Conservative vs. Optimistic Parallel Simulation of DEVS and Cell-DEVS: A Comparative Study

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

Keywords: Conservative DEVS, discrete-event simulation, dynamic lookahead, optimistic DEVS.

Abstract
The conservative Parallel DEVS protocol offers a novel approach that allows conservative simulation of DEVS-based PDES systems. The protocol is based on the classical Chandy-Misra-Bryant synchronization mechanism, and it extends the DEVS abstract simulator to provide means for lookahead computation and null-messages. We present a purely conservative simulator, called CCD++, designed for running large-scale DEVS and Cell-DEVS models in parallel and distributed fashion. A comparative performance analysis is presented, analyzing the performance of CCD++ compared to an optimistic DEVS simulator. Several DEVS-based environmental models with different characteristics are studied. The experiments indicate that the conservative simulator improves performance in terms of execution time, memory usage, operational cost, and system stability for very large models.

1. INTRODUCTION

Many studies have been carried out to show the differences between parallel synchronization mechanisms. Conservative and optimistic approaches are the two major classes of parallel synchronization techniques proposed for parallel discrete event simulation (PDES). Conservative synchronization [1] has more limited parallelism when compared to optimistic algorithms. Conservative approaches strictly avoid causality violations while optimistic approaches [2] allow violations and recover from them by providing a rollback mechanism (which is more costly in rollback and state saving overhead). The Chandy-Misra-Bryant (CMB) [3,4] conservative mechanism prevents deadlocks by introducing null-messages and the notion of lookahead.

We are interested in combining advanced parallel simulation algorithms for large scale simulations. We want to combine the formal advantages of the DEVS (Discrete Event System Specification) [5] formal modeling and simulation (M&S) frame work with parallel simulation algorithms. DEVS provides a discrete-event approach to construct hierarchical models, and P-DEVS [6] provides a more elegant mechanism for handling simultaneous events (allowing efficient execution of parallel models). Cell-DEVS [7] allows defining n-dimensional cell spaces to represent complex discrete event spatial models where each cell is a DEVS component.

Parallel simulation of complex DEVS-based models requires a robust simulator with low synchronization overhead. PCD++ [8] is an optimistic simulator for DEVS and Cell-DEVS which implements the Lightweight Time Warp (LTW) protocol. Although PCD++ reduced the overhead of optimistic protocols to minimum, the issue of enormous memory consumption due to state savings and rollbacks still remains. This is especially apparent when the number of participating nodes increases; resulting in cascaded rollbacks, and in further memory and computation overhead. In order to experiment with both methods within the DEVS modeling framework, we introduced a conservative DEVS protocol and implemented a purely conservative simulator for DEVS and Cell-DEVS [9]. This simulator, called CCD++ (Conservative CD++), is based on the classical CMB approach with deadlock avoidance. As we will show, the dynamic lookahead (which can be automatically derived from the model specification) and the efficient low-cost lookahead computation strategy of CCD++ have outperformed our results obtained for the optimistic algorithm (PCD++) when large models are simulated (some of them with 250,000 cells in total). In this paper we present a comparative study of the performance of conservative and optimistic simulation of DEVS-based environmental systems in terms of memory consumption and execution speedup.

The rest of the paper is organized as follows. Section 2 introduces the motivation and related work. Section 3 recaps the conservative DEVS protocol in the context of CCD++. Section 4 gives the performance analysis. Conclusion and future work are presented in Section 5.

2. RELATED WORK AND GOALS

A real system modeled using DEVS is described as a composition of behavioral (*atomic*) and structural (*coupled*) model components. A DEVS atomic model is formally defined by:

$$M = <X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta>$$

At any time, a DEVS atomic model is in a state $s \in S$. In the absence of external events, the model will stay in this state for the duration specified by *ta(s)*. When the elapsed time *e*, is equal to *ta(s)*, the state duration expires and the atomic model sends the output $\lambda(s)$ and performs an internal transition to a new state specified by $\delta_{int}(s)$. Transitions that occur due to the expiration of *ta(s)* are called internal transitions. However, the arrival of an external event can also cause state transition, which places the model into a new state specified by $\delta_{ext}(s,e,x)$; where *s* is the current state, *e* is the elapsed time, and *x* is the input value. Atomic models can be integrated into larger *coupled* models, which can include one or more atomic or coupled models hierarchically composed. DEVS Abstract simulation algorithm defines two different simulation entities (called DEVS *Processors*): *Simulators* and *Coordinators*. Simulators execute atomic DEVS models, and Coordinators are paired with coupled models. In fact, Simulators are in charge of invoking the *internal* and *external* functions, while the Coordinators must route input/output events and schedule the imminent dependant(s) in the model hierarchy.

