Global Lookahead Management (GLM) Protocol for Conservative DEVS Simulation

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

Keywords: dynamic lookahead, conservative DEVS, distributed systems, message-passing systems, discrete event simulation, parallel algorithms.

Abstract
An approach to carrying out asynchronous distributed simulation of multiprocessor message passing architectures is presented. Aiming at achieving better performance on Conservative DEVS-based simulations, we introduce the GLM protocol which borrows the idea of safe processing intervals from the conservative time window algorithm and maintains global synchronization in a fashion similar to the distributed snapshot technique. Under the GLM scheme, a central *lookahead manager* (LM) exists which is in charge of receiving every LP's lookahead, identifying the global minimum lookahead of the system, and broadcasting it via null messages to all LPs. The simulation is divided into cycles of two phases: *Parallel* phase and *Broadcast* phase. The GLM protocol is asynchronous and the central lookahead manager is not expected to be a bottleneck since the only message transmissions involving it take place when all LPs are blocked waiting for permission to advance their LVTs. The results presented in this paper show that the GLM protocol not only significantly reduces the total number of null messages, but it improves the performance and higher speedups are achieved.

1. INTRODUCTION

Modeling and simulation (M&S) has become a cost-effective tool for analyzing and designing complex systems in a broad variety of fields. Varied parallel and distributed simulation (PADS) techniques have been proposed to deal with large and complex models. In particular, parallel discrete event simulation (PDES) has become the technology of choice to speed up large-scale discrete-event simulations. Various synchronization techniques have been proposed for PDES systems, and they generally fall into two major classes: conservative, which strictly avoid causality violations [1]; and optimistic, which allow violations and recover from them by providing a rollback mechanism [2]. The conservative approaches were the first synchronization algorithms proposed in the late 1970s by R. E. Bryant [3], K. M. Chandy and J. Misra [4]. The Chandy-Misra-Bryant (CMB) algorithm prevents the occurrence of causality errors, and it prevents deadlocks by making use of null-messages. In order to improve performance, these algorithms use lookahead information provided by the simulation designer.

Although PDES techniques have been very successful, the methods do not address the definition of simulation models. With this goal, we are trying to combine PDES algorithms with advanced modeling methods. In particular, we propose to use the DEVS (Discrete Event System Specification) formalism [5], which provides a discrete-event approach to construct hierarchical and modular models. We focus on two extensions: P-DEVS [6], which allows adequate handling of simultaneous events in parallel and distributed environments, and Cell-DEVS [7], which allows representing complex discrete event spatial models. These formal methods have been implemented in the CD++ toolkit [8], raising various issues related to performance, scalability, and complexity. In order to be able to deal with complex models, we implemented optimistic PDES algorithms, and the first purely conservative simulator for Cell-DEVS, called Conservative CD++ (CCD++) [9]. CCD++ uses a Conservative DEVS synchronization algorithm based on the classical CMB approach with deadlock avoidance [9].

Conservative algorithms can distribute a large number of null messages throughout the simulation, degrading performance. In order to decrease the overhead of the original conservative DEVS algorithm, we propose the **Global Lookahead Management** (GLM) protocol which is based on the Conservative Time Window algorithm [11] and the Distributed Snapshot mechanism [12]. GLM dramatically reduces the number of null messages by organizing the conservative execution in such a way that every logical process (LP) reports its lookahead only to the global manager rather than to every neighboring LP.

This work will present the details of the GLM protocol for conservative simulation of large scale DEVS-based models. The implementation details and different strategies of the protocol such as scheduling, null message distribution mechanism, lookahead extraction, deadlock avoidance technique, as well as LP block and resume mechanism will be discussed thoroughly. Following the

simulation overview, we present statistical analysis by running both communication- and computation-extensive large scale DEVS-based models. The experiments are conducted over both the original conservative DEVS protocol and the proposed GLM mechanism to show the performance gain. The results shown by this research clarify the robustness of GLM protocol in terms of handling large and complex simulations while significant speedups are noticed, indicating that the protocol is well suited for simulating such models.

2. BACKGROUND

Although the field of PDES has advanced steadily in the last 20 years [1], most existing techniques focus on the simulation algorithms only, making modeling of the system of interest a difficult task. The DEVS formalism, instead, focuses on modeling methods that can be executed by independent abstract simulation engines [5]. DEVS is a sound formal M&S framework for discrete-event systems, which allows the construction of hierarchical models in a modular manner. A system defined by DEVS is described as a composition of behavioral (*atomic*) and structural (*coupled*) model components.

DEVS has been proven to be the most generic formalism to represent DEDS (Discrete Event Dynamic Systems), and it has been extended to deal with continuous and time-based models. P-DEVS [6] provides an elegant mechanism for handling simultaneous events at the modeling level, and it also prevents serialization constraints, allowing more efficient execution of models in parallel and distributed environments. Cell-DEVS [7] makes use of DEVS for defining cellular models, by defining every cell as an atomic DEVS component. The formalism defines complex cell behavior with simple instructions, allowing the construction of n-dimensional cell spaces, providing advanced constructions to represent the cell's timing behavior.

CD++ [8] is an open source M&S environment which implements DEVS, Cell-DEVS, and P-DEVS and supports standalone and parallel/distributed simulations on different platforms. A parallel optimistic simulator, called as PCD++ [14] was developed for high-performance simulation of these models. In [9] we introduced the Conservative DEVS algorithm which is based on CMB null message protocol with deadlock avoidance. The algorithm overcomes the limitations of the original conservative DEVS [5] by: (i) implementing the mechanism at the top most level of the DEVS abstract simulator hierarchy, thus reducing the frequency of information computation, and (ii) performing a single lookahead computation rather than two types of calculations (*Earliest Input Time* **EIT** *and Earliest Output* Time **EOT** as in [5]). This results in a significant reduction of number of null messages.

