TECHNICAL ARTICLE

SIMULATION 72:3, 170-186

ISSN 0037-5497/99
Printed in the United States of America

Survey of Languages and Runtime Libraries for Parallel Discrete-Event Simulation

Yoke-Hean Low[1], Chu-Cheow Lim[1], Wentong Cai[2],
Shell-Ying Huang[2], Wen-Jing Hsu[2], Sanjay Jain[1], and Stephen J. Turner[3]

[1] Gintic Institute of Manufacturing Technology, Singapore
[2] School of Applied Science, Nanyang Technological University, Singapore
[3] Department of Computer Science, University of Exeter, Exeter, United Kingdom

To develop a parallel discrete-event simulation from scratch requires in-depth knowledge of the mapping process from the physical model to the simulation model, and also a substantial effort in coping with the numerous issues concerning the underlying synchronization protocols in use. Languages and libraries could reduce the development effort significantly by providing the user with a pre-built parallel simulation kernel as well as application development tools. This paper contains a survey of the existing languages and libraries for parallel discrete-event simulation. It is divided into two major sections: one on the languages, the other on the libraries. The discussions are mainly focused on the following aspects: user model, programming framework and language features, library API, protocols, and system support and environment. The reported performances of some packages are also summarized.

Keywords: Parallel discrete-event simulation, languages, libraries, APOSTLE, Parsec, ParaSol, SPaDES, WARPED, language survey, library survey

1. Introduction

Parallel Discrete-Event Simulation (PDES) has been a widely researched area in recent years. There are two main objectives in using PDES. First, the use of parallel processors promises an increase in execution speed. Second, the potentially larger amount of available memory on parallel processors will enable the execution of larger simulation models, as compared to the uni-processor version. Simulation of large-scale models such as a virtual factory [1] would thus become feasible.

The widespread interest in PDES in the research community, however, did not bring about the widespread deployment of PDES in real-world applications [2]. Most research has been focused on inventing new algorithms and getting performance measurements on specific applications [3]. There are two main reasons for this phenomenon. First, parallel processing machines, though in much abundance, are still mainly used by the research community. Most users would only have access to uni-processor machines. Second, the effort of porting over a sequential simulation application to a parallel one requires a substantial amount of programming effort and expertise.

The first obstacle is purely an economical one. It is in some ways related to the second obstacle in the sense that companies would need justification to purchase parallel processing machines. The easiest way to do this would be to implement a prototype of the existing model in PDES. The purpose is to demonstrate

to the management the potential benefits that can be obtained by using PDES. However, this is not a simple task in general. The personnel implementing the prototype would have to be well versed in PDES issues, such as synchronization, partitioning and load balancing, to even begin building a simple abstract model. Because of this, very few attempts have been made by commercial companies to experiment with PDES, not to mention deploying parallel machines in the companies.

To encourage the adoption of PDES, the migration process from a sequential simulation to PDES would have to be made as smooth as possible. The user should be able to concentrate his/her effort on the modeling process and not be distracted by the underlying PDES synchronization protocol issues. To achieve this goal, researchers have come forward and delivered languages and libraries for PDES implementation. An early survey on PDES languages and libraries can be found in [4].

Research on PDES languages and libraries has been especially active in the past few years. This is evident by the availability of new packages, such as APOSTLE [5], Parsec [6], ParaSol [7], SPaDES [8] and WARPED [9]. Therefore, there is a need to once again compare and contrast these languages and libraries to see how the simulation community, in general, could benefit from using them.

This paper contains a survey of the existing PDES languages and libraries which could potentially be of use in certain applications. We limit our discussion to general-purpose PDES languages and libraries. Application-specific PDES languages such as TeD (Telecommunications Description Language) [10] are not included in this survey.

We divide the packages surveyed into two categories: languages and libraries. A simulation language usually provides a full set of well-defined language constructs for the user to design simulation models, whereas a library only provides a group of routines to be used with a base programming language (e.g., C or C++). Our list could not possibly include all the existing PDES languages and libraries in the literature. To the best of our knowledge, they represent the more commonly used and widely researched ones. A fuller version of this paper, with a summary of each package surveyed, can be found in [11].

A quick introduction on the synchronization protocols used in PDES is discussed in Section 2. This would enable the readers to grasp the various concepts pertaining to synchronization protocols discussed in the subsequent sections. Section 3 gives a quick description of the desired features needed in a PDES language or library. Section 4 provides a description and comparison of the existing PDES languages surveyed. Section 5 provides a description and comparison of the existing PDES libraries surveyed. Section 6 summarizes the observations made on these packages together with future work recommended along this line.

2. Parallel Simulation Protocols

Many parallel simulation protocols have been proposed in the last 15 years [12, 13]. In general, PDES can be broadly classified into two categories: those implemented using a conservative algorithm and those using an optimistic algorithm. Within every PDES environment, be it conservative or optimistic, is the concept of a Logical Process (LP), although some languages or libraries may present a higher level of abstraction to the user (for example, an object-oriented language will allow an application to be modeled using simulation objects that may not correspond directly to LPs). Each LP interacts with one another using time-stamped event messages. To model the application to be simulated, the user must find a mapping from the physical processes to the logical processes. The sequence of steps to go about doing the mapping is referred to as the modeling process. To preserve the correctness of the simulation in a PDES simulation system, each LP must execute all incoming event messages in a non-decreasing timestamp order. This is commonly known as preserving the causality constraint.

PDES implemented using a conservative algorithm [14] strictly enforces the causality constraint. Each LP only processes an incoming event message if it is certain that no other event messages with a lower timestamp will be sent by other LPs in the system connected to it. With this constraint, there is thus a possibility that deadlock will occur. Various algorithms have been proposed to solve the deadlock issue. The most commonly used algorithm involves the use of null-messages to ensure continual progression of simulation time on all communication links between LPs. The main drawback of this algorithm is the overhead in sending a potentially large number of null-messages relative to actual event messages.

To generate the null-messages, the system must be able to predict lookahead information in the simulation model. Consider two LPs, P_i and P_j, sharing a communication link L_{ij}. Lookahead for the link L_{ij} refers to the guaranteed time interval T_{la} for P_i not to generate new events to P_j using the link L_{ij}. Suppose P_i is at local simulation time T_{local} and no pending events are to be executed. If the lookahead information T_{la} is known, P_i would be sure that the next time it is going to send an event to P_j would be at time $T_{local}+T_{la}$. It would then send a null-message with timestamp $T_{local}+T_{la}$ to P_j. This would allow P_j to go ahead in the simulation time if it is only waiting for events from P_i.

However, lookahead depends very much on the user's knowledge of the simulation model and the runtime characteristics of the system. A good lookahead in the system is almost certain to improve the performance of a conservative simulation significantly. Other algorithms such as carrier null-messages [15] and conservative time windows [16] have been proposed in recent years. An algorithm that exploits both local and global information to advance local simulation time has also been demonstrated in [17].

An optimistic algorithm [18], on the other hand, allows events to be executed without considering if executing the events will result in any casuality violation. Each LP in this case will proceed to execute each incoming event as it happens. If an event message arrives and has a timestamp that is lower than the ones that have been processed, the LP must correct this error by undoing the events that have been executed. This is commonly known as rollback.

Anti-messages are usually used by a rollback mechanism to force local or remote LPs to roll back to a previous consistent state. In general, PDES using an optimistic algorithm tends to require more memory than the conservative one, due to the state-saving requirement of the optimistic algorithm. By periodically calculating the Global Virtual Time (GVT), the system can reclaim memory from obsolete saved states on LPs as well as committed event messages. This process is usually referred to as fossil collection. Many variations of the optimistic protocol have been proposed, which can be found in [12].

Selecting a suitable synchronization protocol to use is a common issue faced by a user whether he/she chooses to use a PDES language or library. Very often, a simulation model may exhibit good performance running with one protocol, but can deliver a slowdown in performance when switching to another protocol.

To date, there is no good solution that would allow users to know in advance what synchronization protocol best suits their applications. Different factors such as the available platforms, the communication characteristics of the model, granularity of each event, partitioning, and load balancing issues need to be considered. In some cases, these factors can be determined by constructing an abstract model of the actual simulation application and executing it using both types of synchronization protocols. Using the abstract model, the user can then generate code for different simulation systems or even for performance evaluation tools such as N-MAP [19]. An example of constructing an abstract model for manufacturing applications and using the abstract model for protocol selection can be found in [20].

However, one should always bear in mind that in general, an abstract model can be completely unrepresentative of the actual model to be simulated. The decision on selecting which protocol to use can often be determined only after the actual model has been tested on a specific synchronization protocol.

