Parallel Discrete Event Simulation: A Survey

Voon-Yee Vee Wen-Jing Hsu

Centre for Advanced Information Systems, SAS
Nanyang Technological University
Nanyang Avenue, Singapore 639798
vyvee@singnet.com.sg, hsu@ntu.edu.sg

Abstract

In the past decade, parallel processing has gained very significant advances in all fronts of the theory, systems, and applications. However, despite years of research and its apparent significance, parallel simulation remains a major outstanding challenge. In particular, there has been no simulation system which facilitates an early prediction of the program performance.

In this report, we document a survey of the major existing approaches for parallel simulation as well as a comparative study of two leading computational models, namely, Valiant's BSP and Leiserson's Cilk, which are useful formal models for performance prediction of simulation programs.

1 Introduction

Simulation has been heavily relied upon by computer scientists, physicists, circuit designers, mathematicians, military force, and even video game designers [LK91, Fis95, Chi92]. For decades, simulationists have been devising simulation models for large and complex systems to facilitate performance analysis, study of system behavior, optimization, and essentially all aspects of the systems under simulation.

The speed of sequential processors increases every year, while the complexity of the systems we are interested in simulating also increases every year. Today, some of the systems we wish to simulate are so complex that they appear "intractable" to even the fastest uniprocessor we have built to date. Furthermore, the power of a processor will eventually reach an upper bound imposed by the limits of the physical world (such as the speed of light and the finite size of a molecule). Given this, performing simulation in parallel is acknowledged as the most promising solution to simulating those "intractable" systems.

Unfortunately, despite over two decades of research, the technology of parallel simulation has not significantly impressed the general simulation community [Fuj93b]. Considerable efforts and expertise are still required to develop efficient simulation programs. There are no "golden rules" that a programmer can follow to guarantee an efficient program. Generally speaking, parallel simulation is very hard [Fuj90a].

Nevertheless, the field of parallel simulation does deserve future research and cannot be simply ignored. Presently, because of its importance, there is much active research to further develop the parallel simulation technology.

In this paper we give an overview of the existing parallel simulation technology. We first cover some basic concepts in simulation, followed by various ways of decomposing a simulation model. We survey the major existing approaches for the parallel discrete-event simulation. Two major models of parallel computation are also introduced, followed by the existing approaches based on these models. Finally, we offer our views of the prospects of parallel simulation.

2 An Overview of Simulation

A simulation model can be viewed as a representation of the physical system under simulation. *Discrete Event Simulations* (DESs) are characterized by *discrete-state models* (as opposed to *continuous-state models*) and the *event-driven* approach. In a *continuous*-state model, or *continuous simulation*, the *state* of the simulation changes continuously over time; while in a *discrete*-state model, or *discrete simulation*, the change of state is instantaneous and occurs at discrete points in simulated time [Jai91, Fer95, LK91]. Notice that the term *discrete* applies only to the state changes but not to the values of the simulated time.

The discrete event simulation can further be classified according to how the simulation time advances. In *time-driven* discrete simulation (sometimes also referred to as the *unit time* approach), the simulation time is incremented by a constant *time step* $\triangle$. In the *event-driven* approach, the increment of simulation time is triggered by the next earliest occurring event. It is well-known that the time-driven approach is not a good choice for general-purpose simulation; in particular, the approach is inefficient for events irregularly dispersed over time [Fer95]. It is often difficult to fix a proper $\triangle$: to maintain the accuracy of the simulation generally requires a smaller $\triangle$; to run the simulation efficiently requires a larger value of $\triangle$. The event-driven approach however does not suffer from this difficulty.

In Sections 2.1 and 2.2, we present overviews of performing DES in sequential and in parallel respectively.

2.1 Sequential Simulation

All sequential DESs have a common structure. The set of future events is maintained in a *global event list*. These events are to be scheduled by an *event scheduler* in a nondecreasing timestamp order, one after another. After processing an event (of the earliest occurring time among all outstanding events), the *simulation clock* recording the current (simulated) time is then advanced to the time of the next earliest occurring event. The components and organization of a sequential DES model can be found in a large number of literature, see, e.g., [LK91, Fer78, Jai91].

The simulation community, however, is no longer satisfied with the performance of sequential simulators. The systems we desire to simulate today are so complex that the tasks of executing these simulation models are often beyond the capability of sequential simulators [Fuj93b].

2.2 Parallel Simulation

Seminal work of DES in parallel dates back roughly 20 years. Hundreds of papers have been published since then, the majority appearing within this decade (see References and Bibliography). Various ways of decomposing a simulation for processing on multiple processors have been proposed, as outlined below [RW89, Fer95, Cal96]:

1. *Parallelizing Compilers.* In this approach, parallelizing compilers are used to exploit the parallelism available in a given sequential program. This approach requires no changes in the code for sequential simulation, and thus is readily applicable to many existing sequential simulation programs. However, since the compiler completely ignores the structure of the problem, the parallelism exploited is quite

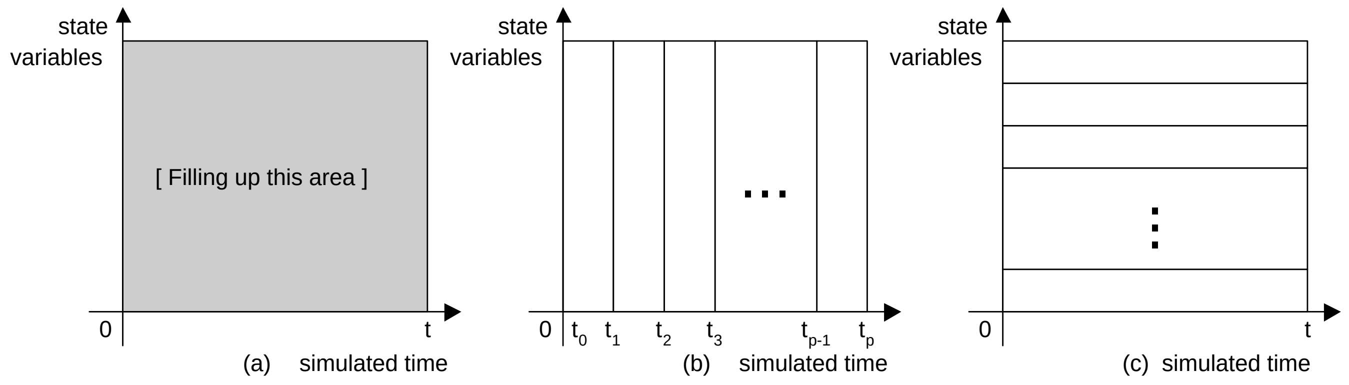

Figure 1: (a) The space-time graph. Simulation execution, which runs from simulated time 0 to time t, is equivalent of filling in the shaded area. Possible decompositions with respect to (b) space domain and (c) time domain.

limited. The program may have to be rewritten to exploit more parallelism of the underlying problem, but this is completely ignored with this approach.

2. *Replicated Trials* (or *Parallel Independent Runs*, *Distributed Experiments*). N sequential simulations are run independently on N processors, and their results are averaged in the end. Since no coordination is required among the trials, high efficiency can be expected. However, the parameters of all simulation runs must be decided before any run takes place. This requirement does not encourage interactive decision making of the parameters, which could have been done if there existed a few fast runs of the simulation. In addition, the memory space required is proportional to the number of independent trials. This can impose a severe restriction for certain large simulation or for certain underlying architectures.

3. *Distributed Functions.* In this approach, the essential subtasks of a simulation are assigned to a number of processors. The subtasks may include random number generation, event set processing, file manipulation, statistics collection, etc. This approach requires minimum changes in the code for sequential simulation. However, since the number of such subtasks is limited, not much parallelism can be exploited. Furthermore, the workload among the processors is difficult to balance as well.

4. *Distributed Events (with Centralized Event List).* In this approach, a processor which becomes available continues to process the event with the earliest timestamp in a global event list. The global event list may be maintained either distributively or by a particular processor. To avoid incorrect simulation, each processor has to ensure that the event with the earliest timestamp in the list will not be cancelled by some events currently processed by other processors. It also has to ensure that processing this event with other events currently being processed by other processors is in consistence with the semantics of the system being simulated. This requires knowledge of the simulation model, which may not be extracted easily. Besides, the global event list can become a bottleneck if many processors are involved in the simulation.

5. *Domain Decomposition.* This class of decomposition is based on the view that a simulation execution is equivalent to filling in a two-dimensional region, with one dimension representing the simulated time and another dimension the state variables, as shown in Figure 1(a). According to this view, the

space-time domain can be decomposed along the time dimension, as illustrated in Figure 1(b), or along the space dimension, as illustrated in Figure 1(c).

(a) *Space-Parallel Decomposition* (or *Distributed Simulation*). The simulation model is decomposed into a number of sub-models or components (in space domain). Each component is assigned a process, where several processes may be run on the same processor. This decomposition is attractive because it is applicable to any model and shows the greatest potential in offering scalable performances for large models.

(b) *Time-Parallel Decomposition* (or *Time Parallelism*). The domain is partitioned into a number of intervals $[t_{i-1}, t_i]$ for $0 < i \leq p$, p being the number of processors. Each processor is assigned an interval and is responsible to compute the values of the state variables within that interval. With this approach, the simulation mechanism must ensure that the state of the system at the end of $[t_{i-1}, t_i]$ must match the state at the beginning of $[t_i, t_{i+1}]$. Recomputations of some intervals are required if any mismatch of states is detected. Therefore, the efficiency of the decomposition depends on whether it is possible to find a good way to accurately predict the initial state of each interval. Nevertheless, the idea is elegant and deserves future research.

Among these classes of approaches, the space-parallel decomposition shows the greatest potential and is considered the most promising approach to perform DES in parallel [RW89, Win92]. Since the event list is also decomposed into individual ones, the event lists would not become the bottleneck as with the fourth approach (distributed events with centralized event list). A higher degree of parallelism is expected since this class of approaches encourages concurrent processing of events with different timestamps. We will focus on this approach in the rest of the report. Discussions of the potentials and the limitations of other approaches can be found in, e.g., [Rey88, Fuj90a], and more recent ones such as [Win92, Cal96, NF94, Fer95].

Early work in decomposition of simulation model along the space dimension referred to the approach as *distributed simulation* [CM79, Bry77], as the simulation was intended to be executed on a network of processors. Later work sometimes uses the term *parallel simulation*, or more popularly, *parallel discrete event simulation* (PDES) or *parallel and distributed simulation* (PADS). We will treat these terms as synonyms henceforth.

In Section 3, we will review the important principles of parallel simulation. The PDES *protocols*, which distinguish between major existing PDES approaches, are elaborated in Section 4. Mathematical performance models of the parallel simulations are introduced in Section 5. Besides simulation protocols, there are other important components which can also greatly influence the performance of the simulation runs. These components are briefly described in Section 6. A comparison and critiques of various approaches are elaborated in Section 7. The prospects of constructing an efficient parallel simulator are outlined before ending this chapter.

