Proceedings of the 1993 Winter Simulation Conference
G.W. Evans, M. Mollaghasemi, E.C. Russell, W.E. Biles (eds.)

PARALLEL HYBRID MODELS IN SYSTEM DESIGN

Chien-Chung Shen

Bellcore
331 Newman Springs Road
Red Bank, New Jersey 07701, U.S.A.

Rajive L. Bagrodia

Computer Science Department
UCLA
Los Angeles, California 90024, U.S.A.

ABSTRACT

A parallel hybrid model is a partially implemented design, where some components exist as simulation models and others as operational subsystems, which executes on a parallel[1] architecture. It supports an evolutionary design approach for complex, distributed systems such that an initial simulation model of the proposed design is iteratively refined and elaborated into an implemented system. It also supports hybrid simulation of physical modules and simulation models to facilitate rapid prototyping of complex, distributed systems. The paper describes the use of parallel hybrid models in system design, and discusses their execution issues including cyclic dependency, event causality, and real-time constraints.

1 INTRODUCTION

Our research is motivated by the following two observations.

(1) **Design Methodology**. The conventional *big bang* approach to designing complex, time-critical distributed systems usually consists of the following three steps: (a) given the specifications, constructing a simulation model to evaluate the functional and performance behavior of the proposed design, (b) implementing the simulation model in hardware and software components, and (c) testing the implemented system to ensure that it satisfies the specifications. These steps shall be repeated as many times as needed until the implemented system passes the test. However, this approach may result in the following problems: (a) the tasks of system modeling, design, and implementation are carried out independently, which is expensive due to various notions used by different activities, and (b) the decoupling of simulation model and its implemented system may introduce inconsistencies between them, which may potentially result in implemented system's dissatisfaction of specifications.

[1]The terms 'parallel' and 'distributed' have been used synonymously in the paper.

(2) **Rapid Prototyping**. In designing large scale distributed systems (such as a WAN), the cost constraint and complexity may prevent us from having all the resources available and building a complete system in the initial stages. One alternative is to have a prototype to study the behavior of the designed system. A prototype usually is a hybrid system which consists of physical, operational components executing concurrently with a simulated model of the unavailable components. Execution of such a hybrid system is called hybrid simulation. For the WAN example, a design may start with a few physical nodes and links which interact with a simulation model of the rest of the system. The use of hybrid simulation not only results in significant saving in terms of resources, but also increases accuracy and reality in evaluating the functional and performance behavior of the proposed design.

In light of the above discussion, we propose to use *parallel hybrid models* for system design, which is based on the notion of Partially Implemented Performance Specification proposed in Bagrodia and Shen (1991). A parallel hybrid model is a partially implemented design, where some components exist as simulation models and others as operational subsystems, which executes on a parallel architecture. It supports an evolutionary approach to system design, where a discrete-event simulation model is iteratively refined or elaborated into an operational system. During each stage of model refinement, an intermediate hybrid model may be executed to determine the functional and performance characteristics of the partially elaborated design. This allows functional and performance characteristics of a system to be estimated at an early stage in its design and subsequently monitored during critical stages of refinement and implementation. The accuracy of the estimation increases as more of the design is implemented.

What advantages may be derived from the hybrid model to system design? Use of the hybrid model implies that a separate functional and performance model does not have to be designed and maintained for the system. *The evolving design is its own model.* This can result in significant savings in terms of manpower and resources while ensuring that the evolving system is consistent with its model. Use of the hybrid model also facilitates rapid prototyping of complex systems so that a hybrid system can be developed in the early design stages under the given cost and complexity constraints. This allows system characteristics to be evaluated and design deficiency to be uncovered as early as possible.

This paper describes the use of parallel hybrid model in system design and discusses execution and implementation issues in its use. The next section describes the basic concepts of hybrid model and its relation to system design. Issues which arise in executing hybrid models on parallel architectures are discussed in section 3. Algorithms to execute hybrid models under real-time constraints are investigated in section 4, and section 5 is the conclusion.