We are interested in improving the performance further, thus, we introduce the **Global Lookahead Management** (GLM) protocol. GLM, based on the Conservative Time Window algorithm [11] and the Distributed Snapshot mechanism [12], significantly reduces the number of null messages by directing them to a global manager rather than to neighboring LPs. The GLM protocol implements an asynchronous strategy [13] in the sense that there is no global clock (every process maintains its own local clock) and GVT approximation is performed based on LPs' lookahead information. The centralized fashion of the GLM protocol does not cause any overhead neither it is prone to bottleneck because the algorithm is only invoked when the simulation is suspended and LPs are blocked waiting for null messages. In fact, the GLM protocol is only active when there is nothing else going on in the simulation.

These new algorithms have been implemented in CCD++, a conservative DEVS and Cell-DEVS simulator [9]. CCD++ supports both standalone and parallel conservative simulations. The simulator is built on top of the WARPED kernel [15], which provides services for defining different types of processes (simulation objects). Simulation objects on a physical processor are grouped into an LP, and they communicate through Message Passing Interface (MPI). WARPED has been used as middleware to provide scheduling, memory, file, event, communication and time management.

3. ARCHITECTURE OF CCD++

CCD++ simulation is carried out by two types of DEVS processors: *Simulators* and *Coordinators*. Simulators execute atomic DEVS models, and Coordinators are paired with coupled models. Simulators are in charge of invoking the DEVS functions defined in their atomic models. On the other hand, Coordinators must route messages and schedule the imminent dependant(s). At the beginning of the simulation, one logical process (LP) is created on each machine (physical process). Then, each LP hosts one or more DEVS processors. CCD++ employs a flat structure by creating a *Node Coordinator* (*NC*), a *Flat Coordinator* (*FC*), and a set of *Simulators* on each node. A special Coordinator, called *Root* is created on machine 0, which is in charge of starting the simulation and performing I/O operations among the simulation system and the 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 proces-

sors of the local FC representing the atomic components of DEVS and Cell-DEVS models.

DEVS processors exchange two categories of messages: *content messages* and *control messages*. The first category includes the *external* (*x*) and the *output* messages (*y*), and the second category includes the *initialization* (*I*), *collect* (@), *internal* (*), and *done messages* (*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. Figure 1 illustrates CCD++ processors and the messaging among them.

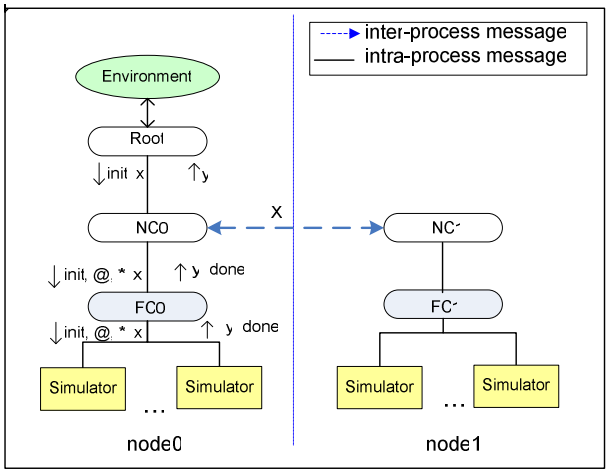

Figure 1. CCD++ processors and messages

The simulation starts by the Root Coordinator residing on *node0* by sending an (*I*, *t*) message to all NCs. At any virtual time, the message flow among the LPs is organized into a multi-phased structure that includes an optional *collect* phase and a mandatory *transition* phase, which in turn may involve multiple rounds of computation to execute state transitions incrementally. The *collect* phase starts with a *collect* message sent from the NC to the FC and ends with the following *done* message received by the NC. The *transition* phase begins with the first *internal* message sent from the NC to the FC and ends at the last *done* message received by the NC at that time. The *transition* phase is mandatory for each individual simulation time. The *output* functions in the imminent atomic models are invoked during *collect* phases, the state transitions for the atomic models are performed in the *transition* phases (as defined in P-DEVS formalism).

4. PHASE-BASED SIMULATION WITH GLM

The GLM protocol borrows the idea of safe processing intervals from the Conservative Time Window algorithm and maintains global synchronization in a fashion similar to the Distributed Snapshot technique. Under GLM, a central *lookahead manager* (LM) exists on LP0 which is in charge of three main tasks: 1) receiving every LP's lookahead, 2) identifying the *global minimum lookahead* of the system, and 3) broadcasting it via null messages to all LPs. This implies that the LPs are no longer required to send their lookahead information directly to each other as in the Conservative DEVS algorithm [9]; rather, they now send their lookahead via null messages to the LM only. In fact, the sole function of the LM is to detect the suspension phase, and to initiate the resume phase by broadcasting the *global minimum lookahead*. The entire algorithm works using the following sequence:

(i) **Parallel phase**: LPs run simulation until suspension.
(ii) **Broadcast phase**: LM broadcasts global minimum lookahead allowing LPs to advance their LVTs.

In the original conservative DEVS algorithm, each LP had to send a null message to every LP and block until all LPs send back their null messages accordingly. With the GLM protocol, each LP sends one null message to the LM and it blocks until the LM sends back a null message carrying the global minimum lookahead value. GLM not only reduces the number of null messages, but it also reduces the blocked time of LPs since they now wait for only a single null message to be received before they can resume as opposed to the Conservative DEVS algorithm where each LP waits on (N-1) null messages every time it blocks. As the number of participating LPs increases, the performance achieved with GLM increases, merely because the total number of LPs has a direct impact on the synchronization overhead. The key characteristic of the GLM protocol is that it is asynchronous and the central LM is not expected to be a bottleneck since the only message transmissions involving it take place at the end of *Parallel* phase and *Broadcast* phase. In fact, the LM does not carry out any computation and it is only invoked when all LPs are blocked and the simulation is suspended. Thus, the centralized fashion of the GLM does not introduce any overhead.

