Transforming DEVS to Non-Modular Form For Faster Cellular Space Simulation

Fahad A. Shiginah, Bernard P. Zeigler
Arizona Center for Integrative Modeling and Simulation
Electrical and Computer Engineering Department
University of Arizona
Tucson, AZ 85719
{shiginah, zeigler}@ece.arizona.edu

Keywords: Discrete Event Simulation, DEVS, Parallel DEVS, Cellular Space Modeling, High Performance Simulation.

Abstract

This paper presents a new approach that enhances the performance of large scale cellular space simulations expressed in modular DEVS. The basic idea is to group cells in cellular space into smaller partitions that are treated as atomic DEVS models. The enhancement is achieved by reducing the large number of messages generated by inter-cell communication. This, in turn, saves large number of simulator iterations that were used to handle such communication. The new approach gives significant simulation speedup over the conventional techniques of representing cells in DEVS. A landslide slope criticality model is used to perform some computational experiments to demonstrate the simulation speedup of our approach.

INTRODUCTION

Cellular Automata (CA), which first introduced by John Von Neumann in the 1950s [3], have been widely used in simulating complex systems. The CA domain of application includes fluid and mass flow, many other sorts of pattern formation, image processing, ecosystems, and highway traffic. Many publications in recent years were dedicated to improve simulation methodologies those take advantage of CA in modeling complex behavioral dynamic systems (e.g., see [1], [3], [11], and [12]).

Based on the fixed time step CA, Avolio and his coworkers developed an empirical approach for modeling and simulating complex dynamic systems [1]. Such an approach can be applied to problems those are very difficult to manage with differential equation systems. However, the use of discrete events, rather than fixed time steps, in simulation gives a significant speedup in many applications [12]. As a result, DEVS (Discrete Event System Specification) has been attracting many researchers as a basis of CA modeling of complex physical systems. Its parallel version, parallel DEVS, was first introduced in [4] to provide a sound framework that exploit the parallelism of the hierarchical DEVS models. Based on this parallelism, it was possible to introduce Cellular DEVS [12] which integrates the theories and algorithms of CA in DEVS.

The cellular space modeling approach divides space into discrete cells where local computations held in each cell are based on its own as well as its neighbor’s states. In conventional DEVS implementation of CA (e.g. [7-11]), a cell is implemented as a DEVS modular atomic or coupled model. When detailed modeling of spatial dynamics is required, large number of cells are typically employed. This results in large number of atomic models those communicate through message passing to carry out the global simulation. Therefore, implementing large scale cellular spaces with highly active cells in DEVS will face the burden of huge numbers of inter-cell messages and hence a performance reduction. Many techniques were introduced to resolve this issue and to gain speedup. Examples of such work can be found in [5], [7], and [11] where the cellular DEVS simulation engine was improved to handle messages and cell activity scanning in more efficient manner. On the other hand, the quantized DEVS approach [12] shows that quantization helps in improving the performance of DEVS simulations by reducing the number of state transitions as well as the number of messages while introducing acceptable errors.

To date, research in DEVS cellular space modeling, has treated each cell as an atomic or coupled model and then either sought to speed up the simulation engine or introduce quantization to the model in order to reduce messages and transitions with attendant error. The simulator enhancements were tackled by either flattening the coordinator hierarchy [11], implementing faster scheduling algorithms those deal with active cells only [5, 7], and/or eliminating unnecessary coordinator objects [7]. The work presented in this paper takes advantage of these enhancements and apply similar methods to the model level. This new error-free approach is designed to reduce the number of messages by
encapsulating and transforming a group of cells into non-modular form. Instead of treating a single cell as an atomic DEVS model, the encapsulation method will group a number of cells in one atomic model. The resulting model will be a non-hierarchal non-modular cell representation that gives a significant speed up which in conjunction with the simulator enhancements done in [5, 7, or 11] will give amazingly high performance on large scale distributed cellular space models over clusters.

In this article, the concept of parallel DEVS formalism is used as the basis for our cellular approach which is then introduced and described in details. A simple landslide slope criticality mass flow model in DEVSJAVA is also introduced as an example base model to test the new approach. Finally, based on the results generated from the test model, a performance analysis study will be used to demonstrate the simulation speedup of our approach over the conventional ones.

