A Performance Evaluation of the Conservative DEVS Protocol in Parallel Simulation of DEVS-based Models

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, parallel DEVS protocol, Cell-DEVS models, DEVS models, dynamic lookahead.

Abstract
We present the performance evaluation of the Conservative DEVS protocol. This conservative algorithm is based on the classical Chandy-Misra-Bryant (CMB) synchronization mechanism, and extends the DEVS abstract simulator by providing means for lookahead computation and null message distribution. The protocol is integrated into the CD++ simulation toolkit, providing a conservative simulator (named CCD++) for running large-scale DEVS and Cell-DEVS models in parallel and distributed fashion. Throughout the experiments, we analyze four types of metrics, the total execution time, the average blocked time per node, the average number of positive events executed on each node, and the average number of null messages per node. We show a study on three environmental Cell-DEVS models, which shows that CCD++ provides considerable speedups, showing its ability for simulating large and complex DEVS-based models.

1. INTRODUCTION

Parallel and distributed simulation (PADS) is a useful method for the Modeling and Simulation (M&S) of large and complex models. In particular, parallel discrete event simulation techniques (PDES) have become the technology of choice to speed up large-scale discrete-event simulation and to make geographically distributed simulations possible. Synchronization techniques for PDES systems generally fall into two major classes of synchronization: conservative, which strictly avoid causality violations [1]; and optimistic, which allow violations and recover from them [2]. Conservative synchronization algorithms, proposed by R. E. Bryant [3], K. M. Chandy and J. Misra [4], prevents the occurrence of causality errors.

The Discrete Event System Specification (DEVS) [5] formalism is a sound M&S framework that provides a discrete-event approach to construct hierarchical modular models. The DEVS formalism has been extended to handle simultaneous event execution [6]. Parallel DEVS (P-DEVS) allows the execution of models in parallel and distributed environments while keeping the properties of the DEVS, and extending it to avoid serialization constraints.

Cell-DEVS [7] is an extension of DEVS to define cellular models, by defining every cell and coupling them together to form a complete cell space. This formalism allows defining complex cell behavior with simple instructions. It also allows construction of n-dimensional cell spaces to represent complex discrete-event models. Advanced timing behavior can be represented by defining different delays among the cells of the cell space. Only the active cells are evaluated, resulting in a noticeable overhead. CD++ [8] is an open source M&S environment, which implements DEVS, Cell-DEVS, and P-DEVS by supporting standalone and parallel/distributed simulations on different platforms. A parallel optimistic simulator, called as PCD++ [9] was developed for high-performance simulation of complex DEVS and Cell-DEVS models based on the WARPED [9] simulation kernel. Examples of other parallel DEVS M&S toolkits include DEVS-C++ [11], DEVS/CORBA [12], DEVSCluster [13], DEVS/P2P [14], DEVS/RMI [15], DEVSim++ [16], and P-DEVSim++ [17]. The DEVS parallel simulator introduced in [18] implements asynchronous simulation algorithms by combining both conservative and risk-free optimistic strategies (using a purely optimistic strategy). In [19] a new simulation algorithm for efficient distributed simulation of P-DEVS models is presented. The algorithm makes use of Java threads and performs sequential execution among the entities on each computing node while the simulation is distributed over remote nodes. We are interested in CMB-based conservative simulation by using null messages and lookahead information to synchronize among participating nodes. The issues related to performance, scalability, and complexity of optimistic-based large-scale parallel simulations motivated us to implement the first purely conservative simulator for Cell-DEVS based on the classical CMB null message synchronization algorithm. The resulting simulator, namely, Conservative CD++ (CCD++) was introduced, and its implementation details were thoroughly discussed in [19]. The purpose of the conservative DEVS protocol is to analyze the effect of conservative synchronization strategies on the overall per-

formance of the simulation. Our goal is to investigate optimistic versus conservative simulation of DEVS-based systems, so that we can provide a reference guide on whether to use a conservative simulator or an optimistic one for a particular simulation, and to find out under what circumstances one outperforms the other.

In the following sections, we will introduce extensive performance analysis of CCD++. By running extensive experimentation for three environmental Cell-DEVS models, we analyze four types of metrics, the total execution time, the average blocked time per node, the average number of positive events executed on each node, and the average number of null messages per node. Our goal is to show how the conservative mechanism of CCD++ provides considerable speedups, showing its ability for simulating large and complex DEVS-based models.