4.1. Lookahead and LVT Computation Strategy

Both lookahead and LVT of the GLM protocol are computed the same way as in the Conservative DEVS algorithm. At every NC, the lookahead computation is performed as follows [9]:

lookahead = MIN(*timestamp of most recently sent output to a remote LP*, *timestamp of first unprocessed input*, *closest state transition time).* (1)

The lookahead is then sent to the LM via a single null message and the LP suspends. Upon receiving the response null message from the LM, the LP resumes by first calculating the new LVT as follows [9]:

LVT = MIN(*timestamp of the input even from the environment, timestamp of most recently sent output to a remote LP, timestamp of first unprocessed input, closest state transition time, time of new global lookahead value received from LM)*. (2)

Thus, the NC computes the new LVT as the minimum value among: (i) the timestamp of the input event received from the environment; (ii) the timestamp of the *external* message recently sent to a remote LP; (iii) the time of the NC Message Bag (minimum timestamp among unprocessed input events); (iv) the closest state transition time of local child processors previously given by the FC in the *done* message; and (v) the new global lookahead value received from LM via null message.

Similar to the Conservative DEVS algorithm, the NC is the local synchronizer at the LP and invokes the GLM protocol at the beginning of every collect phase. The NC issues the collect phase by first performing lookahead computation according to Formula 1, then, it sends the calculated value via a single null message to the LM and blocks the LP. Upon receiving the response null message form the LM, the LP resumes and the NC performs a LVT computation, which will take into account the newly received minimum global lookahead from the LM.

4.2. Null Message Distribution

Based on the strategy used to partition a DEVS/Cell-DEVS model, each LP can only send/receive event messages (*external messages*) to/from those defined by DEVS neighboring. That is, according to DEVS neighborhood specified by the model, if an atomic component (represented by a *Simulator*) on an LP is a neighbor of another atomic component residing on a different LP, then these two LPs are neighbors and they can communicate to each other through inter-LP communication. Therefore, it is possible that some LPs are not direct neighbors of each other because the model's partitions they hold are not DEVS-neighbors with each other. Figure 2-A illustrates the partitioning of an 8x8 Cell-DEVS model on four LPs. The partitioning mechanism divides the cell space into four equal portions (8x2 cells per LP). Figure 2-B shows which LPs send *x* messages to each other (i.e. their child *Simulators* are DEVS neighbors), and Figure 2-C represents the conservative neighboring of LPs where each LP sends null messages to every other LP, and on the other hand, it receives null messages from them. The GLM protocol resolves this tight coupling of LPs by assigning a simple LP connectivity strategy. Under the new scheme, each LP is only coupled with the LP for which the LM resides on (i.e. LP0). Hence, the LP neighborhood is configured as in Figure 2-D.

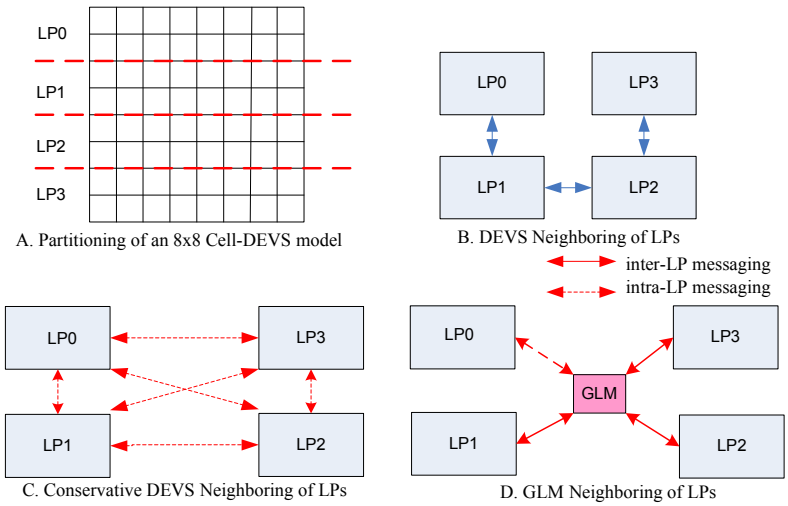

Figure 2. LP Neighboring in Conservative DEVS vs. GLM Protocol.

4.3. LP Block and Resume Mechanism

The GLM block and resume mechanism is slightly different from the Conservative DEVS algorithm in the sense that LPs are no longer distributing null messages to each other, neither they wait for reception of null messages from every LP. In return, each LP sends a single null message at the start of every collect phase to only the LM and stays blocked until the single null message reporting the new global minimum lookahead is received from the LM. This strategy reduces the number of null messages.

4.4. Dynamic Lookahead

The lookahead computation is performed after each LVT computation; hence, it is updated and reported to the LM every time before the LP is suspended. This strategy ensures that the lookahead value of an LP represents the lat-

est LVT update, as there is at least one lookahead computation per LVT update. The dynamic lookahead mechanism of the GLM protocol states that lookahead value is not fixed and every lookahead computation could result in a different value than the previous stage. Unlike other existing conservative algorithms, the modeler is not required to specify the lookahead of the system. The ability of the algorithm in dynamically extracting the lookahead information from the model itself is a main advantage of this approach.

4.5. Low-cost Lookahead Computation

The lookahead computation is a fast, efficient, and low-cost method that involves a simple comparison between existing parameters. In fact, there is neither an actual computation nor a significant computation time required to calculate the lookahead. Rather, the lookahead is extracted from already computed data that existed in the simulator before the conservative protocol was integrated with it. Compared to other existing conservative mechanisms, this benefit reduces the overhead of the algorithm, especially the frequency of invoking lookahead computation, which increases as the model size grows.

4.6. Deadlock Avoidance

Since null message distribution of LPs to the LM occurs before LPs are suspended, deadlock is strictly avoided. NCs only suspend the LP after performing a lookahead computation and reporting it to the LM via a null message. Thus, when an LP is suspended, it has already forwarded its null message, and if every other LP is suspended as well, they would all resume because all required null messages have been already sent to the LM before suspension has taken place.