Various parallel DEVS M&S toolkits have been implemented by different researchers including: DEVS-C++ [10], DEVS/CORBA [11], DEVSCluster [12], DEVS/P2P [13], DEVS/RMI [14], DEVSim++ [15], and P-DEVSim++ [16]. In [17] a distributed simulation strategy for DEVS is presented which combines conservative and risk-free optimistic strategies. [18] presents an implementation of the parallel DEVS simulation protocol that uses a modified Time Warp optimistic algorithm [19] for shared memory multiprocessor machine. In terms of conservative DEVS-based simulations, there has been a large number of work done by integrating DEVS with HLA [20], allowing DEVS tools to use the synchronization services provided by HLA. In this case, DEVS atomic components are defined as HLA federates communicating by exchanging messages through the RTI. In [21] Ziegler proposed the first integrating algorithm of DEVS models into a HLA-compliant environment, which was based on the classical CMB synchronization mechanism, and used the conservative algorithm provided by HLA. However, this approach was prone to deadlock which was later resolved in [22]. Other HLA-based DEVS conservative simulators were proposed in [23-25].

There has been some research done outside the HLA domain. For instance, Zeigler [5] introduced conservative parallel simulation of DEVS models based on the classical CMB approach with deadlock avoidance and the Yaddes [26] algorithm. The principal idea behind this method is to maintain a network of correlated earliest output time (EOT) and earliest input time (EIT) estimates, which matches the output-to-input coupling structure of the DEVS coupled model. The EOT/EIT estimates represent the time information distributed via null-messages. Under this scheme, the lookahead calculation is performed at each DEVS *Simulator*, by looking at input and output ports. There are two limitations associated with this technique: a) a large number of EIT and EOT computations are required (since the algorithm is implemented at the *Simulator* level, the overhead increases as the model size grows; we need one *Simulator* per *atomic* component); b) a large number of null-messages are sent among processors, since both EIT and EOT must be distributed, as opposed to sending only one type of information (i.e. only lookahead).

The need for a robust conservative simulator and the limitations of the original conservative DEVS algorithm led us to propose a novel representation of the conservative DEVS algorithm, and the first conservative Cell-DEVS simulator, namely, CCD++. Our algorithm is implemented at the topmost level of the abstract simulator hierarchy (i.e. the *Coordinator*), reducing the frequency of computation. Also, EIT and EOT calculations are replaced with a single lookahead computation, reducing the total number of null-messages significantly. Deadlock avoidance is maintained through a simple strategy which ensures that a process first sends its latest calculated lookahead and then suspends. In this way, it is guaranteed that a suspended process does not cause deadlock. Another advantage of the algorithm is the dynamic lookahead calculation, done at low-cost and extracted from already existing data (derived from the model specification).

3. DEVS-BASED CONSERVATIVE SIMULATION

This section briefly recaps the conservative DEVS protocol in the context of CCD++, while details of the algorithm can be found in [9].

3.1. CCD++ Architecture

CCD++ is built on top of the WARPED kernel [11], which serves as a service provider for defining different types of processes (simulation objects). Simulation objects mapped on a physical processor are grouped by a logical process (LP). WARPED relies on the Message Passing Interface (MPI) and serves as a middleware to provide scheduling, memory, file, event, communication and time management. The major part of our conservative algorithm was implemented at the CCD++ level. Some

modifications were made at the WARPED layer to comply with the requirements of the conservative algorithm.

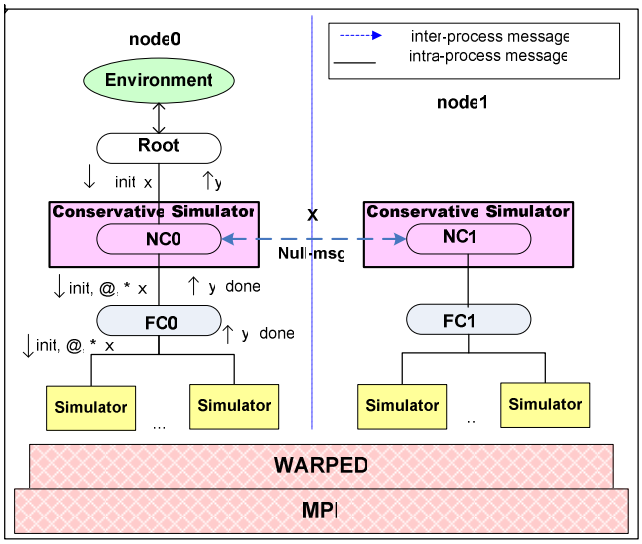

Figure 1. CCD++ Processors Hierarchy and Messaging.

The simulation is executed in a message-driven fashion. Figure 1 represents the processors hierarchy as well as the messages interchanged. At the beginning of the simulation, one LP is created on each machine (physical process). Then, each LP hosts one or more DEVS *Processors*. CCD++ employs a flat architecture, including a *Node Coordinator* (*NC*), a *Flat Coordinator* (*FC*), and a set of *Simulators* on each node. A special Coordinator, called *Root* is created on node 0, which interacts with other NCs using inter-process messaging (for remote NCs) and intra-process messaging (for local NCs). *Root* is in charge of starting the simulation, and performing I/O operations among the simulation system and the surrounding environment. Only one NC is created on each machine, acting as the central controller on the hosting LP. The NC is the parent coordinator for the FC, and routes remote messages received from *Root* or other remote NCs to the FC. The Simulators are the child processors of the local FC. Our conservative algorithm is implemented at the NC.