3. Desired Features for PDES Languages and Libraries

To compare and contrast various PDES languages and libraries, we will look at the following aspects of each package in the following sections:

- User model (or modeling capability);
- Programming framework (e.g., structured or object-oriented);
- Language features and library API (e.g., for scheduling and communication);
- Protocols (e.g., conservative or optimistic);
- System support and environment (e.g., mapping and program development support).

In addition, the reported performance of some packages will also be summarized.

All PDES languages and libraries require the user to model the physical environment in a certain fashion. This is referred to as the world view presented to the user. The user may be allowed to provide a process interaction world view of the model, where each process represents an arbitrary long sequence of activities. Alternatively, the user may be required to provide an event scheduling view of the model where each state change of a simulation object corresponds to the scheduling of an event. Depending on the needs of the user, the selected language or library must provide a modeling framework that allows the easy mapping of the user's model. The user should not be expected to spend too much effort in converting a model from one view to another just to suit the requirements of the system selected.

The programming framework offered by the PDES system to the user would directly affect the development time and maintenance effort required by the user in developing a PDES application. In general, if the system offers an object-oriented programming framework to the user, the development time of the application will be much reduced. However, the price to pay for using an object-oriented approach comes in terms of the runtime overhead. This often results in slower executing speed as compared to those based on structured languages such as C.

If a PDES language is selected, it is crucial for the PDES language to provide a set of language constructs for simulation model construction. For example, the language may need to provide language constructs for the user to express a sense of time. The user might also need to have certain control over the order of event processing and dispatching. If necessary, the user should also be able to specify a scheduling construct such as *wait until* to suspend an LP for an arbitrary amount of time or until a certain condition becomes true.

In parallel discrete-event simulation, different simulation objects may be simulated concurrently on different processors and thus will be at different simulation times at any given instance. To allow shared states between simulation objects, it is therefore necessary to have a mechanism to exchange information. The communication between simulation objects can be either explicit or implicit, and implemented by either message-passing or sharing of memory. Approaches to the implementation of shared state are discussed in [2].

The advantage of using a PDES library over a PDES language is that the user is given more flexibility in

controlling the simulation application in terms of the behavior of the underlying synchronization protocol. PDES libraries often come packaged with various options on the runtime characteristics of the protocol that the user can selectively turn on. A knowledgeable user may thus be able to exploit further performance from a PDES library by fine-tuning the options provided. But, users of PDES libraries should also note that if the options are not set correctly with respect to the model that is to be simulated, the performance of the simulation may degrade significantly. In this case, the user might want to use a PDES language so as not to get involved in the underlying synchronization issues.

In the case of an optimistic PDES library, the user may have the option of choosing between the different kinds of rollback controls, state-saving mechanisms and GVT calculation mechanisms. In the case of a conservative PDES library, the user may be able to specify lookahead information and assign network topology. However, most PDES languages are designed to hide these synchronization protocol issues from the user. These languages would often adopt a default set of options for the underlying synchronization protocol and would not allow the user the option of changing them.

The mapping of LPs to processors is an important issue relating to partitioning and load-balancing. A PDES system should ideally be able to automatically assign LPs to physical processors based on some partitioning and load-balancing heuristics. Sometimes, the user is in a much better position to know the runtime characteristics of the processes and would want to manually perform partitioning and load-balancing. A PDES system should also allow the user this flexibility.

Statistical information collected from the simulation runs can be classified into two categories: statistics counters (such as maximum queue length and throughput) that provide the user with feedback regarding the simulated model itself; and execution information (such as simulation execution time, the number of null-messages generated, memory usage and number of rollback events) that allows the user to assess the performance of the PDES runtime engine.

Although the user could manually track the desired aspect of the simulation model and simulation execution, the PDES languages and libraries should ideally provide the user with options to collect runtime statistics automatically. The user should be allowed to turn on the execution information collection when debugging the simulation model and doing performance analysis, and turn them off during the actual simulation run. In general, the statistical information will help the user to further understand the runtime characteristics of the model. In the case of PDES libraries, the execution information might help the user to assess the efficiency of the synchronization protocol and to decide which synchronization options to turn on/off.

In addition, all PDES systems should ideally be equipped with visualization and debugging capabilities to facilitate the process of developing an application. Visualization tools allow the user to create the simulation model graphically. They can also allow the user to view the simulation as it progresses. Debugging facilities often come in the form of a source-level debugger or a program execution trace file. A source-level debugger is required for runtime debugging. A program execution trace file allows the user to do post-mortem debugging and performance tuning.

4. Languages for Parallel Discrete-Event Simulation

This section gives a brief description and comparison on the list of existing languages for PDES implementation (see Table 1). An early effort by Jade Simulations in developing the PDES language Sim++ was reported in [21, 22]. As Sim++ is quite old and no continuing research efforts are carried out on it, the language will not be discussed in this paper.

The motivation for the creation of these PDES languages rather than library routines is that users of these languages would not need to be concerned with the underlying PDES protocols, as they will be taken care of by the language runtime system. In the case of a PDES library, the user must explicitly make reference to the library routines provided and very often needs to be aware of certain runtime issues relating to PDES. The main requirement of a good PDES language would thus be the extent of transparency of the programming model provided to the user.

APOSTLE (A Parallel Object-oriented SimulaTion LanguagE) [5] is an object-oriented PDES language based on an optimistic protocol. It is a project under the DERA (Defence Evaluation and Research Agency).

Maisie [23] is a C-based PDES language developed to support a number of different simulation protocols,

Table 1. Existing PDES languages discussed in this paper

Language	Organization	Protocol
APOSTLE	DERA Malvern	Optimistic
Maisie/Moose/Parsec	UCLA	Conservative/Optimistic
ModSim	CACI/Jade Simulations	Optimistic
YADDES	University of Waterloo	Conservative/Optimistic

including both optimistic and conservative. Moose (Maisie-Based Object-Oriented Simulation Environment) [24], is the object-oriented version of Maisie that uses inheritance to support iterative design of efficient simulation models. Parsec [6] is derived from Maisie, with several enhancements incorporated and language constructs rewritten. The three packages were developed at UCLA and have been used widely for general-purpose PDES simulation.

ModSim [25] is an object-oriented simulation language based on Modula-2. It was originally developed by CACI Products, Inc., from 1987 to 1989, under contract with the Army Model Improvement Program (AMIP) Management Office using the Jet Propulsion Laboratory's TWOS [26]. The development was taken over by Jade Simulations from 1990. A version of ModSim based on Jade's Time-Warp was released in 1991. A sequential version of ModSim, MODSIM II [27], was later developed and released by CACI Products.

YADDES [28] was developed at the University of Waterloo, Canada. It is a simulation-specification language, which also provides support for a variety of synchronization protocols.

4.1 User Model

All the PDES languages surveyed provide a process interaction world view to the user. In Parsec, for example, each process is represented by an entity. Each entity will execute its own tasks, and if necessary, communicate with one another by sending and receiving messages. In the following code example from Parsec, the entity *worker* first performs some tasks, then receives a *storeRequest* message from another entity, stores the message, and resumes working on another task.

```
entity worker {
    ... ...;
    doTask();
    receive (storeRequest msg) {
        storeMessage(msg);
    }
    doMoreTask( );
}
```

Most PDES languages supporting the process interaction world view could also be programmed in an event scheduling way. In the event scheduling approach, each task in a process will be activated on the arrival of some triggering event. The process interaction world view is more suitable for direct mapping of a real-world process which consists of an arbitrarily long sequence of activities. Event scheduling is convenient for the direct mapping of a real-world resource-oriented scenario where objects are exchanged between resources.

In fact, one view could be converted to another by performing additional analysis on the existing model [29]. For example, to convert the above code segment into an event scheduling world view, it is sufficient to note that no other task or event comes before the function *doTask()* and thus *doTask()* is the first task in the *worker* process. We could assign *doTask()* to correspond to a *processInit* message. In addition, since *doMoreTask()* is the last task in the process and no task or event comes between it and the *storeMessage()* task, we could group the two tasks together and activate them on receiving a *storeRequest* message. We further assume that the *stopWorker()* function will terminate a worker entity on receiving a *processStop* message. Using Parsec, the process modeled using the event scheduling world view is as follows.

```
entity worker {
  ... ...;
  for (;;) {
    receive (processInit int_msg)
      doTask();
    or receive (storeRequest req_msg) {
      storeMessage(req_msg);
      doMoreTask();
    }
    or receive (processStop stop_msg)
      stopWorker();
  }
}
```

However, this conversion cannot be easily done in the general case. For example, it would be very difficult to directly translate the Parsec code given above to the event scheduling view if the receive(*storeRequest msg*) statement is in a nested construct (e.g., a loop).

