Proceedings of the 1988 Winter Simulation Conference
M. Abrams, P. Haigh, and J. Comfort (eds.)

A spectrum of options for parallel simulation

Paul F. Reynolds, Jr.
Department of Computer Science and
Institute for Parallel Computation
The University of Virginia
Charlottesville, Virginia 22903

ABSTRACT

Conventional wisdom has it there are two basic approaches to parallel simulation: *conservative* (Chandy-Misra) and *optimistic* (time warp). All known protocols are thought to fall into one of these two classes. This dichotomy is false. There exists a spectrum of options that includes these approaches. We describe a design space that admits these as alternatives, we show how most of the well known parallel simulation approaches can be derived using our design alternatives, and we explore the implications of the existence of the design space we describe. In particular, we note there are *many* as yet unexplored approaches to parallel simulation.

INTRODUCTION

Parallel simulation, generally called *distributed simulation* in the literature, is concerned with the parallel execution of discrete event simulations. Beginning with the research of Chandy and Misra [ChMi79] and Peacock et al. [PeWo79], a number of approaches have been described for coordinating cooperating processes so that the outcome of a parallel simulation is the same as would occur in a more conventional sequential simulation. Algorithms are *conservative* if they satisfy the property that no process receives information from any other process that predates the current simulation time of the receiving process. They are *optimistic* if processes can act on incomplete information, thus admitting the case where messages may arrive "in the past." Optimistic methods have typically required that some sort of rollback mechanism exist to allow for repair of incorrectly sequenced events. A survey of parallel simulation appeared in [Misr86].

Examples of additional approaches generally considered conservative include the blocking table algorithm [PeMa80], deadlock detection [ChMi81], SRADS [Reyn82], appointments [NiRe84], feed-forward [Kuma86], conditional knowledge [ChMi87] and bounded lag [Luba87]. The optimistic approach has its foundation in the time warp method [JeSo82]. Others have explored variations on the optimistic approach including [Jeff85] and [Soko88]. As we shall see below, SRADS and moving time window [Soko88] have features that bridge these characterizations. A partial chronology of approaches to parallel simulation is shown in Figure 1.

Parallel simulation was originally named distributed simulation in the early Chandy and Misra paper [ChMi79]. More recently distributed simulation has come to be associated with geographically distributed simulations, for example the National Testbed [Word88]. Without attempting to establish formal definitions here, we distinguish distributed simulations from parallel simulations on the basis of inter-process communication times and goals for employing multiple processors. Distributed simulations tend to incur communication delays on the order of seconds, and they tend to be employed for the purpose of bringing physically separated resources together for simulation purposes. Distributed simulations may not be concerned with processor utilization or minimum finishing time, although real-time requirements may bring these into play. Parallel simulation, on the other hand, has been studied primarily for the purpose of maximizing processor utilization and/or minimizing simulation finishing time. Inter-processor communication times have generally been presumed to be relatively small (e.g. on the order of milliseconds). Parallel simulation research has not been concerned with real-time issues, human-in-the-loop or hardware-in-the-loop.

Whether a simulation is discrete event or timestepped is another important consideration. Parallel simulation research has been concerned with discrete event simulations. Parallel timestepped simulation assumes that simulation time advances in (generally) fixed increments, where the chosen increment is sufficiently small that no two significant events that should occur at distinct simulation times are simulated in the same time interval. Parallel discrete event simulation is suitable for simulations which incorporate event sequences that occur on widely disparate time and/or space scales.

There are times when timestepped simulations are appropriate. This includes cases where the generally low overhead of the timestepped approach may make it a better method than an equivalent discrete event simulation. Discrete event simulations are clearly more appropriate in cases where a timestepped approach would have many intervals where no significant events occurred. The introduction of multiple processors may affect relative performance, as may inter-processor communication times and the practicality of attempting to synchronize processors.

We address both parallel discrete event simulations and parallel timestepped simulations here. Our primary goal is to establish the fact that, contrary to prevailing beliefs, there is a spectrum of possible methods. We do not address the important issues relating to choosing among methods, including the appropriateness of one method over the other when the simulation becomes more distributed in nature. Also, we do not address the impact of attempting to employ parallel simulation in a distributed simulation environment, although that is currently a pressing issue.

In the following sections we describe the two traditional approaches to parallel discrete event simulation and we discuss briefly the behavioral properties of each. Then we show that a different view of the design space allows for an infinite variety of options, including adaptive methods. We show how many of the well known parallel simulation approaches can be derived using our design alternatives, and we explore the implications of the existence of the design space we describe.

DEFINITIONS

We assume there is a mapping from the *physical processes* (PP's) being simulated to the *logical processes* (LP's) that represent them in a simulation. For example, in a factory simulation, physical processes could be workstations and logical processes could be direct representations of the workstations. In this case, *events* would be related to job completion and job arrival at independent workstations (LP's).

