the message acceptance horizon of LP_h is 6, which is equal to the receive-timestamp of the earliest available message. As a result, LP_h can be unblocked to break the local deadlock.

3.4. DAI Algorithm

In Fig. 3, we present a conservative parallel simulation algorithm that utilizes the message acceptance horizons computed using the results in the previous subsections. In this algorithm, we assume that the simulation of an LP, LP_j, is performed by a dedicated host processor. Therefore, process scheduling is not included. Also, we assume that a host processor can obtain all necessary simulation information for computing the lower bound, $\mathcal{H}_j(t)$, instantaneously. (Further relaxation of these as-

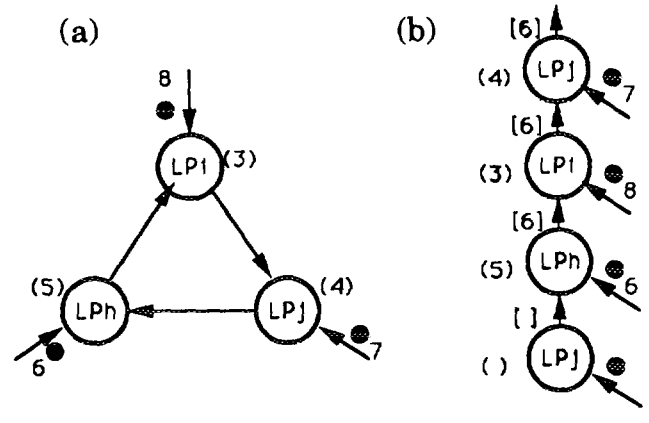

FIG. 2. (a) A snapshot of a local deadlock consisting of three blocked LPs. (b) A part of the search tree of LP_j.

```
while (T_j(t) < stop time)
   if (∃ messages for LP_j) {
      for (each (LP_i, LP_j))
         if ((LP_i, LP_j) is not empty)
            B_i(t) = receive-timestamp of the latest message on (LP_i, LP_j);
         else use Theorem 1 to compute B_i(t);
      use Corollary 1 to compute H_j(t);
      while (∃ messages with receive-timestamps ≤ H_j(t)) {
         if (T_j(t) < min_i{M_ij(t)}) T_j(t) = min_i{M_ij(t)};
         process one message with min_i{M_ij(t)};
         advance T_j(t);
         remove the message from (LP_i, LP_j);
         if ((LP_i, LP_j) becomes empty) {
            use Theorem 1 to compute B_i(t);
            use Corollary 1 to update H_j(t); }}}
```

FIG. 3. A non-deadlocking algorithm for simulating LP_j.

sumptions is discussed later.) Additional information of a physical system such as lookahead or minimum delay can be incorporated to improve our DAI scheme [9].

4. IMPLEMENTATION OF DAI SCHEME

The assumption that a host processor can access all simulation information for computing the message acceptance horizon specified in Corollary 1 without delays can only be realized by putting a global lock on all simulation information at the current time instant. In order to avoid performance degradation due to locking, we use the algorithms in Fig. 4 to compute $\mathcal{B}_i(t)$ and $\mathcal{H}_j(t)$ in the main algorithm (see Fig. 3) without any locking. This new implementation approach is shown to be correct in [9].

Recall that to determine a lower bound of an LP in Theorem 1, we search through all empty channels between the LP and its ancestors. This search space can easily become explosive when the branching factor of LPs and the number of empty channels are large. To reduce the overhead of this search, we transform the search tree of an LP to a game tree with interleaved maximizing and minimizing levels. Figure 5 shows a part of the search tree for an LP, LP_j, and the corresponding game tree. Now, we can use pruning techniques, such as the well known Alpha–Beta procedure [16], to prune our search space. The basic idea of the Alpha–Beta procedure is to prune a subtree if exploration of the subtree will not affect the final result. As illustrated in Fig. 5b, the Alpha–Beta procedure is able to identify that all subtrees below the diagonal cuts do not need to be explored. As a result, a lower bound, $\mathcal{B}_j(t)$, of 10 is computed for LP_j after visiting only three of LP_j's ancestors. To further limit the search space, we also use two heuristic pruning procedures to augment the Alpha–Beta procedure [9].

```
determine_B(LP_i) {
   if (LP_i is in the stack) { add the channel just traversed to Set_i; return ∞; }
   if (LP_i is active) return T_i(t);
   push LP_i into the stack;
   Set_i = {}; lb = ∞;
   for (each (LP_h, LP_i))
      if ((LP_h, LP_i) is not empty) lb = min(lb, M_hi(t));
      else lb = min(lb, determine_B(LP_h));
   if (LP_i sent out messages along channels in Set_i after they were added to Set_i)
      lb = 0;
   pop LP_i out of the stack;
   return max(T_i(t), lb); }

determine_H(LP_j) {
   mah = ∞;
   for (each (LP_h, LP_i))
      mah = min(mah, determine_B(LP_i));
   return mah; }
```

