Breadth First Rollback in Parallel Discrete Event Simulations

Ewa Deelman
and
Boleslaw K. Szymanski
Computer Science Department, Rensselaer Polytechnic Institute

The efficiency of Parallel Discrete Event Simulations that use the optimistic protocol is strongly dependent on the overhead incurred by rollbacks. This paper introduces a novel approach to rollback processing which limits the number of events rolled back as a result of a straggler or antimessage. The method, called Breadth-First Rollback (BFR), is suitable for spatially explicit problems where the space is discretized and distributed among processes and simulation objects move freely in the space. The BFR uses incremental state saving, allowing the recovery of causal relationships between events during rollback. These relationships are then used to determine which events need to be rolled back. This paper presents an application of BFR to the simulation of Lyme disease. The results demonstrate an almost linear speedup—a dramatic improvement over the traditional approach to rollback processing.

1. INTRODUCTION

Discrete Event Simulation (DES) is an attractive simulation technique which has been used in modeling communication systems, electronic circuits, battlefield scenarios etc... As the size and the complexity of the simulations increases, so does the demand for use of more powerful computing systems such as parallel machines. Moving to a parallel or distributed platform increases the complexity of the simulation engine. In sequential DES, the physical system is represented by a single Logical Process (LP). Parallel Discrete Event Simulation (PDES) [Fujimoto 1990] brings with it multiple processes. The physical system is viewed as a set of Physical Process (PP). In PDES, each PP is modeled by an LP. The LPs work together to simulate the underlying physical system by sending event messages between each other. The computational challenge is to keep the simulation consistent among the

The Breadth First Rollback was previously published in PADS'97.

Address: Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180-3590

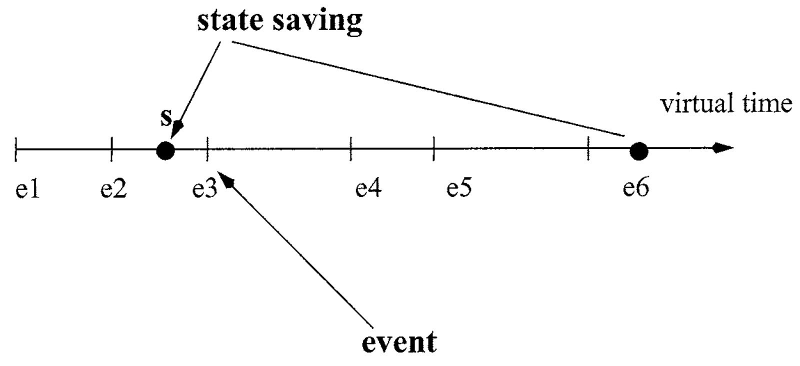


Fig. 1. Non-Unit Checkpointing Interval

processes.

Each Logical Process is composed of state variables, a priority event queue and a clock, which keeps track of the local virtual time (lvt) of the process. The LPs process events asynchronously according to the timestamps of the events. The clock and the state variables are updated during the processing of an event. LPs can communicate with each other by sending event messages. When an LP receives an event message with a timestamp lower than its lvt, a causality error occurs, because it is possible that the incoming event will change the state of the LP in such a way, that the state of the LP at the current lvt will no longer be correct. Two major protocols have been developed to deal with causality errors.

The conservative protocol proposed by Chandy and Misra [Chandy and Misra 1979] prevents causality errors from occurring by allowing the LPs to process events only when it is absolutely certain that there is no event in the system with a timestamp smaller than the timestamp of the event about to be processed. The conservative approach derives its performance from *lookahead*, the ability to determine the minimum delay between dependent events. The lookahead is application dependent and can sometimes be hard to determine. It can also vary throughout the simulation.

The optimistic protocol, on the other hand, does not constrain the LPs' ability to process events. However, when causality errors occur, the protocol enforces recovery from them. Time Warp (TW), introduced by Jefferson, [Jefferson 1985] is the best know protocol of this type. The processing of events is done optimistically, hoping, that no causality errors will occur. When an event with a timestamp smaller than the current lvt is received (such a message is known also as a *straggler*), the LP is rolled back to the last state saved just before the timestamp of the straggler. The event is then processed and the computation is restarted. If messages where sent out between the timestamp of a straggler and the lvt at which that message was received, *antimessages* have to be sent to cancel the original messages released. Upon receiving of an antimessage, the LP checks if the corresponding event has been processed; if it has, a rollback occurs, otherwise the event is annihilated.

There a several issues that need to be addressed in Time Warp. A major one is the amount of memory that TW-based simulators consume because of periodic state saving. The basic memory management tool in TW relies on the *Global Virtual*

Time (GVT) calculation. The GVT is the minimum of the local virtual times of the Logical Processes and of all the timestamps of the messages sent but not yet received. Any information saved with a timestamp smaller than the GVT will not be needed for rollback, so it can be discarded. Deleting obsolete information is referred to as *fossil collection.* Often, fossil collection will not be able to reclaim a sufficient amount of memory for a simulation to proceed efficiently.

To reduce the amount of memory used, the state of a simulation does not need to be saved after each event [Lin et al. 1993]. The side effect of the decreased checkpointing frequency is the increased cost of performing a rollback. Figure 1 illustrates that side effect. When a rollback occurs for the time just prior to event e_6, the LP has to restore the state s, since it is the last state saved before the occurrence of e_6. As a result events e_3–e_5 have to be recomputed. Obviously performance might suffer. Setting up the appropriate checkpointing interval involves balancing the time it takes to save the state, the time is takes to *coast forward* (reexecuting correct events after the rollback) and the amount of memory an LP consumes.

It is also possible that an LP runs out off memory even when checkpointing is done infrequently. When a memory stall occurs, memory reclaiming algorithms are used [Preiss and Loucks 1995]. Some techniques involve discarding information saved in the hopes that it will not be needed. When the deleted objects are messages, as in *Message Sendback* [Jefferson 1985], the messages sent back will trigger rollbacks at the original senders, thus reclaiming some memory. *Gafni's Protocol* [Gafni 1988] extends the class of objects that can be reclaimed to output messages and state vectors. When an output message is reclaimed, a corresponding antimessage is sent out. This might cause a rollback at the receiving LP. Gafni's Protocol also allows state vectors to be reclaimed, thus making an LP rollback to the previous state. Similarly, *artificial rollback* [Lin and Preiss 1992] is used to rollback a process when the available memory is exhausted. All of the above techniques rely on the rollback mechanism. By contrast, *Pruneback* [Preiss and Loucks 1995] is a less intrusive method. It simply deletes some saved states without rolling back the LP. The drawback is that when a rollback occurs, additional coasting forward might be needed.