Discrete event simulations are implemented using a *central events list* and a *logical clock*. The central events list is generally

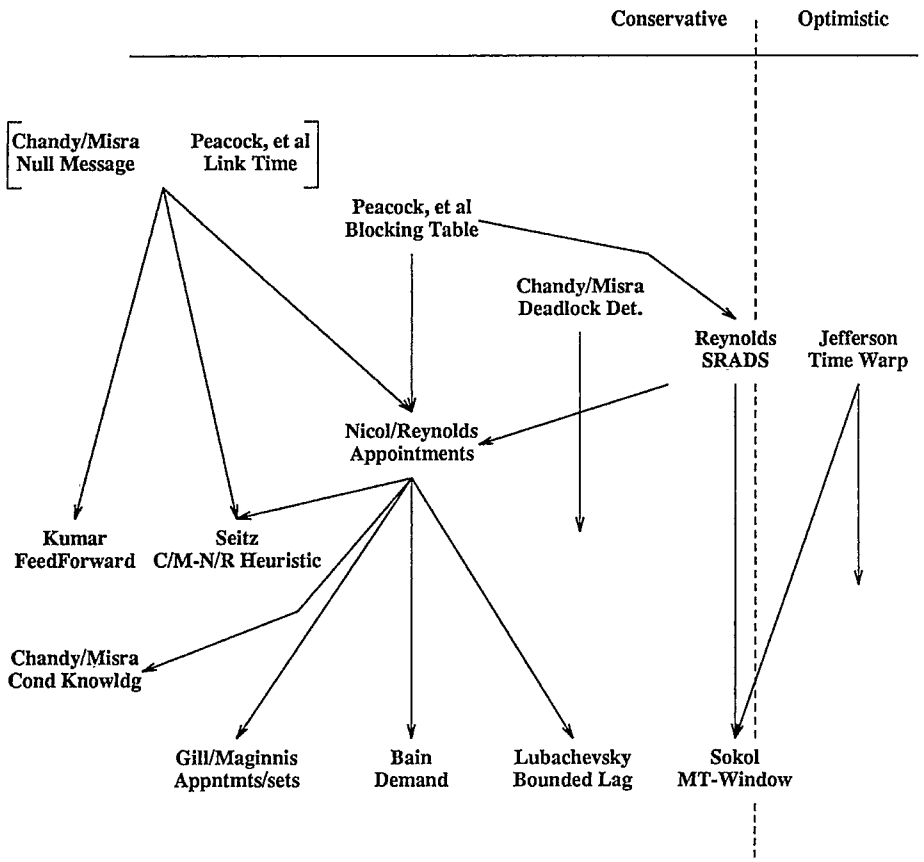

Figure 1. A Partial Chronology of Parallel Simulation

an ordered list from which the next event to be simulated can be selected from the head. Ordering of events in the list is determined by the *logical time* at which they will occur. Logical time in a discrete event simulation is determined by event times. Each time an event is selected from the central events list the logical clock is set to the time at which that event is to occur in the simulation. Thus, for any given value of the logical clock, the central events list is a list of known future events *at that logical time.* The simulation of events has the effect of advancing logical time and changing the contents of the central events list.

Parallel simulation assumes the placement of LP's on multiple processors, thus requiring that LP's have some means of communicating. Placement of LP's on different processors also requires that separate events lists and separate logical clocks be created for each LP (or each cluster of LP's) on a processor. Parallel discrete event simulation research has focused primarily on the issue of coordinating the asynchronous advancement of logical clocks so that the potential outcomes of a simulation are identical to those possible for a sequential counterpart.

LP's can pass messages to each other representing either event or non-event information. Messages contain *timestamps* representing the logical time at which the message was sent. That is, if one LP sends a message to a second then the timestamp in the message is the logical time of the sending LP. *Event messages* carry information that can be related to events in the physical system being simulated. *Non-event messages* are all other messages. We note that messages can be simulated in environments where message passing is not required. That is, message passing is a part of an assumed logical model that may not be required in an actual implementation.

We denote the i^{th} LP as LP_i and its logical clock as LC_i. On occasion we refer to LP_i's *simulation time*. We mean the value of its logical clock, LC_i.

CONSERVATIVE PARALLEL DISCRETE EVENT SIMULATIONS

Conservative approaches are generally regarded as methods that ensure the following conservative rule:

> **CR:**$\forall$ i,j guarantee that LP_j never receives a message from LP_i such that the message timestamp (representing LC_i at the time the message was sent) is less than LC_j.

In other words, an LP should never receive a message in its logical past.

There are a number of ways to guarantee this. The first is to require that an LP have event messages from each of the LP's that could ever send it an event message. The receiving LP needs only to choose the pending message with the smallest timestamp to proceed correctly. Unfortunately, it is easy to demonstrate that this simple approach could deadlock even though the underlying physical system would not have deadlocked, i.e. deadlock is an artifact of the simulation protocol.