3 Principles of Parallel Simulation

In PDES, a model generally comprises N *logical processes* (LPs) $LP_0, LP_1, \ldots, LP_{N-1}$ which interact among themselves by sending timestamped event messages. A *link* exists from LP_i to LP_j if LP_i may send messages to LP_j. Each message is a tuple (E_k, T_k) consisting of an event E_k and its associated timestamp T_k. Each

LP LP_i is associated with a *local clock*, or *local virtual time* (LVT), $Clock_i$ which refers to the simulated time up to which it has progressed. When E_k is processed, $Clock_i$ is automatically advanced to T_k.

It is very important that *causality constraint* be observed in order to ensure the correctness of the simulation.

Let E_1 and E_2 denote two events with timestamps T_1 and T_2 respectively. Assume that $T_1 < T_2$. If E_2 depends on E_1, then there is a causality constraint over the order of execution of E_1 and E_2. For instance, if E_1 causes a change of a state variable which will be referenced by E_2, then E_1 must be executed before E_2 or a *causality error* would occur.

To be concrete, consider a battlefield simulation, in which two tanks A and B are from two opposing sides. Assume that a bomb shell from one tank can reach its enemy within one second. Suppose that event E_1 with timestamp 0900 denotes "tank A fires at tank B (expected hit rate: 95%)", and E_2 with timestamp 0905 denotes "tank B fires at tank A (expected hit rate: 95%)". If tank A does succeed to annihilate tank B, then it makes no sense to process E_2. In other words, whether E_2 should be processed or be discarded is known only after processing E_1, and there is causality constraint from E_1 to E_2.

Violating causality constraint means that the future can affect the past. This can result in anomalous behavior and consequently incorrect simulation. It is the responsibility of the *synchronization mechanism* to ensure proper and correct interactions among the LPs.

It has been shown in [Mis86] that *a parallel simulation obeys the causality constraint if and only if every LP processes events in nondecreasing timestamp order*—a condition formulated as the *local causality constraint* in [Fuj90a]. It is important to note that the local causality constraint is *a sufficient condition but not a necessary condition.* For instance, if two events in the same LP have different timestamps but there is no (direct or indirect) dependency between them, then they can be executed in parallel without causing causality errors.

Parallel simulation approaches broadly fall into two categories—*conservative* and *optimistic*—according to the ways they adhere to the local causality constraint. *Conservative* approaches avoid all possible causality errors by strictly adhering to the local causality constraint. *Optimistic* approaches, on the other hand, attempt to exploit the nonzero probability of producing no causality error by not strictly adhering to the local causality constraint. An optimistic approach guarantees to detect any occurrences of causality errors and provides a *rollback* mechanism to restore the simulation to a correct state. These two approaches are discussed in the following sections.

4 Protocols

4.1 Conservative Protocols

Historically, the first parallel simulation mechanism was based on the conservative approaches. In late 1970s, Chandy, Misra and Bryant [Bry77, CM79, Mis86] developed the algorithms which are often referred to as the *Chandy-Misra-Bryant* (CMB) protocols.

In a conservative approach, the synchronization protocol ensures that a logical process LP_i with LVT $Clock_i$ will not receive a message (E_k, T_k) in which $T_k < Clock_i$. This can be achieved by allowing LP_i to process E_k only if it is impossible for LP_i to later receive an event with a timestamp less than T_k. The events which can be processed safely are called *safe events.*

With this simple mechanism, an LP must block if it does not own any safe event. This may lead to a deadlock situation if the synchronization mechanism is not properly designed.

4.1.1 Deadlock Avoidance

Assume that the sequence of timestamps on the messages sent over a link is nondecreasing. There is a FIFO queue and a clock associated with each link. For a particular link, if the associated queue is empty, then its clock is normally set to the timestamp of the last received message; otherwise, the clock is set to the timestamp of the message in front of the queue.

Consider the following algorithm. An LP repeatedly selects the link with the smallest clock value, and processes the first event in the front of the link's queue. If the queue is empty, then the LP blocks; it resumes when the queue of the link with the smallest clock value becomes nonempty. It can be shown that the protocol adheres to the local causality constraint, and therefore no causality error will ever occur [CM79,Mis86]. This simple algorithm, however, does not prevent the simulation from running into a deadlock. It is possible that some LPs become blocked and each of them wait indefinitely for each other in a cyclic fashion.

In the CMB approach, *null messages* are used to avoid deadlock. In this *deadlock avoidance* algorithm, an LP sends a *null message* $(null, T_{null})$ as a "pledge" that it will not send a message with a timestamp smaller than T_{null}. A null message is only for the synchronization purpose and does not correspond to any event in the physical system. It can be shown that the null message algorithm avoids deadlock situations so long as there is no cycle of zero timestamp increment [CM79], i.e., a cycle in which a message could traverse in zero time (although rarely arises in a physical system).

An LP determines the timestamp of a null message it plans to send by checking the clock value of each incoming link and the simulated time which will elapse when processing an event. Whenever an LP finishes processing an event, it can send null messages to all LPs whom it links to. A well-known problem with this approach is the possibility of having a large amount of null messages which can flood the communication bandwidth and cause a significant slow down of the simulation.

Variations of the algorithm have been proposed to trim down the number of null messages. In some variations, such as in [Mis86, BS88, NR84], null messages are sent on a demand basis. The null message traffic can also be reduced by the *lookahead* information, whose description follows.

4.1.2 Lookahead

Lookahead refers to the capability of predicting what will happen in the simulation. An obvious form of lookahead is the minimum increment timestamp increment of an LP for processing any event. If LP_i with local clock $Clock_i$ requires at least l units of simulated time to process any event, then LP_i is said to have a lookahead of l, because it can guarantee that it will no longer generate messages with timestamps less than $Clock_i + l$. In addition to reducing the null message traffic, the lookahead information also helps to reduce the possibility of deadlocks and thus speed up simulation.

Many papers related to the derivation and the use of lookahead information have appeared. For example, Nicol [Nic88] proposes a scheme to improve the lookahead capability by precomputing the service times for some future events. Cai and Turner [Cai90, CT90] introduce *carrier null messages*, which play a similar role as that of the null messages, but carry additional information on lookahead and the route taken. This approach aims to further improve the quality of lookahead to reduce the message traffic. Fujimoto [Fuj88] demonstrates the importance of lookahead with experimental results, which show that simulators with poor lookahead properties can easily become overburdened by excessive overhead. Cota and Sargent [CS90] suggest a framework for automatic lookahead computation. They discuss the use of control flow graph as a representation of process behavior and show how lookahead information can be extracted from the graph. Lin, Lazowska and Baer [LLB90] consider the class of systems in which no lookahead information is available, and report that these systems cannot be efficiently simulated by using the CMB approach with deadlock recovery algorithm. They suggest to reconfigure the system such that there is no feedback loop, and then use the CMB approach to perform the simulation. Wagner and Lazowska [WL89] derive the expressions for the lookahead for certain queuing network simulation.

4.1.3 Deadlock Detection and Recovery

An alternative of dealing with deadlocks has been proposed by Chandy and Misra [Mis86, CM81]. Instead of avoiding deadlocks, a simulation is allowed to enter a deadlock. With this approach, two mechanisms are required: one is used to detect a deadlock, while the other is used to resolve it.

A deadlock can be detected by using an approach similar to that used in general distributed computing. The deadlock can be resolved by using the fact that the message with the smallest timestamp in any point of the simulation is always safe to process. Details of these mechanisms are described in [Mis86, CM81, LT90].

The advantages of the deadlock detection-and-recovery method lie in completely avoiding the null message traffic as well as allowing cycles with zero timestamp increment (although the latter usually leads to poor performance). The method, however, often results in sequential executions prior to a deadlock. This could adversely affect the overall performance if the simulation model is prone to deadlock. Deadlocks may occur frequently if there are relatively few messages compared to the number of links in the network.

4.1.4 Synchronous Method or Conservative Time Windows

Several researchers have suggested synchronous conservative methods which rely on LPs cooperating within some intervals (in simulated time) to determine safe events and then to process them. An important notion in this area is the *distance*, which is part of the concept of lookahead. The *distance* is a lower bound of the increment of the simulated time for an unprocessed event to propagate. In other words, it is the minimum amount of simulated time it takes for an event at one LP to affect the state of another LP.

Lubachevsky [Lub88, Lub89] demonstrates a window based approach based on the *bounded lag* protocol. In the bounded lag protocol, if two events are to be processed concurrently, the difference of their timestamps must be bounded from above by a constant. A problem common to the window-based approaches is the size of the time window. To tackle this problem, unfortunately, requires information which is application-specific.

4.1.5 Conditional Events

Chandy and Sherman [CS89a] suggest a *conditional events* approach which classifies events into *definite events* and *conditional events*. Definite events, which generally have smaller timestamps, will definitely be processed and will not be cancelled or disabled by other events. Conditional events, on the other hand, will be processed only when certain criteria are satisfied. Clearly, the event with the lowest timestamp in the system is always a definite event.

It is always safe to process definite events. Conditional events approach explores this property and is able to distinguish definite events from the conditional ones using conditional knowledge.

4.1.6 Other Enhancements and Approaches

Several researchers have proposed approaches which make use of application-specific information, network topology of LPs in a model, hardware architecture, etc. to exploit greater parallelism. For example, Kumar [Kum89] proposes a simplified synchronization protocol for acyclic networks. Nevison [Nev90] proposed a simplification to the coordination of simulated time for networks with many closed loops. Vries [dV90] considers networks with feedforward and feedback components.

4.2 Optimistic Protocols

Jefferson and Sowizral [Jef85, JS85] proposed an approach called Time Warp in the early 80s, which becomes one of the most well-known optimistic protocol.

In an optimistic approach, the synchronization mechanism does not prevent the LPs from receiving and processing messages whose timestamps may be out of order. Since this does not adhere to the local causality constraint, causality errors may occur.

A causality error is detected whenever an LP with local clock $Clock_i$ receives a message (called a *straggler*) with a timestamp $T_{str} < Clock_i$. Without considering the straggler message, the simulation may have become incorrect from the simulated time T_{str}. To restore the simulation to a correct state, the LP performs a *roll back* (or *rollback*) to a saved state of a simulated time no later than T_{str}, and restarts the simulation from that state. The roll back mechanism can be accomplished by saving the state of each LP periodically.

Note that an LP might have sent messages to other LPs after the simulated time T_{str} and in effect have propagated the error. To "unsend" a previously sent message m_a, the LP sends a *negative*- or *anti-message* to annihilate the original one, referred to as the matching *positive* message of m_a. Anti-messages can be generated by examining the record of positive messages sent. When an LP receives an anti-message, it examines if it has processed the corresponding positive message. If it has not processed the positive message, the positive message and the anti-message simply cancel each other. Otherwise, the LP has to perform a rollback and may have to send anti-messages to other LPs in turn. Repeating this procedure recursively guarantees to restore the simulation to a correct state.