The DEVS processors exchange *content* and *control* messages. The first category includes the *external* (*x*) and the *output* messages (*y*); the second category includes the *initialization* (*I*), the *collect* (*@*), the *internal* (*), and the *done* messages (*D*). In order 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 in the atomic models, and *done* messages carry scheduling information.

3.2. Conservative Mechanism

In [9] we proposed our Conservative DEVS algorithm where processes communicate only through messages with their neighbors; there are no shared variables and there is no central process for message routing or process scheduling. Although each LP has its own Local Virtual Time (LVT) no events are received at the virtual past time. Synchronization is merely maintained through null-messages carrying out lookahead information. The mechanism is based on the classical CMB approach with deadlock avoidance. Here, we provide a summarized report on the conservative simulation in CCD++ while the details have been reported earlier in [9].

In CCD++, the conservative algorithm is implemented at the NC which is the central synchronizer for driving the simulation on that machine. The key focus of the algorithm is on computing the lookahead and sending it via null-messages and the decision to suspend or resume the LP. Thus, the NC is responsible for lookahead calculation and sending it via null-messages, suspending the LP, receiving null-messages from other LPs while the LP is blocked, and resuming the LP when all remote null-messages are received. Hence, the NC drives the simulation at the LP while other DEVS processors (i.e. *FC*, *Simulator*, *Root*) are unaware of the algorithm.

The *Root* Coordinator residing on *node0* starts the simulation by sending an (*I*, *t*) message to all NCs. The simulation 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). The conservative mechanism is invoked at the beginning of every *collect* phase at the NC. The LP suspension also takes place during the *collect* phase.

Simulators can only communicate with their parent FC, which means there is no direct communication between *Simulators* (even local ones), and FCs are always aware of the timing of state changes of their child *Simulators*. When a *Simulator* sends a (*y*, *t*) message to its parent FC, the FC knows if the recipient is a local *Simulator* or a remote one (residing on another LP). In case the destination *Simulator* is a local one, it simply translates it into an

(*x*, *t*) message and sends it to the recipient *Simulator*. However, if the destination is a remote one, the FC forwards the received message (*y*, *t*) to the parent NC. The NC translates it into an (*x*, *t*) message and sends it through inter-LP communication to the parent NC of the recipient. Note that outgoing inter-LP communication happens only during the collect phases, whereas incoming inter-LP communication can occur at any phase. This implies that since *output* functions of imminent components are invoked only at 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 at any phase.

The NC is invoked when it receives a *done* message from the FC. The *done* message could be in response to an (*I*, *t*), (*@*, *t*), or (***, *t*) message previously sent to the FC. On each node, the simulation time is advanced by the NC. The NC updates the LVT of the LP at the beginning of every *collect* phase. The local FC and the *Simulators* do not send messages with a timestamp different than the current LVT.

When the NC receives a (*done*, *t*) message form the FC, it checks if the next phase of the simulation is a *collect* or *internal*. The conservative algorithm is only invoked if the next phase to take place is a *collect* one. If the NC decides to issue an *internal* phase, it first sends an (***, *t*) message to the FC. The FC will then forward this message to all imminent child *Simulators*. Internal transitions are triggered at these *Simulators* followed by *done* messages emitted to the FC reporting their next state transition time (t_N). The FC sends the closest state transition time (minimum among all t_N values) to the NC through a *done* message. In processing (*done*, *t*), the NC issues a *collect* phase and invokes the conservative mechanism. First, it performs lookahead computation. Then, the NC propagates the lookahead value to other NCs via null-messages *null* (*lookahead*, *LVT*) and gets suspended. During the suspension, the LP is still able to receive messages; however, these are only inter-LP events which are either remote *x* messages or null-messages. When the NC receives all null-messages it resumes and first calculates the new LVT.

4. PERFORMANCE EVALUATION

4.1. Experimental Platform and Metrics

To obtain a comparative study between the conservative and the optimistic DEVS simulator, both CCD++ and PCD++ were used to run extensive tests. The optimistic simulator, PCD++, implements the LTW protocol which was proposed for high-performance parallel optimistic simulation of large-scale DEVS and Cell-DEVS models [8]. The LTW protocol includes a rule-based event-scheduling mechanism using two types of event queues, an aggregated state-saving technique for optimal risk-free state management, and a new rollback algorithm that recovers lightweight LPs from causality errors without sending anti-messages. 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. For the test cases executed on multiple nodes, the results were also averaged over the participating nodes to obtain a *per-node* evaluation (i.e. MEM).

TABLE 1. PERFORMANCE METRICS

Metrics	Description
T	Total execution time of the simulation (sec)
MEM	Maximum memory consumption (MB)

4.2. Simulation Models

Three Cell-DEVS models were tested in our experiments. Two of them (namely *Fire1* and *Fire2*) simulate forest fire propagation [27] in a two dimensional cell space based on Rothermel's mathematical definition [28]. *Fire1* and *Fire2* differ in the way the spread rates are calculated. The first model uses a predetermined rates at reduced runtime computation cost, while the second one invokes the fireLib [29] 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 in a three dimensional cell space [30]. 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.