Various approaches have come about as a result of attempting

to avoid the deadlock problem. The most straight-forward extension was that proposed by Chandy and Misra [ChMi81] that employed distributed deadlock detection to identify the LP that had enough knowledge (unbeknownst to it) to proceed while not violating CR.

The null message [ChMi79] and link time [PeWo79] protocols employed non-event messages to prevent deadlock. It is known that non-event messages generated in this manner can proliferate and essentially choke the simulation. (However, this is not always the case.) Studies by Reed [ReMa87, ReMa88] indicate that the null message approach is not suitable for the simulation of queuing networks, although generalization of this result is tenuous. There has been some question about flexibility assumptions in these studies as well [Chan88].

An alternate method, first proposed in [PeMa80] and later expanded in [Reyn82] uses a "demand" approach, where blocked LP's use non-event messages to acquire information from other LP's, so that they may advance their simulation time (LC) without violating CR.

A mix of these approaches led to the use of simulation-dependent information in order to enhance methods for advancing LP's simulation times. Nicol and Reynolds first employed this strategy in their appointment protocol [NiRe84]. More recently, Bain [Bain88] has proposed variants on this approach. Chandy and Misra have defined a framework based on conditional and unconditional knowledge for reasoning about how simulations can advance simulation time effectively [ChMi87].

Some conservative methods rely on communication topology to ensure CR and deadlock freedom. The feed-forward approach described in [Kuma86] assumes that networks can have forks and joins but no cycles. This assumption is sufficient to demonstrate deadlock freedom for a conservative protocol.

In terms of the variables defined below, conservative methods have traditionally been non-aggressive (SRADS [Reyn82] and a variation of moving time window [Soko88] being noted exceptions), accurate, and without risk. They have employed knowledge embedding, knowledge acquisition and knowledge dissemination. They have generally been asynchronous and none has employed adaptability in a dynamic sense.

OPTIMISTIC PARALLEL DISCRETE EVENT SIMULATIONS

Optimistic approaches relax CR somewhat, guaranteeing instead the following optimistic requirement:

> **OR:** $\forall$ j guarantee that LP_j's state can be restored to a simulation time not exceeding time t_i whenever a message with timestamp t_i arrives in LP_j's logical past (i.e. t_i is less than LC_j).

From another point of view, optimistic methods will act on conditional knowledge. If an event message is awaiting processing by LP_j, then LP_j will process it, even though a message with a smaller timestamp may arrive at a later time.

Optimistic methods, most notably, time warp [JeSo82] and its variants, assume that rollback can be employed in order to guarantee that ultimately the outcome of the parallel simulation matches the outcome of an equivalent sequential counterpart (with allowances for non-determinism in the parallel simulation). This assumption requires that state space be saved periodically (i.e. checkpointing) so that rollbacks can be performed. Optimistic methods have traditionally allowed LP's to pass the outcome of their optimistic assumptions on to succeeding LP's. As a result a rollback at one LP can cause a rollback at other LP's to occur as well.

In terms of the variables defined below, optimistic methods have traditionally been aggressive, accurate and with risk. They have not employed knowledge embedding or knowledge acquisition. They have employed knowledge dissemination in the form of anti-messages. They have been asynchronous and non-adaptable.

EXPANDED HORIZONS

We describe a set of design variables that define a vastly expanded design space. A major goal has been to make the set of variables both comprehensive and orthogonal. If we have failed to be comprehensive, it only makes our central thesis stronger.

At a system level, one design variable is the option to partition the LP's in a simulation so that clusters of them employ different strategies. This is the only system-level design variable we have identified, the rest being applicable at the LP level. We define the partitioning design variable as follows:

> **DV.0: Partitioning** - Determine clusters of LP's based on distinct sets of design variable bindings.

For a partition $P = \{p_1, p_2, ..., p_m\}$, where each p_i represents a non-empty set of LP's, m = 1 means all LP's employ the same set of options. Typically, this is what has been proposed in parallel simulation protocols. We may choose m > 1 in order to match strategies to behavioral properties of LP's. For example, it may be wise to apply an aggressive strategy (see below) to a portion of a simulation and a non-aggressive strategy to the rest. This has been minor experimentation with partitioning, as we discuss later, but no method has has had partitioning as a major design goal from the outset.

The remaining design variables can be viewed as LP-level design decisions. The first of these is *adaptability*, which, as its name implies, suggests that an LP may change design value bindings dynamically. This, in turn, can affect the partitioning. We define adaptability as follows:

> **DV.1: Adaptability** - Changing design variable bindings based on knowledge of selected aspects of the simulation state.

An LP, in the course of a simulation, may have information that would cause it to change to a different set of bindings for design variables, including ceasing to be adaptable. This design variable, as well as some others, immediately offers us an infinite set of options since adaptations can be based on functionally continuous data.