2. STRUCTURE OF CCD++

CCD++ implements the P-DEVS formalism, in which the system of interest is described as a composition of behavioral (*atomic*) and structural (*coupled*) model components. The simulation is carried out by DEVS processors which are of two types: *Simulator* and *Coordinator*. The *Simulator* represents an atomic DEVS model, where the *Coordinator* is paired with a coupled model. The *Simulator* is in charge of invoking the atomic model's *transition* and *external event* functions. On the other hand, the *Coordinator* has the responsibility of translating its children's output events and estimating the time of the next imminent dependent(s). At the beginning of the simulation, one *logical process* (LP) is created on each machine (physical process). Then, each LP will host 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 [21]. A special coordinator, called *Root* is created on machine 0 which interacts with other *NC*s using inter-process messaging (for remote *NC*) and intra-process messaging (for local *NC*). Only one *NC* is created on each machine and acts as the local 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 *NC*s to the *FC*. The *Simulators* are the child processors of the local *FC* representing the atomic components of DEVS and Cell-DEVS models. The DEVS processors exchange two categories of messages: *content* and *control*. The first category includes the *external* (*x*) and the *output* (*y*) messages, and the second includes the *initialization* (*I*), *collect* (*@*), *internal* (*), and *done* (*D*) messages. *External* and *output* messages exchange simulation data between the models, *collect* and *internal* messages trigger the output and the state transition functions respectively (in atomic DEVS models), and *done* messages handles scheduling 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.

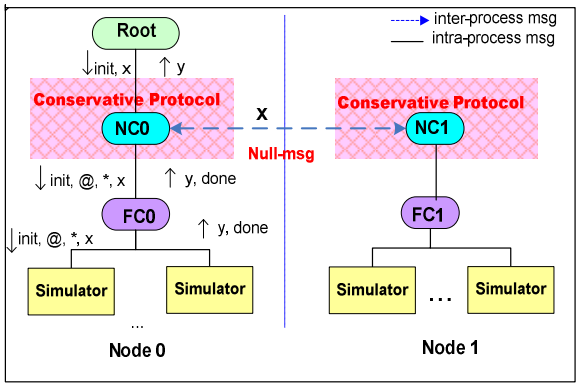

Figure 1. CCD++ Processors and Messages

3. CONSERVATIVE SIMULATION IN CCD++

In [20], we introduced a conservative DEVS algorithm which serves as the synchronization mechanism for the CCD++ simulator. The protocol is based on the Chandy-Misra-Bryant approach with deadlock avoidance. The key contribution of our approach was on an automated method to compute lookahead, and when to suspend/resume the LPs. Since the conservative algorithm is implemented at the *NC*, the simulators are unaware of its existence. The *NC* plays the role of the local conservative synchronizer by calculating lookahead and distributing it via null messages to every participating node in the simulation.

3.1. Concepts and Assumptions

The simulation starts by the *Root* coordinator (on *node0*) which sends an (*I*, *t*) message to all *NC*s. After the initialization phase, the simulation is carried on as a sequence of a mandatory transition phase and an optional collect phase on each node. Based on the LP structure and the division of functionalities in CCD++, the key characteristics of the simulation process (and the main assumptions of the conservative DEVS algorithm), are:

1. All messages from the *Simulators* must go through the parent *FC*. Hence, there is no direct communication between the *Simulators*, even local ones. This implies that *FC*s are always aware of the timing of state changes at their child *Simulators*.
2. Outgoing inter-LP communication happens only during the *collect* phase, whereas incoming inter-LP communication can occur in any phase. Since the output functions of the imminent models are invoked only in the collect phases, at any given simulation time, all the *external* messages going to remote *NC*s 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* initiates every *collect* and *transition* phase. The conservative DEVS algorithm is invoked at the *NC* when it receives a *done* message from the *FC*. The done message could be in response to a (*I*, *t*), (@, *t*), or (*, *t*) which was previously sent to the *FC*.
4. On each node, only the *NC* advances the simulation time. The *NC* calculates the Local Virtual Time (LVT) of the LP at the beginning of every *collect* phase. Local *FC*s and *Simulators* do not send messages with a timestamp different from the LVT.