In the next section, we will show that our conservative DEVS protocol is well-suited for improving simulation performance in such scenarios.

4.3. Test Results and Analysis

For all the three models, a simple partition strategy was used which evenly divides the cell space into horizontal rectangles. In the following tables, the best execution time (T) in each series is shown in bold. The fire propagation models (*Fire1* and *Fire2*) were tested using different sizes

of cell spaces: 100 x 100, 200 x 200, 300 x 300, and 500 x 500. The *Watershed* model was tested with 25 x 25 x 2, 30 x 30 x 2, 50 x 50 x 2, and 100 x 100 x 2 cells. Each of these cases was tested on 2 to 26 nodes for both conservative (CCD++) and optimistic (PCD++) simulators. A "–" mark stands for a case where the model cannot be divided further with the given partition scheme. The highlighted entries of the table show the poor performance due to severe memory swapping, while an "x" mark indicates an out-of-memory scenario resulting a failed test case. Table 2 gives the resulting total execution time and maximum memory consumption for *Fire1* of varied sizes on different number of nodes. It is clearly shown that the conservative simulator outperforms the optimistic one in terms of memory consumption in all cases and execution time in most cases. Meaning that, significantly lower memory consumption is obtained with the conservative simulator and minimum execution times are achieved at smaller number of nodes for the four different sizes. Although the smallest execution time for different series was achieved by the optimistic simulator, but this was gained at the cost of a significantly larger memory consumption and larger number of participating nodes. The memory consumption by the optimistic simulator tends to increase as the model size increases leading to failed test cases due to memory exhaustion such that the simulator was unable to run the tests for 300 x 300 cells on 2, 6, and 8 nodes, and for 500 x 500 cells for any number of nodes.

With the conservative simulator, the maximum memory usage on each node is reduced by 50% up to 95%, making it possible to execute the model using smaller number of nodes at a remarkably lower simulation cost. An outstanding improvement is achieved in terms of the total execution time where a reduction of 40% up to 90% is observed while a much smaller memory is consumed at the same time.

As we can see, in each different size, using the conservative simulator reduces the execution time as the number of nodes increases until it reaches the best execution time (the T value in bold). Meaning that, for the four mentioned model sizes, the smallest execution time is achieved at 4, 6, 12, and 16 nodes respectively and after that the execution time starts to increase as more nodes are engaged. Therefore, for each scenario, the number of nodes at the smallest execution time represents the threshold value where adding more nodes would not reduce the execution time. This is merely due to the fact that when a model, especially a small one, is partitioned onto more and more nodes, the increasing overhead involved with inter-LP communication and null-messages eventually degrades the performance. On the other hand, with the optimistic simulator, the smallest execution time for each size is achieved at larger number of nodes (20, 22, and 24 nodes for 100x100, 200x200, and 300x300 cells respectively). This indicates the high memory consumption drawback of the optimistic mechanism resulting the need for a larger number of nodes to obtain smaller execution times, making it impossible to execute very large models (in case of *Fire1*, the simulator fails to run sizes larger than 300 x 300 cells).

Figure 2 illustrates the speedups for different sizes of *Fire1* with respect to the results of the sequential simulator. From the speedup graphs of the conservative simulator it is clearly observed that the simulator shows better performance as the model size grows. That is, the highest speedups are achieved when the model size is 500x500 cells. Also, for each particular size, the speedup starts dropping after the threshold node. The optimistic simulator behaves differently where better speedups are achieved at smaller model size.

TABLE 2. FIRE1 EXECUTION RESULTS

Size	Sequential	Simulator	Metric	2	4	6	8	10	12	14	16	18	20	22	24	26
100x100	34.25 (T) 50.68(MEM)	Cons.	T	29.42	**28.73**	30.58	32.79	36.27	38.61	42.46	46.91	52.06	56.01	62.82	70.22	76.21
			MEM	32.63	22.49	19.16	17.32	16.36	15.71	15.29	14.95	14.67	14.43	14.27	14.14	14.01
		Optim.	T	48.06	27.15	24.23	22.68	22.32	21.36	21.03	21.34	21.21	**20.66**	21.44	21.32	21.44
			MEM	373.25	271.62	160.26	110.94	82.65	66.75	55.65	48.18	43.55	38.92	36.22	34.05	32.30
200x200	584.759(T) 196(MEM)	Cons.	T	438.57	392.55	**391.26**	394.09	404.56	416.14	428.58	444.41	462.60	487.74	502.43	522.99	552.16
			MEM	122.00	83.42	69.85	62.80	58.70	56.03	54.00	52.61	51.48	50.54	49.81	49.16	48.67
		Optim.	T	5149.62	971.42	1729.93	794.50	411.21	382.92	360.69	340.97	338.66	334.94	**329.54**	333.14	334.64
			MEM	1589.5	1065.5	1531.33	1035.63	731.3	582.417	459.571	373.375	316.667	272.5	240.545	218.125	199.077
300x300	2846.37(T) 456(MEM)	Cons.	T	2280.49	1883.65	1818.61	1787.42	1766.11	**1754.99**	1784.68	1839.56	1849.42	1882.57	1924.10	2077.70	2182.76
			MEM	273.00	184.75	153.67	138.50	128.80	122.75	118.29	114.81	112.33	110.45	108.64	107.46	106.15
		Optim.	T	x	19440.20	x	5820.52	x	4996.71	5639.72	3977.63	3471.41	2447.56	1919.28	**1669.19**	1678.89
			MEM	x	1960.75	x	1483.00	x	1994.92	1904.57	1489.31	1273.72	1064.60	921.96	810.63	717.04
500x500	22883.3(T) 1272(MEM)	Cons.	T	16555.50	13843.60	13199.50	12841.20	12621.10	12451.00	12385.70	**12259.50**	12297.20	12492.70	12534.00	12671.70	13009.60
			MEM	757.00	510.00	425.00	381.63	355.90	338.83	326.43	317.31	310.06	304.00	299.46	295.58	292.42
		Optim.	T	x	x	x	x	x	x	x	x	x	x	x	x	x
			MEM	x	x	x	x	x	x	x	x	x	x	x	x	x