4.2 Programming Framework

The approach taken by the PDES languages towards programming a simulation application can directly affect the learning curve for using the language. A PDES language can provide the user with a traditional structured programming approach, or it can provide the user with an object-oriented programming approach. It can hide the underlying process to processor mapping, or make users responsible for the mapping themselves. The language can also choose to allow or disallow the user to exercise control on the event scheduling. Depending on the needs of the user, some languages may be deemed to be more restrictive than others.

It is generally agreed that object-orientation reduces development time and provides better code management and reusability. However, the price to pay for all the advantages comes in the form of runtime overhead.

APOSTLE, ModSim and Moose provide the users with an objected-oriented environment to develop their applications. For example, in APOSTLE each process is defined as an object and the behavior of a process is indicated by its associated methods. ModSim and Moose support class-based inheritance, whereas APOSTLE has a delegation-based inheritance

that supports both *part-of* and *inherits-from* object relations [30].

In Maisie / Parsec, though the language used is C-like, each process is modeled as a variable of type entity which has a definition similar to that of a C function. To create an instance of the entity, the keyword new is used. All entities in Parsec can be cast to type ename. The following code segment from Parsec illustrates how entities are created:

```
entity worker() {
  ... ...
}

entity manager() {
  ... ...
}

entity factory() {
  ename wl, ml;
  wl=new worker();
  ml=new manager();
  ... ...
}
```

where the entity factory defines two enames and assigns one to be of type *worker* and the other of type *manager*.

YADDES requires the user to provide a specification of the desired simulation in the form of a specification source file. This will in turn be translated into a collection of C language routines and linked to a runtime synchronization library to produce the executable code.

When two applications, one written using a traditional structured programming approach and the other using an object-oriented approach, are having comparable performance, the advantage of using an object-oriented approach can sometimes outweigh the overhead incurred by it. This is illustrated in the case of APOSTLE, where it is reported in [31] that the performance of a Petri Net simulation with places and transitions defined as objects is very close to the one in which places and transitions are defined as arrays.

4.3 Language Features

APOSTLE supports interrogative scheduling [32]. This is useful if the user wants to model the passage of time for simulation entities, or a simulation entity waiting for an event to happen. The wait until construct in APOSTLE allows waiting for simulation time to elapse as well as a Boolean condition to become true.

The same behaviour can be achieved in Parsec using the wait construct together with a guarded message. The wait construct in Parsec allows the user to explicitly specify that a process is to be suspended for a certain amount of simulation time. A guarded message in Parsec allows the user to specify a side effect free Boolean expression to be true for the process to wake up. For example, the entity executing the following statement would suspend itself until the message queue contains a message of type *storeRequest* whose message attribute *unit* is greater than 5.

```
receive (storeRequest msg) when (msg.units>5)
```

This message will then be removed from the queue and the entity can resume execution. The entity would potentially be waiting forever if the required message never arrives. Thus, a language construct:

```
timeout after (t)
```

is also provided in Parsec to allow the entity to resume execution after a simulation time period (i.e., *t* in the above example) has elapsed.

As has been illustrated above, communications in Maisie / Moose / Parsec are explicit. Entities communicate with each other via buffered message-passing. For instance, Parsec uses typed messages. Asynchronous send and receive primitives are provided to respectively deposit and remove messages from the message buffer of an entity.

Communications in APOSTLE and ModSim are implicit via method invocation. In APOSTLE, method invocation is always synchronous. However, any expression containing one or more statement or method invocation may be executed asynchronously using the async construct. The result of the expression can be collected later based on the handle returned. In the following example, the first expression is executed concurrently with the second one.

```
async(job())
async(wait for time_t; job())
```

ModSim supports both synchronous and asynchronous method invocations. In ModSim, a method can be invoked synchronously using the ask method, which may return values; the tell method is used for asynchronous method invocation, which does not return values.

In APOSTLE, concurrency control is supported by predicate path expression and ordering rules. There is no concurrency control construct in either ModSim, Maisie / Moose / Parsec, or YADDES.

YADDES is a specification language in the style of Yacc and Lex, which provides a minimal set of language constructs. The basic components of a YADDES program are model specifications, process specifications, and connection specifications. Model specifications are used to describe general state machines. Process specifications are used to create logical processes by instantiating models. Connection specifications are used to describe the connections between the inputs and the outputs of the logical processes.

4.4 Protocols

APOSTLE and ModSim support only optimistic simulation based on the time-warp protocol, whereas

Maisie / Moose / Parsec and YADDES provide support for a variety of synchronization protocols. APOSTLE and ModSim are more transparent to the user on the underlying protocol implementation. In Maisie / Moose / Parsec and YADDES, for the conservative protocol in particular, the user is often required to provide additional information about the application for the underlying runtime system.

All the PDES languages surveyed allow the user the option of running the simulation sequentially. It is often easier to first develop the simulation model on the sequential runtime system. After the simulation model is tested, the user can then turn on the parallel runtime system and execute the simulation.

Maisie / Moose / Parsec supports both conservative and optimistic simulations. In conservative simulation, for instance, Maisie provides a function *zzcla()* to allow each entity to specify a *lookahead* value. In the case of optimistic simulation, Maisie also allows the user to set a number of parameters to control the frequency of state-saving, the interval for processing GVT, the window within which an entity can process events aggressively, and the interval to collect the state memory. Different runtime libraries can be linked into the program using command line parameters during compilation time.

YADDES provides four runtime libraries to support four different execution mechanisms:

- Sequential discrete-event simulation using a single event list;
- Distributed discrete-event simulation based on multiple synchronized event lists;
- Conservative distributed discrete-event simulation using null-messages; and
- Optimistic distributed discrete-event simulation using time-warp.

However, the runtime library implementing the conservative distributed discrete-event simulation mechanism leaves the prevention of deadlock (that is, the generation of null-messages) to the user.

4.5 System Support and Environment

The mapping of logical processes to processors is an important issue relating to partitioning and load balancing. More often than not, the PDES languages leave the task to the programmers. This is found to be the case for the list of PDES languages surveyed. Users are required to assign the logical processes or objects manually to the underlying physical processors. The exception to this is ModSim, which can perform automatic mapping of one ModSim object to the underlying TWOS kernel process. However, an object-oriented design, such as that supported by ModSim, typically results in relatively fine-grained object definitions. For a big application, tens of thousands of objects may be created. In CACI's implementation, multiple ModSim objects cannot be explicitly mapped onto a single TWOS process [33]. To manage a large number of TWOS processes will inevitably generate considerable overhead due to the time-warp mechanisms.

Statistics collection in a PDES language can be done automatically by the runtime system, or manually tracked by the user. Both YADDES and Maisie / Moose / Parsec provide automatic runtime statistics collection on items such as execution time, number of events received per process, and (if a conservative protocol is used) the number of null-messages generated. APOSTLE and ModSim do not provide any known built-in statistics collection facilities.

Of the four PDES languages surveyed, only Maisie / Moose / Parsec has a visual programming environment called MVPE (M*aisie* V*isual* P*rogramming* E*nvironment*) [41]. A Maisie / Moose / Parsec program can be verified using standard output facilities provided by the C / C++ language. Debugging of a Maisie / Moose / Parsec program can be done using traditional source-level debuggers with the sequential kernel turned on via compile time switches. In the case of APOSTLE, debugging can only be done through a simple form of unformatted stream output that prints out debug messages during runtime. No known debugging facilities are reported by YADDES and ModSim. However, runtime messages for a YADDES program can be printed out by specifying a command line parameter when executing the program.

4.6 Summary

Table 2 shows a summary of the various features of the PDES languages outlined above. Note that the statistics feature denotes whether the PDES language has built-in functions for statistical purposes.

Table 3 shows some selected performance figures reported in the publications related to the PDES languages surveyed. The performance of YADDES is unknown.

As can be seen from Table 3, all packages reported positive speedup. The performance figure of APOSTLE, as indicated in [5], is a speedup of 2 on 16 processors on a 22-nodes Meiko CS-2. It is reported that the APOSTLE granularity control mechanism reduces the simulation runtimes by as much as 80%. For ModSim, in general, the performance is poor for fine-grain event processing. For example, it is reported that only a speedup of 2 was achieved on 48 processors when event processing time is two milliseconds [37]. Maisie / Parsec is used in many applications, including queueing networks, VLSI design, mobile wireless communication and ATM networks simulation [35, 36, 38, 39,].