Adaptability has not appeared in the parallel simulation literature before now. This is somewhat odd because it is akin to dynamic load balancing, a topic that has been studied extensively in parallel systems. For example, a simulation employing an optimistic strategy (actually, an aggressive, accurate strategy with risk; see below) may, on the basis of information about the number of rollbacks caused by inputs from a given LP, reduce or terminate its aggressiveness with respect to inputs from that LP. As with dynamic load balancing, this could be done in order to attempt to minimize processor finishing time.

We define *aggressiveness* as follows:

> **DV.2: Aggressiveness** - Processing messages based on conditional knowledge; that is, relaxing the requirement that messages be processed in a strict monotonic order with respect to message times.

Aggressiveness allows messages to be processed when it is still possible that a message with a timestamp that predates the timestamp of the processed message may arrive later. Aggressiveness is the most visible aspect of Jefferson and Sowizral's optimistic approach: time warp [JeSo82]. Time warp and optimism have come to be equated with maximal aggressiveness: if a message arrives and an LP has no other events to process, process the message.

We mention, without elaboration, that there appear to be at least two interesting forms of aggressiveness: initiating and transfer. We expand on this after defining the design variable, *risk*.

In contrast with aggressiveness, which allows out-of-sequence processing of events, we define next (a compatible concept) *accuracy*, which addresses the *ultimate* sequencing of events in a simulation:

> **DV.3: Accuracy** - Requiring that events within LP's *ultimately* be processed in a correct (monotonic) sequence.

This definition would require, in most cases, that the set of all final states for a given parallel simulation be equivalent to the set of all final states for its sequential counterpart. If no two events in a simulation can have the same simulation time then this should be the case.

There is no requirement that parallel simulation protocols always meet the accuracy definition. To date, only SRADS [Reyn82] and moving time window [Soko88] have allowed for inaccuracy to occur, although even in these cases the potential for inaccuracy is regarded as something to be avoided through other actions. However, there are some indications that a strategy employing controlled inaccuracies may produce useful results relatively efficiently [Theo84].

It is important to distinguish accuracy from aggressiveness. Accuracy requires that *ultimately* all events are processed, *or have the effect of having been processed*, at the simulation time at which they "should" be processed. Aggressiveness does not really address this issue.

To further aid in making the distinction between accuracy and aggressiveness, it is worth noting that purely optimistic approaches are both aggressive and accurate. Despite their aggressiveness, they still guarantee accuracy by ensuring that LP's will back up, as necessary, to guarantee that events are processed when they "should" be (i.e. LP's simulation times match the times of the events they process). There is no requirement that this be the case.

The design variable that rounds out the variables contributing to optimistic approaches is *risk* which we define as follows:

> **DV.4: Risk** - Passing messages which have been processed based on aggressive or inaccurate processing assumptions in an LP.

It is entirely possible that an LP could utilize what would otherwise be idle time through aggressive or inaccurate processing *without passing potentially (or known) inaccurate or out-of-sequence messages to other LP's*. Risk is that design variable that allows inaccurate or out-of-sequence messages to be passed on. A design decision that employs aggressiveness but no risk guarantees that all rollbacks are strictly local. Thus, there would be no need for techniques such as anti-messages [Jeff85] to cope with situations where rollbacks are not guaranteed to be contained locally. As is always the case, the impact of the choice made is problem dependent.

We define *risk messages* as messages that are the product of actions taken based on incomplete (conditional; see [ChMi87]) knowledge, or as a result of processing that leads to the transmission of out of order messages. Recall that messages can be events or information about the simulation state. Inaccurate or aggressive processing coupled with risk can create or pass along risk messages.

We must distinguish situations in which LP's receive risk messages from those in which an LP is *at risk*. If an LP receives risk messages it is at risk. However, we extend this definition to the following case: an empty input queue contains implicitly the message "do nothing from now to time infinity." We define an LP as being *at risk* if there exists a risk message at the head of at least one of its input queues or if there is no message in at least one of its input queues. We note that withholding knowledge increases the probability of putting an LP at risk. Also, non-aggressive, accurate methods can and are likely to put LP's at risk because it is very likely that LP's will not have pending messages from all LP's that may send them messages each time the (receiving) LP is ready to process a new message.

Optimistic approaches, as traditionally defined in the simulation literature, are aggressive, accurate methods that employ risk. The degree to which LP's can be at risk can vary. For example, time warp with cancellation [Jeff85], [LCUW88] is more likely to put LP's at risk than conventional time warp. We discuss this in more detail in the next section.