Note that at any point of simulation the unprocessed event with the smallest timestamp among all LPs can always be processed. This timestamp is called the *Global Virtual Time* (GVT). Since no LP will produce an event which has a timestamp smaller than the GVT and hence causes a rollback, therefore, for each LP, all but one of the saved states with timestamps smaller than GVT can be discarded safely. We need one saved state with a time not greater than GVT in case a rollback to GVT occurs. Many algorithms for efficiently computing or estimating GVT have been proposed, some of which can be found in [Bel90, Win92, SLWN95, XGUC95, Mat93, VCW95, VCW94, Fer95, BCC91].

There are certain operations such as input and output operations for which it is impossible to undo the effects. Therefore, such *irrevocable* operations should be committed only when it is certain that the simulator will not roll back to a time before the irrevocable operations occur. In other words, such operations are committed only after when the GVT has advanced beyond the simulated occurring times of these operations. The process of "clearing" the unneeded data entries, including reclaiming memory from the state saving mechanism and committing irrevocable operations, is called *fossil collection.*

4.2.1 Lazy Cancellation

The original Time Warp algorithm uses *aggressive cancellation.* In this cancellation strategy, an LP which rolls back to simulated time T immediately sends out anti-messages for any previously sent messages with timestamp greater than T. To reduce the anti-message traffic and the frequency of roll backs, a *lazy cancellation* strategy has been proposed [Gaf88].

In lazy cancellation, instead of immediately sending anti-messages upon receiving a straggler message, an LP examines if the rerun of the simulation generates the same positive messages. An anti-message is sent only if its matching positive message is not regenerated. Adopting lazy cancellation provides an additional benefit of exploiting lookahead implicitly [Fuj90a]. Unlike in conservative approaches where lookahead must be specified explicitly, lookahead can be exploited implicitly with lazy cancellation, even when it is not statically guaranteed. However, lazy cancellation does require additional storage for recording the positive messages sent; it also entails the overhead of matching an anti-message with an earlier positive message.

A comprehensive comparison of the aggressive cancellation scheme with the lazy cancellation scheme is given in [RFBJ90]. Lazy cancellation does reduce the traffic overhead, but it may also allow erroneous computations to propagate further. This is because an LP requires additional time overhead to check whether same messages are created. One can always construct cases where lazy cancellation outperforms aggressive cancellation, and vice versa. However, theoretical and empirical studies suggest that lazy cancellation tends to perform at least as well as aggressive cancellation, if not better [RFBJ90].

4.2.2 Lazy Reevaluation

Lazy reevaluation, also known as *jump forward* or *lazy rollback*, is similar to lazy cancellation. However, it deals with the state vectors rather than messages. Assume that an LP with local clock $Clock_i$ receives a straggler with timestamp t_{str}. If there is no state change in the LP within the period from t_{str} to $Clock_i$, the LP needs not rollback but instead can "jump forward" over the rolled back events.

Lazy reevaluation does prevent the LPs from performing unnecessary recomputation of states. It however requires additional storage and bookkeeping overhead which can significantly complicate the coding of optimistic protocols [Fuj90a].

4.2.3 Wolf Calls

With the goal of preventing the erroneous computation from spreading widely, Madisetti, Walrand, and Messerschmitt [Mad88] have proposed the *Wolf Calls* scheme. An LP which just receives a straggler may send a special control message immediately to all other LPs. An LP which receives this message immediately freezes its computation.

This scheme however may cause some correct computations to be frozen unnecessarily. Alternatively, it requires application-specific knowledge such as the speed at which erroneous computation may spread, and the speed at which the control message may broadcast. It may be difficult to extract such information for some simulation models.

4.2.4 Optimistic Time Windows

Similar to conservative simulation, window-based approaches have also been proposed for optimistic protocols with the goal of reducing the number of causality errors. Sokol, Briscoe and Wieland [SBW88] have proposed the *Moving Time Window* (MTW) approach that uses a time window with a fixed size W. The LPs can only process the events within the interval $[GVT, GVT+W]$.

Similar to the time windows approaches in conservative protocols, critics point out that it is not clear how to determine the size of the window. Furthermore, it is rather difficult to distinguish good computations from the erroneous ones within the window. The correct computations beyond the upper bound of the window cannot be processed as well. Empirical results are reported in [SBW88, RWJ89, SS90]. It is found that MTW offers some improvements in certain cases but only offers little help for others.

Lubachevsky, Shwartz and Weiss [Lub89, LSW89] propose *filtered rollback* which is a combination of the Time Warp and the bounded lag simulation algorithms. The algorithm contains tunable parameters which at one extreme make it identical to the bounded lag algorithm, and at the other extreme make it identical to the MTW approach.

4.2.5 Other Enhancements and Approaches

Many enhancements have been proposed to improve the optimistic protocols. There are also a number of hybrid approaches combining certain aspects of conservative approaches and optimistic approaches. Some of the important ones are outlined here.

Fujimoto [Fuj89] proposes *direct cancellation* which is optimized for multiprocessors with shared memory architecture. Let Γ_E be the set of events scheduled when an event E is processed. In this strategy, the simulator keeps the pointers from E to all elements in Γ_E. When an LP rolls back and needs to cancel the effect due to the processing of E, it can track the events generated (i.e., Γ_E) due to E by following the pointers kept with E. On a shared memory multiprocessor, this strategy should outperform the anti-message approach. Empirical results presented in [Fuj89, Fuj90b] show improvements in performance.

Chandy and Sherman [Win92, CS89b] proposed the *space-time* approach which views a simulation execution as filling in a two-dimensional space-time region (as shown in Figure 1), with one dimension representing the simulated time and another dimension the state variables. The key idea of the approach is to partition the space-time graph into disjoint regions and to assign each process one of these regions. Each process is then responsible for filling up its own region. It is important that given a region, the conditions along the boundary must be consistent with the conditions of the regions adjacent to it. Recomputations are required if there are inconsistencies. The computation proceeds until there is no more inconsistency. It is found that this approach resembles Time Warp with lazy cancellation strategy [Fuj90a].

Prakash and Subramanian [Win92, PS91] propose the *filter algorithm* to reduce cascaded rollback by checking the spread of erroneous computation. However, unlike Wolf calls, the algorithm attempts not to freeze the correct computation by. It requires each LP to keep some information such as the number of messages sent so far, the rollbacks carried out, etc. This adds additional overheads to the standard Time Warp approach.

In Time Warp, if there are excessive numbers of rollbacks, explosion of anti-messages can result, which in turn leads to unstable performance of the simulation. *Breathing Time Buckets* (BTB) approach [Ste93, Ste92], proposed by Steinman, is a window-based approach which does not require anti-messages and therefore does not suffer from the this problem. *Breathing Time Warp* (BTW) [Ste93] is later proposed also by Steinman aiming to combine the best aspects of Time Warp and BTB while eliminating their shortcomings. Preliminary results show that BTW outperforms BTB and Time Warp for certain applications.

5 Mathematical Performance Models

Mathematical performance modeling of parallel simulations has been studied for years with the goal to compare the efficiencies of various strategies under various assumptions. Simple models generally provide intuitive insights into the qualitative behavior of PDES, while complex models provide better performance predictions. A few interesting results are outlined below.

In [FK90], Felderman and Kleinrock show that the average potential speedup obtained from using an asynchronous protocol over a synchronous protocol is no more than $O(\log P)$, where P denotes the number of processors. The analysis assumes exponentially distributed execution time in every processor. For uniformly distributed time, the potential speedup obtained is no more than 2. Although the model is simplistic, the result suggests that one should lower the expectation of getting great speedups when switching from a synchronous scheme to an asynchronous scheme.

The performance of Time Warp has been studied and demonstrated extensively. In [LL90], Lin and Lazowska prove that, assuming (unrealistically) zero-cost rollbacks and infinite memory space, Time Warp approach outperforms the CMB approach in feedforward network[1] simulation. Lipton and Mizell [LM90] show that there exists a simulation model for which Time Warp outperforms the CMB approach by a factor of p, p being the number of processes—but the reverse is not true; the CMB approach can only outperform Time Warp by a constant factor. In [Gun94], Gunter shows that Time Warp with lazy cancellation can beat the *critical path* lower bound, where the critical path is a lower bound on the time for a conservative approach to execute a simulation.

For a discussion of more complex models and a more comprehensive survey on mathematical performance models, the readers are directed to [NF94], a survey conducted by Nicol and Fujimoto.

6 Other Important Issues

Besides simulation strategies, which define the operational principle of a simulation framework, there are other components which can also critically affect the simulation behavior. This section briefly covers the basic concepts of some of these topics [FN92, NF94].

Hardware Support. Hardware supports for PDES have been discussed for years. However, most of the supports developed are application-specific. For example, circuit simulations with hardware support have been researched and applied in the industries for years. Work for general-purpose simulation has been carried out. Hardware supports for optimistic protocols have been studied, with the goal of minimizing various overheads incurred in optimistic protocols. See, e.g., [Con89, FTG92].

Load Balancing. A load balancing algorithm is responsible for distributing the workload evenly among all available physical processors. This generally involves a many-to-one mapping, from the logical processes to the physical processors. The algorithm can either be static, which employs only one fixed mapping throughout a simulation run, or dynamic, which allows the mapping to change during execution. See, e.g., [NXG85, LK87, NR90].

Memory Management. Memory management is particularly important in optimistic protocols because optimistic protocols generally require enormous memory consumptions (mainly for state saving) which can even cause memory exhaustion. Many memory managements schemes have been proposed. For instance,

[1]A feedforward network, or feedforward graph, is a directed-acyclic graph (dag)

many have proposed efficient fossil collection strategy based on faster computations of GVT; others have suggested schemes based on infrequent state saving, etc. See, e.g., [LP91, PL95, WH95].

Random Number Generation. Just like sequential DESs, PDESs also require random quantities and therefore parallel pseudorandom number generators (parallel PRNGs) are needed. To guarantee good quality of randomness, there are more stringent requirements for parallel PRNG compared to that of sequential PRNG. For example, in PDES, it is required that sequences generated on any pair of processors be free of mutual correlations. Therefore, replicating the same sequential PRNG may not guarantee a good parallel PRNG. Although PRNG is a relatively traditional topic, it remains an active research area. See, e.g., [Alu97, L'E94, BR87].

Event-Set Algorithm. Similar to the case in DES, event-set algorithm is used to schedule the events in proper order in PDES. However, a basic priority queue structure cannot meet the requirements of some simulation approaches, because operations on the event set have to be serialized and may become a bottleneck. A concurrent priority queue data structure is required in this case. See, e.g., [Ste96, RAFD93, Jon89].