5. Libraries for Parallel Discrete-Event Simulation

The following section consists of a list of existing libraries for PDES implementation. The list is by no means an exhaustive one. There are several other PDES

Table 2. Comparison chart for different PDES languages

Features	APOSTLE	Maisie/Parsec	Moose	ModSim	YADDES
Protocol	Optimistic	Various	Various	Optimistic	Various
World View	Process/ Event	Process/ Event	Process/ Event	Process/ Event	Process
Programming	OO	Structured	OO	OO	Specification
Mapping	Manual	Manual	Manual	Automatic	Manual
Visualization	None	MVPE	MVPE	None	None
Debugging	None	Source Level	Source Level	None	None
Statistics	None	Present	Present	None	Present

Table 3. Selected performance measurement for different PDES languages

Package	Application	Platform	Speedup
APOSTLE	Super-ping [5]	22-nodes Meiko CS-2	9 on 16 processors
Maisie	Gate-level circuit simulation [35]	8-nodes SP-1	4
Parsec	Wireless network simulation [36]	32-nodes SP-2	7.5 on 16 processors
ModSim	Ground combat simulation [37]	48-nodes Computing Surface	6-10 on 32 processors

libraries found in the literature but they are either quite outdated, or no continuing research activities are carried out on them. Examples of these libraries are OLPS (Object Library for Parallel Simulation) by Abrams [40] and TWOODS (Time Warp Object-Oriented Distributed Simulation) by Jon Agre and Pete Tinker [41]. The PDES libraries surveyed in this paper are listed in Table 4, below.

Except for CPSim, which is a commercial conservative PDES library package offered by Boyan Tech, Inc., the rest of the library packages surveyed in this paper are all research-oriented and based on optimistic protocols. In general, compared with optimistic protocols, conservative protocols can be implemented with less effort, and the optimization of the protocol is more application-specific. For instance, *lookahead* information, which is crucial for a conservative protocol to be effective, is usually application-dependent. Therefore, to implement a conservative protocol, the most effective approach is for the programmer to use a general-purpose parallel runtime system, and implement optimizations specific to the simulation application.

Due to the inherent complex and intractable nature of the time-warp mechanisms (e.g. *state saving*, *rollback* and *fossil collection mechanisms*), optimistic protocols are far more difficult to implement than conservative ones. Thus, contrary to conservative protocols, in order to reduce the effort in developing a simulation application using optimistic protocols, the most effective way is for the programmer to use a parallel simulation language or library that hides the implementation details of the time-warp mechanisms. This may explain why most of the libraries are developed for optimistic protocols.

CPSim [42] is a C-based commercial PDES library with a conservative simulation engine. CPSim was initially developed as a parallel simulation tool by Dr. Bojan Groselj. This was later commercialized and distributed by Boyan Tech, Inc., in March 1995.

Table 4. PDES libraries surveyed in this article

Library	Organization	Protocol
CPSim	Boyan Tech, Inc.	Conservative
GTW	Georgia Tech	Optimistic
ParaSol	Purdue University	Optimistic
PSK	Royal Institute of Technology	Optimistic
SimKit	University of Calgary	Optimistic
SPaDES	National University of Singapore	Optimistic
SPEEDES	Jet Propulsion Laboratory	Optimistic
TWOS	Jet Propulsion Laboratory	Optimistic
WARPED	University of Cincinnati	Optimistic

GTW [43] and TWOS [26] are PDES libraries implemented using C. GTW started out as a research project at Georgia Institute of Technology, Atlanta, Georgia. Early versions of GTW for shared-memory machines were used as the basis for other time-warp systems, including University of Calgary's *Warpkit* system [44] and SAIC's *Tempo* system [45]. TWOS was developed originally at the Jet Propulsion Laboratory (JPL), funded by the U.S. Army Improvement Program (AMIP) Management Office (AMMO). It was no longer supported after the project was terminated.

ParaSol [7], PSK [46], SimKit [47], SPaDES [8], SPEEDES [48] and WARPED [9] are object-oriented PDES libraries implemented using C++. ParaSol was developed at Purdue University. PSK stands for P*arallel* S*imulation* K*ernel* and was developed at the Royal Institute of Technology, Sweden, in 1994. It has since been used to study applications specific to mobile telecommunication. SPaDES stands for S*tructured* P*arallel* D*iscrete*-E*vent* S*imulation* and was developed at the National University of Singapore. SPEEDES was developed at JPL, California Institute of Technology, during 1990-1991. The project was sponsored by the U.S. Air Force Electronics-Systems Division, Hanscom Air Force Base, Maryland. WARPED was developed at the University of Cincinnati.

Some of these packages are created to support further research interest in the development of synchronization protocols. For example, PSK has been used extensively over the years to evaluate the effect of using different state-saving strategies on the time-warp system. ParaSol is used as a testbed for the study of the advantage of using thread-based active-transactions in optimistic parallel simulations.

5.1 User Model

Library packages such as CPSim, GTW, PSK, SPEEDES, TWOS and SimKit present the user with an event scheduling world view. For example, to model the *worker* process given in Section 4.1, the following code in CPSim specifies the corresponding event handling routine, *appl()*, for the *worker* (i.e., *node* or LP in CPSim terms). The CPSim runtime system will call the *appl()* function to activate the event handling routine on a particular node. Note that in CPSim a single event handling routine is defined for all the nodes.

```
appl(int senderLP, int receiverLP, int indexRec,
  struct e_node *event, struct lst_node *p_lst)
{
  ... ...
  switch (event->type) {
    case processInit:
      doTask();
      break;
    case storeRequest:
      storeMessage();
      doMoreTask();
      break;
    default:
      printf("appl->Unknown type: %d \n", event->type);
      break;
  }
  ... ...
}
```

In the above code, different functions will be executed depending on the type of events received. In executing a function, new events may be generated for other nodes.

In these packages, new events generated need to be passed explicitly to the simulation kernel for scheduling. This can be done by calling some pre-defined library routines. For example, in CPSim, the function *esend()* is used, and in TWOS, the function *schedule()* is provided.

Both Parasol and SPaDES support the process interaction world view in simulation modeling. For example, ParaSol provides the user with the process interaction world view based on *active thread transactions* [7]. Passive simulation entities (e.g., servers in a queueing network) are modeled as *objects* (or LPs); and active simulation entities (e.g., *jobs* in queueing network) are modeled as *active transactions*, each of which is implemented using a *thread*. An active transaction describes a sequence of activities that an active simulation entity (e.g., a job) needs to accomplish. In ParaSol, objects are mapped to different processors. Thus, to use an object, threads are transparently migrated to the processor where the object is currently resident. For example, the following code describes a job in a queueing network that first uses server A then server B:

```
ATHREAD jobThread(... ... ) {
    Server[serverA]- >reserve();
    Server[serverA]- >hold(expon(5) );
    Server[serverA]- >release();
    Server[serverB]->reserve();
    Server[serverB]- >hold (expon(4) );
    Server[serverB]->release();
    ... ...
}
```

If server A and server B are mapped to different processors, the thread will be migrated to where server B is after releasing server A.

5.2 Programming Framework and Library API

Although CPSim, GTW and TWOS are C-based libraries, they provide users with function templates for simulation program construction. For example, in GTW, the user needs to specify three procedures for each LP: *initialization*, *event-handling* and *clean-up*. In CPSim, the user needs to: initialize the simulation environment (e.g., simulation end time) using function *init_app()*; define the nodes (i.e., LPs), the connection between the nodes and the mapping of nodes to processors using

the function *init()*; and specify event handling procedures for all the nodes in *appl()*.

ParaSol, PSK, SimKit, SPaDES, SPEEDES and WARPED provide an object-oriented framework for simulation program construction. In SimKit, for example, the programming interface consists of three classes:

- *sk_lp*, from which LPs are derived;
- *sk_event*, from which messages (or events) are derived; and
- *sk_simulation*, that provides the initialization interface to the kernel.

In SPaDES, the user is presented with two predefined classes: *processes* and *resources*, which model job entities and servers respectively. A process has four predefined methods:

- *activate()*, to create and schedule a new process;
- *work()*, to make a request for a resource;
- *wait()*, to suspend a process for a specified time interval; and
- *terminate()*, to destroy a process that has completed.

In addition, SPaDES also provides an *executive class* that allows the user to initialize and control the simulation run. Its predefined methods include:

- *initServer()*, to create a new LP for each server in the model; and
- *initProcess()*, to initialize a process object.

Using an object-oriented PDES library, the user usually creates a simulation application by deriving objects from some pre-defined kernel classes. In general, to provide support for object-oriented model construction, the simulation kernel itself also needs to be implemented using an object-oriented language. Although object-oriented design methodology has been recognized as a very useful approach to simulation model construction, the price paid for an object-oriented design can be high. As reported in [49], a significant amount of time was spent on tuning and modifying the design of the PSK kernel in order to obtain acceptable performance.