When the state of the LP is large, saving the entire state can be too costly. If additionally the occurrence of an event changes only a small subset of state variables, then the incremental state saving, developed in SPEEDES [Steinman 1993b], can be efficient. In this state saving method, the part of the state that is affected by an event is saved within the event structure. Upon rollback, the affected events are undone and the modified state variables are restored.

Obviously performance is also an issue in TW. The major source of performance degradation is the cost of rollbacks. The time spent on processing events that will later be rolled back is pure overhead. So is the time spent restoring the state (especially in incremental state saving) and coasting forward. Two most general categories of rollback cost reduction can be classified as follows: (i) protocol modification, and (ii) problem partitioning that defines allocation of parts of the physical system to LPs and the associated issue of mapping of LPs to processors. Section 2 will sketch both categories. Section 3 describes the traditional approach to simulating spatially explicit problems. This approach will be used as the basis of comparison for the new Breadth First Rollback (BFR) algorithm described in section 4. This new algorithm is designed to minimize the impact of rollbacks. BFR is applicable to spatially explicit problems where the space is discretized. In such problems, objects are located in the space and are allowed to move freely. Incremental state saving is used to expose dependencies between events. Our current application is described in section 6.1. Sections 7 and 8 will summarize the results and discuss open problems.

2. RELATED WORK

2.1 Protocol Innovations

One way to improve performance of Time Warp is to decrease the cost of rollbacks by lessening the impact that a rollback of one LP has on the LPs communicating with it.

Lazy cancellation [Lin and Lazowska 1991] can be used to reduce the number of antimessages sent. When a rollback occurs, the LP does not automatically send antimessages corresponding to the positive messages that have been sent after the time to which the computation is rolled back; rather it recomputes the states from the rollback time. If the processing of events causes the same messages to be sent, then there is no need to cancel them, and the cost of canceling "good" messages is saved. Also, the spread of the rollback in minimized. If, however, there are changes, antimessages have to be sent. The idea is that if during the recalculation phase after rollback, if the message would be sent anyway, then it is better not to send any antimessages. These antimessage could cause the receiving LP to roll back unnecessarily. Lazy cancellation, of course, involves comparing messages, which is not always easy.

Lazy re-evaluation [Fujimoto 1990] has been used to determine if a straggler or antimessage had any effect on the state of the simulation. If after processing of the straggler or canceling of an event, the state of the simulation remains the same as before, than there is no need to re-execute any events from the time of the rollback to the current time. The problem with this approach is that it is hard to compare the state vectors to determine if the state has changed.

Both lazy cancellation and lazy reevaluation assume that a rollback at one LP is not likely to affect others. Conversely, the Wolf calls protocol [Madisetti et al. 1988] was designed to minimize the impact that an LP's rollback has on others. It assumes, that a rollback at an LP will most likely trigger rollbacks of other processes. When an LP rolls back, it checks which neighboring LPs are likely to be affected. It then sends a message to these LPs letting them know that a causality error has occurred. If an LP receives such a message, it will stop processing of events. The big disadvantage here is that correct computation might be held back unnecessarily.

Limiting the optimism in a simulation can also reduce the chance of an expensive rollback. Window based algorithms [Reiher et al. 1989; Dickens et al. 1994] set a limit as to how far ahead into the future LPs can proceed without synchronizing. This prevents LPs to get far ahead of others and incur frequent rollbacks. Unfortunately, window based algorithms do not distinguish between useful and erroneous work.

Breathing Time Warp [Steinman 1993a] is composed of two phases: time warp and *breathing time buckets.* In the first phase, the simulation progresses as in Time Warp until the predetermined number of events is reached. Then, the simulation switches to the next phase—breathing time buckets. In this phase events are processed but their messages are not released. The *event horizon* determines the length of the second phase. The event horizon is the timestamp of the earliest new event generated in the current phase. After both phases are completed, the messages are released and the GVT is calculated.

2.2 Problem Decomposition

The Local Time Warp (LTW) [Rajaei et al. 1993] approach combines two simulation protocols by using the optimistic simulation between LPs belonging to the same cluster and by maintaining a conservative protocol between clusters. LTW minimizes the impact of any rollback to the LPs in a given cluster.

Clustered Time Warp (CTW) [Avril and Tropper 1995; Avril and Tropper 1996] takes the opposite view. It uses conservative synchronization within the clusters and an optimistic protocol between them. The reason behind this method is that, LPs in a cluster share the same memory space, so their tight synchronization can be performed efficiently. Two algorithms for rollback are presented: clustered and local. In the first case, when a rollback reaches a cluster, all the LPs in that cluster are rolled back. This way the memory usage is good because events that are present in input queues, and that were scheduled after the time of the rollback, can be removed. In the local algorithm, only the affected LPs are rolled back. Restricting the rollback speeds up the computation, but increases the size of memory needed, because entire input queues have to be kept.

The Multi-Cluster Simulator [Schlagenhaft et al. 1995], in which digital circuits are modeled, takes a different look at clustering. First, the cluster is not composed of a set of LPs; rather, it consists of one LP composed of a set of logical gates. These LPs (clusters) are then assigned to a simulation process. This approach is taken because assigning an LP to a single logical gate would result in high scheduling overhead.

3. SPATIALLY EXPLICIT MODEL AND THE TRADITIONAL APPROACH

The research presented in this paper concentrates on a class of problems known as spatially explicit, problems that have a structure which naturally contains spatial information. In general, the application consists of a multidimensional space, which is discretized into a multidimensional lattice. Each lattice node can contain information which characterizes a given area. There are also mobile objects inhabiting the space. There are two general classes of events in the system: *local* and *non-local.* A local event affects only the state of one lattice node and the objects present in it. A non-local event, for example the Move Event, which moves an object from one location to the next, affects at least two nodes of the lattice.

The simulation currently runs on an IBM SP2, a distributed memory machine. We show results for up to 16 processors. The model was designed in an object oriented fashion and implemented in C++. The communications between processes use the MPI [Gropp et al. 1994] message passing library.