4.7. I/O Operation

The only messages an LP sends out are *external* messages and null messages. The *external* messages are sent out to neighbor LPs defined by DEVS neighboring, while null-messages are sent out to the LM on node 0. Similarly, an LP can only receive *external* messages from its DEVS neighbor LPs, and null messages from the LM.

4.8. Termination

The simulation terminates if the *Stop Time* (specified by the model) is reached or all LPs are idle and have no unprocessed event in their input queues. The NC sets the LP to idle when the LM sends a null message reporting *infinity* as the global lookahead value, all local child Simulators have next transition time of *infinity*, and there is no unprocessed event in the NC message bag. The last message sent out by a NC before setting the LP to idle is a null messages carrying *infinity* lookahead. This ensures that other LPs do not stay blocked awaiting last null-message from the idle LP.

5. SUMMARY OF THE ALGORITHM

Based on the LP structure and the division of functionalities in CCD++, the key features and assumptions of the simulation process are as follows:

1. All messages originating form *Simulators* must go through the parent FC. Hence, there is no direct communication between *Simulators* (even local ones). FCs are always aware of the timing of state changes at their child *Simulators*.
2. Outgoing inter-LP communication happens only in the *collect* phases, whereas incoming inter-LP communication can occur in any phase. Since the *output* functions of imminent models are invoked only in the *collect* phases, at any given simulation time, all *external* messages going to remote NCs are sent out by the end of the *collect* phase. On the other hand, an *external* message from a remote source can arrive at the destination NC in any phase.
3. The NC is the starter for every *collect* and *transition* phase. The NC is invoked when it receives a *done* message from the FC. The *done* message could be in response to a (I, t), (@, t), or (*, t) previously sent to the FC.
4. On each node, the simulation time is advanced by only the NC. The NC calculates LVT of the LP at the beginning of every *collect* phase. The local FC and the *Simulators* do not send messages with a time-stamp different than the current LVT.

6. PERFORMANCE EVALUATION

6.1. Experiment platform and metrics

To analyze the performance of CCD++, extensive tests were carried out on a cluster of 26 compute nodes (dual 3.2 GHz Intel Xeon processors, 1 GB PC2100 266 MHz DDR RAM) running Linux WS 2.4.21 interconnected through Gigabit Ethernet and communicating over MPICH 1.2.6. Table 1 lists the metrics collected in the experiments through extensive measurements.

Table 1. Performance metrics

Metrics	Description
T	Total execution time of the simulation (sec)
NMR	Null event message reduction factor

Three Cell-DEVS models were tested in our experiments. Two of them (namely Fire1 and Fire2) simulate forest fire propagation [16] over 50 hours in a two dimensional cell space based on Rothermel's mathematical definition [17]. Fire1 and Fire2 differ in the way the spread rates are calculated. The first model uses a prede-

termined rates at reduced runtime computation cost, while the second one invokes the fireLib [18] library to calculate spread rates dynamically based on a set of parameters such as fuel type, moisture, wind direction and speed. The spread rate computations are performed at the Simulators when executing (*, t) messages. Hence, the time for executing a (*, t) message reflects the computation intensity of the state transition which was calculated to be 112 µs for Fire1, and 748 µs for Fire2.

The third model used, called as Watershed, was a simulation of environmental influence on hydrological dynamics of water accumulation over 30 minutes in a three dimensional cell space [19]. Although Watershed model (with a 577 µs state transition time) is not as compute-intensive as Fire2, it is a large 3D model with high communication requirements.

6.2. Test results and analysis

For all the three models, a simple partition strategy was used which evenly divides the cell space into horizontal rectangles. The fire propagation model (Fire1 and Fire2) was tested using different sizes of cell spaces: 100 x 100, 200 x 200, and 300 x 300. The Watershed model was tested with 25 x 25 x 2, 30 x 30 x 2, and 50 x 50 x 2 cells. Each of these cases was tested on 2 to 26 nodes.

Figure 3, 5, and 7 illustrate the total execution time achieved on 1 to 26 nodes for Fire1 model with 100x100, 200x200, and 300x300 cells respectively. As we can see, on each different size, the GLM protocol reduces the execution time compared to the Conservative DEVS algorithm for every given number of nodes. Meaning that, for the three mentioned model sizes, the smallest execution time is always achieved by the GLM protocol. It is also shown that for any given number of nodes, the execution time always increases with the size of the model. While the Conservative DEVS protocol increases the execution time after it reaches the smallest execution time, the GLM keeps reducing the execution time as the number of nodes increase in most cases. In fact, the GLM protocol shows much better performance compared to the Conservative DEVS algorithm for all scenarios. This is merely due to the significant null message reduction that the GLM provides as shown in Figure 4, 6, and 8 for the three different model sizes.

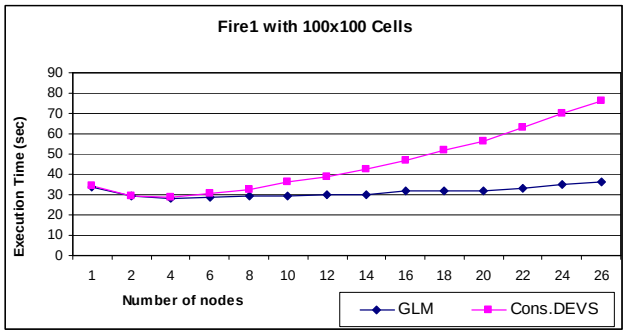

Figure 3. Total Execution Time of Fire1 (100x100 cells)