FIG. 4. Algorithms for computing $\mathcal{B}_i(t)$ and $\mathcal{H}_j(t)$.

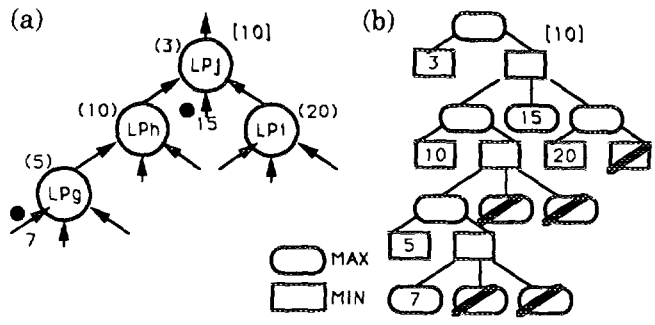

FIG. 5. (a) A part of the search tree for LP_j with a lower bound of 10. (b) The corresponding game tree that can be used to prune the search and obtain the lower bound.

5. QUEUEING NETWORK SIMULATOR USING DAI

We implemented a concurrent FCFS queueing network simulator to evaluate our DAI scheme. Queueing networks are commonly used for parallel simulation studies because they resemble the conventional simulation model [2, 14, 4]. More importantly, they provide a stress test for parallel simulators because little time is needed to simulate the processing of an event and, consequently, the time to unblock LPs becomes relatively long [12]. Our experiments are based on the algorithms shown in Figs. 3 and 4.

A 26-processor Sequent Symmetry S81 was used. The Sequent Symmetry is a private-cache bus-based MIMD shared-memory multiprocessor. Each of its Intel 80386 processors has a floating point co-processor and a 64-kilobyte cache. The processor clock cycle time is 62.5 ns, the bus clock rate is 10 MHz, and the bus bandwidth is 80 Mbytes per second. In this section, we first describe several experimental aspects of the simulator and then present the performance results.

5.1. Partitioning and Scheduling LPs

In our algorithms, the simulation of an LP was assumed to be performed by one dedicated host processor. In our experiments, each host processor simulates a group of LPs within a single process. The LPs are partitioned and assigned to host processors so that the interactions between host processors are minimized. For instance, we partition the toroid network into squares or rectangular blocks and assign each block to a host processor. We use a static scheduling scheme because the global scheduling queue can become a bottleneck in dynamic scheduling. In our implementation, a host processor visits and tries to simulate its LPs in a round-robin fashion. We do not maintain a queue of ready LPs because the overhead of maintaining the queue is relatively high in queueing network simulation applications.

5.2. Simulator Implementation

The results derived in the previous two sections and the scheduling mechanism were used to construct a parallel queueing network simulator. For comparison, a sequential simulator was constructed by modifying our parallel simulator and eliminating all unnecessary overhead for parallelization. The event list in the sequential simulator was implemented as a splay tree to improve performance [7].

We also constructed a parallel simulator that examines only parent LPs' simulated times for comparison. Again, in order to obtain a fair comparison, the simulator was constructed by modifying our parallel simulator and eliminating all unnecessary overhead for accessing global simulation information. Since this simulator cannot avoid global deadlocks, a mechanism for detecting/breaking global deadlocks was implemented by maintaining a global counter that indicates the number of blocked or idle LPs.

5.3. Empirical Results

Using the three simulators, we conducted experiments to simulate various queueing networks. Table I shows the parameters used in our experiments [9]. Our major goal was to determine the parameters which affect the performance of the DAI scheme and to show the problem domain in which the DAI scheme is able to achieve good speedups.

Two distributions were used for the service times of the queue servers. They are *uniform* (DIST = uniform = LA + 2 × (10-LA) × rand) and *exponential* distributions (DIST = expntl = LA-(10-LA) × ln(rand)), where LA is the lookahead and rand is a random value uniformly distributed between 0 and 1. Each LP independently generates its random number using a unique seed.