7 Parallel Simulation: A Summary

This section presents a comparison and some critiques to the approaches for parallel simulation. A comparison of the strategies employed by conservative approaches with that of optimistic approaches is also presented.

7.1 A Comparison of Existing Strategies

Table 1 compares the main features of the conservative approaches and the optimistic approaches. The table summarizes some operational principles of both approaches, and presents some critiques to them.

Many analytical and empirical studies have been conducted to evaluate various strategies. The performance of the conservative approaches are studied in, e.g., [CT97, KRR96, Fuj88, MRR90]; performance of the optimistic approaches in, e.g., [Gil88, LCUW88, Bel93, Fuj90b]. Some of the important findings are summarized below:

- There has been no single approach which is able to provide good performance for all (or most) kinds of applications. An approach which is good for certain applications may perform badly for the other applications.

- Conservative approaches are constantly accused of lacking robustness in terms of their performances. First, they heavily rely on lookahead. Stripping the lookahead information off the specification can degrade the performance considerably. Second, The simulation time taken can be sensitive to small changes in the system (which can affect the lookahead values). In contrast, optimistic approaches do not rely heavily on lookahead and hence their performance is less sensitive to small changes in the specification.

- Optimistic approaches inherently have more overheads which are not shared by conservative approaches. The overheads include state saving, rollback, GVT calculation, fossil collection, etc. The degree at which they may affect the overall performance depends on factors such as the granularity of each LP, the frequency of state saving, supports from hardware, etc.

Strategy	Conservative Approaches	Optimistic Approaches
Principle	strict adherence to local causality constraint	allow violation of local causality constraint; provide rollback mechanism to recover if causality errors occur
Parallelism	allow less parallelism for exploitation	allow aggressive exploitation of parallelism
Synchronization	block LPs which may violate local causality constraint; may cause deadlocks if precaution is not taken	rollback mechanism to recover from causality errors; incur state-saving overhead
Deadlock	adopt avoidance strategy, detection-and-recovery strategy, or synchronous approaches	no deadlock problem
Lookahead	rely heavily on lookahead to achieve good performance; lookahead however might not be statically guaranteed but dynamically available	no dependence on lookahead to achieve good performance; lookahead can be exploited automatically with lazy cancellation or lazy reevaluation
Configuration of LPs	require static configurations by most existing conservative techniques	network configuration can be changed dynamically
Memory Requirement	require less memory than that of optimistic protocols	require more memory; more complex and complicated memory management
Implementation	straightforward implementation and data structures	notoriously hard to implement; complex data manipulations

Table 1: A comparison between conservative approaches and optimistic approaches

Optimistic methods appear to offer greater hope for general purpose simulation, if state-saving overhead is kept within a manageable level [Fuj90a]. Recently, many successful cases of using optimistic methods have been reported, as can be seen from the number of publications in this field.

Nevertheless, conservative methods have also been found to offer great potential for certain classes of applications, particularly when we have ample application-specific knowledge of the systems being simulated. See, e.g., [PN98, MRR90].

7.2 Prospects of Parallel Simulation

Despite two decades of research, parallel simulation technology has not had significant impact to the simulation community. Computer scientists have to admit that developing a good parallel simulation with current technology remains a specialized technique which is mastered by only a small portion of researchers.

It is generally agreed that the future success of parallel simulation depends on whether it is possible to reduce the expertise and effort required to develop efficient simulation programs [Fuj93b]. Ideally, the users who develop simulation models should not be concerned about the low-level mechanisms adopted by the underlying simulation approach. Rather, they should be able to concentrate on correctly modeling the physical systems. Fujimoto [Fuj93b, Fuj93a] has suggested four potential "silver bullets" which would make parallel simulation more accessible: application-specific libraries, better languages for parallel simulation, good solutions to the support for shared state, and automatic parallelization of sequential simulations.

Abrams [Abr93] also outlines several scenarios which would foster wider use of parallel simulation technology.

8 Computational Models for Parallel Simulations

In the previous section, we have identified some critical issues which must be tackled to ensure the future success of PDES. In particular, the effort and the expertise required must be reduced; the performance concerns must also be alleviated.

Recently, to address the above concerns, a few researchers in this field have started to look into certain models of parallel computation and have attempted to design PDES algorithms based on these models. It is generally agreed that these computational models can provide a framework to simplify the design of correct and efficient algorithms, and to provide a *cost model* for performance prediction.

An introduction to the models of parallel computation is presented in Section 8.1. The BSP model and several existing PDES algorithms based on BSP are studied in Sections 8.2 and 8.3. The Cilk model is introduced in Section 8.5 while several existing PDES algorithms by using Cilk are reviewed in Section 8.6.

8.1 Models of Computation

Modeling complex phenomena is an old art. The purposes of modeling are to capture the salient characteristics of phenomena with clarity, and to provide the right degree of accuracy to facilitate analysis and prediction [MMT95].

In sequential computing, the *Random Access Machine* (RAM) has been a very successful model of computation that promoted consistency and coordination among algorithm developers, computer architects and language experts. It has elegantly expressed the important characteristics of sequential computation. In the realm of parallel computing, however, there has been no similar success [MMT95]. *Parallel Random Access Machine* (PRAM), the natural parallel analog of RAM, does not provide similarly good characteristics for modeling parallel computations. Surveys on PRAM and its variations can be found in, e.g., [MMT95, Goo93, McC93].

Since the PRAM model has not been able to reflect the key characteristics of the existing parallel computers, many other models have been proposed, see, e.g., the surveys in [MMT95, Ham96, CG96]. Of late, the BSP model and the Cilk model have gained much attention of the parallel computing community.

The BSP model and the Cilk model were proposed in the early 90s. Of particular interest, both models allow the programmers to make early prediction of the performance of a parallel program. The Cilk model even offers a guarantee of highly scalable performance, as long as the parallel program adheres to the Cilk programming paradigm [BJK+95].

8.2 The BSP Model

The *Bulk-Synchronous Parallel* (BSP) model was proposed in 1990 by Valiant [Val90] as a *bridging model* of parallel computation. The notion and the role of a *bridging model* was aptly captured and introduced by Valiant in [Val90]: *"...the von Neumann model is the connecting* bridge *that enables programs from the diverse and chaotic world of software to run efficiently on machines from the diverse and chaotic world of hardware."*

As shown in Figure 2, a BSP model, or a *bulk-synchronous parallel computer* (BSPC), has the following elements:

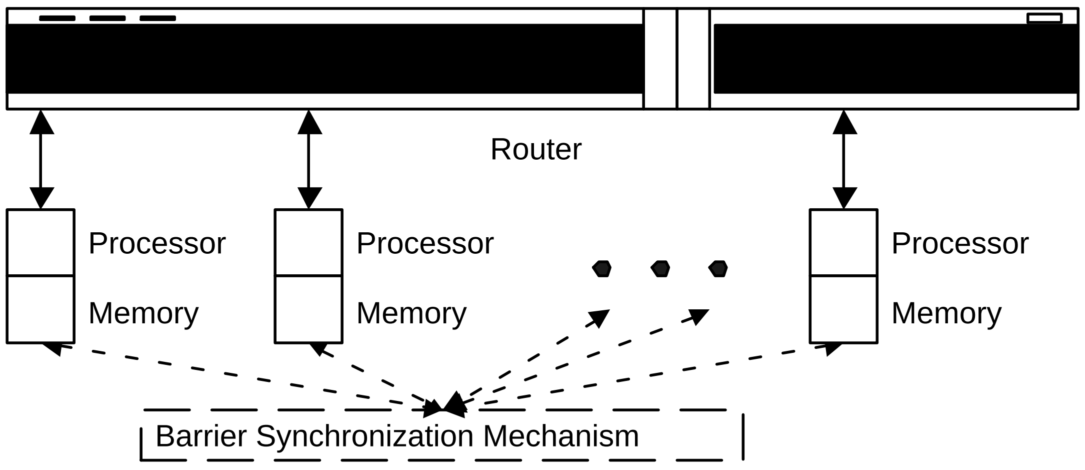

Figure 2: A Bulk-Synchronous Parallel Computer (BSPC)

1. A number of *components*, each performing processing and/or memory functions. In practice, a component can be viewed as consisting of a processor with a private local memory.
2. A *router* that delivers messages point-to-point between pairs of components.
3. A *global barrier synchronization mechanism* for synchronizing all, or a subset of, the components. The interval from a synchronization to the next synchronization is referred to as a *superstep*.

A BSP computation consists of a sequence of supersteps. In each superstep, each component may perform a number of operations on the data held *locally*. A component can also communicate with the other components by sending and/or receiving messages, but all messages will only be routed after the next synchronization, i.e., at the end of the current superstep. For clarity, a superstep can be divided conceptually into a computation superstep and a communication superstep.

The BSP model facilitates the prediction of program performance based on two parameters: L, the synchronization period; and g, the bandwidth parameter, defined as:

$$\begin{aligned} g &\triangleq \frac{\text{computation rate}}{\text{communication rate}} \\ &= \frac{\text{total local operations performed by all processors per time unit}}{\text{total messages delivered by the rounter per time unit}} \end{aligned}$$

which attempts to summarize the characteristics of network bandwidth as well as the latency or the startup cost of routing.

For a superstep with a *h-relation*, i.e., a routing request where each processor sends and receives at most h messages, its cost model is given by [GV94]:

$$\max\{L, x + hg\}$$

where x is the maximum number of local computations executed by any processor in that superstep. There are also alternative charges, such as $\max\{x, L+gh_s, L+gh_r\}$ [GV94], or $\max\{L, x, gh_s, gh_r\}$ [McC93,McC94], where h_s and h_r are the maximum number of messages sent or received by any processor in that superstep respectively. These alternatives are due to the small variations in interpreting and realizing the BSP model. Nevertheless, the difference among the costs will not, in general, be significant.

With this cost model, the performance of a BSP algorithm can be predicted by using the parameters L, g, and p parameters regardless of the underlying architecture, p being the number of components. In addition, the BSP model is generally able to assure good performance if there exists sufficient *parallel slackness* [Val90], i.e. $v \gg p$ for an algorithm written for v virtual parallel processors running on p physical processors. In this case, it is said that there exists a *parallel slackness factor* of $\frac{v}{p}$.

8.3 The BSP Algorithms for PDES

8.3.1 Conservative Protocols based on BSP

Calinescu [Cal95] and Marin [Mar97a, Mar97b] propose several conservative approaches for parallel simulation with the BSP model. A characteristic shared by most of these algorithms is that messages sent between LPs during a superstep are processed by the recipients in the *subsequent* rather than the current superstep. This is because in BSP all messages are routed only at the end of a superstep. Another important characteristic is the use of double buffering technique for communication between LPs.