3.2. A Simulation Scenario in CCD++

In this section we present a simulation scenario based on the conservative mechanism of CCD++, its flat architecture, and the messaging mechanism introduced in Section 2. As illustrated in Figure 2, only two nodes are participating. The simulation starts with message (I_1) from *Root* at the *NC* at time 0. This initialization phase ends when the two local *Simulators* (S1 and S2) send back *done* messages (D_5, D_6) to the *FC*, which causes forwarding message (D_7) to the *NC*. Every time the *NC* receives a *done* message from its *FC*, it starts the next phase immediately. However, a lookahead computation and null message distribution is performed at every collect phase. After computing the lookahead, the *NC* sends a message ($null_1$) to the remote *NC* and blocks (shaded area). The *NC* remains blocked until all remote lookaheads (carried by remote null messages) are received at the LP. During suspension the LP can still receive messages; however these messages are only inter-LP events which are either remote *x* messages or null messages. When the *NC* receives all null messages ($null_2$) it resumes and calculates the new LVT, which is equal to the state transition time that was reported by the *FC* via *done* message (D_7). At this time, all *Simulators* are imminent. Thus, the NC starts the first *collect* phase by sending a *collect* message ($@_8$) to the *FC* where it further distributes this message as two collect messages ($@_9$, $@_{10}$) to each of the *Simulators*.

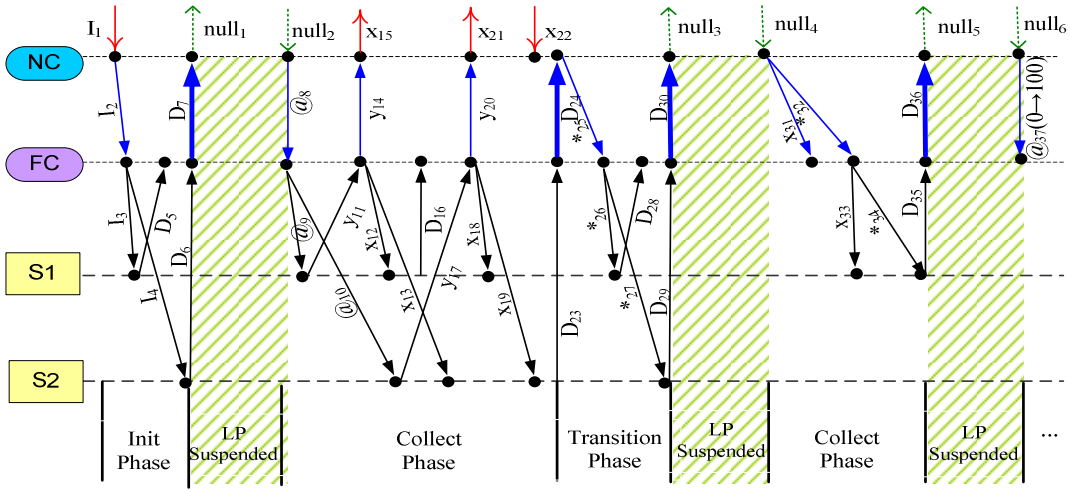

Figure 2. Sample Simulation Scenario in CCD++

Upon receiving the *collect* message, imminent *Simulators* execute their output functions and send *output* messages to their parent FC. S1 processes $@_9$ first and sends an output message (y_{11}) to the FC. If it must be sent to S1, S2 (as well as to remote *Simulators*), FC translates it into an *external* message and sends one copy to each local *Simulator* (x_{12}, x_{13}). For the remote *Simulators*, the FC then forwards the output message (y_{14}) to the NC, which translates the message into an *external* one (x_{15}) and sends it remotely to all destination NCs. Similar actions are performed when the FC processes the output message (y_{17}) from S2 (x_{18}, x_{19}, y_{20}, x_{21}). During these steps, the remote external message x_{22} is received at the NC which inserts it into the NC's Message Bag. When the FC receives the corresponding *done* messages (D_{16}, D_{23}) from S1 and S2, it sends a done message (D_{24}) to the NC, reporting the end of output operations at the local *Simulators*. This *done* message triggers the next phase at the NC, thus the first *transition* phase starts immediately by sending an *internal* message ($*_{25}$) to the FC. This message is then forwarded to imminent *Simulators* S1 and S2 ($*_{26}$,$*_{27}$). Internal transitions are triggered at these *Simulators* followed by *done* messages emitted to the FC (D_{28}, D_{29}). The FC then sends the closest state transition time to the NC through a *done* message (D_{30}). When processing D_{30}, the NC performs lookahead computation, sends the new value through $null_3$ to all remote NCs, and blocks. When the only remote null message ($null_4$) is received at the NC, the suspensions ends, and the NC calculates the new LVT. This LVT turns out to be equal to the timestamp of the external message x_{22} recently received from a remote NC and added to the NC Message Bag. Therefore, the NC sends it to the FC,