The *branching factor* of a network is the average number of incoming channels per node. In our experiments, we used LPs (representing queues and their servers) configured as a toroid (a two-dimensional mesh with wrap

TABLE I
Parameters, Notations, and Their Values Used in the Experiments

Parameter	Notation	Values
Number of host processors	NHP	1, 2, 4, 8, and 16
Timestamp increment distribution	DIST	Uniform and exponential
Branching factor	BF	4, 6, and 8
Computation granularity	GRN	0, 1000, 2000, 3000, and 4000
Message population ratio	MPR	0.5, 0.75, 1, 1.5, and 2
Lookahead	LA	0, 1, 2, 3, 4, and 5
Number of LPs	NLP	64, 256, and 1024

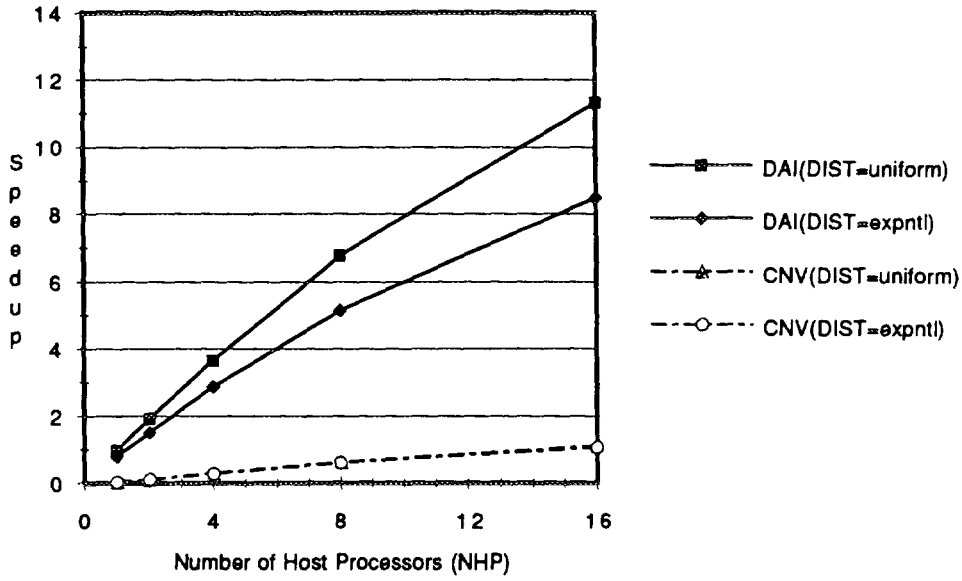

FIG. 6. Speedup with the toroid topology. (BF = 4, GNR = 0, MPR = 1, LA = 0, NLP = 256.)

around connections), cuboid (a three-dimensional mesh with wrap around connections), or hypercube network having branching factors of four, six, and eight, respectively. In our presentation of experimental results, the branching factor represents one of the three topologies described. The *computational granularity* represents the additional time spent in processing a message. The *message population ratio* is the total number of messages in the system divided by the number of channels. Initially, the messages are evenly distributed on channels and the total number of messages is conserved during the simulation.

All of the simulations in our experiments were run until the simulated times of all LPs reached 5000. Since the mean service time is 10 for both timestamp increment distributions, each LP will process a large number of messages before the simulation stops. The experiments were replicated with many different random number seeds and the variation in the performance results were observed to be negligibly small.

The values of most of the parameter settings were chosen to be similar to those used by other researchers in evaluating parallel simulation algorithms [4, 13, 15]. Figure 6 presents the results of the experiments on the toroid topology. In the figure, we compare our DAI scheme with the conventional scheme which examines only parent LPs' simulated times. For each scheme, there are two lines representing the speedups with different timestamp increment distributions. The keys DAI and CNV represent the performance of our DAI scheme and the conventional scheme, respectively.

Our scheme does not deadlock and, therefore, does not spend time on deadlock detection/resolution. On the other hand, the conventional scheme deadlocks approximately 3000 and 4000 times using the uniform and exponential distributions, respectively, during the course of a simulation run. Both schemes perform better with the uniform distribution than with the exponential distribution.

