Flattened Conservative Parallel Simulator for DEVS and CELL-DEVS

Shafagh Jafer, Gabriel Wainer
{sjafer,gwainer}@sce.carleton.ca
Dept. of Systems and Computer Engineering
Carleton University Centre of Visualization and Simulation (V-Sim),
1125 Colonel By Dr. Ottawa, ON, Canada.

Abstract

Synchronization as the key to parallel and distributed computing requires a robust mechanism to handle communication among concurrent processes. In this paper we propose a flattened parallel simulator for DEVS and Cell-DEVS based on the classical null-message and lookahead based synchronization mechanism. We first present how flattening the architecture reduces communication overheads among participating nodes. Then we propose our blocking mechanism used to suspend the logical processes. After that we present our lookahead computation and null-message mechanism used to avoid causality errors as well as deadlock. Our conservative mechanism is implemented in WARPED kernel and can be used by any DEVS and Cell-DEVS parallel simulators who wish to adapt a conservative synchronization approach.

1. Introduction

Parallel and distributed simulation (PADS) techniques were proposed to resolve the issues of complex models simulation. Not only the shortage of resources, but also the long execution times brought up the idea of Parallel discrete event simulation (PDES) studies. Fujimoto [1] classifies three major research categories in the area of parallel and distributed simulation. The first research group is the high performance computing community which started in late 1970's and 1980's aiming at reducing execution time by using multiple processors. This community developed the world wide known fundamental ideas by proposing two synchronization algorithms: Chandy-Misra-Bryant [2, 3] and Time Warp [4]. The second group is the Defense community, which mainly focuses on facilitating interoperability and software reuse. Finally, the third group is the gaming and Internet community which is interested in developing realistic scenarios in distributed environments.

Among the existing modeling and simulation techniques, DEVS (Discrete Event System Specification) formalism [5] provides a discrete-event approach which allows construction of hierarchical models in a modular manner. In this work, our main focus is on discrete-event M&S approach and DEVS formalism which has been proven to be a universal formalism to represent DEDS (Discrete Event Dynamic Systems).

The Timed Cell-DEVS formalism [6] is an extension to the traditional Cellular Automata which makes use of DEVS by defining every cell to represent an atomic DEVS model and coupling them together to form a complete cell space representing a coupled DEVS model. Parallel Cell-DEVS formalism extends the standard formalisms of Cell-DEVS to allow a higher degree of parallelism in parallel and distributed environments. CD++ [7] is a modeling toolkit that implements the DEVS and Cell-DEVS theories by applying the original formalisms.

In this work we propose the implementation of the classical conservative synchronization mechanism in WARPED kernel [8] to support parallel and distributed simulation of DEVS and Cell-DEVS.

3. Previous Works

The first parallel version of CD++ [9] was based on a central synchronizer approach exploiting the parallelism inherent to the DEVS formalism. Under that scheme, a single root coordinator acts as a global scheduler for every node participating in the simulation. Based on this structure, all events with the same timestamp are scheduled to be processed simultaneously on the available nodes. The simulator introduced two different types of coordinators; master and slave to reduce inter-process communication. The simulation is carried out by DEVS processors which are of two types: simulator and coordinator. The simulator represents an atomic DEVS model, where the coordinator is paired with a coupled model. Aside from these two types of coordinators, a special coordinator namely, root coordinator was introduced. The root resided above the topmost coordinator. It was responsible for driving the simulation and advancing the virtual simulation time. It was also responsible for handling external events that arrive from the environment.

At the beginning of the simulation, one logical process (LP) would reside on each machine (physical process). Then, each LP would host one or more DEVS processors. Thus, not all of a coordinator's children would necessarily sit on the same LP. A coordinator would communicate with its child processors through intra-process messaging if they reside on the same LP, and through inter-process messaging if they were sitting on remote LPs. Figure 1 represents the scenario.

The Parallel CD++ runs the abstract simulator described in [9]. The DEVS processor (root coordinator, simulator, slave coordinator or master coordinator) were the simulation objects that ran on the available LPs. The simulator and master and slave coordinator were designed in such a way that upon receiving any messages, any other message that gets sent in response had the same timestamp. This was due to the fact that, the root coordinator was the only DEVS processor that advanced time by sending a new message with the time of the next imminent model or external event. In the scope of this abstract simulator, a message was only considered a straggler if its timestamp t was less than the local virtual time (LVT) of the receiving LP. An LP was allowed to receive multiple messages with a timestamp equal to its LVT. The only constraint that needed to be placed was that two or more events sent from a source object S to a destination object D shall preserve the same ordering upon arrival to D. Due to the existence of root being the central synchronizer, this parallel simulator could never produce a straggler message thus no synchronization mechanism at the LP level was used since the synchronization was provided by the application itself.