We note, as an aside, that there appear to be two interesting attributes of aggressiveness and risk. In both cases it is interesting to distinguish between initiating and transferring forms. Initiating aggressiveness occurs when an at risk LP processes a non-risk message. Transfer aggressiveness occurs when an LP processes a risk message (implying the LP is at risk). Similarly, an LP initiates risk if it passes a message created from initiating aggressiveness, and it transfers risk if it passes a message created from transferring aggressiveness. A parallel simulation that limits pairwise aggressiveness and risk between LP's along these dimensions could be quite practical. The limiting could be done based on the presence of forks, joins and cycles in the LP connection topology so that, for example, every cycle contains at least one LP that does not initiate aggressiveness with any other LP in the cycle. We are exploring the utility of this idea.

A slight extension of the meaning of initiating risk creates another interesting possibility for a simulation protocol. Consider including the case where an LP holds back knowledge for whatever purpose, e.g. in order to reduce rollbacks incurred by an overly aggressive successor. Our definition of risk admits this because a lack of knowledge creates an at risk situation.

The following design variables pertain to the sharing of knowledge about the state of a simulation. Both event and non-event messages can be used to carry this knowledge. Knowledge can be made available through means such as modifying a simulation to provide it, or extraction of pertinent knowledge from a simulation designer or user. As an alternative, it can be gathered by a simulation protocol without direct input from the simulation or the simulation author or user. The next design variable addresses this:

> **DV.5: Knowledge embedding** - Knowledge about LPs' behavioral attributes is embedded in the simulation.

Chandy and Misra [ChMi87] have recently advocated embedding knowledge about a simulation into the simulation itself in order to make it perform more efficiently. For example, a case that poses performance problems for non-aggressive protocols is one where two LP's each have only conditional knowledge about the state of the other, but collectively, the conditional knowledge is

sufficient to generate useful unconditional knowledge. This unconditional knowledge, in turn, could lead to vastly improved performance through the reduction of non-event message exchanges.

Use of embedded knowledge prevents a simulation protocol from being fully general. The benefits and limitations of using embedded knowledge have not been fully explored. An alternative is for a simulation protocol to attempt to determine critical behavioral properties through dynamic analysis. This could be done, for example, by performing execution-time analysis leading to an estimation of chosen properties. While such an approach may allow for a general (black box) approach, costs may outweigh benefits. It is clear that further investigation is required in this area.

Knowledge can be embedded in the simulation and/or in the simulation protocol. Knowledge embedding in the simulation would most likely be done either manually or with automated support systems. This approach is likely to lead to inflexibility because changes to the simulation could lead to changes in the code representing embedded knowledge. Embedding knowledge in a protocol would most likely be done through parameterization and/or automated means. This approach could be more flexible but may not exploit embedded knowledge as efficiently. Where to embed knowledge and how to embed it are open issues.

Embedded knowledge or dynamic analysis may allow an LP to determine its own future behavior or the future behavior of other LP's. Acquisition and dissemination of knowledge are important. The next two design variables address knowledge acquisition and dissemination.

DV.6: Knowledge dissemination - LP's initiate the transmission of knowledge to other LP's.

Knowledge dissemination occurs when an LP sends messages to other LP's that may create the opportunity for them to make progress. Knowledge dissemination appears in a number of protocols, including Chandy and Misra's null messages [ChMi79], Peacock, et al.'s link time algorithm [PeWo79], Nicol and Reynolds' appointments [NiRe84], and others. It can also appear in optimistic methods, e.g. in the form of anti-messages.

We distinguish between two important approaches to knowledge dissemination. The null message approach has a prescribed method for generating non-event messages: each time an LP sends a message along one output path, it sends non-event (null) messages with the same timestamp along all other paths. This approach requires no embedded knowledge. The appointment approach, in contrast, requires LP's to generate future possible communication times based on an internal analysis of the LP's current state. An LP must be programmed to analyze all possible future internal events in order for this strategy to work.

One problem with approaches that rely strictly on dissemination is that they can lead to the computation and transmission of knowledge that is not used. The use of knowledge acquisition can alleviate this problem.

DV.7: Knowledge acquisition - LP's initiate requests for knowledge from other LP's.

Strategies employing knowledge acquisition are sometimes known as "demand" methods. Here, LP's send out requests for simulation information so that more knowledgeable decisions can be made about pending input messages. We note that knowledge acquisition and knowledge dissemination are orthogonal in that either can be used to any degree independent of the use of the other.

The next design variable captures the spectrum of options that admits both timestepped and discrete event parallel simulation.

DV.8: Synchrony - Degree of temporal binding among LP's.

How much is the temporal progress of individual LP's controlled by the simulation? At one extreme LP's may advance simulation time one unit at a time, in a SIMD-like manner, with no LP's going forward until all have completed the previous step. At the other extreme LP's could advance simulation time independently, with the most likely first constraint being a concern for accuracy. Fox [FoJo88] has suggested three basic categorizations: synchronous, loosely synchronous and asynchronous. Loosely synchronous characterizes LP's that work independently for a short duration, synchronize, work independently, etc.

The synchrony design variable allows us to capture timestepped simulation (loosely synchronous) as well as all of the major proposed methods (asynchronous). It is important to regard timestepped parallel simulations as members of the set of possible parallel simulation strategies; this new design variable is sufficient to do that.