**PARALLEL DEVS FORMALISM**

Discrete EVent System Specification (DEVS) [12] supports object orientation over modeling environments. Its theory provides a mathematical formalism for representing dynamic systems. The DEVS formalism was revised in [4] to reduce sequential processing and enable full parallel executions. The resulting parallel DEVS has the basic atomic model defined as:

\[
M = \langle X, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \delta_{\text{con}}, \lambda, \tau_a \rangle
\]

where \( X \) is a set of input values.

\( Y \) is a set of output values.

\( \delta_{\text{int}}: S \to S \) is the internal transition function.

\( \delta_{\text{ext}}: Q \times X^b \to S \) is the external transition function, where \( Q = \{ (s, e) \mid s \in S, 0 \leq e \leq \tau_a(s) \} \) is the total state set.

\( e \) is the time elapsed since last transition.

\( X^b \) denotes the collection of bags over \( X \).

\( \delta_{\text{con}}: Q \times X^b \to S \) is the confluent transition function.

\( \lambda: S \to Y^b \) is the output function.

\( \tau_a: S \to R_{0,\infty} \) is the time advance function.

An atomic model \( M \) in parallel DEVS remains in a state \( s \in S \) for \( \tau_a(s) \) amount of time if no external event occurs. When that time advance expires, i.e., when the elapsed time, \( e = \tau_a(s) \), the system outputs the values, \( Y^b = \lambda(s) \), just before it changes to state \( \delta_{\text{int}}(s) \). When an external event \( x \) in \( X^b \) occurs before this expiration time, i.e., at \( e < \tau_a(s) \), the system changes to state \( \delta_{\text{ext}}(s, e, x) \). However, in case of internal and external transitions collide, \( \delta_{\text{con}} \) is employed to resolve the conflict and determine the next state. In all cases, the model then goes to some new state \( s' \) with some new resting time, \( \tau_a(s') \) and the same story continues[13].

Note that input or output values \( X^b \) and \( Y^b \) are bags of elements. This means that one or more elements can appear on a port at the same time. This capability comes from the parallel implementation of DEVS which allow components to send to the ports simultaneously. These basic components may be coupled in DEVS to form a multi-component model which is defined by the following structure:

\[
CM = \langle X, Y, D, \{ M_i \}, \{ I_i \}, \{ Z_{i,j} \} \rangle
\]

where

\( X \) is the set of input values.

\( Y \) is the set of output values.

\( D \) is the set of components.

for each \( i \) in \( D \): \( M_i \) is a component which is an atomic model \( M_i = \langle X_i, Y_i, \delta_{\text{int}}^{i}, \delta_{\text{ext}}^{i}, \delta_{\text{con}}^{i}, \lambda_i, \tau_{a_i} \rangle \)

for each \( i \) in \( D \cup \{ \text{self} \} \): \( I_i \) is the influences of \( i \), \( i \) is not in \( I_i \).

\( \text{self} \) is the coupled model itself \( CM \) which allow external inputs and outputs.

for each \( j \) in \( I_i \): \( Z_{i,j} \) is the \( i \) to \( j \) output translation function (coupling).

\[
Z_{\text{self}, i} : X_{\text{self}} \to X_i
\]

\[
Z_{i,\text{self}} : Y_i \to Y_{\text{self}}
\]

\[
Z_{i,j} : X_i \to Y_j
\]

**CELLULAR SPACE MODELS IN DEVS**

(The Conventional approach)

Parallel DEVS is an object oriented environment in which models can be defined as instances of atomic or coupled models.
model classes. The cellular space is a hierarchical coupled parallel DEVS model (CM) that consists of a number of cells which are created as instances of atomic or coupled model. The cells should be arranged as a grid and hence each cell has a well defined address in the space. According to this addressing scheme, the coupling relations \( \{Z_{ij}\} \) will connect each cell to its neighboring cells defined by the neighboring rules followed by the model.

Figure 2. An example of 2-D cellular space in DEVS.

THE NEW APPROACH

Our approach is based on reducing the volume of communication inside the coupled cellular model by making atomic models encapsulating a group of cells (subspace) rather than a single cell. Inside that non-modular subspace, no inter-cell messages are required since all cells can know their neighbor states using internal variables. The subspaces will only communicate to know the states at the cells by their boundaries. Figure 3 explains, in general, the idea of message reduction in the encapsulation method of a two dimensional cellular space.