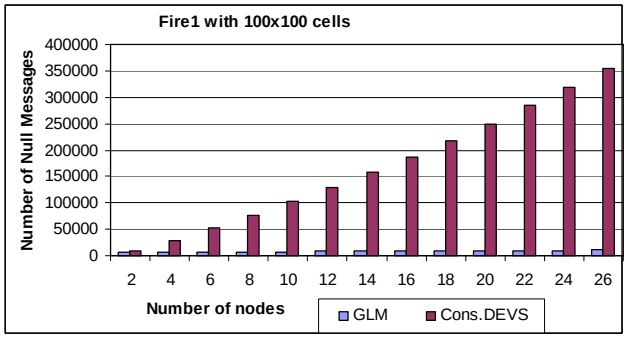

Figure 4. Total Number of Null Messages of Fire1 (100x100 cells)

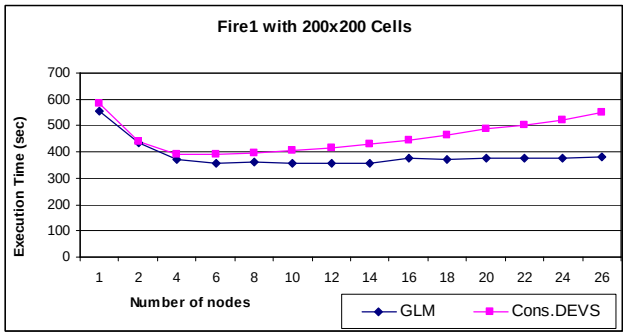

Figure 5. Total Execution Time of Fire1 (200x200 cells)

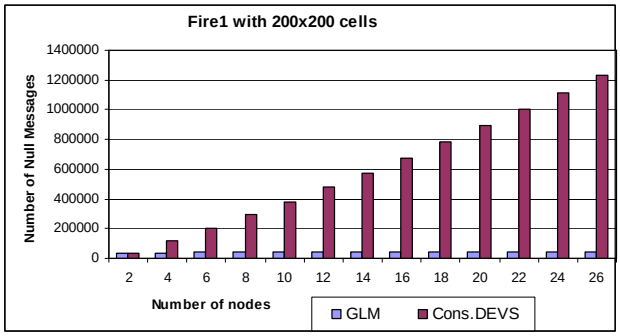

Figure 6. Total Number of Null Messages of Fire1 (200x200 cells)

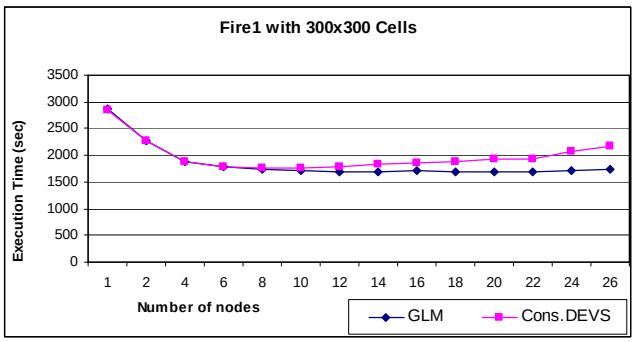

Figure 7. Total Execution Time of Fire1 (300x300 cells)

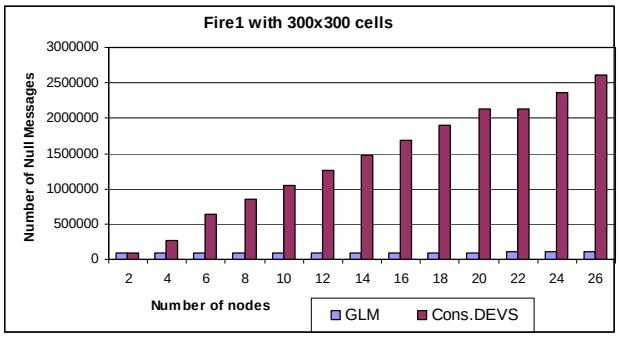

Figure 8. Total Number of Null Messages of Fire1 (300x300 cells)

The calculated null message reduction factor for Fire1 is given in Table 2. The speedups achieved with respect to the sequential simulator (results obtained on 1 node) are given in Table 3 where it is shown that the GLM protocol always results in better speedup compared to the Conservative DEVS algorithm.

Table 2. NMR for Different Sizes of Fire1 Model

Size	Metric	2	4	6	8	10	12	14	16	18	20	22	24	26
100x100	NMR	1.23	4.00	6.77	9.52	12.21	14.85	17.46	20.02	22.55	25.08	27.53	30.02	32.43
200x200	NMR	1.00	3.13	5.28	7.44	9.59	11.75	13.90	16.05	18.20	20.34	22.48	24.61	26.75
300x300	NMR	1.00	3.02	7.25	9.35	11.44	13.54	15.63	17.72	19.82	21.90	21.64	23.71	25.77

Table 3. GLM vs. Conservative DEVS Speedups for Fire1 Model

Size	Protocol	2	4	6	8	10	12	14	16	18	20	22	24	26
100x100	GLM	1.16	1.21	1.19	1.17	1.17	1.14	1.14	1.08	1.08	1.07	1.04	0.98	0.95
100x100	Con.DEVS	1.16	1.19	1.12	1.04	0.94	0.89	0.81	0.73	0.66	0.61	0.55	0.49	0.45
200x200	GLM	1.35	1.57	1.63	1.62	1.64	1.64	1.63	1.56	1.57	1.55	1.56	1.55	1.53
200x200	Con.DEVS	1.33	1.49	1.49	1.48	1.45	1.41	1.36	1.32	1.26	1.20	1.16	1.12	1.06
300x300	GLM	1.25	1.50	1.59	1.63	1.65	1.68	1.69	1.67	1.68	1.68	1.68	1.67	1.64
300x300	Con.DEVS	1.25	1.51	1.59	1.61	1.62	1.59	1.55	1.54	1.51	1.48	1.48	1.37	1.30

Similarly, Figure 9, 11, and 13 illustrate the total execution time achieved on 1 to 26 nodes for *Fire2* model with 100x100, 200x200, and 300x300 cells respectively. As in *Fire1*, on each different size, the GLM protocol reduces the execution time and total number of null messages significantly. The performance achieved by the GLM protocol stays high as the number of nodes are increased which is not the case for the Conservative DEVS protocol where the performance starts to drop down as more nodes are engaged. The outstanding null message reductions are presented in Figure 10, 12, and 14 for the three different sizes of the model.