KNOWN POINTS IN THE DESIGN SPACE

We characterize briefly the major approaches to parallel simulation based on the design variables we have just defined.

We have summarized the classification of major approaches to parallel simulation in the table in figure 2. We find the best view of this table is a macroscopic one. For example, no approach has proposed adaptability. Partitioning has barely been explored. It was first proposed in [NiRE84], and an implementation combining null messages and timestepped simulation was reported in [HaDo88]. In both cases partitioning was accomplished through the combination of two existing approaches. The table reflects the fact that no protocol has been designed from the outset with partitioning as a primary goal. Relaxation of the accuracy requirement appears in SRADS and Moving Time Window. The following notes apply to the subscripts in the table.

Note-1: The Chandy/Misra deadlock detection algorithm cycles through two phases: 1) process until deadlock and 2) detect and repair deadlock.

Note-2: SRADS is accurate, non-aggressive and without risk only if the embedded knowledge is sufficient that acquisition occurs at correct times.

Note-3: Reader appointments uses acquisition, Writer appointments uses dissemination.

Note-4: Moving time window has three repair options when inaccuracy occurs: rollback (i.e. time warp), historical records and do nothing (i.e. SRADS).

We add that the design variables are not binary. For example, adaptability has an infinite number of options on which to base decisions affecting adaptations. Similarly, knowledge dissemination has a large number of possible implementations.

Optimistic methods such as time warp [JeSo82] are aggressive, accurate and with risk. They avoid the use of embedded knowledge; in fact lack of embedded knowledge is regarded as a strength. Knowledge acquisition is not employed. Knowledge dissemination is employed in the form of anti-messages. The degree of risk varies. The variation that delays sending anti-messages [Jeff85] places successors at greater risk because they are more likely to process messages that must be rolled back.

Null messages [ChMi79] are non-aggressive, accurate and without risk. They do not use knowledge embedding or knowledge acquisition. Knowledge dissemination is employed in the form of null messages.

Blocking table [PeMa80] is non-aggressive, accurate and

	partit-ioning	adapta-bility	aggress-iveness	accu-racy	risk	Knldg Embed	Knldg Dissem	Knldg Acquis	Synch-rony
Null Msgs (Chandy)				Y			Y		A
Blocking Tbl (Peacock)				Y				Y	A
Ddlk Detect (Chandy)				Y			Y		LS^1
Time Warp (Jefferson)			Y	Y	Y		Y		A
SRADS (Reynolds)			Y/N^2	Y/N^2	Y/N^2	Y		Y	A
Appointments (Nicol)				Y		Y	Y/N^3	Y/N^3	A
MT-Window (Sokol)			Y/N^4	Y/N^4	Y/N^4	Y/N^4	Y		A
Timestep				Y		Y	Y		LS

Figure 2. Categorizing Known Techniques.

without risk. Knowledge embedding and knowledge dissemination are not employed. Knowledge acquisition is employed in that LP's request simulation times from predecessors in order to increase knowledge about the future.

The deadlock detection approach of [ChMi81] is non-aggressive, accurate and without risk. Knowledge embedding is not required. Knowledge dissemination and/or acquisition are employed when necessary to perform distributed deadlock detection and resolution.

SRADS [Reyn82] is aggressive, potentially inaccurate and potentially with risk. Knowledge embedding is employed. Knowledge dissemination is not, but knowledge acquisition is. SRADS assumes that acquisition can be performed at periodic intervals such that accuracy is guaranteed, or that inaccuracies can be controlled by adjusting acquisition intervals. Identification of acquisition intervals is what makes SRADS rely on embedded knowledge.

Appointments [NiRe84] are non-aggressive, accurate and without risk. Knowledge embedding is employed extensively. Two approaches have been proposed, one employing knowledge acquisition and the other employing knowledge dissemination. The more recent protocol of Bain [Bain88] is similar. In [Nico88] Nicol discusses an implementation for a combined approach, employing both knowledge acquisition and knowledge dissemination.

Moving time window [Soko88] has characteristics that depend on repairs made when out of sequence messages arrive. In essence the method processes messages aggressively within bounds. If no rollback occurs then the method becomes potentially inaccurate and at risk. In this sense it is akin to the SRADS protocol [REYN82]. If rollback is employed then it is accurate, in which case it is akin to time warp [JeSo82].

Conditional knowledge [ChMi87] is a loosely formed approach that advocates the use of embedded knowledge to advance the determination of unconditional knowledge. As presented, it is an accurate approach. Aggressiveness and risk are options, as are knowledge acquisition and dissemination.

NEW POINTS IN THE DESIGN SPACE

We have established that the design space for parallel simulation is essentially infinite. Many of the design variables have, effectively, an infinite set of options. That means we have only begun to explore the potential alternatives for performing parallel simulation. We explore two new alternatives for the purpose of demonstrating that the taxonomy developed here does more than simply capture existing methods.