Recall that in the BSP model, a component can only operate on the data held locally. If we associate a link from LP_i to LP_j with only one buffer, then, in any superstep, the buffer can only be operated exclusively by either LP_i or LP_j, because the same buffer cannot be simultaneously read and written by different components. An algorithm employing this single buffering technique can ensure such mutual exclusive access by, say, allowing all LPs to send messages only (and cannot read any messages received) in some supersteps, and to read the messages received in the other supersteps. Alternatively, such inefficiency can be eliminated by using the double buffering technique.

With double buffering, a link is associated with two buffers. One buffer is used for reception while another for transmission. These two buffers are swapped in the communication superstep.

The buffer size must be properly chosen as it can affect the overall performance significantly. A smaller buffer size implies more supersteps which will incur more overheads. However, a larger buffer size does not necessarily increase the overall performance, because it may result in many idling LPs in any superstep and hence limit the potential parallelism of the simulation [Cal95]. To see this, consider the simulation of a cyclic queuing network with only 3 LPs, as illustrated in Figure 4. Assume that LP_0 initially generates a total of 3,000 messages to be passed to LP_1. Whenever an LP receives a message, the message will be redirected to another LP after arbitrary simulated time.

If the buffer can hold at least 3,000 messages, then in any superstep, only one LP will be busy processing these messages from the input link. However, if the buffer can hold only 1,000 messages, every LP will be busy processing 1,000 messages. Clearly, the loads among the LPs are more balanced in the latter case. The smaller buffer has helped to distribute the messages more evenly, and better parallelism is obtained in the latter case. A proper choice of buffer size is therefore crucial to the performance. This example serves also to illustrate the subtlety of (and the interplay between) a myriad of factors may affect the performance in parallel simulations.

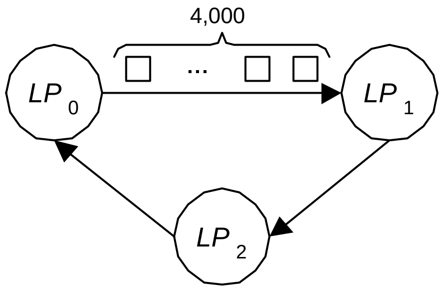

Figure 3: A cyclic queuing network

Deadlock Resolution

With BSP, the null messages approach for deadlock resolution can be inefficient, because supersteps are needed additionally for sending null messages. The deadlock detection and recovery scheme which uses a marker in [Mis86] is not appropriate neither. Since an LP is restricted to using only local data, the marker requires p supersteps to traverse all p LPs. This can increase the number of supersteps significantly as well.

The deadlock detection and recovery algorithm proposed in [Cal95] is based on the algorithm by Reed, Malony and McCredie in [RMM88]. With this algorithm, each physical processor is responsible for reporting whether it is deadlocked to a special *guardian* processor which is responsible for monitoring the global system state. The algorithm entails only two supersteps, with p-relation for each superstep, to detect a deadlock and to resume the simulation. This requires a cost of $2(L+pg)$. Thus, the algorithm is not quite suitable for large g and also not scalable with respect to p. An algorithm which generalizes the pattern of broadcasting has been proposed in [Cal95]. The algorithm requires only k-relation in each superstep, where $1 < k \leq p$, but requires more supersteps. This algorithm costs $2\lfloor \log_k p \rfloor (l + kg)$.

Calinescu presents some empirical results using his approach in [Cal95]. It is found that the performance of the simulation is heavily influenced by two parameters: the parallel slackness of the algorithm and the inter-process buffer size. A speedup of about 8 is obtained with 14 processors in simulating a 42-LP tandem network. However, it is not clear how the buffer size should be chosen to maximize the performance.

In [Mar97b], Marín demonstrates an algorithm which uses null messages with lookahead values to reduce the null message traffic and the likelihood of getting deadlocked. A deadlock is resolved by using a deadlock detection and recovery algorithm which recognizes that it is always safe to process the event E_{min} with the smallest timestamp T_{min} in the system. A master process is responsible to determine T_{min} and broadcast the value to all LPs. However, no experimental results of Marín's approaches have been published.

In [LLC$^+$98], Lim *et al.* has evaluated the performance of a conservative approach using BSP for manufacturing simulation based on the *safe time* algorithm proposed in [CLT97], which will be introduced in Section 8.6. They show good speedups for LP with medium to large granularity. However, it is believed that the approach is relatively less scalable when the granularity is small.

8.3.2 Optimistic Protocols based on BSP

GVT Computation

In [Cal96], Calinescu suggests an optimistic approach for parallel simulation with the BSP model, and considers some important issues related. One of the issues is the efficient algorithms of GVT computation. Since the BSP model guarantees that any messages sent in a superstep are delivered by the start of the next superstep, no acknowledgments are required. This property greatly simplifies the design of the algorithms for GVT computation.

In this approach, the processors are first organized into a complete q-ary tree, i.e., a tree whose non-leaf nodes have exactly $q \geq 2$ sons, and all leaves are in the same level. In the first superstep, each of the leaf processors estimates its local GVT over the set of LPs they simulate, and sends it to its parent. In the subsequent supersteps, a processor which receives the estimates from its children computes its own estimate of the GVT, and then sends the estimate to its parent. Such computation is repeated until the root processor receives the estimates. After computing the global GVT estimate, the root processor sends the value to its children. In the subsequent supersteps, each processor which receives the estimate propagates it to its children in the subsequent superstep. The propagation stops when all processors have received the estimate.

Recursive-Rollback Avoidance

If the parallel simulation progresses in a way that all messages (including anti-messages) are handled and propagated (if necessary) in the same superstep, *recursive-rollback* can occur. Consider a simple simulation model with a number of LPs connected in a directed cycle. If an LP ever sends an anti-message to another LP, the anti-message can chase its matching positive message for the rest of the simulation. This phenomenon, referred to as *recursive-rollback*, can degrade the performance of the simulation dramatically due to the heavy traffic of the anti-messages.

The algorithm presented in [Cal96] addresses the problem by keeping a *rollback history queue* on each LP. It keeps a record of the anti-messages generated. Basically, if an LP receives an anti-message which is recorded in the queue, the anti-message will no longer be propagated. With this algorithm, it can be shown that any anti-message will eventually stop propagating.

Semi-Optimistic Approach

Marín also suggests a *semi-optimistic* [Mar97b] algorithm in BSP which allows a processor to execute events optimistically if it has no safe events to process in that superstep. When causality errors occur, a rollback message is broadcast to all processors. Notice that no anti-messages are ever required since a global rollback is used.

Optimistic Approaches

An BSP approach to optimistic PDES with filtered rollback is outlined in [Cal96]. A BSP version of Time Warp (BSP TW) is given in [Mar97b, Mar98c]. Marin also compares the effectiveness of Time Warp and Event Horizon in [Mar98a]. An important result is that the total number of supersteps required by Time

Warp is at most that required by Event Horizon approach [Ste94]. This implies that Time Warp generally offers more parallelism.

Empirical Results

At the time of this writing, few experimental results have been published for the optimistic approaches based on BSP. Calinescu has not published the empirical results based on his approach. Marín presents in [Mar98b] some preliminary results of several optimistic PDES approaches on BSP. Several BSP version of approaches such as BSP, MTW, BTB, BTW are compared in terms of important parameters such as the number of supersteps involved and the balance in communication. He concludes that in many situations, BSP TW should outperform the other approaches.

8.4 Critiques of BSP Approaches to PDES

The research of using the BSP model for PDES is still in early stage. A lot of work is still needed. Some of the drawbacks for the existing approaches are discussed in this section.

With current approaches, the programmers still need to be concerned about the details of the underlying simulation mechanism in order to achieve good performance. For example, the performance of the algorithms as presented in [Cal95] is sensitive to the buffer size associated with each link and to the bandwidth of the system.

Perhaps more seriously, as JáJá argues in [JáJ96], the BSP model is not a suitable bridging model because the parameters of the BSP model have to be adjusted across various platforms. This can hinder the developments of tools and algorithms to improve performance. Furthermore, although the model provides early prediction of performance, it is not able to provide a *performance guarantee*. In summary, we therefore argue that the performance concerns have not been alleviated significantly with BSPC.

In the following section, we will introduce the Cilk model, which is the first and only model that provides the programmers with a guarantee of application performance. Moreover, since the Cilk language is a natural extension of the C language, it allows a C programmer to adopt the Cilk programming paradigm with minimum efforts. In contrast, we believe that it requires longer time for a C programmer to get used to the BSP programming paradigm. [LLC+98] also reports that the BSP version of the *safe time* algorithm [CLT97], which will be introduced in Section 8.6, is more complex than the Cilk version, mainly because each component in the BSPC is allowed to operate only on the data held locally.

In the following sections, some current PDES approaches using the Cilk model will also be introduced. Some general considerations and implications of applying Cilk in PDES are also presented.

8.5 The Cilk Model

Cilk [BJK+95, MIT98, Joe96, Ran98] is a C-based runtime system for multithreaded parallel programming. It was developed and is being maintained by a team led by Leiserson at MIT. The Cilk model comprises a model for parallel computation with an algorithmic performance guarantee.

8.5.1 Computation Model

A *Cilk procedure* is a parallel analog of a C function which is identified by the keyword `cilk`. Parallelism is introduced when a Cilk procedure is invoked with the keyword `spawn`. Instead of waiting for the children

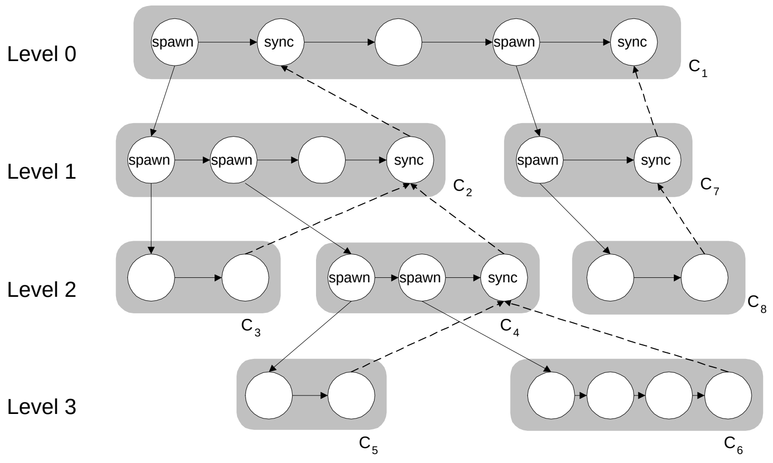

Figure 4: The Cilk model of computation. Threads, which are shown as white circles, are embedded into a tree of Cilk procedures, shown as shaded boxes with rounded corners. The edges denote the ordering constraints. The sequence of instructions within a Cilk procedure is order by horizontal edges. Each downward edge corresponds to a `spawn` of a child, while each upward, dashed edge corresponds to the dependency introduced by `sync` statement.