followed by another internal message (x_{31}, $*_{32}$). When FC executes $*_{32}$, it flushes x_{31} to S1 followed by $*_{34}$. External message x_{33} is added into S1's message bag, accepting the value previously transmitted by x_{22} from a remote sender. After that, the internal message $*_{34}$ invokes S1's external transition, which consumes the value wrapped in x_{33}. The resulting *done* message (D_{35}) is sent to the *FC*. When NC executes D_{36}, another lookahead computation takes place, *null5* is sent out, and the LP is blocked. After receiving $null_6$, NC calculates the new LVT. In this case there is no message in its NC Message Bag, and the remote lookahead reported by $null_6$ is larger than the closest state transition (time=100), therefore, the NC advances the local simulation time from 0 to 100 and sends to the FC a collect message ($@_{37}$) that has a send time of 0 and a receive time of 100, thereby starting a new cycle of simulation similar to that initiated by $@_8$.

4. PERFORMANCE EVALUATION

In order 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. The experiments were conducted using only one core per node. However, it is possible to make use of both cores.

Table 1 lists the metrics collected in the experiments through extensive measurements. The experimental results for each test case were averaged over 10 independent runs to strike a balance between data reliability and testing effort. For those test cases executed on multiple nodes, the results were also averaged over the participating nodes to obtain a *per-node* evaluation (i.e. BT, PEV, and NEV represent the corresponding results per one node). The PEV values present the total number of DEVS messages executed during the simulation. Note that the different memory management strategy that has been used for these experiments produced different results than those previously reported in [20].

Table 1. Performance Metrics

Metrics	Description
T	Total execution time of the simulation (sec)
BT	Total blocked time during the simulation (sec)
PEV	Total number of positive events executed
NEV	Total number of null events executed

4.1. Test Models

Three Cell-DEVS models were used in our experiments. Two of them (*Fire1* and *Fire2*) simulate forest fire propagation in a two dimensional cell space based on Rothermel's mathematical definition [22]. *Fire1* [23] and *Fire2* differ in the way the spread rates are calculated. The first model uses a predetermined rate at reduced runtime computation cost, while the second one invokes the fireLib library [24] 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, called *Watershed*, is a simulation of the environmental influence on hydrological dynamics of water accumulation in a three dimensional cell space [25]. Although *Watershed* model is not as compute-intensive as *Fire2* (577 μs state transition time), its large size (due to its 3D aspect) increases the communication intensity.

Cell-DEVS models execute a great number of simultaneous events at each virtual time, thus, we need a robust parallel simulator to handle these scenarios while keeping operational cost low. In the next section, we will show that our conservative DEVS algorithm is well-suited for improving simulation performance in such situations.

4.2. Test results and analysis

For all the models we used a simple partition strategy (evenly divide the cell space into horizontal rectangles). In the following tables, the best execution times (T) in each series are shown in bold. The *Fire* model was tested using cell spaces of 100x100, 200x200, 300x300, and 500x500. The *Watershed* model was tested with 25x25x2, 30x30x2, 50x50x2, and 100x100x2 cells. Each of these cases was tested on 1 to 26 nodes. The highlighted entries show the maximum BT value for each model's size scenario. Table 2 gives the resulting total execution time, total blocked time, as well as the number of positive and null events (T, TB, PEV, and NEV) for *Fire1* of varied sizes on different numbers of nodes.

As we can see the conservative DEVS algorithm 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 sizes, the smallest execution time is achieved at 4, 6, 12, and 14 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 execution time, which is due to the conservative overhead. For any given number of nodes, the execution time always increases with the size of the model. Moreover, the execution time rises less steeply when more nodes are used in the simulation. For example, as the model size increases from 10000 to 20000 cells, the execution time increases sharply by nearly 1562% (from 34.03 to 565.44 s) on 1 node, whereas it only rises by 1047% (from 33.24 to 381.11 s) on 8 nodes.