Method: Consider a method, as shown in Figure 3, that is aggressive, accurate and without risk. Settings of other design variables are important but not of primary concern to us here. This technique would process inputs as they became available but would not pass event messages unless input information was sufficient to guarantee that output messages posed no risk. Rollbacks would occur only locally, making this approach much less costly, in terms of space at least, than traditional optimistic approaches.

Variations on this method can be constructed by considering other design variable bindings. For example, the employment of knowledge acquisition or knowledge dissemination could lead to a significantly improved aggressive strategy, i.e. aggressiveness could be tempered by knowledge acquired (or received) from other LP's that could potentially send messages. Knowledge acquisition or knowledge dissemination would also be required in order to prevent deadlock in the simulation. The method for disseminating or acquiring knowledge could be quite flexible and could even be adaptable.

Method: Consider a method, as shown in Figure 4, that is adaptive with respect to knowledge acquisition and knowledge dissemination. The appointments method [NiRe84] assumes that LP's exchange knowledge, generated in part from embedded knowledge, for the purpose of allowing LP's to advance their simulation time non-aggressively. The question of how much knowledge dissemination to employ as opposed to (or in conjunction with) how much knowledge acquisition to employ is difficult to capture statically. An adaptive method could be employed where these questions could be answered dynamically through e.g. simple dynamic statistical analysis of the effectiveness of each of the approaches: acquisition and dissemination.

	partit-ioning	adapta-bility	aggress-iveness	accu-racy	risk	Knldg Embed	Knldg Dissem	Knldg Acquis	Synch-rony
Local rollback			Y	Y	N				A

Figure 3. Local Rollback Protocol

	partit-ioning	adapta-bility	aggress-iveness	accu-racy	risk	Knldg Embed	Knldg Dissem	Knldg Acquis	Synch-rony
Adap Knldg Sharing		Y		Y		Y	Y	Y	A

Figure 4. Adaptive Knowledge Sharing Protocol

Variations on this method can be derived by considering bindings for other design variables. Mixing aggressiveness (possibly adaptive) in with the others could produce an approach that is quite different from a non-aggressive approach. As with any method, a partitioning of methods could be beneficial.

It is not our intention to suggest that the alternatives outlined here are the newest and best approaches to parallel simulation. We have little idea how well they would perform. Our goal is to demonstrate the utility of the framework our taxonomy provides. Another goal is to make quite clear the fact that a large number of very reasonable approaches to parallel simulation are as yet unexplored.

SPECTRUM: A TESTBED

Up to now parallel simulation algorithms have been studied in isolation. That is, individual algorithms have been analyzed in one or a small number of locations. We know of no cases where two or more algorithms have been applied to the same application in the same environment. Coupled with the observation that a multitude of potential algorithms exist, we have embarked on constructing a testbed that will allow for the testing of a variety of parallel simulation algorithms in a *common* environment.

The SPECTRUM (Simulation Protocol Evaluation on a Concurrent Testbed with ReUsable Modules) testbed is functioning in a prototype state and is expected to be in a fully operational state by late Fall of 1988. It is operating on an INTEL iPSC/2 32-node hypercube and will soon be ported to a BBN 32-node GP-100 in the University of Virginia's Institute for Parallel Computation. The testbed is designed to support efficient evaluation of a variety of applications and protocols. Early experience indicates that experiments can be constructed in a small number of days. The testbed will be described in a forthcoming paper.

CONCLUSIONS

Early in the development of operating systems theory there was a lot of research directed at the determination of "the best" CPU scheduling algorithm. As intuition would have it, a fairly simple algorithm, round robin, emerged as a good general purpose algorithm. In the same sense many of the alternatives created by the design variables presented could probably be rejected fairly quickly. However, the analogy should not be carried too far. Round robin is a good general purpose algorithm for general purpose computing. Parallel simulation on the other hand tends to be less general purpose in nature and it has higher performance demands. Many of the parallel simulation options that might otherwise be discarded because of their limited generality must be considered. The higher the performance demands the more special cases we must consider.

The design variables defined here provide us with a starting point for exploring a large number of options that have, to date, been overlooked. They provide us with a more general taxonomy than "optimistic" and "conservative." However, their most important contribution is to provide us with a clear indication that the determination of high performance parallel simulation algorithms is not only not completed, the explorations have only begun. A key issue at this point is whether analytic methods can help us contain what looms as an expansive and time-consuming empirical investigation.

ACKNOWLEDGMENTS

This research was supported in part by the Department of Defense, through the Jet Propulsion Laboratory, contract number 957721. Thanks also to Craig, Phil and the SPECTRUM group for their invaluable input.

REFERENCES

[Bain88] Bain, W. and D. Scott, "An Algorithm for Time Synchronization in Distributed Discrete Event Simulation," *Proc., SCS Multi-conference*, San Diego, CA, Feb., 1988.

