Parallel Discrete Event Simulation

Dr.N.Sairam & Dr.R.Seethalakshmi
School of Computing,
SASTRA Univeristy,
Thanjavur-613401.

Contents

1. Parallel Discrete Event Simulation..... 3
1.1 Parallel Discrete Event Simulation..... 4
1.2 Conservative Synchronization..... 6
1.3 Optimistic Synchronization..... 7
1.4 Time Warp Algorithm..... 7

1. Parallel Discrete Event Simulation

Parallel discrete event simulation is nothing but the discrete event simulation on parallel computers**.** Parallel discrete event simulation has dominated the serial execution in reducing the time and memory. It is used to solve large-scale composite models. Simulation systems are nothing but models of mathematical and logical relationships. In simulation, computers are used to evaluate the models numerically, where data is gathered and used to estimate the behavior of target systems. A simulation model depicts the state evolution over time. The target system can be viewed as a continuous system, discrete system or as a hybrid system.

In continuous systems, state changes continuously with respect to time. In discrete system the state changes only at specific period of time. The hybrid system is a combination of both continuous system and discrete system. There are two simulation methods available. They are time driven simulation and event driven simulation. In a time driven (or time-stepping) simulation, time is measured at small intervals giving the impression that the system evolves continuously over time. As such, it is naturally more suitable for simulating continuous systems. In an event-driven (or discrete-event) simulation, time leaps through distinct points in time, which we call events. Consequently, event-driven simulation is more appropriate for simulating discrete systems. Note that one can combine both types of simulations, for example, in a computer network simulation, using discrete events to represent detail network transactions,
such as sending and receiving packets, and using continuous simulation to capture the fluid dynamics of overall network traffic.

A discrete-event simulation maintains a data structure called the event-list, which is basically a priority queue that sorts events according to the time at which they are scheduled to happen in
the simulated future. A clock variable T is used to denote the current time in simulation. At the heart of the program is a loop; the simulator repeatedly removes an event with the smallest timestamp from the event list, sets the clock variable T to the timestamp of this

event, and processes the event. Processing an event typically changes the state of the model and may generate more future events to be inserted into the event-list. The loop continues until the simulation termination condition is met, for example, when the event-list becomes empty or when the simulation clock has reached a designated simulation completion time.

1.1 Parallel Discrete Event Simulation

Parallel Discrete Event Simulation (PDES) is a research area is parallel simulation and high performance computing. PDES is nothing but execution single event simulation on parallel systems like clusters, shared memory multiprocessors or a combination of both. As such, PDES can bring substantial benefit to time-critical simulations, and simulations of large-scale systems that demand an immense amount of computing resources. The simplest form of parallel simulation is called replicated trials, which executes multiple instances of a sequential simulation program concurrently on parallel computers. This approach has the obvious advantage of simplicity and it can expedite the exploration of a large parameter space. The disadvantage is that each replicated trial does not provide any speed up and cannot overcome the memory limit due to sequential execution. To address the latter problem, Hybinette and Fujimoto introduced a cloning method as an efficient parallel computation technique to allow simultaneous exploration of different simulation branches resulted from alternative decisions made in simulation. Another form of parallel simulation is to assign different functions of a simulation program, such as random number generation and event handling, to separate processors. This method is called functional decomposition. The main problem is the lack of ample parallelism. Also, the tight coupling of the simulation functions creates an excessive demand for communication and synchronization among the parallel components, which can easily defeat the parallelization effort.

More generally, one can view simulation as a set of state variables that evolve over time. Chandy and Sherman presented a space-time view of simulation, where each event can

be characterized by a temporal coordinate, indicated by the the timestamp of the event, and a spatial coordinate, indicated by the location of the state variables affected by the event. Accordingly, the state space of a discrete-event simulation can be perceived as consisting of a continuous time axis and a discrete space axis; the objective of the simulation is therefore to compute the value at each point in the space-time continuum. This space-time view provides a high-level unifying concept for parallel simulation, where one can divide the space-time graph into regions of arbitrary shape and assign them to separate processors for parallel processing. For example, Bagrodia, Chandy, and Liao presented a distributed algorithm using fixed-point computations.