3.1 Partitioning

The space in our application is discretized into a two-dimensional lattice. Similar discretization is used, for example, in personal communication services [Carothers et al. 1995], where the two-dimensional space is discretized into hexagonal or square cells. In these simulations, each cell is modeled by an LP. In some simulations, it would be prohibitively expensive to assign one LP to each spatial entity, so then the entities are "clustered" into a single LP, similarly as in circuit simulations.

In case of spatially explicit problems, the issue of partitioning the space between LPs is of importance. There are two possibilities: either one LP is assigned to each lattice node (which results in high simulation overhead), or the lattice nodes are clustered together and the resulting clusters be assigned to LPs. Our first implementation used the latter approach and assigned spatially close nodes to a single LP, with TW used between the LPs. This would be similar to the CTW, except that our implementation did not have multiple LPs within a cluster, making the simulation of space more efficient. To achieve better performance, the space can also be divided into more LPs than there are available processors [Deelman and Szymanski 1996].

3.2 Protocol and Memory Management

The Time Warp protocol has been chosen for the simulation engine. The LPs in this simulation are called *Space Managers*, because they are responsible for all the events that happen in a given region of space. If the *Space Manager* determines that an object moves out of local space to another partition, the object and all its future events are sent to the appropriate *Space Manager.*

Because the state information is large, incremental state saving of information necessary for rollback was used. When an event is processed, the state information that it changes is placed into the event's local data structure. The event is then placed on a *processed event list.* Events that move an object from one LP to another are also placed in a *message list* (only pointers to the events are actually placed on the lists; the resulting duplication is not costly and speeds up sending of antimessages). If an object moves to another LP, the sending LP saves the object and the events that it is sending in a *ghost list* to be able to restore this information upon rollback.

When a rollback occurs, messages on the *message list* are removed and corresponding antimessages are sent out (using aggressive cancellation). Then the events from the *processed event list* are removed and undone. Undoing an event which involved sending an object to another process entails restoring the objects from the *ghost list* and restoring the future events of the object to the future event queue. For local events, the parts of the state that have been changed by the events have to be restored. During fossil collection, the obsolete information is removed and discarded from the three lists: the *processed event list*, the *message list* and the *ghost list.* We have chosen aggressive cancellation to stop the erroneous simulation on the communicating LPs as soon as possible.

Unfortunately, the traditional approach did not perform as well as expected, especially for large problem sizes, because when a rollback occurred in a cluster, the entire cluster had to roll back.