5.3 Protocols

In addition to the parallel protocols supported, CPSim, GTW, WARPED, TWOS and SimKit also provide the user with a sequential kernel. In the case of WARPED, the use of a sequential kernel is not transparent in the sense that the user needs to change some of the simulation codes before the sequential kernel can be used. In CPSim and SimKit, on the other hand, an application model can be executed on both the sequential and parallel engines without any code change. SimKit has a highly optimized sequential kernel that uses the splay tree implementation for the future event list. TWOS provides a sequential simulator named TWSIM which also provides the same interface as TWOS.

PDES libraries using a conservative protocol are much easier to implement as compared to the optimistic ones. Conservative protocols often do not provide different runtime options for the user to choose from. What the user needs to do at most is to specify the topology and some lookahead information associated with the application. On the other hand, various runtime options can often be found in the optimistic protocols. The user may often need to turn on / off some of these options to gain additional performance improvement.

5.3.1 Conservative Protocol

CPSim is the only conservative library package available in the list of library packages surveyed. It provides a set of C library functions for the user to build the simulation applications. The user is required to initialize certain system variables through CPSim function calls. In some cases, the user also needs to provide the implementation of some predefined functions needed by the CPSim runtime system.

As CPSim uses the conservative protocol, it also requires the use of null-messages to ensure a deadlock will not occur during the simulation. The user needs to supply lookahead information in the simulation model to avoid the deadlock and to improve the performance of the simulation. Lookahead information in CPSim is specified for each pair of links between two LPs. It can be defined during the initialization of the simulation using the function *connect()* and subsequently updated during the execution of the simulation using the function *appoint()*.

```
... ...;
lp1 = new_lp_h("worker", 0, 1);
lp2 = new_lp_h("manager", 0, 1);
connect(lp1, lp2, 5, 0);
... ... ;
```

For example, the above code segment in function *init()* tells the simulation kernel to create a new *worker* and a *manager* node, both with one server and identifier 0; and that the lower bound on service time for an event that is processed at *worker* node and sent to *manager* node is 5.

5.3.2 Optimistic Protocol

The rest of the library packages are implemented using the optimistic protocol. SPEEDES provides the time-warp protocol as an option and implements a default *Breathing Time Buckets* (BTB) protocol. Although BTB is an optimistic protocol, all the rollbacks occur locally on each LP. This is different from the time-warp protocol which allow rollbacks to propagate to other LPs. Other packages provide the user with a default standard implementation of the time-warp protocol and allow the user to turn on certain options to use other variants of the time-warp mechanisms. The options often revolve around choices, such as, which state-

saving mechanism is used, how often GVT computation is performed, how memory is managed, and how rollback is controlled.

State-saving is an important issue in a PDES application running the time-warp protocol. It directly affects the efficiency of the simulation performance and also the amount of memory required to run the simulation. The size of the state to save and the frequency of state-saving affects the performance of the system significantly. Different state-saving strategies have been proposed and some have been incorporated into these packages.

The default state-saving implemented on most of these packages is the *Full State-Saving* (FSS) policy, where the system would save the complete state of an LP after each time increment. This is as opposed to the *Sparse State-Saving* (SSS) policy, where the system only saves the full state of the LP after every fixed time interval. SPEEDES only supports *Incremental State-Saving* (ISS), where the changes of the state variables from the previous saved state are saved. ISS is useful if the state of an LP is large and not changed often. *Adaptive State-Saving* (ASS) is a variant of SSS in that the time interval between each state saving is determined during runtime based on some heuristics and not fixed by the user input.

Packages such as SimKit and GTW allow the user to perform ISS. However, the user is required to explicitly insert incremental state-saving code using the library routines provided whenever the state is changed. In the case of PSK, ISS is done transparently using C++ operator overloading. The user needs only to define the state of the LP using a system-supplied template *State* class. Operators within the *State* class have been overloaded and automatic ISS will be performed whenever the user does any modification to the state variables of an LP. The PSK kernel also provides ASS through runtime statistics analysis.

The message cancellation policy, otherwise known as rollback control, determines how anti-messages are generated. The default rollback control is known as *aggressive cancellation policy*, whereby anti-messages are immediately sent to other LPs on receiving a straggler message.

Another form of message cancellation policy is known as *lazy cancellation*. In lazy cancellation, an LP receiving a straggler message does not send out anti-messages immediately, but does a local rollback and re-simulates up to the simulation time when the rollback occurred. If all output messages generated during the re-simulation are the same as those generated before the rollback, then no anti-messages need to be sent at all. This policy prevents the potential cancellation of correct messages sent to other LPs. However, additional memory would be required to perform bookkeeping on the anti-messages maintained in the rollback queue. The cancellation of actual wrong output messages may be delayed, resulting in further propagation of the rollback chain.

Aggressive message cancellation is employed by the time-warp protocol in SPEEDES and is the default policy in GTW, whereas TWOS provides a default lazy message cancellation policy. WARPED allows the user to specify lazy / aggressive / adaptive message cancellation. In the case of GTW and PSK, automatic direct cancellation is activated if a shared memory machine is used. This is a form of optimization available only on shared memory platforms where all LPs keep track of any output messages they sent using pointers. If cancellation of these messages is required, these pointers can be used to remove or free the messages efficiently. The BTB protocol in SPEEDES only involves local rollback, so message cancellation is never required.

Global Virtual Time (GVT) update is used in a time-warp system to restrict the extent of rollback and also to reclaim unused memory space from committed events. Most of the packages surveyed do not make the GVT update requirement transparent to the user. For example, in GTW, a GVT Period parameter in the configuration file must be set to tell the runtime system how often to do GVT update. A PE_Event_Memory parameter must be set to allocate the size of memory per processing element for events. In the case of WARPED, each LP will manually call the *calculateLGVT()* method of the *GVTManager* class at the end of each simulation cycle to compute an estimation of local GVT (LGVT) based on the simulation time of all the simulation objects local to the LP. Each LP then sends the LGVT computed to a central *GVTManager*. The *GVTManager* will compute the GVT of the system based on the pGVT algorithm [50] and broadcast the new GVT to all the LPs.

5.4 System Support and Environment

Users of these PDES libraries would need to have as much control of the system as possible. In the last section, we have discussed the extent of control the user can exercise on the underlying characteristics of the synchronization protocols. This section will further outline other kinds of control the user would like to have on the programming environment of the systems.

Most of the library packages require the user to specify the mapping of LPs to processors and do not perform dynamic load balancing during runtime. Examples of such packages are SPaDES, WARPED and GTW. The following code segment shows how LP-to-processor mapping is done in GTW.

```
TWLP[0].Map = 0; /* map LP 0 to processor 0 */
if (TWnpe==1)      /* if only 1 processor available */
  TWLP[l].Map=0; /* map LP 1 to processor 0 */
else               /* otherwise */
  TWLP[l].Map=1; /* map LP 1 to processor 1 */
```

TWOS also uses manual allocation of LPs to processors but performs dynamic runtime LP migration based on information about processor utilization. PSK

does a round-robin allocation of LPs to the available processors if no user-defined mapping is specified. CPSim and SimKit, on the other hand, provide both manual and automatic mapping of LPs to processors.

Depending on the PDES libraries used, the user may be able to map multiple simulation objects onto a single LP. If there are more LPs than physical processors, multiple LPs may be mapped onto a single processor. In the case of multiple simulation objects being mapped to a single LP, these simulation objects would typically share the event list of the LP. If multiple LPs are mapped to a single processor, some PDES libraries also allow the user to specify the scheduling of these LPs on the processor. For example, in ParaSol, the user can create a customized scheduler to tell the processor which LP to run next. WARPED also provides two types of schedulers for the user to choose from: a round-robin scheduler and a least-timestamp scheduler.

Visualization tools allow the user to create the simulation model graphically and also perhaps to view the simulation visually as it progresses. PSK has a visualization tool for mobile telecommunication applications [49]. GTW has visualization support known as PVaniM-GTW [51] that is used for parallel simulations in Network Computing Environments. TWOS has a *Time-Warp Progress Graph* [52] which serves as a visualization tool to depict the progress of simulation on multiple processor nodes. However, this visualization tool only runs on Sun workstations using the Sunview graphics package.

Debugging a PDES application is often a tedious and intractable process. Runtime information needs to be extracted to verify the correctness of the simulation executed. This information can either be extracted during runtime via debug messages or gathered after the simulation run has completed using a program execution trace file. Often, these PDES libraries also provide a sequential runtime kernel for the user to run the simulation as well. As sequential debugging tools are more widely available and easier to use, the user could make use of the sequential kernel whenever possible to debug the simulation application, before proceeding to use the parallel kernel to run the simulation. However, using the sequential simulation kernel can only allow the user to verify the correctness of the simulation model, and may not reveal the bugs that only exist in the execution of the parallel simulation.