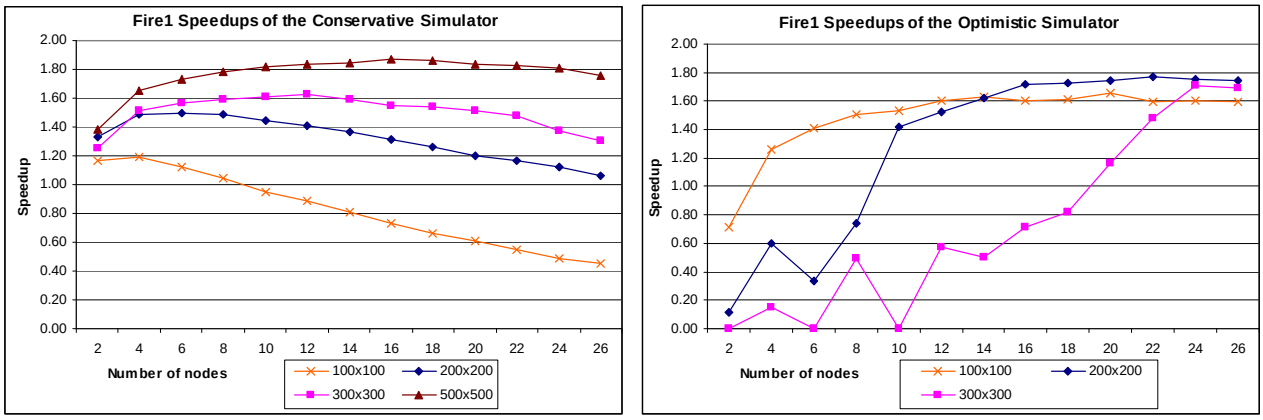

Figure 2. Fire1 Speedups for Both Simulators.

This is clearly due to its high memory consumption nature especially when a small number of nodes are used. High speedups for large sizes are only observed at large number of nodes. A speedup value of zero represents the case where the test was failed due to memory exhaustion.

TABLE 3. FIRE2 EXECUTION RESULTS

Size	Sequential	Simulator	Metric	2	4	6	8	10	12	14	16	18	20	22	24	26
100x100	90.82 (T) 51.1(MEM)	Cons.	T	84.07	**75.05**	78.27	77.51	78.51	83.25	88.69	92.30	96.99	101.36	107.42	115.56	122.60
			MEM	31.13	22.32	19.05	17.47	16.35	15.82	15.33	15.00	14.72	14.50	14.32	14.19	14.08
		Optim.	T	87.33	60.13	50.82	45.59	43.46	41.10	39.50	38.93	36.83	35.28	35.74	35.19	**35.18**
			MEM	300.50	212.50	140.50	105.47	79.13	66.57	55.12	49.22	46.32	44.05	38.19	35.63	34.29
200x200	802.99(T) 205(MEM)	Cons.	T	705.98	624.75	**600.65**	617.68	624.96	638.27	651.11	664.20	683.26	702.83	725.35	749.18	772.63
			MEM	123.50	82.88	69.89	62.67	58.87	55.94	54.11	52.63	51.41	50.51	49.80	49.22	48.65
		Optim.	T	6274.18	1182.70	2184.47	542.37	478.81	458.11	445.64	431.67	425.30	420.36	409.15	406.10	**397.81**
			MEM	1555.50	1065.00	1428.00	640.75	541.20	478.33	451.93	376.31	321.56	275.70	244.23	216.88	198.35
300x300	3338.18(T) 459(MEM)	Cons.	T	2773.40	2475.07	2357.55	2350.39	2350.56	**2349.30**	2364.75	2387.60	2424.94	2482.67	2520.49	2560.70	2618.25
			MEM	275.00	184.50	153.67	138.00	128.70	122.67	118.14	115.06	112.56	110.55	108.64	107.33	106.15
		Optim.	T	x	x	x	x	x	6662.50	7130.92	4684.61	3414.41	2172.08	2109.37	1852.18	**1835.47**
			MEM	x	x	x	x	x	2157.92	1921.00	1490.94	1269.94	1042.05	922.91	792.63	743.35
500x500	24013.2(T) 1279(MEM)	Cons.	T	18792.60	16091.50	15155.40	14762.50	14603.40	**14412.00**	14453.90	14508.10	14624.60	14686.50	14606.60	14926.90	15051.10
			MEM	768.50	510.75	424.83	381.88	356.00	338.75	326.43	317.25	310.06	304.45	299.46	295.63	292.39
		Optim.	T	x	x	x	x	x	x	x	x	x	x	x	x	x
			MEM	x	x	x	x	x	x	x	x	x	x	x	x	x