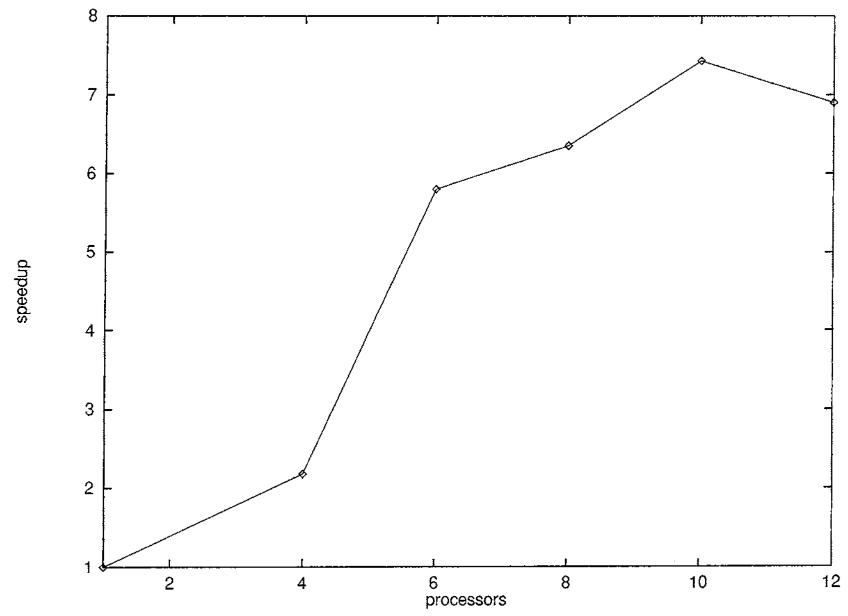


Fig. 2. Speedup For Small Data Set

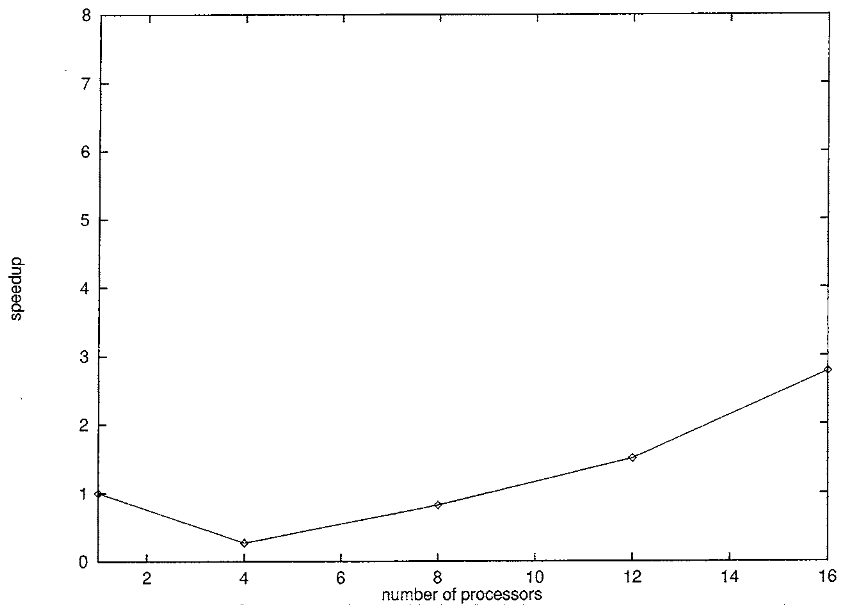


Fig. 3. Speedup For Large Data Set

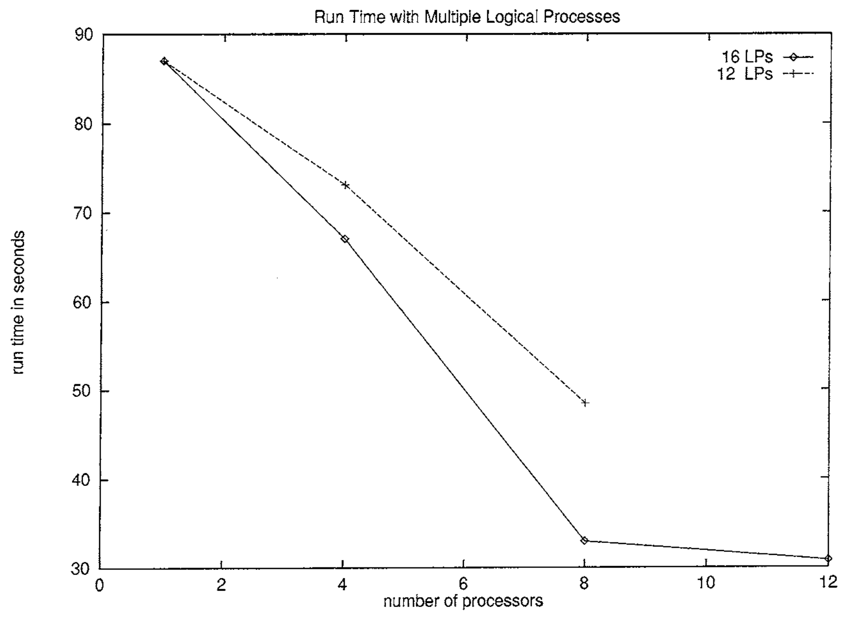


Fig. 4. Running Time for Large Data Set and Multiple LPs per Processor

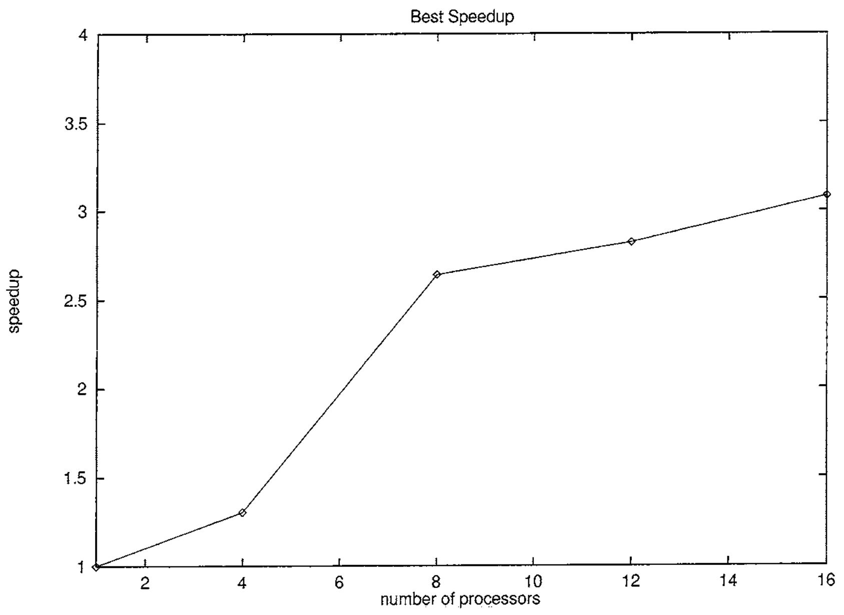


Fig. 5. Speedup with Large Data sets and 16LPs

Initial results obtained for a small-size simulation were encouraging (Figure 2); however, the speedup was not impressive for larger simulations (Figure 3). The performance degradation is caused by the large space allocated to individual processes for the increased problem sizes. When a rollback occurs, the entire space allocated to an LP is rolled back. To minimize the impact of the rollback, the space was divided into more LPs, while keeping the same number of processors. Figure 4 shows the runtime improvement achieved with this approach. For the given problem size, the optimal number of LPs was 16 (Figure 5), and the best efficiency was achieved with 8 processors.

4. BREADTH FIRST ROLLBACK

To improve performance, the nodes of the lattice belonging to an LP (cluster) are allowed to progress independently in simulation time; however, all the nodes in a cluster are under the supervision of one LP. When a rollback occurs in a LP/cluster, only the affected lattice nodes are rolled back, thanks to a breadth-first rollback strategy. This results in an inter-cluster and intra-cluster Time Warp.

The main innovation in BFR is the different forward and rollback processing. All future information is global to the lattice cluster, but information about the past is distributed among the nodes of the spatial lattice. The future information is centralized to handle the proper and fast scheduling of events, and the past information is distributed to limit the effects of a rollback. From the point of view of the future, a partition is treated as a single LP, whereas, from the point of view of the past, the partition is seen as a set of LPs (one LP per lattice node). The performance of the new method results in a close-to-linear speedup.

Breadth-First Rollback is designed for spatially explicit, optimistic PDES. The space is discretized and divided among LPs, giving each LP the responsibility for a set of interconnected lattice nodes. Making the simulation as fast as possible requires optimizing two processes: speeding up the forward simulation process and reducing the impact of a rollback on an LP. The forward computation will progress fastest when there are few LPs per processor, because context switching will be minimized. To reduce the impact of a rollback, its depth and breadth must be kept to a minimum. The rollback should not reach further into the past than necessary, and the number of events affected at a given time has to be minimized. To achieve the latter, a property of spatially explicit problems is useful: if two events happened to nodes so far apart that one event cannot affect the other (given the current logical virtual time (*lvt*) of the LP and the time of the rollback), then at most one of these events needs to be rolled back when a causality error occurs.

Local events, as defined in section 3 are easy to roll back. Assume that a local event e at location l and time t triggers an event e_1 at time t_1 and location l (by definition of a local event). To process a rollback which impacts event e, only the state of location l has to be restored to time just prior to time t. While doing that, e_1 will be undone as part of restoring the state of l. If, however, the triggering event e is non-local and triggers an event e_1 at location $l_1 \neq l$, then restoring the state of l is not sufficient—it is also necessary to restore the state of l_1 just prior to the occurrence of event e_1. Regardless of whether an event is local or non-local, the state information can be restored on a node-by-node basis.