Packages such as GTW, SimKit, WARPED and CPSim allow the user to debug the parallel program via runtime messages. The user may often need to specify some compilation flags to enable these messages. For example, in GTW, the configuration file must be modified to enable debug messages to be printed prior to the compilation of the application. During runtime, a warning will be printed whenever a message is sent or received by an LP.

Statistics generation is an important feature in a PDES library. Very often, the user of the PDES library would like to know how to improve the performance of the simulation further. If the null-message protocol is used, the user would want to know how many messages are generated during the simulation, and how many of those are null-messages. This would allow the user to measure the performance of the synchronization protocol. If the time-warp protocol is used, the user may also want to know on average how many rollbacks occur on an LP, and the memory consumption during runtime. Statistics pertaining to the performance of the model are also needed by the user. If a manufacturing plant planner is looking at the simulation, he/she would want to know information, such as which machine is having the longest queue in the system and the average cycle time of the system.

WARPED relies on the user to track runtime statistics manually using the C++ language itself. The commercial product CPSim provides a standard report generation function which lists information such as throughput, minimum/maximum queue size, and utilization. Other research packages, such as TWOS, also keep runtime statistic counters such as the number of anti-messages, queue statistics, and cache statistics, and output them to a file at the end of a simulation run.

Most of the packages also provide an output mechanism for the system to output user-defined messages. Since most of these packages are operating using an optimistic protocol, to ensure that these output messages do not get rolled back, messages are output only when GVT is updated and the events corresponding to the output are committed. This would often result in messages being output in a burst manner. The output generated from the simulation can potentially be used to drive other visualization packages, such as ParaGraph [53].

5.5 Summary

Table 5 shows a comparison chart based on the discussions presented above. The row on programming indicates the programming style supported by the libraries. For packages like CPSim, although the framework is imperative and structured programming, the simulation model can be constructed using the function templates provided by the library. If the library supports sequential simulation, the debugging of the simulation model can also be carried out using the sequential kernel.

Some of the performance figures as reported in the publications related to these PDES library packages are listed in Table 6. There are no application-specific performance figures available to us on CPSim, ParaSol, and SimKit.

For a fine-grained application such as a 8×8 torus network, SPaDES shows a negative speedup. However, when the network size is increased to 32×32, the speedup is improved from 0.03 to 4.59 on 16 processors.

The application SPEEDES simulates a fully connected FIFO queueing network. The speedup is relative

Table 5. Comparison chart for different PDES libraries

Features	CPSim	GTW	ParaSol	PSK	SimKit
Protocol	Conservative	Optimistic	Optimistic	Optimistic	Optimistic
	Sequential	Sequential			Sequential
World View	Event	Event	Process	Event	Event
Programming	Function Template	Function Template	Object-Oriented	Object-Oriented	Object-Oriented
Mapping	Manual & Automatic	Manual	Manual	Automatic	Manual
Visualisation	None	Present	None	Present	None
Debugging	Debug Message	Debug Message	Not known	Debug Message	Debug Message
Statistics	Present	Present	None	None	None

Features	SPaDES	SPEEDES	TWOS	WARPED
Protocol	Optimistic	Optimistic	Optimistic Sequential	Optimistic Sequential
World View	Process	Event	Event	Event
Programming	Object-Oriented	Object-Oriented	Function Template	Object-Oriented
Mapping	Manual	Manual	Manual & Automatic	Manual
Visualisation	None	None	Present	None
Debugger	Not known	Debug Message	Debug Message	Debug Message
Statist ics	None	None	Present	None

to the doubling of the number of nodes, that is, the improvement on the speedup is ranging from 1.5 to 1.9 every time the number of nodes doubles.

6. Conclusion

Different PDES languages and libraries have been discussed in this paper. A PDES language ideally hides the underlying synchronization protocol from the user and allows the user to get on with the task of doing actual coding using the language constructs. However, most existing PDES languages have not achieved total transparency in this aspect. Without good partitioning heuristics, users must be concerned with the mapping of LPs to processors. Runtime options must still be manually set in order to achieve optimal performance. PDES languages like Maisie / Moose / Parsec that offer support for various kinds of synchronization protocols would allow the user to write one version of the application program and experiment on it with different runtime engines.

Visualization tools are still absent from most packages, be it languages or libraries. Given that most of the packages surveyed are products of research projects, visualization support may come across as a not-immediately-rewarding feature. Nevertheless, a package with visualization support can certainly help to speed up the adoption process of using PDES languages and libraries.

The fact that only CPSim uses a conservative protocol reflects the general consensus that implementing a time-warp optimistic protocol requires much effort. More emphasis is therefore placed by the research community on developing optimistic libraries. This is not to say that languages and libraries implemented using a conservative protocol are not needed, but that most users would be more willing to code a conservative protocol engine from scratch, rather than a time-warp one. However, to the user outside the PDES community, to code a conservative PDES engine would still be a formidable task. Hence, more libraries based on a conservative protocol should be developed to further promote the use of PDES in the sequential simulation community.

An interesting development in recent years is the United State Department of Defense (DoD) High Level Architecture (HLA) [57] effort to “provide a specification of a common technical architecture for use across all classes of simulations.” In particular, the Runtime

Table 6. Selected performance measurement for different PDES libraries

Package	Application	Platform	Performance
GTW	PCS network simulation [43]	42-CPU KSR-2	speedup of 38, with 335,000 committed events per second
PSK	Telecommunication network [49]	8-CPU SparcCenter 2000	speedup of 4
SPaDES	Torus network message routing [54]	Fujitsu AP3000 with 32 143MHz Sun Ultra Sparc processors	speedup ranging from 0.03 to 4.59 on 16 processors
SPEEDES	Queueing network [48]	Caltech.JPL MarkIII 64-node hypercube	1.5-1.9 relative speedup every time the number of nodes doubles
TWOS	Theater-level combat simulation [55]	60-node BBN Butterfly GP1000	speedup of 24
WARPED	Synthetic model [56]	Sun SparcCenter 1000 with 4 processors	6500 events/sec event rate on 100 simulation objects

Infrastructure (RTI) component of the HLA provides a distributed operating system to support simulation interaction and management. Synchronization mechanism for event ordering is provided in the time management component of the HLA [58]. By expanding its services to include repeatable execution and ordering of simultaneous events [59], HLA provides a framework into which research in the PDES community can be readily applied onto real-world systems.

Most of the PDES libraries surveyed offer the user a whole range of options to choose from if the optimistic protocol is used. The different kinds of runtime options offered by these optimistic PDES libraries allow the user to fine-tune the performance of the simulation application. However, to someone new to PDES, these options may mean nothing to them. All they are concerned with is how a PDES engine is going to make his/her application run faster, or run with a larger model. Research into automatically setting these options during runtime would make the PDES libraries more readily acceptable by new users to PDES.

As has been summarized in Bagrodia's 1994 survey [60] and also evident in the above discussion, the research directions for PDES languages and libraries are still mainly in the areas of application partitioning support, automatic mapping and load balancing, and program development environments (including GUI for model construction, debugging,visualization, and statistics collection). It is believed that PDES languages and libraries will remain as an active research area in the years to come. Therefore, this survey could help the researchers in this area to understand how the present situation has come about, what was tried, what worked and what did not.

7. References

[1] Jain, S. "Virtual Factory Framework: A Key Enabler for Agile Manufacturing." In *Proceedings of 1995 INRIA/IEEE Symposium on Emerging Technologies and Factory Automation*, pp 247-258, IEEE Computer Society Press, 1995.

[2] Fujimoto, R.M. "Parallel Discrete-Event Simulation: Will the Field Survive?" *ORSA Journal on Computing*, Vol. 5, No. 3, pp 213-230, Summer 1993.

[3] Nicol, D. "Parallel Discrete-Event Simulation: So Who Cares?" Keynote Speech, 11th Workshop on Parallel and Distributed Simulation (PADS'97), June 1997.

[4] Rajaei, H., Ayani, R. "Design Issues in Parallel Simulation Languages." *IEEE Design and Test of Computers*, pp 53-63, Dec. 1993.

[5] Wonnacott, P., Bruce, D. "The APOSTLE Simulation Language: Granularity Control and Performance Data." In *Proceedings of 10th IEEE/ACM/SCS Workshop on Parallel and Distributed Simulation (PADS'96)*, pp 114-123, May 1996.

[6] Bagrodia, R. *PARSEC User Manual, Release 1.0*, UCLA Parallel Computing Lab, 1997.