The execution time for the 100 x 100 model decreases from 34.03 to 29.35 s when the number of nodes climbs from 1 to 4, leading to a speedup of 1.16. For the 200x200 size, a speedup of 1.52 is observed from 1 to 6 nodes (565.44 to 373.07 s). When larger sizes are tested (300 x 300, and 500 x 500) speedups are higher. For the first case, the execution time decreases by 39% (from 2872.10 to 1765.33 s), obtaining a speedup of 1.63 from 1 to 12 nodes. Similarly, for the 500x500 model, the results show an execution time decreased by 44% from 1 to 14 nodes, achieving a speedup of 1.79. These results show that as the model's size increases, the threshold for the number of nodes also increases.

Table 2. Fire1 Test Results

Size	Metric	1	2	4	6	8	10	12	14	16	18	20	22	24	26
100x100	T	34.03	30.45	**29.35**	30.74	33.24	36.57	39.63	45.48	48.88	53.84	58.70	65.29	72.04	79.38
	BT	0	5.26	8.13	10.30	12.60	15.40	18.12	21.89	25.34	29.50	33.83	39.53	45.23	51.37
	PEV	480893	257710	141588	103212	84373	73300	66193	61224	57704	55048	53027	51555	50308	49423
	NEV	0	8434	28880	51584	76012	101798	129282	157676	187619	218517	250267	284150	317744	353674
200x200	T	565.44	437.88	379.47	**373.07**	381.11	393.54	405.42	436.87	448.82	464.80	494.47	514.08	537.55	564.70
	BT	0	74.32	68.22	71.96	77.06	86.28	97.94	113.33	127.08	142.68	160.51	181.82	203.81	227.25
	PEV	1974693	1045294	568033	409477	330925	284391	253979	232673	217086	205298	196196	189067	183319	178711
	NEV	0	36746	117476	202479	291024	382850	478378	576952	678719	783563	891688	1003337	1117362	1234774
300x300	T	2872.10	2234.75	1890.38	1812.64	1773.66	1777.89	**1765.33**	1865.92	1896.32	1908.25	1998.13	2027.88	2064.38	2140.48
	BT	0.00	367.69	299.11	255.51	241.18	248.49	266.71	295.39	322.71	356.96	391.73	433.40	479.33	528.58
	PEV	4468493	2368444	1278652	919106	739836	633439	563234	513768	477310	449441	427676	410257	396157	384612
	NEV	0	86646	265151	453414	645567	843632	1046352	1253979	1466744	1684036	1906573	2133515	2365411	2602349
500x500	T	22537.10	16476.80	14209.30	13377.10	13031.00	12891.80	12793.90	**12570.60**	13058.20	13186.60	13526.80	13491.40	13614.30	13731.90
	BT	0	2717.72	2118.52	1675.06	1427.44	1307.35	1233.48	1228.48	1247.09	1288.40	1334.66	1423.76	1512.68	1624.93
	PEV	12456093	6604744	3559387	2546967	2045699	1745542	1546399	1405946	1301356	1220766	1157317	1106088	1063961	1028964
	NEV	0	246446	745412	1252474	1776228	2305178	2840595	3387877	3941669	4502517	5073360	5651855	6237300	6831124

For example, the threshold for 100x100 size was 4 nodes while it increased to 14 nodes for the 500x500 model. However, when the number of nodes gets closer and closer to the threshold value, the difference among execution times is not significant (for example, for the 200 x 200 model, the execution time decreases by only 1.7% from 4 to 6 nodes, and for 500 x 500 cells it only decreases by 1.8 % from 12 to 14 nodes). This is because when a model, especially a small one, is partitioned onto more and more nodes, the increasing overhead involved in inter-LP communication and the increase in the number of null messages eventually degrades the performance. We need to consider the tradeoffs between the benefits of higher degree of parallelism and the associated overhead needs when choosing different partitioning strategies. The experimental results also showed that better performance is achieved on a larger number of nodes as the model's size increases. The four different sizes we chose were large enough so that the shortest execution time was achieved on 4 or more nodes.