To show the impact of a rollback on an LP, consider a straggler or an antimessage

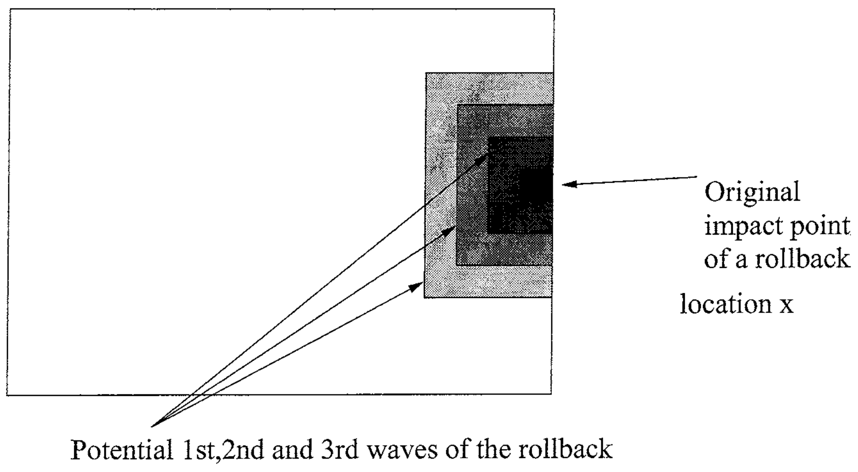

Original impact point of a rollback
location x
Potential 1st,2nd and 3rd waves of the rollback

Fig. 6. Waves of Rollback

arriving at a location (x), marked in the darkest shade in Figure 6. The rollback will proceed as follows. The events at x will be rolled back to time t_r, the time of the straggler or antimessage. Because incremental state saving is used, events have to be undone in decreasing timestamp order to enable the recovery of state information. Since the rollback involves undoing events that happened at x, each event processed at that node will be examined to determine if that event caused another event to occur at a different location. If an event e at location x is being examined, and it is determined that it triggered an event e_1 at location $x_1 \neq x$, then location x_1 has to be rolled back to the time prior to the occurrence of e_1. Only then is e undone (this breath-first order of processing events motivated the name to the new method).

Objects can move only from one lattice node to a neighboring one, so that a rollback can spread from one site only to its neighbors. The time of the rollback at the new site must be strictly greater than the one at site x, assuming a non-zero delay between causally-dependent events. This assumption is justified in spatially explicit problems because of a finite speed of objects. The breadth of the rollback is limited by the speed with which simulated objects move in space.

Figure 6 shows potential waves of rollback, from the initial point of impact through three more layers of rollback. Often, the size of the affected area will usually be smaller than the shaded area in Figure 6, because events at one site will most likely not affect all their neighboring nodes. Obviously, if an event at location x triggered events on a neighboring LP, antimessages have to be sent.

It is interesting to note that each location belonging to a given LP can be at a different logical time. In fact, events in a given LP may not be processed in an increasing-timestamp order. If two events are independent, an event with a higher timestamp can be processed ahead of an event with a lower timestamp. A similar type of processing was mentioned briefly in [Steinman 1992] as CO-OP (Conservative-Optimistic) processing. The justification is that the requirement of processing events in timestamp order is not necessary for provably correct simulations. It is only required that the events for each simulation object be processed in a correct time order.

Consequently, when an event is processed, the local time of the lattice node where the event is scheduled to occur is checked. If that time is greater than the time of the event, the node has to roll back.

5. CHALLENGES OF THE NEW APPROACH

To implement BFR, some changes had to be introduced the model. A major modification was made to the Move Event in response to the following question: If an object is moving from location (x, y) to location (x_1, y_1), where should the event be placed as "processed"? If it is placed in the processed event list at location (x, y), and location (x_1, y_1) is rolled back, then there would be no way of determining that the event affected location (x_1, y_1). If it is placed at location (x_1, y_1), and location (x, y) is rolled back, a similar difficulty arises. Placing the Move Event in both processed lists is also not a good solution, because, in one case, the object is moving out of the location, and, in the other case, it is moving into the location. This dilemma motivated the of split of the event into two: the MoveOut and MoveIn Events. Hence, when an object moves from location (x, y) to location (x_1, y_1), the MoveOut is placed in the processed event list at (x, y), the MoveIn at location (x_1, y_1). When the move is non-local, (location (x_1, y_1) belongs to another LP), the MoveOut is placed in the *processed event list* at location (x, y). The MoveIn event is then sent to the appropriate process. When this event is received and processed, it will be placed at location (x_1, y_1). Upon rollback, if a MoveOut to another LP is encountered, an antimessage is sent together with a location (x, y) to which the original message was addressed, to avoid searching the lattice nodes for this information.

Since the MoveOut Event indicates when a message has been sent, no *message list* is necessary. Another affected structure is the *ghost list.* In the original approach, objects and their events were placed on the list in the order in which they left the partition. In BFR, the time order is not preserved. Objects are placed on the list in any timestamp order because the nodes of the lattice can be at different times. The non-ordered aspect of the *ghost list* poses problems during *fossil collection.* The list cannot be merely truncated to remove obsolete objects. The solution, again, is to distribute that list among the nodes. This is useful for load balancing, as described in the final section. However, the *ghost list* is relatively small (compared to the *processed event list*), so it might not be necessary to distribute the list if no load balancing is performed. It is only necessary to maintain an order in the list based on the virtual time at which the object is removed from the simulation.

Additionally, the trigger information about the event must be preserved. In the original implementation, when an event was created, the id of the event that caused it was saved in one of the tags (the trigger) of the new event. When an event was undone, the dependent future events were removed by their trigger tags from the future event queue. In BFR, it is possible that the future event is already processed, and its assigned location has not been rolled back yet. It is prohibitively expensive to traverse the future event list and then each *processed event list* in the neighborhood in search of the events whose triggers match the given event tag. The solution is to create dependency pointers from the trigger event to the newly created event. This way, a dependent event is easily accessed, and the location where it resides can be rolled back. Pointer tacking has been previously implemented for shared memory machines [Fujimoto 1990] to decide whether an event should be canceled or not. In addition, it is also necessary to determine if a dependent event has been processed or not, to be able to quickly locate it either in the future event queue or in a *processed event list.*