[7] Edward, M., Felipe, K., Veron, R. "ParaSol: A Multithreaded System for Parallel Simulation Based on Mobile Threads." In *Proceedings of 1995 Winter Simulation Conference*, pp 690-697, Arlington, VA, Dec. 1995.

[8] Teo, Y.M., Tay, S.C., Kong, S.T. "SPaDES: An Environment for Structured Parallel Simulation." Technical Report, TR20/96, Department of Information Systems and Computer Science, National University of Singapore, Lower Kent Ridge Road, Singapore, Oct. 1996.

[9] Dale, E.M., Wilsey, P.A., Timothy, J.M. *WARPED Simulation Kernel Documentation (Version 0.5)*, Sept. 1995.

[10] Special issue on TeD, *Performance Evaluation Review*, Vol. 25, No. 4, March 1998.

[11] Low, Y.H., Lim, C.C., Cai, W., Huang, S.Y., Hsu, W.J., Jain, S., Turner, S.J., "Survey of Languages and Runtime Systems for Parallel Discrete-Event Simulation." *Upstream Project Technical Report*, Gintic Institute of Manufacturing Technology, Singapore, Jan. 1998.

[12] Ferscha, A. "Parallel and Distributed Simulation of Discrete-Event Simulation." In *Handbook of Parallel and Distributed Computing*, McGraw-Hill, 1995.

[13] Fujimoto, R.M., "Parallel Discrete-Event Simulation." *Communications of the ACM*, Vol. 33, No. 10, pp 31-53, Oct. 1990.

[14] Misra, J., "Distributed Discrete-Event Simulation." *ACM Computing Surveys*, Vol. 18, No. 1, pp 39-65, 1986.

[15] Cai, W., Turner, S.J. "An Algorithm for Distributed Discrete-Event Simulation - The 'Carrier Null Message' Approach." In *Proceedings of the SCS MultiConference on Distributed Simulation*, Vol. 22, No. 1, pp 3-8, Jan. l990.

[16] Nicol, D. "Performance Bound on Parallel Self-initiating Discrete-Event Simulations." *ACM Transactions on Modeling and Computer Simulation*, Vol. 1, No. 1, pp 24-50, Jan. 1991.

[17] Cai, W., Letertre, E., Turner, S.J. "Dag Consistent Parallel Simulation: a Predictable and Robust Conservative Algorithm." In *Proceedings of 11th Workshop on Parallel and Distributed Simulation (PADS'97)*, pp 178-181, Lockenhaus, Austria, June 1997.

[18] Jefferson, D. "Virtual Time." *ACM Transactions on Programming Languages and Systems*, Vol. 7, No. 3, pp 404-425, 1985.

[19] Ferscha, A., Johnson, J., Turner, S.J. "Early Performance Prediction of Parallel Simulation Protocol." In *Proceedings of the 1st World Congress on System Simulation*, pp 282-287, Aug. 1997.

[20] Turner, S.J., Lim, C.C., Low, Y.H., Cai, W., Hsu, W.J., Huang, S.Y. "A Methodology for Automating the Parallelisation of Manufacturing Simulations." In *Proceedings of the 12th Workshop on Parallel and Distributed Simulation (PADS'98)*, pp 126-133, May 1998.

[21] Lomow, G., Baezner, D. "A Tutorial Introduction to Object-Oriented Simulation and Sim++." In *Proceedings of the 1991 Winter Simulation Conference*, pp 157-163, 1991.

[22] Baezuer, D., Lomow, G., Unger, B. "Sim++: The Transition to Distributed Simulation." In *Proceedings of the 1990 SCS MultiConference on Distributed Simulation*, D. Nicol, editor, Vol. 22, No. 2, pp 211-218, San Diego, CA, Jan. 1990.

[23] Bagrodia, R. and Liao, W.-T. *Maisie User Manual, Release 2.2*, UCLA Parallel Computing Lab, 1995.

[24] J. Waldorf, Bagrodia, R. "MOOSE: A Concurrent Object-Oriented Language for Simulation." *International Journal of Computer Simulation*, Vol. 4, No. 2, pp 235-257, 1994.

[25] J. West, Mullarney, A. "ModSim: A Language for Distributed Simulation." In *Proceedings of SCS MultiConference on Distributed Simulation*, pp 155-159, 1988.

[26] Peter, L.R. "Parallel Simulation Using the Time Warp Operating System." In *Proceedings of 1990 Winter Simulation Conference*, Dec. 1990.

[27] CACI. *MODSIM II—The Language for Object-Oriented Programming: Reference Manual*, CACI Products Company, l993.

[28] Preiss, B.R. "YADDES—Yet Another Distributed Discrete-Event Simulator: User Manual." Technical Report, Department of Electrical and Computer Engineering, University of Waterloo, Canada, 1990.

[29] Mathewson, S. "Simulation Program Generators." *SIMULATION*, Vol. 23, No. 4, pp 181-189, 1974.

[30] Lieberman, H. "Using Prototypical Objects to Implement Shared Behavior in Object-Oriented Systems." In *Proceedings of 1st Conference on Object-Oriented Programming: Systems, Languages and Applications*, pp 214-223, Portland, USA, 1986.

[31] Turner, S.J. "Simulating Timed Petri Net using the APOSTLE Language." Research Report 382, Department of Computer Science, University of Exeter, United Kingdom, 1998.

[32] Evans, J.B. *Structures of Discrete-Event Simulation: An Introduction to the Engagement Strategy*, Ellis Horwood Ltd., Chichester, 1988.

[33] Rich, D.O., Michelsen, R.E. "An Assessment of the ModSim / TWOS Parallel Simulation Environment." In *Proceedings of the 1991 Winter Simulation Conference*, pp 509-518, 1991.

[34] Meyer, R., Bagrodia, R. "MVPE: Visual Design of Parallel Simulation Models." In *Proceedings of the Fourth International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp 227-230, San Jose, CA, Feb. 1-3, 1996.

[35] Bagrodia, R., Chen, Y., Jha, V., Sonpar, N. "Parallel Gate-level Circuit Simulation on Shared Memory Architectures."In *Proceedings of the Ninth Workshop on Parallel and Distributed Simulations (PADS'95)*, pp 170-174, Lake Placid, NY, June 1995.

[36] Bagrodia, R., Meyer, R., Takai, M., Chen, Y., Zeng, X., Martin, J., Park, B., Song, H. "Parsec: A Parallel Simulation Environment for Complex Systems." *IEEE Computer*, Vol. 31, No. 10, pp 77-85, Oct. 1998.

[37] Baezner, D., Rohs, C., Jones, H. "U.S. Army ModSim on Jade's TimeWarp." In *Proceedings of the 1992 Winter Simulation Conference*, pp 665-671, 1992.

[38] Bagrodia, R., Li, Z., Jha, V., Chen, Y., Cong, J. "Parallel Logic Level Simulation of VLSI Circuits." In *Proceedings of the 1994 Winter Simulation Conference*, pp 1354-1361, 1994.

[39] Bagrodia, R., Gerla, M., Kleinrock, L., Short, J., Tsai, T.-C. "A Hierarchical Simulation Environment for Mobile Wireless Networks." In *Proceedings of the 1995 Winter Simulation Conference*, pp 563-570, Arlington, VA, Dec. 1995.

[40] Abrams, M. "The Object Library for Parallel Simulation." In *Proceedings of the 1988 Winter Simulation Conference*, pp 210-219, 1998.

[41] Agre, J.R., Johnson, A., Tinker, P.A., Vopova, S. "Time Warp Object Oriented Distributed Simulation System (TWOODS)." In *Proceedings of Rockwell Software Engineering Symposium (SES) III*, Dallas, TX, pp 9.2.1-9.2.13, Oct. 1989.

[42] Groselj, B. "CPSim: A Tool for Creating Scalable Discrete-Event Simulations." In *Proceedings of 1995 Winter Simulation Conference*, pp 579-583, Arlington, VA, Dec. 1995.

[43] Das, S., Fujimoto, R.M., Panesar, K., Allison, D., Hybinette, M. "GTW: A Time Warp System for Shared Memory Multiprocessors." In *Proceedings of 1994 Winter Simulation Conference*, pp 1332-1339, Dec. 1994.

[44] Xiao, Z., "Preliminary Report on Warpkit Performance." Technical Report, Department of Computer Science, University of Calgary, Canada, March 1994.