As discussed earlier, other metrics were collected as well. Considering the total blocked time (BT) illustrated by Figures 3-6, we can see that there is a relation between BT and the number of nodes: when more nodes participate, the BT increases mainly (because each node has to wait for a larger number of null messages). This is clearly observed for smaller sizes (100x100 and 200x200), where the maximum BT is observed when there are 26 nodes in the simulation. For larger sizes, adding more nodes reduces the BT because when more nodes are engaged, the partition on each machine is smaller, thus each computation cycle is shorter and the nodes are kept waiting for a smaller amount of time. However, the BT starts to increase when more nodes are added beyond the threshold. This is when the conservative overhead and the large number of null messages cancel out the benefit of having more nodes. Aside, there are some factors that affect the BT value, such as memory congestion and inter-LP communication delays (the case for the 500x500 model where maximum BT was observed when the simulation run on 2 nodes). A "−" mark stands for a case where the model cannot be divided further with the given partition scheme.

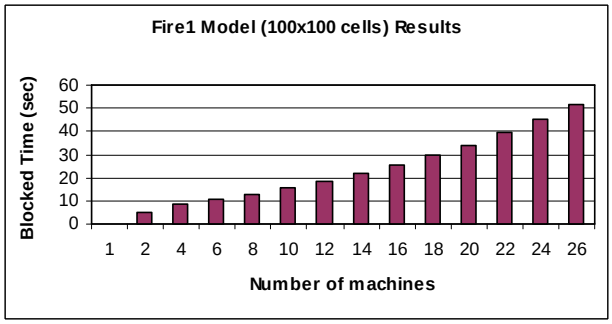

Figure 3. Fire1 Average BT per Node (100x100)

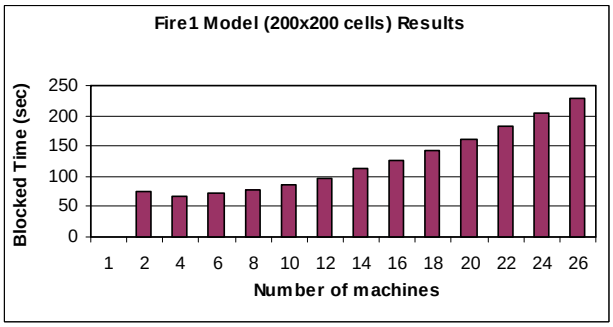

Figure 4. Fire1 Average BT per Node (200x200)

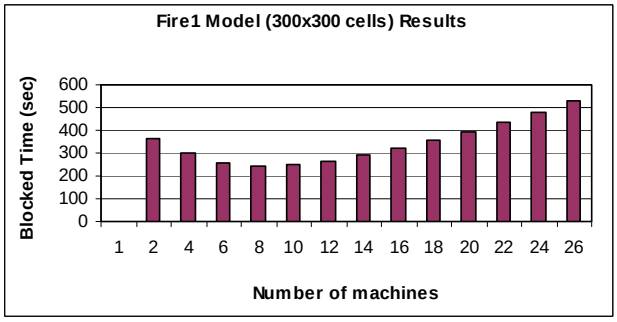

Figure 5. Fire1 Average BT per Node (300x300)

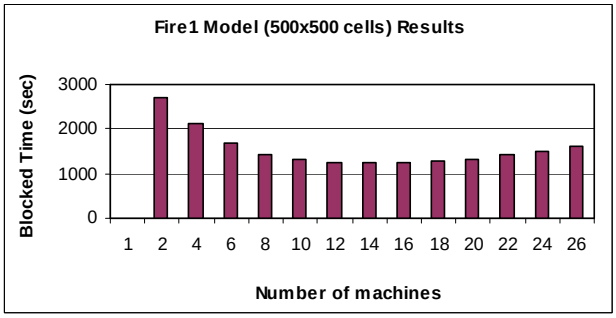

Figure 6. Fire1 Average BT per Node (500x500)