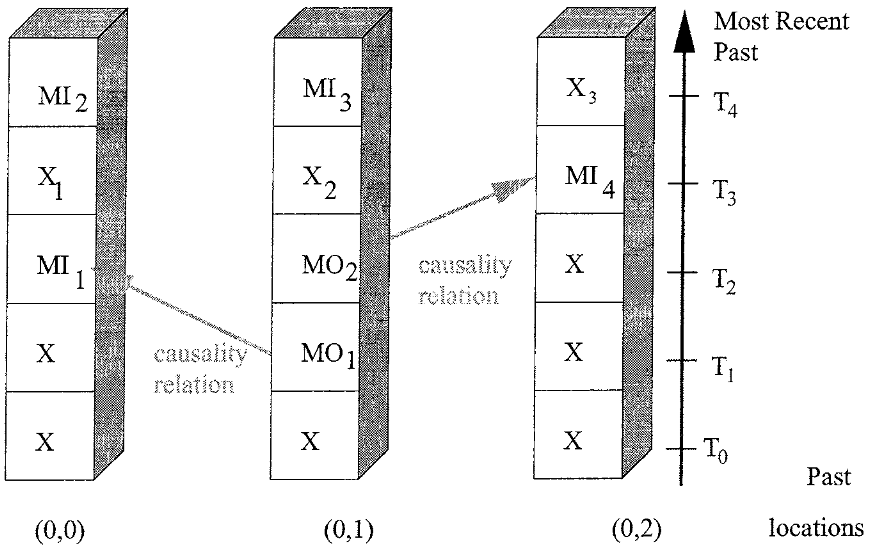
One more change was required for the random number generation. In the original simulation, a single random number stream was used for an LP. Now, since the sequence of events executed on a single LP can differ from run to run (because the sequence of executed events is not necessarily the same), the same random number sequence can give two different results! Obviously, non-repeatability cannot be accepted. The solution that we found was to distribute the random number sequence among the nodes of the lattice. Initially, a single random number sequence is used to seed the sequences at each node. From there, each node generates a new sequence.

With BFR, time is not uniform across the space, so the global virtual time (GVT) calculation cannot be invoked based upon the distance of the local virtual time (lvt) from the previous GVT (as in the original version). Instead, it is invoked after a certain number of messages has been received from other processes since the previous GVT.

6. EXAMPLES

The following code constitutes the skeleton of the Breath-First Rollback (unoptimized, for clarity).

```
rollback_space(t,x,y)  // rollback location x,y to time t {
   For every event E processed at x,y after and including time t {
      // the events are undone in the order opposite
      // to the one in which they were processed
      // update the local virtual time (lvt) to the time of the event
      if ( E.eventTime < lvt ) {
         lvt = E.eventTime;
      }
      // make sure to undo the dependent events first
      while there exists an unprocessed dependent event D of E  {
         // (E triggered D)
         if (D is at location (x1,y1) != (x,y) ) {
            if ( time of  (x1,y1) >= D.eventTime )  {
               rollback_space( D.eventTime, x1, y1  );
            }
         }
      }
      if (E.event_type == MOVE_OUT_EVENT)  {
         if the new location is outside_bounds of the current LP  {
            send out an anti-message for event E
            Obj = object affected by E
            // restore events that were scheduled for Obj when
```



X could be any event
MI- MoveIn event
MO- MoveOut

Fig. 7. View of Processed Lists at Three Nodes of the Lattice.

```
          // message was sent
          restore_events_from_ghost_list(Obj, t);
        }
      }
      undo_event(E);
      insert_event(E);      // into the future event queue
      // remove events that E triggered (from the event queue)
      remove_scheduled_events(E.id);
  }
}
```

To demonstrate the behavior of the BFR algorithm, let's consider the example in Figure 7, which shows processed event lists at three different lattice nodes: (0,0),(0,1), and (0,2). The event MO is a MoveOut event, MI a MoveIn event, and X can be any local event.

If there is a rollback for location (0,1) at time T_0, the following will happen: First, MI_3 is undone and placed on the future event queue. Then, X_2 undergoes similar processing. Next, MO_2 is being considered and the dependence between it and MI_4 is detected, so a rollback for location (0,2) and time T_2 is performed. As a result, X_3 and MI_4 are undone. Both are placed on the future event queue. Next MO_2 is undone, which causes MI_4 to be removed from the future event queue. MO_1 is examined, and (0,0) is rolled back to time T_1. MI_2, X_1 and MI_1 are

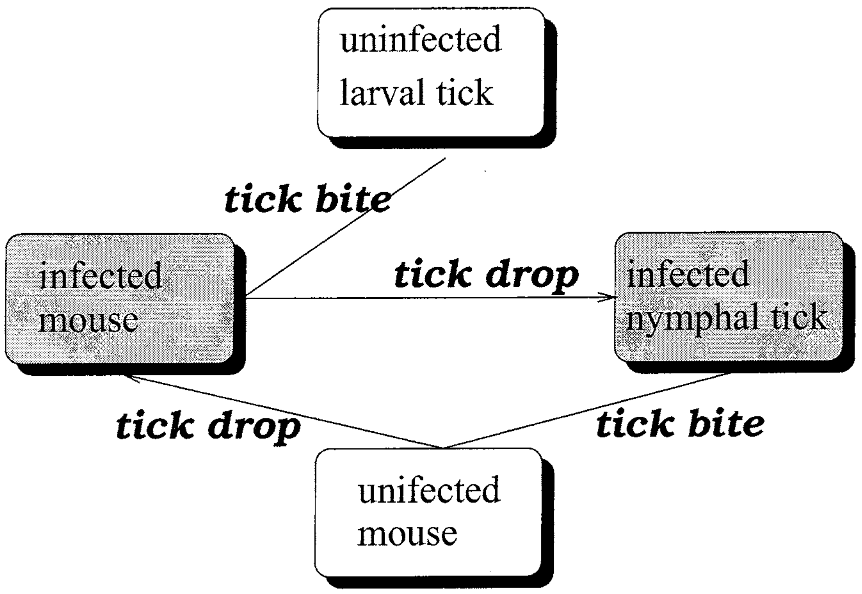


Fig. 8. The Cycle of Lyme Disease

undone and placed on the future event queue. MO_1 is undone and MI_1 is removed from the future event queue. This process can be optimized by not placing events that are soon to be removed, in the queue. If an event was scheduled during the time affected by the rollback, it does not need to be placed back on the future event queue.

If the rollback occurred at location (0,0) for time T_1, then the three most recent events at location (0,0) will be undone and placed on the future event queue, and no other location will be affected during the rollback. It is possible that the other locations will be affected when the simulation progresses forward. If, for example, an event MO_z was scheduled for time T_2 on (0,0) and triggered an event MI_z on (0,1) for time T_3, then location (0,1) would have to roll back to time T_3.

Interesting aside: location (x,y) can be at simulation time 10. The next event in the future list is scheduled for time 20 and location (x_1,y_1), and processed. If an event comes in from another process for time 15, a rollback may not occur. If the event is to happen at location (x,y), then no rollback will happen. If, however, it is destined for location (x_1,y_1), localized rollback will occur. Clearly, comparing the timestamp of an incoming event to the lvt is not enough to determine if a rollback is necessary.