Figure 6 shows that our scheme outperforms the conventional scheme and achieves acceptable speedups. The experimental results obtained with the cuboid and hypercube topology also indicate that the speedups obtained with the DAI scheme are much higher than the conventional scheme. However, the DAI scheme is found to be more sensitive to the branching factor because it aggressively searches through empty channels. With the same message population ratio, a higher branching factor increases the search space and reduces the performance of our scheme. For instance, the speedups obtained using the hypercube topology (with a branching factor of eight) were a factor of two lower than those using the toroid topology. In contrast, the conventional scheme is not sensitive to the branching factor.

The message population ratio is a critical parameter to the DAI scheme. Figure 7 shows the speedup as the message population ratio (MPR) is varied. As the MPR is increased from 0.5 to 1.5, the performance of the DAI scheme increases rapidly. Adding more messages not only increases parallelism but also decreases the number of empty channels. Since the DAI scheme aggressively searches through empty channels, a higher message population ratio reduces the search space and lowers the simulation overhead in DAI. The performance of the DAI

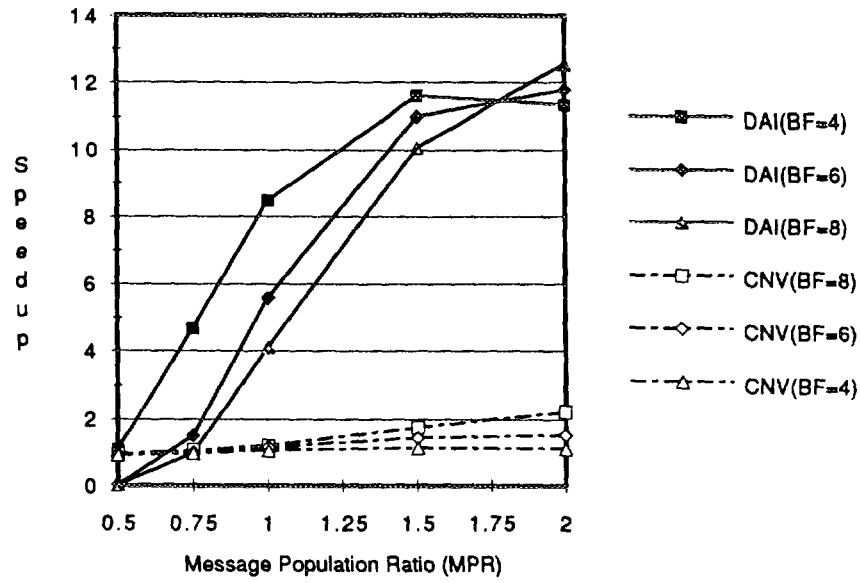

FIG. 7. Speedup as message population ratio is varied. (NHP = 16, DIST = expntl, GRN = 0, LA = 0, NLP = 256.)

scheme drops below that of the conventional scheme for low MPRs. For the entire set of experiments we conducted, this is the only region of the parameter space where the DAI scheme is worse than the conventional scheme.

Previous research has demonstrated good parallel performance when lookahead is used [4, 5]. The speedup of the DAI scheme increases linearly from 6 to 10 as the lookahead is increased from zero to five time units, versus from 1 to 3 for the conventional scheme with BF = 6, NHP = 16, DIST = expntl, GRN = 0, MPR = 1, NLP = 256. This suggests that exploiting global simulation information can improve performance beyond that achievable by only exploiting physical system information such as lookahead. The DAI scheme is complementary to other schemes that exploit information about the physical system, such as lookahead. We also explored the performance effects of varying the computational granularity and the number of LPs [9].

Our experimental results demonstrate some limitations of the DAI scheme. In important discrete event simulation applications, the MPR is much lower than one; e.g., in circuit simulation the MPR is approximately 0.1. The DAI approach does not appear to be well-suited for such applications. Also, at high message populations, deadlock avoidance is usually more efficient than deadlock resolution. In our experimentation, we compared the DAI scheme exclusively to the conventional scheme with deadlock resolution. Our work in conjunction with previous empirical research comparing deadlock resolution to deadlock avoidance appears to indicate that the DAI scheme is better than either of the conventional schemes for systems with high MPRs. However, further experimentation comparing the DAI scheme to the conventional scheme with deadlock avoidance is required before the superiority of the DAI scheme for high-MPR simulation applications can be clearly established.