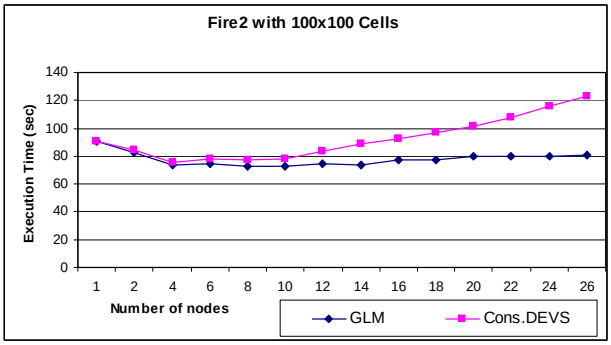

Figure 9. Total Execution Time of Fire2 (100x100 cells)

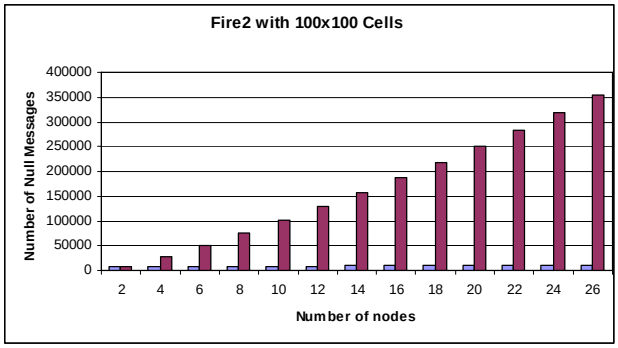

Figure 10. Total Number of Null Messages of Fire2 (100x100 cells)

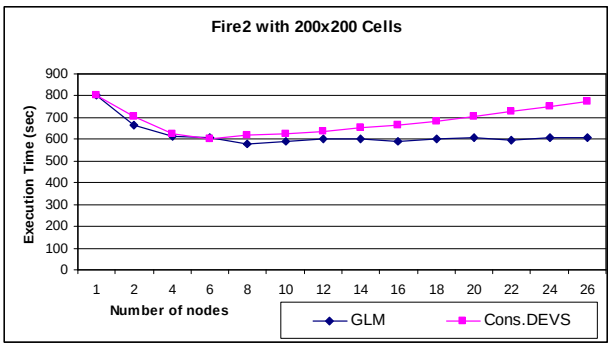

Figure 11. Total Execution Time of Fire2 (200x200 cells)

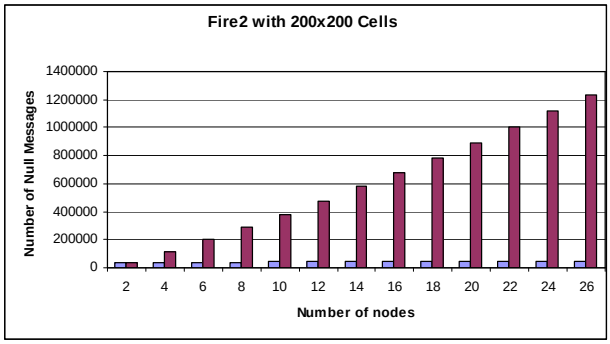

Figure 12. Total Number of Null Messages of Fire2 (200x200 cells)

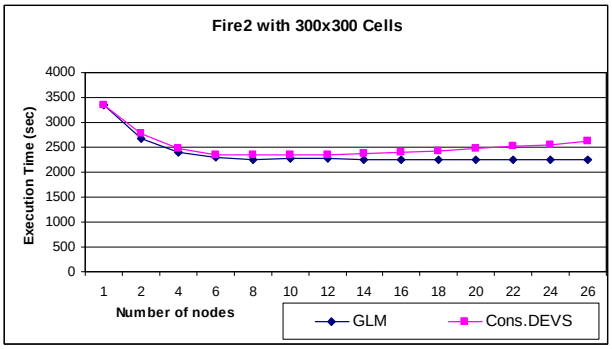

Figure 13. Total Execution Time of Fire2 (300x300 cells)

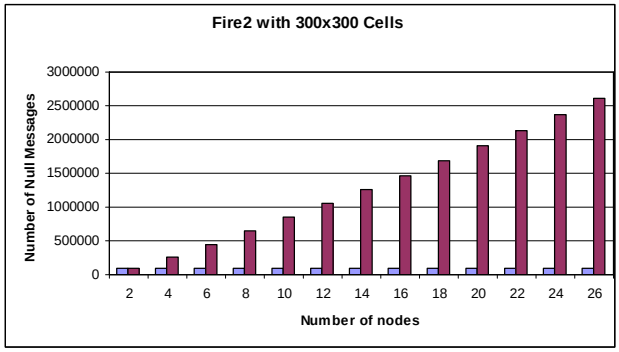

Figure 14. Total Number of Null Messages of Fire2 (300x300 cells)

The calculated null message reduction factor for *Fire2* is given in Table 4. Table 5 shows the speedups of both algorithms compared to the sequential results (running the simulation on a single node with a sequential simulator). It is shown that the GLM protocol always outperforms the Conservative DEVS algorithm.