(the spawned procedures) to complete as is done in C, the parent procedure instance continues to execute in parallel. To synchronize, the parent procedure executes a `sync` statement, which serves as a local "barrier", to wait for the completion of *all* its children.

A Cilk procedure is broken down into a sequence of *threads*, which are considered as atomic execution units. Each thread is a maximal sequence of instructions ending with a `spawn`, a `sync` statement, or a return from a Cilk procedure.

With the Cilk call/return semantics (`spawn` and `sync`) for parallelism, the computation of a Cilk program can be viewed as a series-parallel dag [FL97] (called a *Cilk dag* henceforth) that unfolds dynamically as the computation progresses. Figure 4 shows an example of the computation of a Cilk program. The Cilk procedure C_1, shown as the shaded box lying in level 0, is broken down into 5 threads, shown as the white circles. The edges denote the ordering constraints. For instance, the first thread in C_1 spawns C_2, which in turn spawns C_3 and C_4. Notice that C_3 and C_4 can execute concurrently, along with the third thread of C_2. C_2 waits for the completion of C_3 and C_4 at the `sync` statement before returning to C_1.

8.5.2 Performance Model

The Cilk runtime system provides an algorithmic model of performance based on two parameters: T_1 (*work*) and T_∞ (*critical path length*). The execution of any parallel program can be measured in terms of these two parameters [BJK+95, Blu95, BL94, KR90]. The *work* of a program corresponds to its execution time on one processor. The *critical-path length* corresponds to its execution time on an infinite number of processors, which is also the time required to execute the threads along the longest dependency path in the Cilk dag.

The performance of the Cilk runtime system depends on the efficiency of its work scheduler. Cilk uses a provably efficient scheduler which is based on the concept of *randomized work-stealing* [Blu95, BL94]. With this technique, a processor (the *thief*) who runs out of work selects another processor (the *victim*) randomly, from whom to steal a ready thread. It has been shown that the scheduler guarantees that the expected execution time of a lock-free Cilk program[2] on P processors is given by [Blu95, BL94]

$$T_p = T_1/P + O(T_\infty)$$

which provides a near-linear speedup and is asymptotically optimal, since T_1/P and T_∞ are both lower bounds [BL94, BJK+95]. The equation also resembles Brent's theorem [CLR90, p. 709] which yields the upper bound of $T_p \leq T_1/P + T_\infty$ [Blu95]. It has been verified empirically that the constant factor hidden by the order notation is small, so that $T_p \approx T_1/P + T_\infty$ is a good approximation for a wide range of applications [Blu95, BJK+95, FLR98, BL94].

8.6 The Cilk Algorithms for PDES

PDES by using Cilk is a relatively new research topic. Only two existing results based on conservative protocols have been presented in the literature. Both results and their empirical findings are presented below.

8.6.1 The Safe Time Approach

Cai, Letertre and Turner describe in [CLT97] a general conservative simulation algorithm, referred to as the *safe time* approach, by using Cilk. The algorithm is robust in that it can be applied for simulation applications where lookahead information is not available or is difficult to extract.

The algorithm comprises a number of iterations, where all LPs wait at every transition of iterations for the computation of the *gst*, the global simulation time. The *gst* is the minimum of all LVTs (Local Virtual Times) and the timestamps of all messages in the simulation. Let *inclock* be the minimum of the input link clocks of a given LP. In any iteration an LP first computes the *safe time* which is the maximum of (1) the *inclock* in the current iteration and (2) the *gst* computed at the beginning of the current iteration. The algorithm guarantees that, given an LP, all events within the LP with timestamps not greater than the safe time are safe events, which are always safe to process.

At each transition after *gst* has been computed, it is propagated to all LPs in a divide-and-conquer manner. The divide-and-conquer procedure creates a hierarchy (specifically, a binary tree *cf.* Figure 5) of Cilk procedure instances, in which each Cilk procedure instance is assigned a number of LPs. When the number of LPs assigned to the same Cilk procedure instance is smaller than a fixed threshold, they are executed serially.

At the end of an iteration, a Cilk procedure returns to its parent the minimum of (1) the LVTs of the subset of the LPs it is simulating, and (2) the timestamps of all messages generated in the current iteration. The root procedure instance computes the *gst* by taking the minimum of all values returned. It then broadcasts the value to all LPs, which starts another iteration. Detailed descriptions and the pseudo-code of the algorithm is given in [CLT97].

Figure 5 shows a trace of a simulation program with this algorithm. At the beginning of an iteration, a procedure instance "broadcasts" the *gst* to its child procedure instances in a divide-and-conquer manner.

[2]The statement applies only to Cilk programs that contain no *lock* constructs. Cilk guarantees that critical sections guarded by the same lock act atomically with respect to each other. If the lock contention is reasonably low, this performance model should still apply.

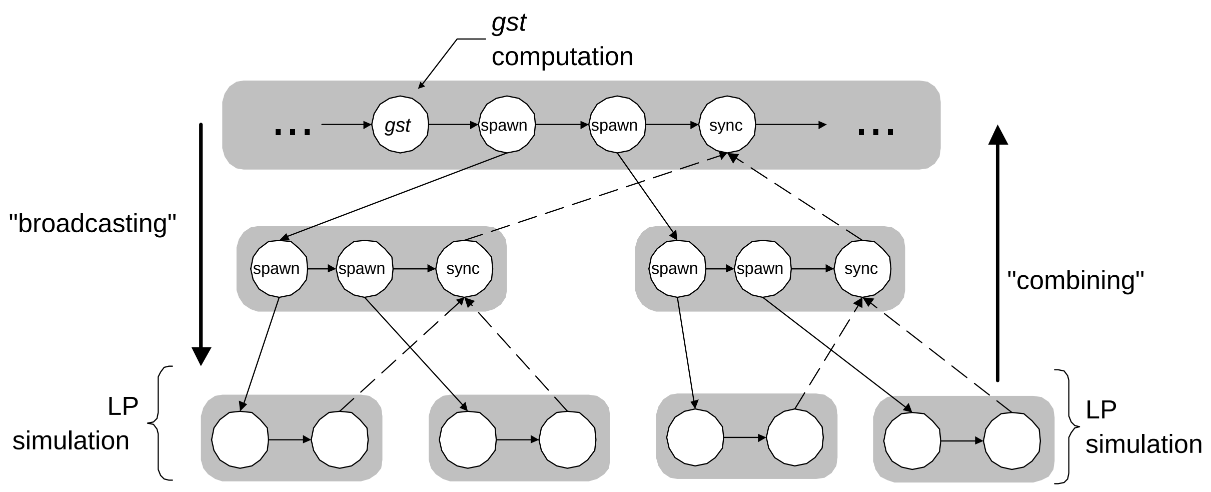

Figure 5: A trace of a simulation program with the safe time approach. The trace illustrates various steps involved in an iteration.

After simulating the LPs, each child procedure instance returns a value for *gst* computation. These values are "combined" (by taking the minimum) as the new *gst* for the next iteration.

Empirical Results

Promising results have been reported using the safe time approach [CLT97]. With 16 processors, a speedup of about 9.4 has been achieved for the super-ring simulation, while a speedup of about 13 has been achieved for a simplified manufacturing simulation. However, as also further confirmed in [LLC+98], large grain size is required for the approach to be scalable to more processors. Low speedup is obtained if the grain size is small, since in this case the overhead of spawning Cilk procedures would dominate the execution of the simulation run.

8.6.2 Nops: A Conservative PDES Engine

Poplawski and Nicol implemented a parallel simulator, Nops, to support the TeD (Telecommunications Description Language) by using Cilk [PN98]. Their approach is to modify the Cilk runtime system in order to map a TeD process to a Cilk thread naturally. They have included a mechanism to suspend and to restart a Cilk thread—a feature not in the original Cilk model. Nevertheless, they retain Cilk's mechanisms for spawning threads, saving state, a limited degree of scheduling, etc.

However, in their approach the Cilk's work-stealing mechanism has been suppressed, which happens to be the main element that guarantees the application performance. It is thus of a major interest to see whether the performance can be improved by applying load balancing, or by using the original Cilk model.

Empirical Results

Although the Cilk runtime system has been modified, Nops is reportedly able to provide near-optimal speedup [PN98]. Nevertheless, when the grain size is small, the algorithm is also less scalable with respect to the increase in the number of processors. In spite of this restriction, Poplawski and Nicol have shown that Nops outperforms most of the other existing simulation packages [PN98].

Discussion

Encouraging results have been reported in applying Cilk to certain parallel simulation applications. We believe that there are still ample rooms for enhancement to the current approaches. In particular, we aim to further exploit the potentials of Cilk and design approaches that target wider range of simulation applications, including medium- to small-grained simulation applications.

9 Conclusions

Simulation has been heavily relied upon in many fields of engineering and science. The simulation technology is developing rapidly in recent years. In this paper, we have surveyed the major existing approaches and examined the outlook of the field. We also cover the approaches based on two leading models of parallel computation. Such approaches facilitate certain degree of performance prediction. We also give a snapshot of some important issues related to the parallel simulation and a comparison of existing strategies.

References

[Abr93] Marc Abrams. Parallel discrete event simulation: Fact or fiction? *ORSA Journal on Computing*, 5(3):231–233, Summer 1993.

[Alu97] Srinivas Aluru. Lagged Fibonacci random number generators for distributed memory parallel computers. *Journal of Parallel and Distributed Computing*, 45(1):1–12, August 1997.

[BCC91] Reid Baldwin, Moon Jung Chung, and Yunmo Chung. Overlapping window algorithm for computing GVT in time warp. In *The 11th International Conference on Distributed Systems*, pages 534–541, Washington, D.C., May 1991.

[Bel90] Steven Bellenot. Global virtual time algorithms. *Proceedings of the SCS Multiconference on Distributed Simulation*, 22(1):122–127, January 1990.

[Bel93] Steven Bellenot. Performance of a riskfree time warp operating system. In *Proceedings of 7th Workshop on Parallel and Distributed Simulation (PADS'93)*, pages 155–158, San Diego, California, May 1993.

[BJK$^+$95] Robert D. Blumofe, Christopher F. Joerg, Bradley C. Kuszmaul, Charles E. Leiserson, Keith H. Randall, and Yuli Zhou. Cilk: an efficient multithreaded runtime system. In *Proceedings of the 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 207–216, Honolulu, Hawaii, July 1995.

[BL94] Robert D. Blumofe and Charles E. Leiserson. Scheduling multithreaded computations by work stealing. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 356–368, Santa Fe, New Mexico, November 20–22, 1994.

[Blu95] Robert D. Blumofe. *Executing Multithreaded Programs Efficiently.* PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, September 1995.