Thus, as described above, the parallel CD++ presented in [9] did not implement any of the two practical synchronization mechanisms namely, conservative and optimistic. This was the motivation for us to propose the first conservative parallel and distributed simulator for Cell-DEVS by extending the WARPED kernel [8] to support such synchronization algorithm. In the next section we will describe the details of the proposed conservative simulator.

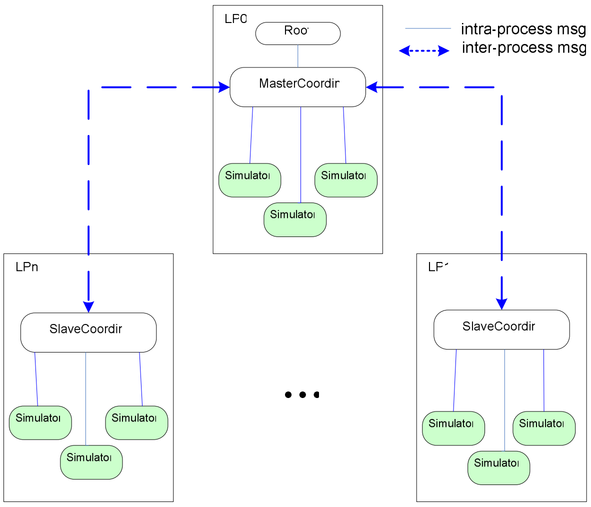

Figure 1. Hierarchical architecture of the first parallel CD++ simulator

4. Proposed Approach

The first issue that must be considered for implementing the proposed conservative simulator is to modify the hierarchical architecture of the simulator in [9] to a flattened one. Using a flat simulation mechanism rather than the traditional hierarchical one reduces the overhead in communication by reducing the number of exchange messages (especially inter-process message) to minimum. This is achieved by simplifying the underlying simulator structure, while keeping the same model definition and preserving the separation between model and simulator [10]. Researchers have shown that flat simulators outperform hierarchical ones significantly [10, 11, 12]. They have also showed that al-though the hierarchical simulator presented in [9] tried to reduce the communication overhead by introducing two specialized DEVS coordinators, but in some cases the communication overhead was still significantly high. The flattened architecture that will be presented here is depicted from [10] which was the first attempt to re-design the parallel CD++ simulator to adopt a flattened structure.

4.1. Flattened Architecture for the Proposed Conservative Simulator

To achieve our goals, the whole abstract simulator is redesigned to reflect the two major modifications; i.e. the departure from centralized root-based simulator to a conservative-based simulator, and flattening the structure of the simulator. As a result, a new Parallel DEVS simulator is implemented which deals with the communication overhead dilemma by using a flattened structure rather than the old hierarchical approach. The new flattened architecture would be as presented in Figure2.

As shown in the flattened architecture diagram, one LP is created on each machine encapsulating the DEVS processors. Only one Root is created on machine 0 (LP0) which interacts with other NCs using inter-process messaging (for remote NC) and intra-process messaging (for local NC). The Root coordinator is in charge of starting the simulation and performing I/O operations among simulation system and the surrounding environment. Only one NC is created on each machine and acts as the local central controller on its hosting LP. The NC is the parent coordinator for FC and routes remote messages received from the Root or from other remote NCs to the FC. The Simulators are the child processors of the local FC which represent the atomic components of DEVS and Cell-DEVS models. When a Simulator needs to communicate with a remote Simulator residing on another LP, it sends the message to its FC, then the message is forwarded to the NC above it. Once the message is at the NC, it will further be routed to the destination NC. There is no direct communication among Simulators; all messages must be forwarded to the parent FC. This is why the FC is known as the local central controller of its hosting LP.

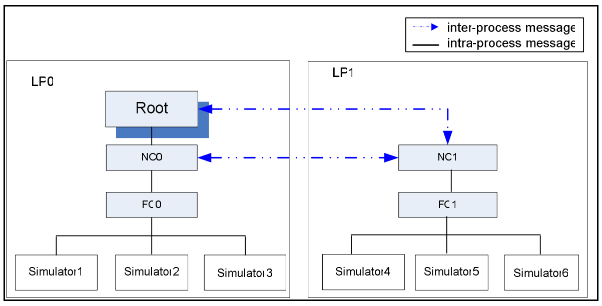

Figure 2. Flattened architecture for the proposed conservative simulator

4.2. Message definition

The processors exchange two categories of messages: content messages and control messages. The first category includes the external message (x) and the output message (y), and the second category includes the initialization message (I), the collect message (@), the internal message (*), and the done message (D). To describe these messages, external and output messages are used to exchange simulation data between the models, initialization messages start the simulation, collect and internal messages trigger the output and the state transition functions respectively in the atomic DEVS models, done messages handle synchronization by carrying the model timing information. The simulation is executed in a message-driven manner. Each type of PCD++ processor, defines its own receive functionality for each type of messages.