2 HYBRID MODEL AND SYSTEM DESIGN

A distributed system consists of a collection of communicating sequential processes that execute concurrently on a number of processors linked by an arbitrary interconnection network. A processor may interleave the execution of multiple processes, and processes communicate exclusively via messages

In a hybrid model, each process in the physical system is either represented by an operational module (also called a *physical process* or *pp* for short) or abstracted by a (partially implemented) *logical process* (*lp*). Message communication among processes in the physical system is represented by message communications among the corresponding *pp* or *lp* in the hybrid model. A hybrid model uses the notion of *logical element* (*le*) to model hardware resources such as processors or communication channels. (Notice that symbols like *le*, *lp*, and *le* represent both singular and plural forms of the corresponding term.) Each *pp* and *lp* must be mapped to a specific *le*, and multiple *pp* and *lp* may be mapped to a common *le*. All *pp* and *lp* mapped to a common *le* are executed sequentially and those mapped to different *le* are executed in parallel. The hardware and software configurations of a physical system are thus directly represented in its hybrid model; moreover, alternative mappings may be easily tried in the hybrid model to evaluate their impact on system performance. *Virtual clocks* are used to model physical time. Each *le* is associated with a virtual clock. The value of a virtual clock denotes the current time at the corresponding *le*. A virtual clock is updated when a *pp* or *lp* mapped to the corresponding *le* receives a message and executes a computation step. The virtual clocks of different *le* are implicitly synchronized by the underlying algorithm used to execute hybrid models.

On receiving a message, a process executes either an operational step or a simulation step. An *operational step* refers to the statements executed by an operational module to process a message received by it. The physical time consumed by an operational step is measured by a physical clock and used to update the virtual clock of the corresponding *le*. In contrast, a *simulation step* models or simulates the activities that would be executed by the corresponding operational step. The physical time used to execute a simulation step is ignored; rather the virtual clock is updated by an interval estimated to be the physical time that would be required for the corresponding operational step. The use of simulation steps abstracts away the speeds of the underlying physical processors, and hence facilitates performance modeling when the physical architectures are not available. A *pp* executes only operational steps, whereas a partially elaborated *lp* executes an operational step in response to some messages and a simulation step in response to others. A simulation model is a special case of a hybrid model, where each process executes only simulation steps.

The use of hybrid models in system design is to iteratively transform each logical process to a physical process and eventually replace each logical element by a physical hardware. Given the functional, performance, and architectural specifications for a proposed system and an initial system design, an analyst first develops a simulation model of the software modules that have not yet been implemented and defines logical elements for the hardware processors that are not yet available. The simulation model estimates resource requirements of the physical object that it represents. The satisfaction of constraints imposed on the proposed design can be evaluated by executing the simulation model under the assumed operating environment. If the proposed design does not satisfy the specifications, it is modified appropriately to increase resources or attempt a different decomposition on the primary software modules. After being evaluated to satisfy specifications, the model is refined iteratively, where a refinement may be elaborating a simulation step into an operational step, or replacing a logical element by the physical hardware.

The above process of refinement and elaboration implies that at the intermediate stages, the model may contain some software modules that are (at least partially) operational, and the operational steps of these modules must be included in determining the overall functional and performance characteristics of the evolving system. This intermediate form of the model is referred to as a hybrid model. The refined hybrid model may again be evaluated to ensure that its functional and performance characteristics are satisfied. The hybrid model is refined iteratively, while its behavior is continuously monitored to ensures that the refinements do not violate the specifications. If so, appropriate modifications must be made to the assumed resource parameters, such as increasing processor speed. If the behavior of the refined model is satisfactory, the iterative refinement proceeds further. This process is repeated until the software models are transformed into operational statements, and the logical elements are replaced by actual target architectures.

3 EXECUTION OF HYBRID MODELS

3.1 Sequential Execution

In Bagrodia and Shen (1991), a centralized environment to execute hybrid models on a uni-processor is described, which is a straightforward adaptation of a sequential discrete-event simulation algorithm. As a hybrid model contains both simulated components and implemented components, the environment must be able to distinguish the execution of a simulation step from that of an operational step. The centralized environment uses two main data structures: a set of virtual clocks, one for each *le* in the hybrid model, and an event-list. Each virtual clock gives the time up to which the processes mapped to the associated *le* have been executed. The event-list is a partial order of timestamped messages. Messages from the event-list are delivered to a destination process in the order determined by their timestamps. The message timestamps are generated on the basis of the virtual clock[2] of the transmitting process. A special *timeout* message is defined to be scheduled by an *lp* for delivery to itself at a future time to simulate the physical time that would be required by the physical process to execute the corresponding operational step. When a timeout message is delivered to a process, its virtual clock is simply incremented by the duration of the simulation step specified in the timeout message. For other messages, the virtual clock is advanced by the physical time interval corresponding to the duration of the operational step executed by the process on a processor.