Table 4. NMR for Different Sizes of Fire2 Model

Size	Metric	2	4	6	8	10	12	14	16	18	20	22	24	26
100x100	NMR	1.23	4.00	6.77	9.52	12.21	14.85	17.46	20.02	22.55	25.08	27.53	30.01	32.43
200x200	NMR	1.00	3.13	5.28	7.44	9.59	11.75	13.90	16.05	18.20	20.34	22.48	24.61	26.75
300x300	NMR	1.00	3.02	5.09	7.15	9.23	11.30	13.36	15.44	17.50	19.57	21.64	23.71	25.77

Table 5. GLM vs. Conservative DEVS Speedups for Fire2 Model

Size	Protocol	2	4	6	8	10	12	14	16	18	20	22	24	26
100x100	GLM	1.10	1.24	1.22	1.25	1.26	1.21	1.23	1.18	1.18	1.14	1.14	1.13	1.12
100x100	Con.DEVS	1.08	1.21	1.16	1.17	1.16	1.09	1.02	0.98	0.94	0.90	0.85	0.79	0.74
200x200	GLM	1.21	1.30	1.33	1.39	1.35	1.33	1.34	1.36	1.34	1.32	1.35	1.32	1.32
200x200	Con.DEVS	1.14	1.29	1.34	1.30	1.28	1.26	1.23	1.21	1.18	1.14	1.11	1.07	1.04
300x300	GLM	1.24	1.39	1.46	1.49	1.48	1.46	1.48	1.48	1.49	1.48	1.49	1.49	1.49
300x300	Con.DEVS	1.20	1.35	1.42	1.42	1.42	1.42	1.41	1.40	1.38	1.34	1.32	1.30	1.27

The total execution time results for Watershed model are given in Figure 15, 17, and 19 for the three different sizes respectively. Watershed model is three dimensional and communication-intensive, thus, higher speedups are achieved with both protocols compared to Fire1 and Fire2. However, again the GLM protocol significantly shows better speedups (Table 7) in all scenarios. The total numbers of null messages for different Watershed simulations are shown in Figure 16, 18, and 20. As presented by Table 6, the null message reduction factor is significantly high.

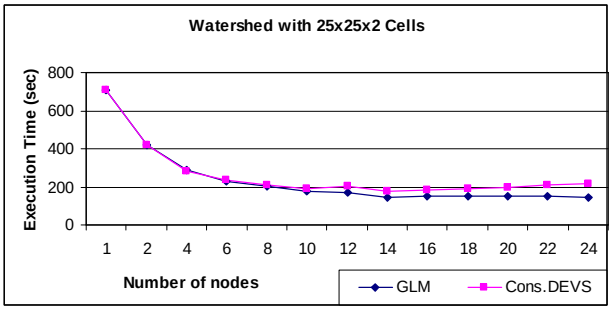

Figure 15. Watershed (25x25x2 cells) Execution Time

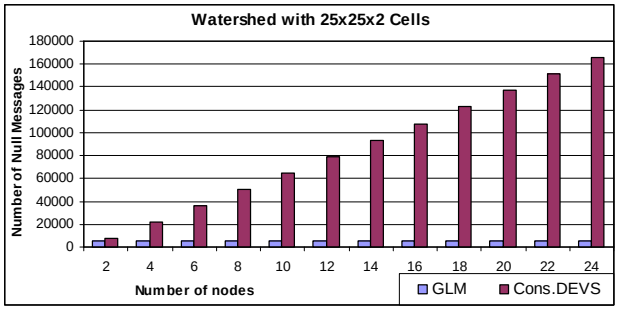

Figure 16. Watershed (25x25x2 cells) Total Number of Null Messages

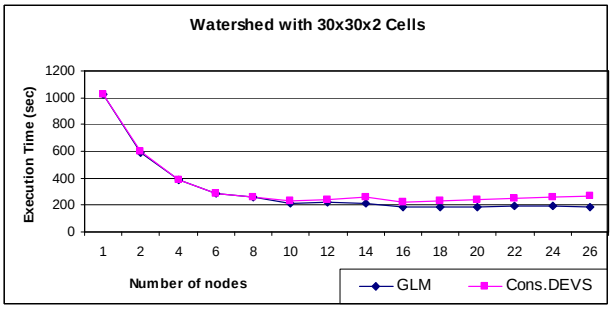

Figure 17. Watershed (30x30x2 cells) Execution Time

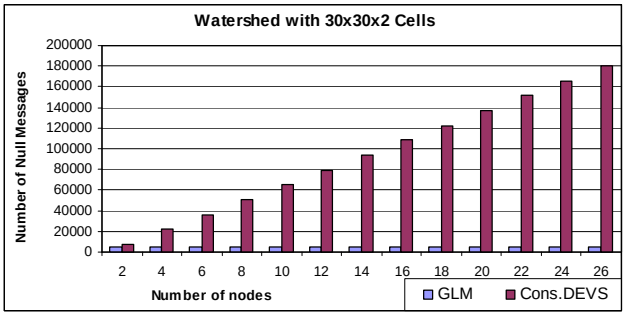

Figure 18. Watershed (30x30x2 cells) Total Number of Null Messages

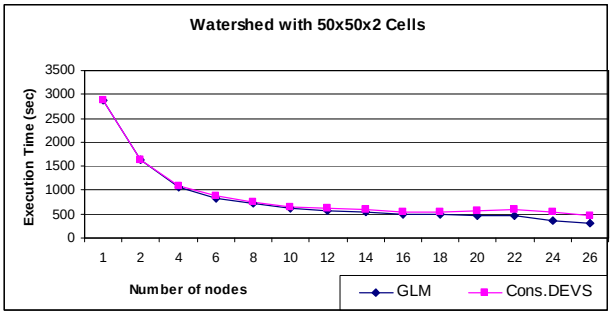

Figure 19. Watershed (50x50x2 cells) Execution Time

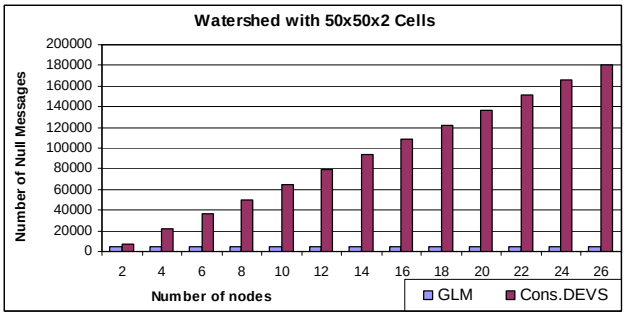

Figure 20. Watershed (50x50x2 cells) Total Number of Null Messages

Table 6. NMR for Different Sizes of Watershed Model

Size	Metric	2	4	6	8	10	12	14	16	18	20	22	24	26
25x25x2	NMR	1.33	4.00	6.67	9.33	12.00	14.66	17.33	19.96	22.62	25.32	27.98	30.31	-
30x30x2	NMR	1.33	4.00	6.67	9.33	12.00	14.66	17.33	20.00	22.65	25.32	27.96	30.61	33.22
50x50x2	NMR	1.33	4.00	6.67	9.33	12.00	14.66	17.33	20.00	22.66	25.33	27.99	30.66	33.33