6.1 The Application

The application that motivated this work is the simulation of the spread of Lyme disease in nature [White et al. 1991]. The space, in this case, is two-dimensional, and it is discretized into a two-dimensional lattice. There are two types of objects in the simulation: the mobile objects that are moving freely in space (mice) and the stationary objects located at the lattice nodes (ticks). The two main groups of events are: (i) local to a node (such as tick bites, death, etc..) and (ii) non-local (such as a move from one node to another–*Move Event*). Lyme disease is prevalent in the Northeastern United States [Barbour and Fish 1993; Miller et al. 1990]. People can acquire the disease by coming in contact with an infected tick. If a tick is infected with the spirochete, the spirochete may be transferred into the host's blood stream, causing an infection. Since the ticks are practically immobile, the spread of the disease is driven by their mobile hosts, such as mice and deer. Even though the most commonly known cases of Lyme disease are reported in humans, the main infection cycle consists of ticks and mice (Fig. 8). If an infected tick bites a mouse, the animal becomes infected. The disease can also be transmitted from an infected mouse to an uninfected, feeding tick.

Larvae hatch uninfected and quest for a blood meal. If they find a mouse, they feed on it and molt into nymphs. The nymphs overwinter and in the spring begin to look for a blood meal. If successful, they will feed, drop off the animal, and molt into adult ticks. The latter deposit eggs the following spring. The duration of the simulation is 180 days, starting in the late spring [Deelman et al. 1996]. This time is the most active for the ticks and mice. Mice, during that time, are looking for nesting sites and may carry ticks a considerable distance [Ostfeld et al. 1996].

The mice are modeled as individuals, and ticks, because of their sheer number (as many as 1200 larvae/400m^2 [Ostfeld et al. 1996]) are treated as "background". The space is discretized into nodes of size 20x20m^2, which represents the size of the home range for a mouse. Each node may contain any number of ticks in various stages of development and with various infection status. Mice can move around in space in search of empty nesting sites. The initiation of such a search is represented by the *Disperse Event*, and the moves are modeled by the *Move Event.* Mice can die (*Kill Event*) if they cannot find a nesting site or by other natural causes, such as old age, attacks by predators, and disease. Mice can be bitten by ticks (*Tick Bite*) or have ticks drop off (*Tick Drop*). From the above list of events, only the *Move Event* is non-local.

7. RESULTS

Figure 9 shows the performance of BFR and illustrates the almost linear speedup. The BFR is considerably faster than the traditional approach thanks to its properties. The most important among them is localization of rollbacks. When a rollback occurs, only the absolutely necessary events are undone. In the traditional approach, the number of events that needed to be rolled back was ultimately proportional to the number of lattice nodes assigned to a given LP. When a rollback occurred, all the events that happened in that space had to be undone. On the other hand, when a rollback occurs in the BFR version, the number of events being affected by a rollback is proportional to the length of the edges of the space that interface with other LPs. When space is divided into strips, the number of events affected by a given rollback is proportional to the length of the two communicating edges. Consequently, we observe that the number of events rolled back using BFR is an order of magnitude smaller than that in the traditional approach.

There are also fewer antimessages being sent as a result of the hybrid form of

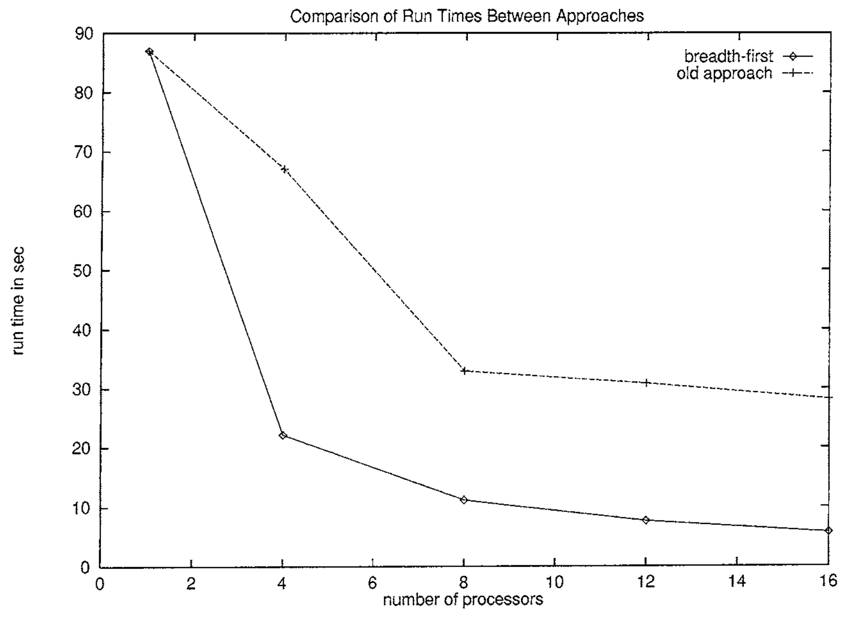


Fig. 9. Results: Comparison of Runs With BFR and the Traditional Approach

lazy cancellation. In general, having one LP per processor eliminates on-processor communication delays. There are, of course, some drawbacks to the new method. Fossil collection is much more expensive (because lists containing past information are distributed); therefore, it is done only when the GVT has increased by a certain amount from the last fossil collection. It is harder to maintain dependency pointers than triggers, because, when an event is undone, its pointers have to be reset. The pointers have to be maintained when events are created, deleted, and undone, whereas triggers are set only once. There is no aggressive cancellation, but, as it can be seen from the results, that does not seem to have an adverse impact on performance.

8. CONCLUSIONS AND FUTURE WORK

We have described a new algorithm for rollback processing in spatially explicit problems. The algorithm is based on the optimistic protocol and relies on the space being partitioned into a multi-dimensional lattice. Rollbacks are minimized by examining the processed event list of each lattice node during rollback, in search of causal dependencies between events which span the lattice nodes. The rollback impacts the minimum number of sites, making the simulation very efficient. The number of events rolled back in BFR is an order of magnitude less than the number of events rolled back using the traditional approach. As a result, an almost linear speedup is achieved. Obviously this performance is attainable thanks to a large degree of parallelism existing in the application.