[BR87] Komiko O. Bowman and Mark T. Robinson. Studies of random number generators for parallel processing. In Michael T. Heath, editor, *Proceedings of the Second Conference on Hypercube Multiprocessors*, pages 445–453, Knoxville, Tennessee, September 29–October 1, 1987. SIAM.

[Bry77] R. E. Bryant. Simulation of packet communication architecture computer systems. Technical Report MIT-LCS-TR-188, Massachusetts Institute of Technology, 1977.

[BS88] William L. Bain and David S. Scott. An algorithm for time synchronization in distributed discrete event simulation. *Proceedings of the SCS Multiconference on Distributed Simulation*, 19(3):30–59, February 3–5, 1988.

[Cai90] Wentong Cai. *Parallel Program Monitoring: the Logical Clock Approach and its Deadlock Avoidance.* PhD thesis, University of Exeter, 1990.

[Cal95] Radu Calinescu. Conservative discrete-event simulations on bulk synchronous parallel architectures. Technical Report PRG-TR-16-95, Oxford University Computing Laboratory, 1995.

[Cal96] Radu Calinescu. Bulk synchronous parallel algorithms for optimistic discrete event simulation. Technical Report PRG-TR-8-96, Oxford University Computing Laboratory, 1996.

[CG96] Thomas H. Cormen and Michael T. Goodrich. A bridging model for parallel computation, communication, and i/o. *ACM Computing Surveys*, 28(4es):208, December 1996.

[Chi92] James A. Chisman. *Introduction to Simulation Modeling Using GPSS/PC.* Prentice-Hall, Englewood Cliffs, New Jersey, 1992.

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introductions to Algorithms.* MIT Press, 1990.

[CLT97] Wentong Cai, Emmanuelle Letertre, and Stephen J. Turner. Dag consistent parallel simulation: a predictable and robust conservative algorithm. In *Proceedings of 11th Workshop on Parallel and Distributed Simulation (PADS'97)*, pages 178–181, Lockenhaus, Austria, June 10–13, 1997.

[CM79] K. Mani Chandy and Jayadev Misra. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering*, SE-5(5):440–452, September 1979.

[CM81] K. Mani Chandy and Jayadev Misra. Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM*, 24(11):198–205, November 1981.

[Con89] A. I. Concepcion. A hierarchical computer architecture for distributed simulation. *IEEE Transactions on Computers*, C-38(2):311–319, 1989.

[CS89a] K. M. Chandy and R. Sherman. The conditional event approach to distributed simulation. *Proceedings of the SCS Multiconference on Distributed Simulation*, 21(2):93–99, March 28–31, 1989.

[CS89b] K. M. Chandy and R. Sherman. Space, time, and simulation. *Proceedings of the SCS Multiconference on Distributed Simulation*, 21(2):53–57, March 28–31, 1989.

[CS90] Bruce A. Cota and Robert G. Sargent. A framework for automatic lookahead computation in conservative distributed simulations. *Proceedings of the SCS Multiconference on Distributed Simulation*, 22(1):56–59, January 1990.

[CT90] Wentong Cai and Stephen J. Turner. An algorithm for distributed discrete-event simulation—the "carrier null message" approach. *Proceedings of the SCS Multiconference on Distributed Simulation*, 22(1):3–8, January 1990.

[CT97] J. G. Cleary and J.-J. Tsai. Performance of a conservative simulator of atm networks. In *Proceedings of 11th Workshop on Parallel and Distributed Simulation (PADS'97)*, pages 142–145, LockenHaus, Austria, June 10–13, 1997.

[dV90] Ronald C. de Vries. Reducing null messages in Mirsa's distributed discrete event simulation method. *IEEE Transactions on Software Engineering*, SE-16(1):82–91, January 1990.

[Fer78] Domenico Ferrari. *Copmuter Systems Performance Evaluation.* Prentice-Hall, Englewood Cliffs, New Jersey, 1978.

[Fer95] Alois Ferscha. Parallel and distributed simulation of discrete event systems. In *Handbook of Parallel and Distributed Computing.* McGraw-Hill, 1995.

[Fis95] Paul A. Fishwick. Computer simulation: The art and science of digital world construction. Available on the Internet from `http://www.cis.ufl.edu/~fishwick/introsim/paper.html`, September 21, 1995.

[FK90] Robert E. Felderman and Leonard Kleinrock. An upper bound on the improvement of asynchronous versus synchronous distributed processing. *Proceedings of the SCS Multiconference on Distributed Simulation*, 22(1):131–136, January 1990.

[FL97] Mingdong Feng and Charles E. Leiserson. Efficient detection of determinacy races in Cilk programs. In *Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'97)*, Newport, Rhode Island, June 22–25, 1997.

[FLR98] Matteo Frigo, Charles E. Leiserson, and Keith H. Randall. The implementation of the Cilk-5 multithreaded language. In *1998 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'98)*, Montreal, Canada, June 17–19, 1998.

[FN92] Richard M. Fujimoto and David M. Nicol. State of the art in parallel simulation. In J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson, editors, *Proceedings of the 1992 Winter Simulation Conference*, pages 246–254, December 1992.

[FTG92] Richard M. Fujimoto, J. Tsai, and G. Gopolakrishnan. Design and evaluation of the rollback chip: Special purpose hardware for time warp. *IEEE Transactions on Computers*, C-41(1):68–82, January 1992.

[Fuj88] Richard M. Fujimoto. Performance measurements of distributed simulation strategies. *Proceedings of the SCS Multiconference on Distributed Simulation*, 19(3):14–20, February 3–5, 1988.

[Fuj89] Richard M. Fujimoto. Time warp on a shared memory multiprocessor. *Transactions Society for Computer Simulation*, 6(3):211–239, 1989.

[Fuj90a] Richard M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, October 1990.

[Fuj90b] Richard M. Fujimoto. Performance of time warp under synthetic workloads. *Proceedings of the SCS Multiconference on Distributed Simulation*, 22(1):23–28, January 1990.

[Fuj93a] Richard M. Fujimoto. Future directions in parallel simulation research. *ORSA Journal on Computing*, 5(3):245–248, Summer 1993.

[Fuj93b] Richard M. Fujimoto. Parallel discrete event simulation: Will the field survive? *ORSA Journal on Computing*, 5(3):213–230, Summer 1993.

[Gaf88] Anat Gafni. Rollback mechanisms for optimistic distributed simulation systems. *Proceedings of the SCS Multiconference on Distributed Simulation*, 19(3):61–67, February 3–5, 1988.

[Gil88] John B. Gilmer, Jr. An assessment of "time warp" prallel discrete event simulation algorithm performance. *Proceedings of the SCS Multiconference on Distributed Simulation*, 19(3):45–49, February 3–5, 1988.

[Goo93] Michael T. Goodrich. Parallel algorithms column 1: Models of computation. *SIGACT News*, 24:16–21, 1993.

[Gun94] Michial A. Gunter. Understanding supercritical speedup. In *Proceedings of 8th Workshop on Parallel and Distributed Simulation (PADS'94)*, pages 81–87, Baltimore, Maryland, July 1994.

[GV94] Alexandros V. Gerbessiotis and Leslie G. Valiant. Direct Bulk-Synchronous Parallel algorithms. *Journal of Parallel and Distributed Computing*, 22(2):251–267, August 1994.

[Ham96] Susanne E. Hambrusch. Models of parallel computation. In *Proceedings of Workshop on Challenges for Parallel Processing*. International Conference on Parallel Processing, 1996.

[Jai91] Raj Jain. *The Art of Computer Systems Performance Analysis: Tehcniques for Experimental Design, Measurement, Simulation, and Modeling.* John Wiley & Sons, 1991.

[JáJ96] Joseph JáJá. On combining technology and theory in search of a parallel computation model. In *Proceedings of Workshop on Challenges for Parallel Processing*, pages 115–123. International Conference on Parallel Processing, 1996.

[Jef85] David R. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, July 1985.

[Joe96] Christopher F. Joerg. *The Cilk System for Parallel Multithreaded Computing.* PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, January 1996.

[Jon89] Douglas W. Jones. Concurrent operations on priority queues. *Communications of the ACM*, 32(1):132–136, January 1989.

[JS85] David Jefferson and H. Sowizral. Fast concurrent simulation using the Time Warp mechanism. *Proceedings of the SCS Multiconference on Distributed Simulation*, 15(2):63–69, January 24–26, 1985.

[KR90] Richard M. Karp and Vijaya Ramachandran. Parallel algorithms for shared-memory machines. In Jan van Leeuwen, editor, *Handbook of Theorectical Computer Science*, volume A, chapter 17, pages 869–941. Elsevier / The MIT Press, 1990.

[KRR96] Jorg Keller, Thomas Rauber, and Bernd Rederlechner. Conservative circuit simulation on shared-memory multiprocessors. In *Proceedings of 10th Workshop on Parallel and Distributed Simulation (PADS'96)*, pages 126–134, Philadelphia, Pennsylvania, May 22–24, 1996.

[Kum89] Devendra Kumar. An approximate method to predict performance of a distributed simulation scheme. In *International Conference on Parallel Processing*, volume 3, pages 259–262, Pennsylvania, August 1989.

[LCUW88] Greg Lomow, John Cleary, Brian Unger, and Darrin West. A performance study of time warp. *Proceedings of the SCS Multiconference on Distributed Simulation*, 19(3):50–55, February 3–5, 1988.

[L'E94] Pierre L'Ecuyer. Uniform random number generation. *Annals of Operations Research*, 53:77–120, 1994.

[LK87] Frank C. H. Lin and Robert M. Keller. The gradient model load balancing method. *IEEE Transactions on Software Engineering*, SE-11(1):32–38, January 1987.

[LK91] Averill M. Law and W. David Kelton. *Simulation Modeling and Analysis.* McGraw-Hill, second edition, 1991.

[LL90] Yi-Bing Lin and Edward D. Lazowska. Optimality considerations of "time warp" parallel simulation. *Proceedings of the SCS Multiconference on Distributed Simulation*, 22(1):29–34, January 1990.

[LLB90] Yi-Bing Lin, Edward D. Lazowska, and Jean-Loup Baer. Conservative parallel simulation for systems with no lookahead prediction. *Proceedings of the SCS Multiconference on Distributed Simulation*, 22(1):144–149, January 1990.

[LLC$^+$98] Chu-Cheow Lim, Yoke-Hean Low, Wentong Cai, Wen Jing Hsu, Shell Ying Huang, and Stephen J. Turner. An empirical comparison of runtime systems for conservative parallel simulation. In *2nd Workshop on Runtime Systems for Parallel Programming (RTSPP'98)*, Orlando, Florida, March 30, 1998.

[LM90] Richard J. Lipton and David W. Mizell. Time Warp vs. Chandy-Misra: A worst-case comparison. *Proceedings of the SCS Multiconference on Distributed Simulation*, 22(1):137–143, January 1990.