[2]We use the virtual clock of a process to mean the virtual clock associated with the *le* to which the process has been mapped.

3.2 Parallel Execution

The execution of a hybrid model on a distributed architecture represents a critical refinement step: replacing some *lp* and its associated *le* in the model (for instance, the process that models a communication network) by the actual hardware (use an operational network to transmit messages between entities). We also have an additional motive – validation. If the hardware architecture of the system being designed is (partially) available, it can be used to validate part of the hybrid model. For instance, assume that the communication network to be used in the implemented system is available. In this case, the *lp* and *le* that modeled the network may be removed from the hybrid model, and the hybrid model can use the available network to transmit messages. Measurement of the transmission times can be used to validate the model. It is important to note that replacing an *lp* by actual hardware may increase the elapsed time for execution of the hybrid model. (This follows because simulating a message transmission may require the execution of only a few instructions on a processor, while the time taken for actual transmission of the message over a network will be determined by the network itself.) Further, the refinement affects the performance characteristics of the hybrid model only to the extend that an *lp* is an abstraction of a actual component, and may not reproduce the exact behavior of the physical device (for instance, the network in our example).

Intuitively, a parallel hybrid model is executed on a distributed architecture by mapping each *le* in the model to a specific physical processor (referred to as a *pe*) in the distributed architecture, and by distributing the event-list of the parallel hybrid model among the multiple *pe*. This implies that although processes mapped to different *le* may execute on different *pe*, processes mapped to a common *le* are executed on the same processor. Distributed execution of a parallel hybrid model raises the following critical issues.

(1) **Logical Time vs. Physical Time**. Let **P** denote a distributed system that consists of a set of message-communicating processes and M_P denote a partial order of messages generated during the execution of processes in **P**. A parallel hybrid model of this system must generate a partial order of messages that is consistent with the partial order M_P. Consistency requires that if a message m_1 generated in the

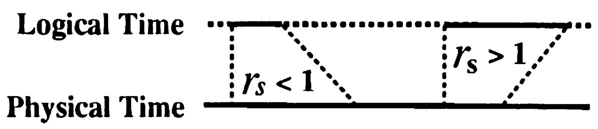

Figure 1: Rate of Progress for Simulation

parallel hybrid model has a timestamp t_1, then a corresponding message m_1' must be transmitted in the physical system at time t_1. Messages in a simulation model are timestamped on the basis of a logical clock, which is decoupled from the physical clock. We define a ratio r_s to represent the (average) rate of progress for a simulation:

$r_s = s_t/p_t$, where
s_t = duration of a simulation step, and
p_t = physical time required by the execution engine to simulate the system for interval s_t

Depending on the application being simulated, r_s may be smaller than one (eg. simulation of a VLSI circuit or a large telephone switching system) or greater than one (eg. sparse traffic simulations), as shown in Figure 1.

In a parallel hybrid model, message timestamps may be generated from a simulation clock and/or the physical clock. For execution of a parallel hybrid program, we define r_s as above and another ratio r_c as follows:

$r_c = c_t/p_t$, where
c_t = duration of an operational step, and
p_t = physical time required to execute the operational step

Typically ratio r_c is 1 as the physical time required to execute an operational step determines its duration. However, r_c may be less than one if the hardware to be used in the operational system is expected to be faster than that used to execute the hybrid model, or be greater than 1 if the operational hardware is expected to be slower. A parallel hybrid model may consist of simulation and operational components that have a variety of r_s and r_c ratios. The incompatible rates of progress among different components complicate the algorithm used to execute parallel hybrid models.

(2) **Cyclic Dependency**. Consider a system that consists of two interacting, concurrent physical components pp_a and pp_b. Assume that the hardware for the operational system is available and consists of a network of two *pe*. In the parallel hybrid model of this system, pp_a is an operational module, whereas pp_b exists as a simulation module lp_b. The hybrid

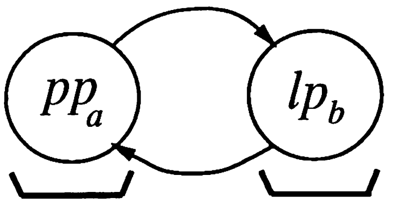

Figure 2: Cyclic Dependency

model is executed on the available hardware by executing pp_a and lp_b on different *pe*, as shown in Figure 2. If pp_a and lp_b are not cyclically dependent on each other, messages for pp_a may be generated (with appropriate timestamps) in an off-line mode by lp_b and then be input by pp_a as desired. However, the presence of cyclic dependencies makes the execution of parallel hybrid model non-trivial.