Figure 3. Message Reduction Approach. A cellular space of \( N \times N \) dimension can be divided into \( S \times S \) subspaces. Each subspace will have \( (N/S) \times (N/S) \) cells. Assuming that in a single iteration a cell will send \( V \) messages in each of the 4 directions (Neumann neighbors) as the worst case scenario. The conventional approach (left) will result in \( 4VN^2 -4VN \) messages per iteration given that the cells at the boundaries will not send to at least one of its neighbors. While in the encapsulation scheme (right), each subspace will just send \( 4VN/S \) and the total number of messages per iteration will be \( 4VNS-4VN \). The percentage reduction of messages will be \( (N-S)/(N-1)\% \) which will result in 100% reduction if \( S=1 \) or 0% if \( S=N \).

Closure under coupling, in parallel DEVS, implies that a coupled model (CM) can be treated as an atomic model (M) which is equivalent to CM (see [4] and/or [13] for detailed proof). This property gives the feasibility of implementing our approach in which a coupled model of a group of cells will be converted into its equivalent atomic model. The methodology underlying this conversion uses the reverse direction of a method explained in [12] for translating non-modular multi-component DEVS models into modular form (see Figure 4). Since that method concentrates on translating variables and couplings only, more effort needs to be done in handling the conversion of states and events. This involves adding a discrete event list handler inside the atomic model to keep track of activities among all cells those belong to that atomic model. Figure 4 shows the resulting parallel DEVS atomic model that has an extension for handling activity scanning and discrete event list handling. The state of this model can be treated as a vector state, i.e. combining all cells states while the time advance will be the minimum among all the encapsulated cells.

Figure 4. Encapsulation Mechanism: Converting a cellular space into an atomic model.
The resulting atomic model, shown in Figure 4, implements an optimal intelligent scanning mechanism in order not to scan all arrays of cellular values at any single iteration and to ensure equivalence to the original coupled DEVS simulation. This mechanism will be dealing with a minimum scan list that contains the imminent cells and their neighbors only. The entries of this list will be updated by the basic DEVS functions like $\delta_{int}$ and $\delta_{ext}$. For example, on receiving external event, $\delta_{ext}$ can decide the cells at the boundaries that should receive these messages and immediately will add these cells as well as their neighbors to the scan list. $\delta_{int}$ involves some mechanisms to carry out the computations for those cells inside the scan list as well as making rigorous predictions of their next events and add them to the events list. In addition, it will be responsible to retrieve the imminent cells from the event list and add them to the scan list.

By adding the scanning list as well as the event list handler to the atomic model, we introduce more complexity to the atomic model level while computational efforts are kept the same. This is because these additional functionalities are implied in the simulator when each cell is treated as an atomic model. All that we have done is moving these functionalities down to the atomic level. However, additional computational overhead might be introduced when dealing with arrays in computations and predictions. Therefore, the task turned out to be an optimization problem to find the best encapsulation size $S$ that minimizes the execution time since we are adding computation overhead in favor of reducing communication. Therefore, a test model should be implemented using this approach in order to show this computation/communication trade off. The following section explains a slope criticality model that was used to put our approach in action.

**SLOPE CRITICALITY MODEL**

Slope criticality has many applications in dynamical natural phenomena like sand piles, debris flows, landslides, and land surface evolutions (e.g. [2] and [6]). In such models a local piece of spatial space is stable in certain slopes till it exceeds a critical slope when it starts initiating a mass flow to the neighbors. This flow might continue propagating over neighboring areas forming a bigger global flow that keeps running until it reaches global stability.

Applying cellular space modeling techniques to these models will result in dividing the spatial space into cells in which each cell will apply automata rules to decide its own local state. This will be based on the slope calculation which is represented by the height difference ($\Delta Z$) between each two neighboring cells given that the distance between them is fixed ($d_{ij}$). When a cell has a slope greater than a certain critical value, it initiates a mass flow to its neighbor with a fixed rate $R$.

![Figure 5. Slope Calculation.](image)

In this two dimensional cellular model, we apply the 4-neighbors rule. Criticality check involves calculating the slopes between a cell and its four neighbors. Whenever a cell finds that the slope at any direction is greater than the known threshold, it calculates the flow rate for that direction and informs the neighbors with its outgoing flow. The neighboring cells then update their states according to the incoming flows. Accordingly, the height of the initiating cell will be reduced by the amount of increase in the receiving neighbor and this conveys the mass conservation. The simulator will iterate until there is no more cells in critical state or by reaching a predefined stopping rule.