Table 7. GLM vs. Conservative DEVS Speedups for Watershed Model

Size	Protocol	2	4	6	8	10	12	14	16	18	20	22	24	26
25x25x2	GLM	1.69	2.48	3.08	3.53	4.07	4.19	4.89	4.81	4.74	4.65	4.79	4.85	-
25x25x2	Con.DEVS	1.69	2.49	3.02	3.34	3.72	3.52	4.09	3.87	3.71	3.57	3.39	3.30	-
30x30x2	GLM	1.75	2.62	3.61	4.00	4.75	4.69	4.73	5.51	5.48	5.47	5.36	5.40	5.47
30x30x2	Con.DEVS	1.72	2.68	3.61	3.94	4.40	4.23	4.01	4.70	4.47	4.27	4.08	3.95	3.78
50x50x2	GLM	1.75	2.69	3.44	3.89	4.68	4.94	5.20	5.71	5.95	6.07	6.03	7.66	8.92
50x50x2	Con.DEVS	1.77	2.66	3.27	3.87	4.43	4.55	4.84	5.19	5.19	5.09	4.74	5.34	6.21

7. CONCLUSION AND FUTURE WORK

We presented the GLM protocol for parallel conservative simulation of large scale DEVS-based models. The protocol overcomes the issue of large number of null messages of the Conservative DEVS algorithm by introducing a global lookahead manager that takes care of null message reception and distribution. The simulation is divided into cycles of two distinguished phases: a *parallel phase* which corresponds to the duration where all LPs are busy performing parallel computations, and a *broad-*

cast phase where LPs are suspended and waiting for the global lookahead manager to allow them advance their LVTs. The GLM protocol is asynchronous and the central lookahead manager is not expected to be a bottleneck since the only message transmissions involving it take place at the end of *Parallel* phase and *Broadcast* phase. In fact, the LM does not carry out any computation and it is only invoked when all LPs are blocked and the simulation is suspended. Thus, the centralized fashion of the GLM does not introduce any overhead. The results presented in this paper showed that the GLM protocol not only reduces the total number of null messages, but it significantly improves the performance and higher speedups are achieved.

8. REFERENCES

[1] Fujimoto, R. M. Parallel and distributed simulation systems. New York: Wiley. 2000.

[2] D. R. Jefferson. 1985. "Virtual time". ACM Trans. Program. Lang. Syst. 7(3), pp. 404-425.

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

[4] 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.

[5] Zeigler, B., T. Kim, and H. Praehofer. 2000. Theory of modeling and simulation: Integrating discrete event and continuous complex dynamic systems. San Diego: Academic Press.

[6] Chow, A. C. and B. Zeigler. 1994. "Parallel DEVS: A parallel, hierarchical, modular modeling formalism". In Proceedings of the Winter Computer Simulation Conference, Orlando, FL.

[7] 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. Argentina. 1998.

[8] Wainer, G. 2002. CD++: A toolkit to develop DEVS models. Software – Practice and Experience, 32:1261-1306.

[9] Jafer, S., Wainer, G., "Conservative DEVS - A Novel Protocol for Parallel Conservative Simulation of DEVS and Cell-DEVS Models". Submitted to DEVS Symposium - Springsim'10. Florida. USA. 2010.

[10] Jafer, S., Wainer, G., "A Performance Evaluation of the Conservative DEVS Protocol in Parallel Simulation of DEVS-based Models". SummerSim'10, Ottawa. 2010.

[11] Lubachevsky, B.D. "Efficient distributed event-driven simulations of multiple-loop networks". Communication. ACM 32, (January 1989), 111-123.

[12] L Lamport, KM Chandy. "Distributed snapshots: Determining global states of distributed systems". ACM Transactions on Computer Systems, 1985.

[13] Chandy, K., and Misra, J. "Asynchronous Distributed Simulation via a Sequence of Parallel Computations". Comm. of the ACM 24, 2 (1981), 198-205.

[14] Q. Liu, G. Wainer, "Parallel environment for DEVS and Cell-DEVS models". SIMULATION 83(6), 2007, pp.449-471.

[15] Radhakrishnan, R., D. E. Martin, M. Chetlur, D. M. Rao, and P. A. Wilsey. 1998. "An object-oriented time warp simulation kernel". In Proceedings of the International Symposium on Computing in Object-Oriented Parallel Environments, LNCS 1505, pp. 13-23.

[16] Ameghino, J., A. Troccoli, and G. Wainer. "Models of Complex Physical Systems Using Cell-DEVS". The 34th IEEE/SCS Annual Simulation Symposium. 2001.

[17] Rothermel, R. "A Mathematical Model for Predicting Fire Spread in Wild-land Fuels". Research Paper INT-115. Ogden, UT: U.S. Department of Agriculture, Forest Service, Intermountain Forest and Range Experiment Station. 1972.

[18] C. D. Bevins, "fireLib User Manual and Technical Reference". http://www.fire.org/, accessed in Dec. 2008.

[19] G. Wainer, "Applying Cell-DEVS methodology for modeling the environment". SIMULATION 82(10), 2006, pp.635-660.