The cyclic dependencies may be unraveled by an iterative computation method based on checkpointing and rollback as follows. A distributed execution checkpoints each *pp* and *lp* periodically. The hybrid model on each *pe* is executed without explicit synchronization with the other *pe*. If the run-time system of a local *pe* receives a message from (a process resident on) a remote *pe* and the timestamp on the message is smaller than the timestamp on the last message processed on the local *pe*, the hybrid model on the local *pe* must be rolled back and recomputed from an appropriately checkpointed state. This procedure is repeated until eventually the computation reaches a fixed-point where further execution does not change states.

(3) **Real-Time Constraint**. In a parallel hybrid model, the execution of physical components may impose real-time constraints on simulation modules such that the simulation of an event must be completed by some deadline to make the execution of the physical component correct. This is typically the case when applying parallel hybrid models to design hard real-time systems. Real-time constraints may also appear when the nature of the physical components disallows checkpointing and rollback, where the physical component must explicitly demand timely processing of events by the simulation model to prevent any late arrival of messages.

Consider a hybrid model of a real-time system which consists of two pairs of interacting, concurrent components (pp_a, lp_c) and (pp_b, lp_d), where lp_c and lp_d execute on the same processor, as shown in Figure 3. The execution of physical components pp_a and pp_b imposes real-time constraints on simulation modules lp_c and lp_d. Due to the competitive nature of lp_c and lp_d as they execute on the same *pe*, some schedul-

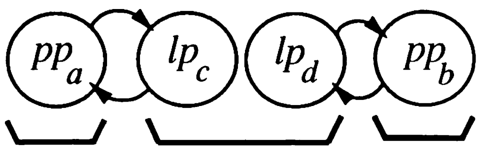

Figure 3: Real-Time Constraint

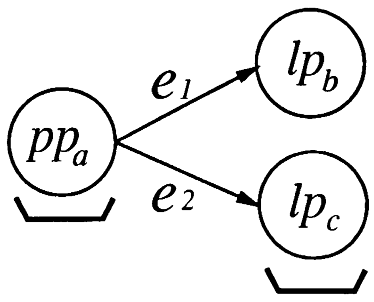

Figure 4: Event Causality

ing algorithm is needed to schedule their execution accordingly to satisfy real-time constraints.

(4) **Event Causality**. The correct execution of parallel hybrid models requires that events are executed in the order of their dependencies. Event dependency is a partial order relation, which is typically determined by (logical) timestamp values associated with events. The use of an event-list maintains a total order of events, which guarantees that events are executed in the correct order. However, in order to satisfy some real-time constraints imposed on events, events from the event-list will be executed in some reverse order.

Consider a parallel hybrid model that consists of three concurrent components pp_a, lp_b, and lp_c, where lp_b and lp_c are executed on the same processor, as shown in Figure 4. Two independent events, e_1 and e_2, are scheduled by pp_a. Assume that e_1 has a smaller timestamp than e_2, while e_2 has a tight deadline than e_1. In order to satisfy the real-time constraints, e_1 and e_2 are executed in reverse order.

4 HYBRID MODELS AND REAL-TIME CONSTRAINTS

Ghosh, Fujimoto, and Schwan (1993) have studied the suitability of using Time Warp mechanism to perform simulation with real-time constrains. They showed that Time Warp using lazy cancellation can meet real-time deadlines, while scheduling guarantees cannot be made even in the absence of false event when aggressive cancellation is used. In contrast, we investigate existing real-time scheduling algorithms and use them to execute hybrid models with real-time constraints, whenever applicable. In this section, we state our assumptions about hybrid models and describe appropriate real-time scheduling algorithms.

In executing hybrid models with real-time constraints, execution of events in timestamp order, although guarantees event causality, may cause real-time deadlines to be missed. Therefore, the idea is to execute events according to their real-time deadlines without violating any causality constraint. This implies that events may be processed in non-timestamp order to meet real-time requirements while satisfying causality constraints.