The results presented here were obtained with the even load distribution. However, if the simulation's load per LP is uneven (for example, when the odd LPs have more load then the even ones), the performance degrades. We believe, that BFR will lend itself well to load balancing, since the local (at the node level) history tracking facilitates load balancing. An overloaded LP can "shed" layers of space to balance the load. The objects and the events scheduled for them are sent along with the space. Nothing special needs to happen on the receiving side. On the sending side, however, the priority queue has to be filtered in order to extract the future events for the area sent to the new process.

9. ACKNOWLEDGMENTS

This work was supported by the National Science Foundation under Grants BIR-9320264 and CCR-9527151. The content of this paper does not necessarily reflect the position or policy of the U.S. Government—no official endorsement should be inferred or implied.

REFERENCES

Avril, H. and Tropper, C. 1995. Clustered time warp and logic simulation. *Workshop on Parallel and Distributed Simulation*, 112–119.

Avril, H. and Tropper, C. 1996. The Dynamic Load Balancing of Clustered Time Warp for Logic Simulations. *Workshop on Parallel and Distributed Simulation*, 20–27.

Barbour, A. and Fish, D. 1993. The biological and social phenomenon of Lyme disease. *Science 260*, 1610–1616.

Carothers, C., Fujimoto, R., and Lin, Y. 1995. A Case Study in Simulating PCS Networks Using Time Warp. *Workshop on Parallel and Distributed Simulation*, 87–94.

Chandy, K. M. and Misra, J. 1979. Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. *IEEE Transactions on Software Engineering 5*, 440–452.

Deelman, E., Caraco, T., and Szymanski, B. K. 1996. Parallel Discrete Event Simulation of Lyme Disease. *Pacific Biocomputing Conference*, 191–202.

Deelman, E. and Szymanski, B. K. 1996. Simulating Lyme Disease Using Parallel Discrete Event Simulation. *Winter Simulation Conference*, 1191–1198.

Dickens, P., Nicol, D., Reynolds, P., and Duva, J. 1994. Analysis of Optimistic Window-based Synchronization. *ICASE Report No. 94-27*.

Fujimoto, R. M. 1990. Parallel Discrete Event Simulation. *Communications of the ACM 33*, 10, 31–53.

Gafni, A. 1988. Rollback mechanism for optimistic distributed simulation systems. *Distributed Simulation*, 61–67.

GROPP, W., LUSK, E., AND SKJELLUM, A. 1994. *Using MPI.* The MIT Press.

JEFFERSON, D. 1985. Virtual Time. *Trans. Prog. Lang. and Syst. 7*, 404–425.

LIN, Y. AND LAZOWSKA, E. D. 1991. A Study of Time Warp Rollback Mechanisms. *ACM Transactions on Modeling and Computer Simulations*, 51–72.

LIN, Y., PREISS, B., LOUCKS, W., AND LAZOWSKA, E. 1993. Selecting the Checkpoint Interval in Time Warp Simulation. *Workshop on Parallel and Distributed Simulation*, 3–10.

LIN, Y.-B. AND PREISS, B. 1992. Optimal Memory Managment for Time Warp Parallel Simulation. *ACM Trans. on Modeling and Computer Simulation*, 283–307.

MADISETTI, V., WALRAND, J., AND MESSERSHMITT, D. 1988. Wolf: A rollback algorithm for optimistic distributed simulation systems. *Winter Simulation Conference*, 296–305.

MILLER, G., CRAVEN, R., BAILEY, R., AND TSAI, T. 1990. The epidemiology of Lyme disease in the United States 1987-1998. *Laboratory Medicine 21*, 285–289.

OSTFELD, R., HAZLER, K., AND CEPEDA, O. 1996. Temporal and Spatial Dynamics of *Ixodes scapularis* (Acari: Ixodidae) in a rural landscape. *Journal of Medical Entomology 33*, 90–95.

PREISS, B. AND LOUCKS, W. 1995. Memory Managment Techniques for Time Warp on a Distributed Memory Machine. *Workshop on Parallel and Distributed Simulation*, 30–39.

RAJAEI, H., AYANI, R., AND L. THORELLI. 1993. The Local Time Warp Approach to Parallel Simulation. *Workshop on Parallel and Distributed Simulation*, 119–126.

REIHER, P., WIELAND, F., AND JEFFERSON, D. 1989. Limitation of Optimism in Time Warp Operating System. *Winter Simulation Conference*, 765–770.

SCHLAGENHAFT, R., RUHWANDL, M., C.SPORRER, AND BAUER, H. 1995. Dynamic Loaad Balancing of a Multi-Cluster Simulation of a Network of Worstations . *Workshop on Parallel and Distributed Simulation*, 175–180.

STEINMAN, J. S. 1992. SPEEDES: A Unified Approach to Parallel Simulation. *Workshop on Parallel and Distributed Simulation*, 75–84.

STEINMAN, J. S. 1993a. Breathing Time Warp. *Workshop on Parallel and Distributed Simulation*, 109–118.

STEINMAN, J. S. 1993b. Incremental State Saving in SPEEDES using C++. *Winter Simulation Conference*, 687–696.

WHITE, D., CHANG, H., BENACH, J., BOSLER, E., MELDRUM, S., MEANS, R., DEBBIE, J., BIRKHEAD, G., AND MORSE, D. 1991. The geographic spread and temporal increase of the Lyme disease epidemic. *Journal of the American Medical Association 266*, 1230–1236.

Rensselaer

SCIENTIFIC COMPUTATION RESEARCH CENTER

MEMORANDUM

TO: Jacob Fish, Joseph Flaherty, Ken Jansen, Antionette Maniatty, Robert Spilker and Bolek Szymanski

FROM: Mark Shephard [illegible]
SCOREC

DATE: January 22, 1998

RE: Technical Papers

Please send all new and/or missing technical papers to Elaine Phillips, CII 7013 that you would like to have included in the 1997 SCOREC Report Series and associated abstract books.

We mail these abstract books, which list papers and give their abstracts, to many different people. This is a good way to generate additional outside interest in your work. If you have any questions regarding the above, you can contact myself or Elaine Phillips at extension 6795.

Elaine, I'll appreciate it if you could add the following two to tech reports.

1. "... cellular automata" will appear in a journal "Parallel and Distributing Computing Practices"

2. "Breadth First" is submitted to "ACM Trans. on Modeling and Computer Simulation"

[illegible]