4.3. Layered architecture

Figure 3 illustrates the layered architecture of the proposed conservative CD++ simulator, where each layer only depends on the layers below it.

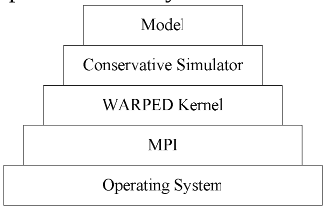

Figure 3. Layered architecture of the proposed conservative CD++ simulator

In order to implement the classical conservative algorithm for the proposed simulator, we have used the WARPED kernel [13, 8] which is a public domain simulation kernel originally developed at the University of Cincinnati. WARPED provides an implementation of Jefferson's original Time Warp algorithm [4]. The WARPED kernel is an attempt to make a freely available Time Warp simulation kernel that is easily ported, simple to modify and extend, and readily attached to new applications. For the purpose of our research, we have only used the services provided by WARPED to facilitate the distribution of the simulation executives on the nodes. We have not used the functionalities that implement the Time Warp algorithm such as state saving and rollback mechanism. Since our purpose is to build a conservative simulator, we have only used WARPED as the middleware that supports creation of model objects (simulation objects) as entities which exchange messages (time-stamped events) with each other and respond to events by applying them to their internal stats. Thus, the kernel was used to provide functionalities for sending and receiving events by simulation objects. The kernel also provides a simple definition of time (which can be redefined by the user) and functions to perform consistent I/O operations.

Our implementation of the conservative mechanism resides on the WARPED kernel layer and is separated from the core simulator code. Therefore it can be viewed as new extension to WARPED kernel which can be used by other researches who would like to use WARPED kernel as a conservative synchronization middleware for their DEVS-based simulator.

5. Design of the Conservative CD++ Simulator

Conservative synchronization approaches were the first synchronization algorithms proposed in the late 1970s by R. E. Bryant [2], K. M. Chandy and J. Misra [14]. This synchronization technique which is known the Chandy-Misra-Bryant (CMB) algorithm, disallows any occurrence of causality errors. In conservative schemes, if a LP has an unprocessed event with timestamp *t* and it is guaranteed that no event with earlier timestamp can be received, then the probability that causality error may happen is zero. When the LP has a list of unprocessed events from all other LPs it can safely process the event with lowest timestamp because the future events will for sure have larger timestamps. As long as there are unprocessed events from all other LPs, then this cycle can be repeated and synchronization is guaranteed. However, if this condition is not met, then there is a risk of deadlock. Technique to resolve this deadlock is to find the model's *lookahead*, which provides the smallest time stamp of the new events that a process can schedule in the future. Null messages are responsible to carry out the lookahead information among LPs. This way each LP, based on the lookahead information that it receives from all other LPs can derive a lower bound on the time stamp (LBTS) of the events that it will receive in future. As a result, the LP would know which event is safe to process. An example of a safe lookahead value is the timestamp of the first unprocessed event in the input queue. The main drawback of the conservative synchronization approach is the time-wasting flow of null messages which degrade the simulation performance significantly.

In the following sections we will describe the design details for our conservative simulator.

5.1. Scheduling mechanism

As we mentioned in the previous section, we use the WARPED kernel as the middleware which provides basic mechanism for handling scheduling at each LP. Every LP has a single Lowest Time Stamp First (LTSF) scheduler and a single input queue which holds all the input messages for all the local simulation objects (local simulators). This single inputQ is therefore, the unprocessed events queue for each LP. Aside from input events that arrive from other LPs and the environment, the inputQ also holds those events that are sent from a local simulation object to another local object. This queuing mechanism provides an easy-to-handle messaging strategy which is only responsible for a single queue for all the simulation objects on an LP. Each simulation cycle starts by removing the first element of the inputQ and processing it by the destination simulation object. The DEVS coordinators explained in Section 4.1 are responsible for delivering the message to the corresponding local simulation object based on the *destination_id* field of the input event. On the other hand, when a simulation object residing on an LP sends a message to another remote object on a different LP, the communication manager is responsible for delivering this message by placing it into the inputQ of destination LP. This whole process of sending and receiving time stamped events among the simulation objects needs to be studied closely to avoid causality errors in order to have a reliable conservative parallel simulator. One important fact that must be considered is that in conservative mechanism each LP has its own local virtual clock (LVT) and thus if the synchronization is performed poorly, there is going to be causality error and the simulation results will be incorrect. Since in DEVS there are different messages exchanged among local and remote processing entities (i.e. coordinators and simulation objects) it is very important to first understand how these different types of messages are sent back and forth and if they only affect local simulation objects or remote ones. These messaging paradigms are described in details next.