[ChMi79] Chandy, K.M. and J. Misra, "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs," *IEEE Trans on Software Engineering.*, SE-5,5, May, 1979, 440-452.

[ChMi81] Chandy, K.M. and J. Misra, "Asynchronous Distributed Simulation via a Sequence of Parallel Computations," *CACM*, 24,4, April, 1981, 198-205.

[ChMi87] Chandy, K.M. and J. Misra, "Conditional Knowledge as a basis for Distributed Simulation," *CalTech Tech*

report, 5251:TR:87, Sept 1987.

[Chan88] Chandy, K.M., Private Communication.

[FJLO88] Fox, G., et al., "Solving Problems on Concurrent Processors," Prentice Hall, 1988.

[Fuji88] Fujimoto, R.M., "Performance Studies of Distributed Simulation Strategies," *Proc., SCS Multi-conference,* San Diego, CA, Feb., 1988.

[HaDo88] Hartrum, T.C. and B.J. Donlan, "Distributed Battle-management Simulation on a Hypercube," *Proc., SCS Multi-conference,*, San Diego, CA, Feb., 1988.

[JeSo82] Jefferson, D. and H Sowizral, "Fast Concurrent Simulation Using the Time Warp Mechanism," *A Rand Note*, N-1906-AF.

[Jeff85] Jefferson, D., "Virtual Time," *ACM TOPLAS*, 7,3, July, 1985, 404-425.

[Kuma86] Kumar, D. "Simulating Feed-forward Systems Using a Network of Processors," Annual Simulation Symposium, Dec. 1985, 127-144.

[LCUW88]Lomow, G., et al., "A Performance Study of Time Warp," *Proc., SCS Multi-conference,*, San Diego, CA, Feb., 1988.

[Luba87] Lubachevsky, B., "Bounded Lag Distributed Discrete Event Simulation", *Proc., SCS Multi-conference,*, San Diego, CA, Feb., 1988.

[Misr86] Misra, J., "Distributed Discrete Event Simulation," *ACM Computing Surveys,* 18,1, March, 1986, 39-65.

[Nico88] Nicol, D.M., "Parallel Discrete Event Simulation of FCFS Stochastic Queuing Networks," *Proceedings of ACM SIGPLAN PPEALS,* Yale University, July, 1988.

[NiRe84] Nicol, D.M. and P.F. Reynolds, "Problem Oriented Protocol Design," *ACM Winter Simulation Conference,* Dallas, Texas, Nov., 1984, 471-474.

[PeWo78] Peacock, J.K., Wong, J.W. and E. Manning, "Distributed Simulation Using a Network of Processors," *Computer Networks,* 3, North Holland Pub., 1979, 44-56.

[PeMa80] Peacock, J.K., Manning, E. and J.W. Wong, "Synchronization of Distributed Simulation Using Broadcast Algorithms," *Computer Networks*, North Holland Pub., 1980, 3-10.

[ReMa87] Reed, D.A., Maloney, A.D. and B.D. McCredie, "Parallel Discrete event Simulation: A Shared Memory Approach," *1987 SIGMETRICS Conf on Measurement and Modeling of Computer Systems*, Banff, Alberta, Canada, May, 1987.

[ReMa88] Reed, M.A., and A.D. Maloney, "Parallel Discrete Event Simulation: The Chandy-Misra Approach", *Proc., SCS Multi-conference,*, San Diego, CA, Feb., 1988.

[Reyn82] Reynolds, P.F. "A Shared Resource Algorithm for Distributed Simulation," *Proc of the Ninth Annual Int'l Comp Arch Conf,* Austin, Texas, April, 1982, 259-266.

[Soko88] Sokol, L., et al. "MTW: A Strategy for Scheduling Discrete Events for concurrent Execution",*Proc of SCS Multi-Conference,*" February, 1988, San Diego, 34-42.

[Theo84] Theofanos, "Distributed Simulation of Queueing Networks," Master's Thesis, The Univ of Virginia, Jan., 1984.

[Word88] Worden, J. "National Testbed Program," *Proc of SCS Multi-Conference: Aerospace Simulation III,*" February, 1988, San Diego, CA.

BIOGRAPHY

PAUL F. REYNOLDS, JR. is an Associate Professor of Computer Science, as well as the Director of the Institute for Parallel Computation at the University of Virginia. He is a member of the Simulation Engineering Group, an oversight group for the National Testbed. His research interests include parallel and distributed simulation, and, in general, parallel language and algorithm design. He has been a consultant to numerous corporations and government agencies in the systems and simulation areas. His Ph.D is from the University of Texas, 1979.

Institute for Parallel Computation
Thornton Hall
The University of Virginia
Charlottesville, VA 22901
(804) 924-1039
pfr@uvacs.cs.virginia.edu