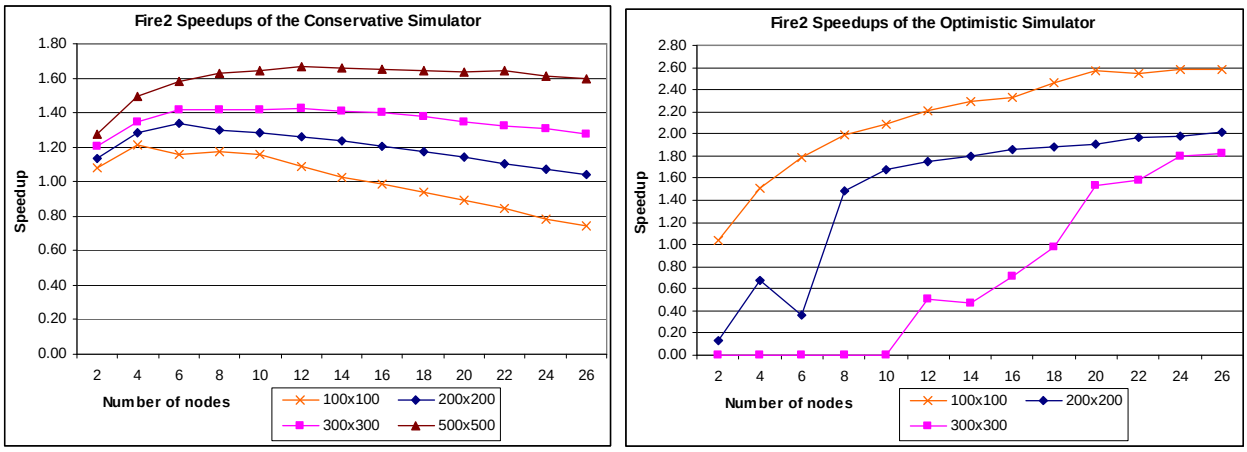

Figure 3. Fire2 Speedups for Both Simulators.

The results for *Fire2* are shown in Table 3. Similar to *Fire1* the conservative simulator reduces memory consumption dramatically in all cases. Also, it starts reducing execution time at much smaller number of nodes compared to the optimistic one where minimum execution times are achieved at the cost of engaging more nodes. The larger the model is the better results are achieved by the conservative one such that when the optimistic simulator is still unable to execute the simulation for the 300x300 cells with up to 10 nodes, the conservative simulator is successfully reducing the execution time starting with 2 nodes. The results show that the smallest execution times reported by the conservative simulator for the first three sizes are almost 1.5 times of the minimum execution times achieved by the optimistic one (i.e. 75.05 vs. 35.18, 600.65 vs. 397.81, and 2349.30 vs. 1835.47 for the first three model sizes respectively). This is

mainly because *Fire2* is computation intensive and each computation takes longer time compared to *Fire1*, resulting longer LP suspensions, thus, longer execution times. Since the suspension-free nature of the optimistic simulator is not affected by this characteristic of the model, the optimistic simulator achieves lower execution times but at the cost of much more number of nodes. The speedups achieved by both simulators are presented in Figure 3. The speedup graphs clearly show higher speedups are achieved by the conservative simulator as the model becomes larger. Similar to *Fire1* the highest speedups are obtained at the 500x500 cells model. As in *Fire1* the optimistic simulator shows higher speedups when a smaller model is simulated. The larger the model is the lower speed up is observed for this simulator. Table 4 gives the results for *Watershed* model. Outstanding performance is achieved by the conservative simulator with high speedups compared to the sequential simulator for the four different model sizes. As in other models presented here, the conservative simulator reduces memory consumption remarkably in all cases compared to the optimistic one. Also, lower execution times are achieved when a small number of nodes are used. Since the *Watershed* model is communication intensive, the simulation requires numerous number of inter-LP messages limiting the parallelism of the conservative simulator as the LPs are required to suspend more often which would result in more null-messages produced and longer suspension periods. This behavior degrades the performance of the conservative simulator resulting higher execution time compared to the optimistic one when larger number of nodes are used. The speedups graphs shown in Figure 4 illustrate the performance achieved by the two simulators compared with the sequential results.