**EXPERIMENTAL RESULTS**

The model explained above was implemented in DEVSJAVA using the new approach for the purpose of performance comparison with the conventional implementations. A 32 by 32 cellular space model was run with different setups including different sizes of encapsulation ($S$) in the case of our new approach. In all cases, identical data (i.e. heights and flows at any given time) results should be generated which ensures the correctness of our implementations.

By implementing the encapsulation method in DEVSJAVA, we achieved an enormous time reduction, messages reduction, as well as number of simulator iterations reduction as shown in Table 1. These new findings are not in favor of the conventional way of implementing cellular automata in DEVS especially when running on a single machine. For example, implementing the new approach by encapsulating all cells in one atomic model ($S=1$) gives a speed up of 46 in execution time over the conventional approach ($N=32$) by saving 121569 message-handling simulator iterations using the conventional DEVSJAVA simulator (DJSim).

In Table 1, the last column shows the execution time when repeating the runs over the implemented activity-based DEVSJAVA simulator (ADJSim) suggested in [5]. This simulator alone achieved a speed up of 3.8 over the
conventional simulator and noticeable speedups when combined with our approach. The last case shows that the new simulator has no gain because the model consists of single cell which is always active during the simulation run.

Table 1. Experimental Results.

<table>
<thead>
<tr>
<th></th>
<th>Message Reduction (%)</th>
<th>Iterations</th>
<th>Execution Time (Min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>N=32</td>
<td>0</td>
<td>171478</td>
<td>DJSim: 63.17</td>
</tr>
<tr>
<td>S=32</td>
<td>0</td>
<td>171478</td>
<td>DJSim: 66.35</td>
</tr>
<tr>
<td>S=16</td>
<td>51.60</td>
<td>150149</td>
<td>DJSim: 21.73</td>
</tr>
<tr>
<td>S=8</td>
<td>77.40</td>
<td>126171</td>
<td>DJSim: 13.89</td>
</tr>
<tr>
<td>S=4</td>
<td>90.30</td>
<td>124755</td>
<td>DJSim: 21.22</td>
</tr>
<tr>
<td>S=2</td>
<td>96.70</td>
<td>124755</td>
<td>DJSim: 39.53</td>
</tr>
<tr>
<td>S=1</td>
<td>100</td>
<td>49909</td>
<td>DJSim: 1.36</td>
</tr>
</tbody>
</table>

Figure 6 shows the performance comparison curves which indicates at some points that the new approach has introduced some computation overhead to the model. This can be noticed by comparing the original model (N=32) with our implementation when we encapsulated one cell only in each atomic model (S= 32). If there was no additional computation added, the execution times at these two points should be equal. At the middle of the execution time curve, there is a saturation point S=8 where the execution time starts to increase. This resulted from the nature of the simulated problem as well as the trade off between reducing messages and introducing computation overhead. However, the last case, where all cells encapsulated under one atomic model, shows the highest performance which is independent of the problem nature. Furthermore, the simulator introduced in [5] shows a significant speed up especially in the cases where relatively few cells of the total number are active. On the other hand, when S gets smaller, majority of cells are active and there is less or no speedup gain over the conventional simulator.

CONCLUSION

Our new approach of transforming cellular DEVS into the non-modular form significantly enhances the performance of cellular space modeling in DEVS. This was achieved by reducing the number of messages transmitted inside the global cellular model during the simulation which in turn reduces the number of simulator iterations that handle external events. It was shown that the best performance was achieved when all cells in the cellular space were treated as one atomic model in a single machine. This may lead us to believe that we can get a great enhancement in distributed cellular simulation by transforming the sub-cellular space in a single machine entirely into non-modular form. This will allow each single machine not to handle any internal messages and the performance bottleneck will be moved to the external messages between machines. Therefore, further investigation is required to test our approach on large distributed cellular space simulation to justify these claims.

In addition to this modeling enhancement, simulator enhancement will be a key issue when running large scale cellular space models. These enhancements can speed up the simulation engine. However, simulation restructuring does not deal with inter-cell messaging overhead. Therefore, both model and simulator enhancements must by employed together to simulate a large scale cellular modeling environment. The approach presented in this paper can be applied to any other non-hierarchal cellular model. However, applying it manually is complex and error prone. Therefore, an automated way of conversion needs to be implemented for fast, accurate conversion process.

REFERENCES