5.2. Messaging paradigms

As was discussed earlier, in DEVS there are five different types of messages exchanged among simulator, flat coordinator, node coordinator (NC), and root coordinator (incase of LP0). Figure 4 illustrates how these messages are propagated during the execution. As you can see in Figure 4, the only message that could arrive from remote LPs is the external (q) message. The rest of the messages are local to each LP and thus could not cause causality errors at other LPs. When a simulator (simulation object) wants to send a message to another remote object, it first sends an output (y) message to its flat coordinator, and then the flat coordinator translates this message to an external (q) message and sends it upward to its node coordinator. Once the node coordinator receives this message, since it knows about which simulation object runs on which LP, it forwards that q message to the LP of the destination simulation object through its communication manager.

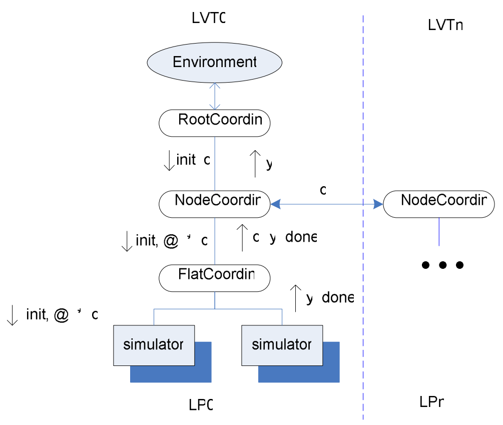

Figure 4. Message exchange in conservative CD++ simulator

Next we will present our implementation of the conservative algorithm in WARPED kernel.

5.3. New extension to WARPED

We have implemented the null-message based conservative algorithm in WARPED in such a way that the simulator does not know about the synchronization algorithm that is used in the WARPED layer below it. This separation enables switching between different synchronization mechanisms without the need to change the simulator code. The two major modifications which took place in the Time Warp kernel of WARPED will be presented next.

5.3.1. LP locking mechanism

We mentioned that there is a single LTSF scheduler residing on each LP which schedules the top most element of the inputQ to be executed next. However, this should only happen if the LP has received a message from all other LPs so that no message with earlier timestamp will be received at the LP later in time. The way we have implemented this is based on the messaging paradigm of DEVS which was presented in Section 5.2 which demonstrated that the only message an LP can receive from other LPs is the external (q) message. Therefore, at every execution cycle, when the top most element of the inputQ is a q message, the LP can process this event if and only if it has already received a q message from every other LP. These q messages could either be real q messages with meaningful data and positive sign, or they could be synthetic q messages with negative sign and no real data (i.e. null-messages) which are sent only for the purpose of synchronization. Thus, if it happens that the LP does not yet have a q message from all other LPs, it will be blocked waiting for other LPs to send their q message. This mechanism is implemented as following:

```
while (!simulationDone){
   if (inputQ->top.type == 'q'){
         gotMsgfromLP(inputQ->type.sourceLP);
             if (receivedMsgfromAll == true)
                        excecute(q);
             else       blockThisLP();
   }
}
```

Figure 5. LP locking mechanism for the conservative CD++ simulator

5.3.1. Lookahead and null-message mechanism

When a simulation object on an LP sends and output message (translated to q message by the node coordinator) to a remote simulation object, this states that there is no way that the remote LP will receive an earlier message from the sender LP later in time. Thus even if only one simulation object is the target of the q message, this is enough for the receiver LP to be assured that it will not receive causality causing messages from that particular LP. Therefore, when an LP sends a q message to one or more LPs, it should also send a synthetic copy of that message to the rest of LPs who are not the destination of this q message. This is done by sending a negative q message that holds the lookahead value of the sender LP. The lookahead value will assure receiver LPs that they will not receive any message from the source LP for the duration stated by the lookahead value. The lookahead value for an LP is basically the minimum time among all output messages that will be sent by its local simulators (simulation objects) minus the current local virtual time of the LP. The lookahead computation algorithm and the null-message sending mechanism are presented below.

```
waitForAllDoneMsgs()
           lookahead = minOutPutMsgTime(outputMsgs) - currentLVT
```