TABLE 4. WATERSHED EXECUTION RESULTS

Size	Sequential	Simulator	Metric	2	4	6	8	10	12	14	16	18	20	22	24	26
25x25x2	710.24 (T) 91.88(MEM)	Cons.	T	419.53	285.01	234.89	212.73	190.93	201.82	**173.80**	183.52	191.39	198.77	209.62	215.12	-
			MEM	48.70	31.96	24.24	19.94	17.38	15.33	14.13	13.01	12.14	11.45	10.88	10.41	-
		Optim.	T	469.65	306.25	257.18	195.16	176.19	172.39	**136.18**	136.37	142.69	143.86	139.54	141.85	-
			MEM	70.46	164.86	128.68	131.07	132.81	132.27	153.82	141.87	128.25	114.39	113.95	103.44	-
30x30x2	1025.64(T) 131(MEM)	Cons.	T	596.24	383.29	284.21	260.23	232.97	242.63	255.48	**218.45**	229.69	240.14	251.47	259.83	271.28
			MEM	69.18	40.08	30.88	25.19	21.82	19.18	17.31	16.17	15.03	14.13	13.39	12.78	12.26
		Optim.	T	616.28	390.68	293.33	237.82	208.26	204.82	198.27	169.12	168.45	168.01	**165.54**	165.64	166.55
			MEM	89.69	163.07	164.18	151.55	171.62	148.91	138.31	117.57	139.45	156.50	149.91	130.20	122.69
50x50x2	2871.02(T) 364(MEM)	Cons.	T	1621.89	1079.23	877.52	741.16	648.49	631.08	593.39	552.99	553.24	563.53	605.10	537.63	**462.34**
			MEM	188.00	99.26	69.58	54.80	46.32	40.33	36.55	33.27	31.09	29.02	27.32	25.91	24.97
		Optim.	T	1749.99	1193.27	921.26	755.63	621.20	544.48	465.78	442.03	349.26	346.71	352.51	358.92	**282.79**
			MEM	241.00	244.25	191.67	161.25	147.60	141.67	149.93	150.81	169.00	156.50	147.09	139.54	170.61
100x100x2	11601.8(T) 1462(MEM)	Cons.	T	6583.68	3960.89	3059.01	2637.42	2298.66	2204.79	2111.54	1998.58	1902.31	1779.51	1853.76	1864.00	**1734.73**
			MEM	746.00	388.00	267.83	207.38	171.40	147.50	130.21	117.50	107.48	99.48	93.11	87.81	83.25
		Optim.	T	x	5650.52	3353.21	2679.50	2217.38	2060.85	1866.57	1761.43	1625.72	1467.14	1468.35	1434.94	**1298.03**
			MEM	x	692.75	485.33	392.88	336.60	305.50	277.71	255.44	241.33	228.55	223.27	211.67	206.92

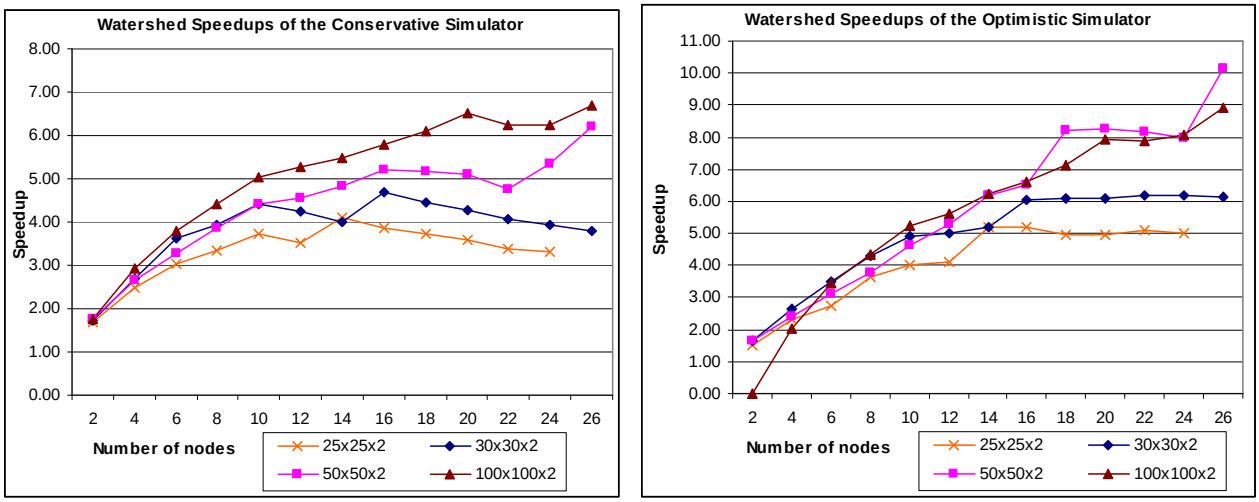

Figure 4. Watershed Speedups for Both Simulators.

As in *Fire1* and *Fire2*, the conservative simulator shows higher speedups when the model size is larger. In fact, much higher speedups are achieved at all cases of the *Watershed* model due to its complexity and size. For the optimistic simulator, unlike in *Fire1* and *Fire2*, better speedups are achieved for larger sizes. However, when the number of

nodes increases beyond a level, the speedups star to drop due to increased number of rollbacks.