A special case of the space-time view is the time-parallel approach, which is based on temporal decomposition of the time-space continuum. **Time- parallel** simulation divides the space-time graph along the time axis into non-overlapping time intervals, and assigns them to different processors for parallel processing. Due to the obvious dependency issue, that is, the initial state of a time interval must match the final state of the preceding time interval, the efficiency of this approach relies heavily on the model's ability of either rapidly computing the initial state or achieving fast convergence under relaxation. For this reason, only a limited number of cases
using time-parallel simulation exist in the literature. Successful examples include trace-driven cache simulations, queuing network and Petri net simulations, and road traffic simulations.

Orthogonal to the time-parallel approach, **space-parallel** simulation is based on data decomposition, where the target system is divided into a collection of subsystems, each simulated by a logical process (LP). Each LP maintains its own simulation clock and event-list, and is only capable of processing events pertaining to the subsystem to which it is assigned. These LPs can be assigned to different processors and executed concurrently. Communications between the LPs take place exclusively by exchanging time stamped events. Space-parallel simulation is in general more robust than the other parallelization approaches, mainly because data decomposition is naturally applicable to most models.

In discrete event simulation the events must be processed in the non decreasing time stamp order. If the events with small timestamp affects the events with a larger time stamp it is referred to as the causality constraint. In the case of parallel simulation, there are a lot of event lists. Each logical process has its own event list and the simulation clock to handle the simulation. Each Logical Process has to maintain its own event list and timestamp ordering. Here the fundamental challenge is therefore associated with the difficulty of preserving the local causality constraint at each LP without the use of a global simulation clock.

1.2 Conservative Synchronization

The first parallel simulation synchronization protocol is the CMB algorithm proposed independently by Chandy and Misra, and Bryant. In CMB,LPs are connected via directional links, through which events are transferred from one LP to another in chronological order (with non decreasing time stamps). Events are enqueued at the receiving LPs—there is one input
queue for each incoming link at an LP. Also, each input queue is associated with a clock variable, set to be either the timestamp of the first event in the queue, or, if the queue is empty, the timestamp of the last processed event (or zero initially). Each LP maintains a loop: at each iteration, the LP selects an input queue with smallest clock value and processes the event
at the beginning of the input queue; if the input queue is empty, the LP blocks until an event arrives at the queue and then continues at the next iteration. Since LPs block on empty input queues (with the smallest clock value),deadlock may happens once a waiting cycle is formed, in which case no progress will be made even if there are events in other input queues. The
CMB algorithm uses null messages to avoid this pathological situation. A null message does not represent any real activities in the model; it carries only a timestamp and is regarded as a guarantee from the sending LP that it won’t send events in the future with

timestamps smaller than the time stamp of the null message. Upon receiving a null message, an LP can advance the clock associated with the input queue. The LP can further propagate the time advancement to its successor LPs, possibly by sending out more null messages. Consequently, no waiting cycle is formed and deadlock is avoided. It is important to note that the use of null messages is not the only way to prevent deadlocks. Alternatively, one can allow deadlock to happen, and subsequently detect and recover from deadlock situations. Deadlock recovery is based on the observation that events with the smallest time stamp in the system can always be processed safely. In cases where deadlock shappen more frequently, however, this may result in a significant amount of sequential execution that can adversely affect the overall performance.

1.3 Optimistic Synchronization

Optimistic synchronization enforces the local causality constraint differently from its conservative counterpart: an LP is allowed to process events that arrive in the simulated past, as long as the simulation is able to detect such causality errors, rewind the simulation clock, and roll back the erroneous computations.

1.4 Time Warp Algorithm

In Time Warp, each LP saves the events received from other LPs in the input queue, and those sent to other LPs in the output queue. Also, the state variables are saved in the state queue each time before an event is processed. When an event arrives with a timestamp smaller than the current simulation clock (called the straggler event), the LP must be rolled back to the saved state immediately before the timestamp of the straggler event. All actions that the LP might have affected on other LPs later than the timestamp of the straggler event must also be canceled. This can be achieved by sending anti-messages corresponding to the original messages that are stored at the output queue. Upon receiving an anti-message, the LP will annihilate the corresponding message in the input

queue. If the anti-message carries a timestamp smaller than the LP's current simulation clock (and thus become a straggler event), the LP will also be rolled back accordingly. PDES has been applied in many areas, such as military applications (including war games and training exercises), on-line gaming, business operations, manufacturing, logistics and distribution, transportation, computer systems and computer networks. Thus PDES plays an important part in computing.