[45] West, D., Mellon, L., Ramsey, J., Cleary, J. Hofmann, J. "Infrastructure for Rapid Execution of Strike-Planning Systems." In *Proceedings of the 1995 Winter Simulation Conference*, pp 1207-1214, Arlington, VA, Dec. 3-6, 1995,

[46] Ronngren, R., Liljenstam, M., Ayani, R., Montagnat, J. "A Comparative Study of State Saving Mechanism for Time Warp Synchronized Parallel Discrete-Event Simulation." *IEEE Proceedings of Simulation*, 1996.

[47] Gomes, F., Franks, S., Unger, B., Xiao, Z., Cleary, J., Covington, A. "SimKit: A High Performance Logical Process Simulation Class Library in C++." In *Proceedings of the 1995 Winter Simulation Conference*, pp 706-713, Arlington, VA, Dec. 1995.

[48] Steinman, J.S. "SPEEDES: A Multiple-Synchronization Environment for Parallel Discrete-Event Simulation." *International Journal in Computer Simulation*, Vol. 2, No. 3, pp 251-286, 1992.

[49] Lijenstam, M., Ayani, R. "MOBSIM++: An Environment for Parallel Simulation of PCNs." In *Proceedings of 1st World Congress on Systems Simulation*, pp 272-281, Singapore, Sept. 1997.

[50] D'Souza, L.M., Fan, X., Wilsey, P.A. "pGVT: An Algorithm for Accurate GVT Estimation." In *Proceedings of the 8th Workshop on Parallel and Distributed Simulation (PADS'94)*, pp 102-109, July 1994.

[51] Carothers, C.D., Topol, B., Fujimoto, R.M., Stasko, J.T., Sunderam, V. "Visualizing Parallel Simulations in Network Computing Environments: A Case Study." In *Proceedings of 1997 Winter Simulation Conference*, Atlanta, GA, pp 110-117, 1997.

[52] JPL. *Time Warp Operating System Version 2.5.1, User's Manual*, Jet Propulsion Laboratory, CA Institute of Technology, Pasadena, CA, Sept. 1991.

[53] Heath, M. and Etheridge, J. "Visualizing the Performance of Parallel Programs." *IEEE Software*, Vol. 8, No. 5, pp 29-39, Sept. 1991.

[54] Teo, Y.M., Tay, S.C. "SPaDES / C++ Distributed Simulation on the AP3000." *8th International Parallel Computing Workshop (PCW'98)*, pp 347-354, Singapore, Sept. 1998.

[55] Wieland, F., Hawley, L., Feinberg, A., Di Loreto, M., Blurne, L., Ruffles, J., Reiher, P., Beckman, B., Hontalas, P., Bellenot, S. "The Performance of a Distributed Combat Simulation with the Time Warp Operating System." *Concurrency: Practice and Experience*, Vol. 1, No. 1, pp 35-50, 1989.

[56] Balakrishnan, V., Frey, P., Abu-Ghazaleh, N., Wilsey, P.A. "A Framework for Performance Analysis of Parallel Discrete-Event Simulators." In *Proceedings of the 1997 Winter Simulation Conference*, Dec. 1997.

[57] Dahmann, J.S., Fujimoto, R.M., Weatherly, R.M. "The Department of Defense High Level Architecture." In *Proceedings of the 1997 Winter Simulation Conference*, Atlanta, GA, pp 142-149, 1997

[58] Fujimoto, R.M., Weatherly, R.M. "Time Management in the DoD High Level Architecture." *1996 Workshop on Parallel and Distributed Simulation*, May 1996.

[59] Fujimoto, R.M., "Zero Lookahead and Repeatability in the High Level Architecture." *1997 Spring Simulation Interoperability Workshop*, March 1997.

[60] Bagrodia, R. "Language Support for Parallel Discrete-Event Simulations." In *Proceedings of 1994 Winter Simulation Conference*, pp 1324-1331, 1994.

8. Additional Reading

Wonnacott, P., Bruce, D. "A Prototype Implementation of APOSTLE and its Performance." In *Proceedings of the l995 SCS Summer Computer Simulation Conference*, pp 197-205, l995.

Teo, Y.M., Tay, S.C., Kong, S.T. "SPaDES: A Parallel Simulation Environment." In *Proceedings of 1st World Congress on Systems Simulation*, pp 293-298, Singapore, Sept. 1997.

Yoke-Hean Low received his BASc in Computer Engineering in 1997 from the Nanyang Technological University, Singapore. He is currently a Research Associate at the Gintic Institute of Manufacturing Technology. His current project involves the use of parallel and distributed simulation in the modeling of a virtual factory. His research interests include parallel and distributed simulation, parallel programming tools and the Internet. He is a member of the Association for Computing Machinery and the Institute of Electrical and Electronics Engineers.

Lim Chu Cheow obtained his BSc in Mathematical and Computational Sciences and his MSc in Computer Science, both from Stanford University. He received his PhD in Computer Science from the University of California at Berkeley. He is a member of Phi Beta Kappa. He worked on the implementation of the Public Domain Compiler (V0.1) for the object-oriented language Sather at the International Computer Science Institute at Berkeley. While with the Defence Science Organization, he worked on the use of high-performance techniques in engineering and scientific applications. He is now a Research Fellow at Gintic Institute of Manufacturing Technology. His current project is applying parallel and distributed simulation techniques to models of a virtual factory and manufacturing supply chains. His research interests include high-performance computing (both system and algorithmic issues), compilers, object-oriented programming languages, simulations and the Internet.

Sanjay Jain is a Senior Research Fellow and the Manager of the Manufacturing Planning and Scheduling Group at Gintic Institute of Manufacturing Technology, Singapore. His research interests include using modeling and analysis techniques in the development and operation of manufacturing systems, and in improving performance of simulation systems through parallel and distributed execution. Prior to joining Gintic, he worked as a Senior Project Engineer with General Motors' North American Operations Technical Center. He earned a Bachelors of Engineering from the University of Roorkee, India, a Postgraduate Diploma from the National Institute for Training in Industrial Engineering, Bombay, India, and a PhD in Engineering Science from Rensselaer Polytechnic Institute, Troy, New York. Dr. Jain has contributed in the area of manufacturing scheduling and simulation at international conferences and in publications such as the Winter Simulation Conference, the INRIA / IEEE Symposium on Emerging Technologies and Factory Automation, and Rensselaer's Conference on Agile Integrated and Computer Integrated Manufacturing. He is a member of the Institute of Industrial Engineers.

Wentong Cai is currently an Associate Professor in the School of Applied Science, Nanyang Technological University, Singapore. He obtained his BSc in Computer Science from Nankai University, China, in 1985, and his PhD from the University of Exeter, United Kingdom, also in Computer Science, in 1991. Dr. Cai was a Post-Doctoral Research Fellow at Queen's University, Canada, and later joined Nanyang Technology University. Dr. Cai is a member of IEEE, and has published more than 50 refereed journal and conference articles in the areas of parallel processing. His current research interests include parallel and distributed simulation, parallel programming tools and environments, and performance analysis.

Shell Ying Huang obtained her BSc degree in Engineering and her PhD from the Department of Computing, Imperial College, London University. After working as a Teaching Assistant and then a Research Assistant at the Department of Computing, Imperial College, she joined the School of Applied Science, Nanyang Technological University, Singapore, as a Lecturer in 1991 and became a Senior Lecturer in 1998. Her research interests include task scheduling for parallel programs, parallel and distributed simulation, scheduling and routing for automated guided vehicles.

Hsu Wen Jing is a Faculty Member of the School of Applied Science, Nanyang Technological University (NTU), Singapore. He is currently the Director of the Centre for Advanced Information Systems, School of Applied Science, NTU. Before joining NTU, he was on the faculty of National Chiao Tung University, Taiwan, and Michigan State University, USA. He also worked as a Visiting Scientist at the IBM Palo Alto Scientific Center and IBM's T.J. Watson Research Center for two years. His present research interests are in the algorithms, architectures and applications of parallel computers, simulation and decision support, the scheduling and routing of automated guided vehicles. His technical papers have been published in reviewed journals and conferences. He is involved in a large-scale project funded by the National Science and Technology Board concerning computer simulations for manufacturing industries. He is also a consultant in industry-related projects, including the anchorage management and conflict prediction systems for the Maritime and Port Authority of Singapore.

Stephen Turner graduated from Trinity College, Cambridge, United Kingdom, with a BA in Mathematics and Computer Science in 1971 and received a PhD in Computer Science from the University of Manchester, United Kingdom, in 1979. He is currently a Senior Lecturer in Computer Science at the University of Exeter, where he directs the research of a group working in the area of distributed and parallel systems. His current research interests include parallel and distributed simulation, visual programming environments, parallel programming models, and agent technology. He served on the ACM/IEEE/SCS Parallel and Distributed Simulation (PADS) steering committee from 1994 through 1997 and is Program Chair of PADS '99.