In the paper, we assume that for a set of time-constrained events, there is enough execution resource to process them so that events are executed to satisfy both real-time and causality constraints. The contrasting *imprecise computation model* (Liu et al. 1991), for the case when hybrid models execute on slower processors and hence real-time constraints cannot be guaranteed, may change the semantics of the hybrid models and is under further investigation. We define the notion of *feasible event set* as follows.

Definition 1 An event set E is said to be *feasible* if there exists an execution such that all its deadlines are met without violating event causalities.

We now apply existing real-time scheduling algorithms to execute hybrid models with real-time constraints. In the current stage of research, the following two categories of events are considered.

1. **Periodic Events**. A set E of n independent periodic events scheduled by the *rate-monotonic algorithm* (Liu and Layland 1973) is feasible if the resource utilization of the tasks is less than $n(2^{1/n} - 1)$. Given a set of independent periodic events, the rate-monotonic scheduling algorithm assigns a fixed priority to each event, where the shorter the period, the higher the priority.

2. **Aperiodic Events**. In the case that all events in an event set E are aperiodic and independent, the *earliest deadline first* and the *least laxity first* scheduling algorithms (Dertouzos and Mok 1989) guarantee a feasible schedule. Earliest deadline scheduling algorithm executes an event whose deadline is the closest, while least laxity scheduling algorithm executes an event which has the smallest laxity.

The scheduling algorithms used to execute hybrid models are preemptive and priority-driven. This means that whenever there is an event that has a higher priority than the one currently being processed, the executing event is immediately interrupted and the higher priority event is started being processed. Thus the specification of such an algorithm amounts to the specification of policy that assigns priorities to events. In the case of independent periodic events, the execution algorithm gives each event a fixed priority and assigns higher priority to events with shorter periods. For aperiodic events, higher priorities are given to events with earliest deadline or least laxity.

5 CONCLUSION

In the paper, we propose to use parallel hybrid models in system design. A parallel hybrid model is a partially implemented design, where some components exist as simulation models and others as operational subsystems, which executes on a parallel architecture. Hybrid models not only supports an evolutionary design approach for complex, time-critical systems, but also facilitate rapid system prototyping by hybrid simulation. We also identify several critical issues which arise when executing hybrid models on distributed architectures. Answers to these issues will shed light on system integration, hybrid, and real-time simulation.

We also discuss the execution of hybrid models with real-time constraints. In the paper, we assume that a feasible schedule exists for an independent event set, and apply existing real-time scheduling algorithms to execute the events to meet real-time requirements. However, the causality constraints which determine the order of event execution in a hybrid model are in general complex and data dependent. Research is in progress to apply other real-time scheduling algorithms (Sha and Goodenough 1990) to execute hybrid models.

REFERENCES

Bagrodia, R. L., and C.-C. Shen. 1991. MIDAS: Integrated design and simulation of distributed systems. *IEEE Transactions on Software Engineering* 17:1042-1058.

Dertouzos, M. L., and A. K.-L. Mok. 1989. Multiprocessor on-line scheduling of hard-real-time tasks. *IEEE Transactions on Software Engineering* 15:1497-1506.

Ghosh, K., R. Fujimoto, and K. Schwan. 1993. Time warp simulation in time constrained systems. In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation (PADS93)* , ed. R. Bagrodia and D. Jefferson, 163-166, San Diego, California.

Liu, C. L., and J. W. Layland. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of ACM* 20:46-61.

Liu, J. W. S., K. J. Lin, W. K. Shih, A. C. Yu, J. Y. Chung, and W. Zhao. 1991. Algorithms for scheduling imprecise computations. *IEEE Computer* 5:58-68.

Sha, L., and K. B. Goodenough. 1990. Real-time scheduling theory and Ada. *IEEE Computer* 4:53-62.

AUTHOR BIOGRAPHIES

CHIEN-CHUNG SHEN is a Member of Technical Staff at Bellcore. He received B.S. and M.S. degrees in computer science from National Chiao Tung University, Taiwan, in 1982 and 1984, respectively, and a Ph.D. degree in computer science from UCLA in 1992. His research interests include network operations and management, distributed real-time systems, programming language semantics and verification.

RAJIVE L. BAGRODIA is an Associate Professor in the Computer Science Department at UCLA. He received a B.Tech. degree in electrical engineering from the Indian Institute of Technology, Bombay in 1981, and M.A. and Ph.D. degrees in computer science from the University of Texas at Austin in 1983 ans 1987, respectively. His research interests include parallel languages, distributed algorithms, distributed simulation, and software design methodologies.