5. CONCLUSION

A comparative performance analysis has been conducted to analyze the performance of CCD++ compared to an optimistic DEVS simulator (PCD++) by simulating several DEVS-based environmental models with different characteristics. The experimental results presented in this paper showed that the optimistic simulator outperforms the conservative one when the model size is small. As the model size increases better performance is achieved in terms of lower memory consumption and lower execution time by the conservative simulator. Although the optimistic simulator shows better performance at large models but this is only achieved at the cost of more participating nodes, and it is limited to a certain size. Meaning that, the optimistic simulator fails to execute very large models making the conservative one the simulator of the choice when limited memory is available and also when very large models are to be executed. We are currently working on optimizing the conservative simulator to reduce the overhead of the synchronization mechanism. We are also investigating dynamic load balancing by introducing dynamic process creation and deletion into the conservative simulator to achieve higher performance.

6. 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] Qi Liu, Gabriel A. Wainer, "A Performance Evaluation of the Lightweight Time Warp Protocol in Optimistic Parallel Simulation of DEVS-based Environmental Models", Proceedings of the 23rd ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS), Lake Placid, New York, USA - June 2009.

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

[10] Zeigler, B.; Moon, Y.; Kim, D.; Kim, J. G. "DEVS-C++: A high performance modeling and simulation environment". The 29th Hawaii International Conference on System Sciences. 1996.

[11] Zeigler, B.; Kim, D.; Buckley, S. "Distributed supply chain simulation in a DEVS/CORBA execution environment". Proceedings of the 1999 Winter Simulation Conference. 1999.

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

[13] Cheon, S.; Seo, C.; Park, S.; Zeigler, B. "Design and implementation of distributed DEVS simulation in a peer to peer network system". Advanced Simulation Technologies Conference – Design, Analysis, and Simulation of Distributed Systems Symposium. Arlington, USA. 2004.

[14] Zhang, M.; Zeigler, B.; Hammonds, P. "DEVS/RMI – An auto-adaptive and reconfigurable distributed simulation environment for engineering studies". DEVS Integrative M&S Symposium (DEVS'06). Huntsville, Alabama, USA. 2006.

[15] T.G. Kim, S.B. Park, "The DEVS formalism: Hierarchical modular systems specification in C++". In *Proceedings of European Simulation Multiconference*. 1992.

[16] Y.R. Seong, S.H. Jung, T.G. Kim, K.H. Park, "Parallel simulation of hierarchical modular DEVS models: A modified Time Warp approach". *Internat. J. Comput. Simulation 5 (3)*, 1995, pp.263-285.

[17] Praehofer, H., Reisinger, G.: "Distributed Simulation of DEVS-based Multiformalism Models". *AIS '94, Gainesville, FL, IEEE/CS Press*, Dec. 1994, pp. 150-156.

[18] James J. Nutaro, "On constructing optimistic simulation algorithms for the discrete event system specification". 2008. Transactions on Modeling and Computer Simulation.

[19] Yi Sun; Nutaro, J.; "Performance Improvement Using Parallel Simulation Protocol and Time Warp for DEVS Based Applications". Distributed Simulation and Real-Time Applications, 2008. DS-RT 2008. 12th IEEE/ACM International Symposium. 2008. Page(s):277 – 284.

[20] IEEE std 1516.2-2000. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification. Institute of Electrical and Electronic Engineers, New York, NY, 2001.

[21] B.P. Zeigler, G. Ball, H.J. Cho and J.S. Lee, "Implementation of the DEVS formalism over the HLA/RTI: Problems and solutions", In Simulation Interoperation Workshop (SIW), number 99S-SIW-065, Orlando, FL, 1999.

[22] T. Lake, B.P. Zeigler, H.S. Sarjoughian and J. Nutaro,"DEVS Simulation and HLA Lookahead" In Simulation Interoperability Workshop (SIW), number 00S-SIW-160,Orlando, FL, 2000.

[23] G. Zacharewicz, N. Giambiasi and C. Frydman,"Improving the DEVS/HLA Environment", In DEVS Integrative M&S Symposium, DEVS'05, Part of the 2005 SCS Spring Simulation Multiconference, SpringSim'05, San Diego, CA, USA, April 3-7 2005.

[24] G. Zacharewicz, N. Giambiasi and C. Frydman, "A New Algorithm for the HLA Lookahead Computing in the DEVS/HLA Environment", In European simulation Interoperability Workshop (EUROSIW),Toulouse, France, 2005.

[25] G. Zacharewicz, N. Giambiasi, C. Frydman, "Improving the Lookahead Computation in G-DEVS/HLA Environment", in: 9th IEEE International Symposium on Distributed Simulartion and Real-Time Applications (IEEE DS-RT 2005), IEEE, pp. 273 - 282, Montreal, Qc., Canada, 10-12 October 2005.

[26] E. DeBenedictus, S Ghosh, M.-L- Yu. "A Novel Algorithm for Discrete Event Simulation". IEEE Computer, June 1991, pp. 21-33.

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

[28] 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.

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

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