[LP91] Yi-Bing Lin and Bruno R. Preiss. Optimal memory management for time warp parallel simulation. *ACM Transactions on Modeling and Computer Simulation*, 1(4):283–307, October 1991.

[LSW89] Boris Lubachevsky, Adam Shwartz, and Alan Weiss. Rollback sometimes works ... if filtered. In E. A. MacNair, K. J. Musselman, and P. Heidelberger, editors, *Proceedings of the 1989 Winter Simulation Conference*, pages 630–639, 1989.

[LT90] L. Z. Liu and C. Tropper. Local deadlock detection in distributed simulations. *Proceedings of the SCS Multiconference on Distributed Simulation*, 22(1):64–69, January 1990.

[Lub88] Boris D. Lubachevsky. Bounded lag distributed event simulation. *Proceedings of the SCS Multiconference on Distributed Simulation*, 19(3):183–190, February 3–5, 1988.

[Lub89] Boris D. Lubachevsky. Efficient distributed event-driven simulations of multiple-loop networks. *Communications of the ACM*, 32(1):111–123, January 1989.

[Mad88] Vijay Madisetti. WOLF: A rollback algorithm for optimistic distributed simulation systems. In M. Abrams, P. Haigh, and J. Comfort, editors, *Proceedings of the 1988 Winter Simulation Conference*, pages 296–305, San Diego, California, 1988.

[Mar97a] Mauricio Marín. Billiards and related systems on the bulk-synchronous parallel model. In *Proceedings of 11th Workshop on Parallel and Distributed Simulation (PADS'97)*, pages 164–171, Lockenhaus, Austria, June 10–13, 1997.

[Mar97b] Mauricio Marín. Direct BSP algorithms for parallel discrete-event simulation. Technical Report PRG-TR-8-97, Oxford University, January 1997.

[Mar98a] Mauricio Marín. Asynchronous (time-warp) versus synchronous (event-horizon) simulation time advance in BSP. Technical report, Oxford University, February 1998. To appear in Euro-Par'98.

[Mar98b] Mauricio Marín. An empirical assessment of optimistic PDES on BSP. Technical report, Oxford University, February 1998. Draft.

[Mar98c] Mauricio Marín. Time warp on BSP computers. Technical report, Oxford University, February 1998. To appear in 12th European Simulation Multiconference.

[Mat93] Friedemann Mattern. Efficient algorithms for distributed snapshots and global virtual time approximation. *Journal of Parallel and Distributed Computing*, 18(4):423–434, August 1993.

[McC93] W. F. McColl. General purpose parallel computing. In A. M. Gibbons and P. Spirakis, editors, *Lectures on Parallel Computation. Proceedings of 1991 ALCOM Spring School on Parallel Computation*, volume 4, pages 337–391. Cambridge University Press, 1993.

[McC94] W. F. McColl. BSP programming. In G. Blelloch, M. Chandy, and S. Jagannathan, editors, *Proceedings of DIMACS Workshop on Specification of Parallel Algorithms*, Princeton, May 1994. American Mathematical Society.

[Mis86] Jayadev Misra. Distributed discrete-event simulation. *ACM Computing Surveys*, 18(1):39–65, March 1986.

[MIT98] MIT Laboratory for Computer Science. *Cilk-5.2 Reference Manual*, July 21, 1998. Available on the Internet from `http://theory.lcs.mit.edu/~cilk`.

[MMT95] B. M. Maggs, L. R. Matheson, and R. E. Tarjan. Models of parallel computation: A survey and synthesis. In *Proceedings of the 28th Hawaii International Conference on System Sciences (HICSS)*, volume 2, pages 61–70, January 1995.

[MRR90] B. C. Merrifield, S. B. Richardson, and J. B. G. Roberts. Quantitative studies of discrete event simulation modelling of road traffic. *Proceedings of the SCS Multiconference on Distributed Simulation*, 22(1):188–193, January 1990.

[Nev90] Chris Nevison. Parallel simulation of manufacturing systems: Structural factors. *Proceedings of the SCS Multiconference on Distributed Simulation*, 22(1):17–19, January 1990.

[NF94] David M. Nicol and Richard M. Fujimoto. Parallel simulation today. *Annals of Operations Research*, 53:249–285, 1994.

[Nic88] David M. Nicol. Parallel discrete-event simulation of FCFS stochastic queueing networks. *ACM SIGPLAN Notices Proceedings of PPEALS, New Haven, Connecticut*, 23(9):124–137, 1988.

[NR84] David M. Nicol and Paul F. Reynolds, Jr. Problem oriented protocol design. In *Proceedings of 1984 Winter Simulation Conference*, pages 471–474, December 1984.

[NR90] David M. Nicol and Paul F. Reynolds, Jr. Optimal dynamic remapping of data parallel computations. *IEEE Transactions on Computers*, C-39(2):206–219, February 1990.

[NXG85] Lionel M. Ni, Chong-Wei Xu, and Thomas B. Gendreau. A distributed drafting algorithm for load balancing. *IEEE Transactions on Software Engineering*, SE-11(10):1153–1161, October 1985.

[PL95] Bruno R. Preiss and Wayne M. Loucks. Memory management techniques for time warp on a distributed memory machine. In *Proceedings of 9th Workshop on Parallel and Distributed Simulation (PADS'95)*, pages 30–39, Lake Placid, New York, June 14–16, 1995.

[PN98] Anna L. Poplawski and David M. Nicol. Nops: A conservative parallel simulation engine for TeD. In *Proceedings of 12th Workshop on Parallel and Distributed Simulation (PADS'98)*, Banff, Alberta, Canada, May 26–29, 1998.

[PS91] A. Prakash and R. Subramanian. Filter: An algorithm for reducing cascaded rollbacks in optimistic distributed simulations. In *Proceedings of the 24th Annual Simulation Symposium*, pages 123–132, 1991.

[RAFD93] Robert Rönngren, Rassul Ayani, Richard M. Fujimoto, and Samir R. Das. Efficient implementation of event sets in time warp. In *Proceedings of 7th Workshop on Parallel and Distributed Simulation (PADS'93)*, pages 101–108, San Diego, California, May 1993.

[Ran98] Keith H. Randall. *Cilk: Efficient Multithreaded Computing.* PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 1998.

[Rey88] Paul F. Reynolds, Jr. A spectrum of options for parallel simulation. In M. Abrams, P. Haigh, and J. Comfort, editors, *Proceedings of the 1988 Winter Simulation Conference*, pages 325–332, San Diego, California, 1988.

[RFBJ90] Peter Reiher, Richard M. Fujimoto, Steven Bellenot, and David Jefferson. Cancellation strategies in optimistic execution systems. *Proceedings of the SCS Multiconference on Distributed Simulation*, 22(1):112–121, January 1990.

[RMM88] Daniel A. Reed, Allen D. Malony, and Bradley D. McCredie. Parallel discrete event simulation using shared memory. *IEEE Transactions on Software Engineering*, SE-14(4):541–553, April 1988.

[RW89] Rhonda Righter and Jean C. Walrand. Distributed simulation of discrete event systems. *Proceedings of the IEEE*, 77(1):99–113, January 1989.

[RWJ89] Peter L. Reiher, Frederick Wieland, and David Jefferson. Limitation of optimism in the time warp operating system. In E. A. MacNair, K. J. Musselman, and P. Heidelberger, editors, *Proceedings of the 1989 Winter Simulation Conference*, pages 765–770, 1989.

[SBW88] Lisa M. Sokol, Duke P. Briscoe, and Alexis P. Wieland. MTW: A strategy for scheduling discrete simulation events for concurrent execution. *Proceedings of the SCS Multiconference on Distributed Simulation*, 19(3):34–42, February 3–5, 1988.

[SLWN95] Jeffrey S. Steinman, Craig A. Lee, Linda F. Wilson, and David M. Nicol. Global virtual time and distributed synchronization. In *Proceedings of 9th Workshop on Parallel and Distributed Simulation (PADS'95)*, pages 139–148, Lake Placid, New York, June 14–16, 1995.

[SS90] Lisa M. Sokol and Brian K. Stucky. MTW: Experimental results for a constrained optimistic scheduling paradigm. *Proceedings of the SCS Multiconference on Distributed Simulation*, 22(1):169–173, January 1990.

[Ste92] Jeffrey S. Steinman. Speedes: A multiple-synchronization environment for parallel discrete-event simulation. *International Journal in Computer Simulation*, 2:251–286, 1992.

[Ste93] Jeff S. Steinman. Breathing time warp. In *Proceedings of 7th Workshop on Parallel and Distributed Simulation (PADS'93)*, pages 109–118, San Diego, California, May 1993.

[Ste94] Jeff S. Steinman. Discrete-event simulation and the event horizon. In *Proceedings of 8th Workshop on Parallel and Distributed Simulation (PADS'94)*, pages 39–49, Baltimore, Maryland, July 1994.

[Ste96] Jeffrey S. Steinman. Discrete-event simulation and the event horizon part 2: Event list management. In *Proceedings of 10th Workshop on Parallel and Distributed Simulation (PADS'96)*, pages 170–178, Philadelphia, Pennsylvania, May 22–24, 1996. Computer Society Press.

[Val90] Leslie G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, August 1990.

[VCW94] George Varhese, Roger Chamberlain, and William E. Weihl. The pessimism behind optimistic simulation. In *Proceedings of 8th Workshop on Parallel and Distributed Simulation (PADS'94)*, pages 126–131, Baltimore, Maryland, July 1994.

[VCW95] George Varghese, Roger Chamberlain, and William E. Weihl. Deriving global virtual time algorithms from conservative simulation protocols. *Information Processing Letters*, 54(2):121–126, April 28, 1995.

[WH95] Yung-Chang Wong and Shu-Yuen Hwang. Prediction of memory consumption in conservative parallel simulation. In *Proceedings of 9th Workshop on Parallel and Distributed Simulation (PADS'95)*, pages 199–202, Lake Placid, New York, June 14–16, 1995.

[Win92] A. J. Wing. Discrete event simulation in parallel. In Lydia Kronsjö and Dean Shumsheruddin, editors, *Advances in Parallel Algorithms*, Advanced Topics in Computer Science Series, chapter 7, pages 179–226. Blackwell Scientific Publications, 1992.

[WL89] David B. Wagner and Edward D. Lazowska. Parallel simulation of queueing networks: Limitations and potentials. In *Proceedings of 1989 ACM SIGMETRICS and PERFORMANCE '89*, volume 17(1), pages 146–155, May 1989.

[XGUC95] Z. Xiao, F. Gomes, B. Unger, and J. Cleary. A fast asynchronous GVT algorithm for shared memory multiprocessor architectures. In *Proceedings of 9th Workshop on Parallel and Distributed Simulation (PADS'95)*, pages 203–208, Lake Placid, New York, June 14–16, 1995.