6. CONCLUSION

This study was motivated by the potential high benefit and relatively low overhead of exploiting global simulation information in a shared memory multiprocessor. Theoretically, the DAI scheme removes non-essential blocking to resolve any local deadlocks. Careful implementation is crucial to the success of the DAI scheme. We showed that extra locking for accessing simulation information is not necessary and can be avoided. Furthermore, we developed efficient search pruning techniques for computing message acceptance horizons in a potentially exponential search space. Experimental results on a Sequent Symmetry S81 demonstrated that global simulation information can be efficiently exploited in a shared memory environment to increase the parallelism and speedups in simulating some important queueing networks.

REFERENCES

1. Ayani, R. A parallel simulation scheme based on distance between objects. *Proc. SCS Multiconference on Dist. Simulation,* 1989, pp. 113–118.
2. Bagrodia, R., Chandy, K. M., and Liao, W. An experimental study on the performance of the space-time simulation algorithm. In *Proc. Work. Par. & Dist. Sim.* 1992, pp. 159–168.

3. Chandy, K. M., and Misra, J. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Trans. Software Eng.* **5,** 5 (Sept. 1979) 440–452.
4. Fujimoto, R. M. Performance measurements of distributed simulation strategies. In *Proc. SCS Multiconference on Dist. Simulation.* 1988, pp. 14–19.
5. Fujimoto, R. M. Parallel discrete event simulation. *Comm. ACM* **33,** 10 (Oct. 1990), 30–53.
6. Goodman, J. R. Cache consistency and sequential consistency. Tech. Rep. 61, SCI Committee, 1989.
7. Jones, D. W. An empirical comparison of priority-queue and event-set implementations. *Comm. ACM* **92,** 4 (Apr. 1986), pp. 300–311.
8. Konas, P., and Yew, P. Synchronous parallel discrete event simulation on shared-memory multiprocessor. In *Proc. Work. Parallel & Distrib. Sim.* 1992, pp. 12–21.
9. Lin, J. M. Efficient parallel simulation for designing multiprocessor systems. Ph.D. thesis, University of Michigan, Ann Arbor, 1992.
10. Lin, J. M. Exploiting dynamic topological information to speed up concurrent multicomputer simulation. In *Proc. Work. Parallel & Distrib. Sim.* 1992, pp. 201–202.
11. Lin, J. M., and Abraham, S. G. Discrete event simulation on shared memory multiprocessors using global simulation information. In *International Conference Parallel Processing,* 1992, Vol. III, pp. 254–261.
12. Nicol, D. M. Parallel discrete-event simulation of FCFS stochastic queueing networks. In *ACM SIGPLAN Symp. Prin. Practice of Parallel Prog.* 1988, pp. 124–137.
13. Prakash, A., and Subramanian, R. An efficient optimistic distributed simulation scheme based on conditional knowledge. In *Proc. Work. Parallel & Distributed Sim.* 1992, pp. 85–94.
14. Reed, D. A., Malony, A. D., and McCredie, B. D. Parallel discrete event simulation using shared memory. *IEEE Trans. Software Eng.* **14,** 4 (Apr. 1988), 541–553.
15. Wagner, D. B., Lazowska, E. D., and Bershad, B. N., Techniques for efficient shared-memory parallel simulation. In *Proc. SCS Multiconf. on Dist. Simulation* 1989, pp. 29–37.
16. Winston, P. H. *Artificial Intelligence.* Addison-Wesley, Reading, MA, 1984.

Received June 15, 1992; revised February 2, 1993; accepted April 22, 1993

JIAJEN M. LIN received the B.S. and M.S. degrees in electrical engineering from National Taiwan University, Taipei, in 1983 and 1985, respectively, and the Ph.D. degree in computer science and engineering from the University of Michigan, Ann Arbor, in 1992. He is currently a senior design engineer at Intel Corporation. His research interests are computer architecture, computer-aided design, and parallel processing.

SANTOSH G. ABRAHAM is currently an Assistant Professor in the Department of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor. From 1984–1987, he was a research assistant in the Center for Supercomputing R&D at the University of Illinois. His research interests are in the areas of multiprocessor and memory hierarchy simulation, compilation for parallel systems, and computer architecture. Santosh Abraham received the B.Tech. degree from the Indian Institute of Technology, Bombay in 1982, the M.S. degree from the State University of New York at Stony Brook in 1983, and the Ph.D. degree from the University of Illinois, Urbana, in 1988—all in Electrical Engineering.