Figure 6. Lookahead computation for the conservative CD++ simulator

```
If (msgToSend type == 'q'){
       for (allLPs who are not targeted by q msg){
           sendNullMsg(lookahead)
     }
}
```

Figure 7. Null-message sending mechanism for the conservative CD++ simulator

The lookahead computation is done by the flat coordinator which is the parent coordinator of all local simulators on that LP. However, detecting the need and sending the null-messages are performed at the WARPED layer. As mentioned earlier, this is to keep the simulator core separated from the synchronization mechanism applied at the WARPED kernel layer.

8. Conclusions

We tackled the problem of executing DEVS and Cell-DEVS models in parallel and distributed environments based on the proposed conservative WARPED synchronization kernel. The hierarchical architecture of the simulator was replaced with a flattened one to reduce communication overhead among the nodes. We showed how the flat architecture simplifies messaging paradigms among remote LPs by enabling each LP to talk to other LPs directly without the need to use a central synchronizer residing on LP0. We then introduced our extension to the WARPED kernel which enables the kernel to support a conservative synchronization mechanism based on the classical null-message and lookahead algorithms. Our proposed synchronization mechanism was kept at the WARPED layer to provide a conservative protocol that is not tied up to the simulator core and thus enables easily switching among different synchronization algorithms without modifying the simulator code. At the end, we proved how our proposed algorithm avoids causality errors and deadlock by presenting a sample execution scenario.

10. References

[1] Fujimoto, R.M. "Parallel and Distributed Simulation Systems." Proceedings of the Winter Computer Simulation Conference. Phoenix, AZ. USA. 2001.

[2] Bryant, R. E. "Simulation of packet communication architecture computer systems". Massachusetts Institute of Technology. Cambridge, MA. USA. 1977.

[3] Chandy, K.; Misra, J. "Distributed Simulation: A Case Study in Design and Verification of Distributed-Programs." IEEE Transactions on Software Engineering, pp. 440-452. 1979.

[4] Jefferson, D. "Virtual Time". ACM Transactions on Programming Languages and Systems. 7(3):405-425. 1985.

[5] Zeigler, B. "Theory of modeling and simulation". First Edition. Wiley. 1976.

[6] Wainer, G.; Giambiasi, N. "Specification, modeling and simulation of timed Cell-DEVS spaces". Technical Report n.: 98-007. Departamento de Computación. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires. Ar-gentina. 1998.

[7] Wainer, G.; Christen G.; Dobniewski, A. "Defining models with the CD++ toolkit". Proceedings of the European Simulation Symposium. Marseille, France. SCS Publisher. 2001.

[8] Martin, D. E.; McBrayer, T. J.; Radhakrishnan, R.; Wilsey, P. A. "WARPED - A Time Warp Parallel Discrete Event Simulator". Available at:
http://www.ececs.uc.edu/~paw/warped/doc/index.html.
1999. [Accessed April, 2009].

[9] Troccoli, A.; Wainer, G. "CD++, a tool for simulating Parallel DEVS and Parallel Cell-DEVS models". Técnica Reporta. Departamento de Computación. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires. Ar-gentina. 2001.

[10] Glinsky, E. "New Techniques for Parallel Simulation of DEVS and Cell-DEVS Models in CD++". M. A. Sc. Thesis. Carleton University. Canada. 2004.

[11] Kim, K.; Kang, W. "CORBA-based, Multi-threaded Distributed Simulation of Hierarchical DEVS Models: Trans-forming Model Structure into a Non-hierarchical One". International Conference on Computational Science and Its Applications (ICCSA). Assisi, Italy. 2004.

[12] Liu, Q. "Distributed Optimistic Simulation of DEVS and Cell-DEVS Models with PCD++". M. A. Sc. Thesis. Carleton University. Canada. 2006.

[13] Radhakrishnan, R.; Martin, D. E.; Chetlur, M.; Rao, D. M.; Wilsey, P.A. "An Object-Oriented Time Warp Simulation Kernel". Proceedings of the International Symposium on Computing in Object-Oriented Parallel Environments (ISCOPE'98). Vol. LNCS 1505, pp. 13-23. Springer-Verlag. 1998.

[14] Chandy, K. M.; Misra J. "Distributed simulation: A case study in design and verification of distributed programs". IEEE Transactions on Software Engineering. pp.440-452. 1978.