Table 3. Speedups for Watershed Model

Size	2	4	6	8	10	12	14	16	18	20	22	24	26
25x25x2	1.61	2.34	2.84	3.11	3.34	3.26	3.54	3.45	3.33	3.20	3.15	3.14	-
30x30x2	1.71	2.53	3.36	3.70	4.03	3.89	3.77	4.26	4.09	3.87	3.75	3.67	3.55
50x50x2	1.69	2.51	3.16	3.61	4.21	4.34	4.39	5.01	5.20	4.82	4.48	4.66	5.24
100x100x2	1.74	2.82	3.66	4.27	4.91	5.06	5.35	5.57	5.88	6.29	6.11	5.99	6.56

Table 4. Fire2 Test Results

Size	Metric	1	2	4	6	8	10	12	14	16	18	20	22	24	26
100x100	T	90.89	84.35	**76.99**	78.22	79.27	83.15	86.99	90.74	93.03	99.70	104.79	112.74	116.88	124.34
	BT	0	27.79	41.01	47.63	51.28	56.65	60.27	63.80	66.42	72.71	77.31	84.18	88.09	94.63
	PEV	480893	257710	141588	103212	84371	73300	66188	61224	57689	55043	53021	51548	50312	49423
	NEV	0	8434	28880	51584	76005	101798	129260	157676	187619	218483	250286	284192	317790	353674
200x200	T	834.40	677.12	635.73	635.97	**616.64**	635.72	643.95	652.21	669.24	684.58	710.11	734.30	755.05	785.13
	BT	0	197.22	253.42	273.95	273.69	297.81	307.98	318.17	340.90	355.68	373.37	396.30	416.22	442.73
	PEV	1974693	1045294	568033	409477	330925	284391	253979	232673	217085	205300	196196	189072	183317	178715
	NEV	0	36746	117476	202479	291024	382850	478378	576952	678719	783580	891688	1003379	1117339	1234749
300x300	T	3390.48	2802.32	2462.25	2340.39	2325.42	**2297**	2328.21	2327.84	2428.36	2466.22	2513.26	2559.47	2607.24	2629.33
	BT	0	647.521	720.498	710.726	738.583	741.036	769.957	788.527	815.489	858.521	899.877	957.888	1001.67	1027.69
	PEV	4468493	2368444	1278652	919106	739836	633439	563234	513768	477310	449441	427674	410257	396157	384612
	NEV	0	86646	265151	453414	645567	843632	1046352	1253979	1466744	1684036	1906554	2133515	2365411	2602349
500x500	T	23693.60	18265.80	15710.00	14771.20	14481.60	14314.50	14190.80	**14140.80**	14399.60	14725.30	14876.20	14717.60	15048.10	15124.90
	BT	0	3605.11	3318.53	3006.09	2852.53	2692.84	2700.40	2659.93	2692.48	2748.89	2810.37	2840.01	2982.60	3081.34
	PEV	12456053	6604744	3559387	2546967	2045699	1745536	1546399	1405946	1301356	1220766	1157311	1106081	1063964	1028964
	NEV	0	246446	745412	1252474	1776228	2305151	2840595	3387877	3941669	4502517	5073303	5651792	6237323	6831124

Table 5. Watershed Test Results

Size	Metric	1	2	4	6	8	10	12	14	16	18	20	22	24	26
25x25x2	T	711.93	442.57	304.57	250.75	229.14	213.18	218.21	**201.12**	206.476	214.101	222.28	226.274	226.508	-
	BT	0	22.97	51.17	49.38	51.50	48.23	60.19	48.60	64.1541	78.5136	90.9462	98.9484	102.827	-
	PEV	11095032	5880901	3266632	2395209	1959475	1698057	1523754	1389608	1263681	1165712	1087343	1023236	969795	-
	NEV	0	7204	21612	36020	50428	64836	79244	93652	108060	122468	136876	151284	165692	-
30x30x2	T	1056.84	618.77	416.90	314.64	285.64	261.95	271.75	280.56	**248.27**	258.57	273.26	281.876	288.095	298.012
	BT	0	18.76	57.42	36.97	43.89	39.99	61.27	76.37	48.79	70.41	91.86	105.885	117.03	130.344
	PEV	16016947	8404934	4591724	3320654	2685074	2303762	2049569	1867996	1731768	1589822	1476254	1383346	1305904	1240391
	NEV	0	7204	21612	36020	50428	64836	79244	93652	108060	122468	136876	151284	165692	180100
50x50x2	T	2912.06	1725.14	1158.10	921.03	805.85	692.07	671.37	663.83	581.65	559.82	603.79	649.42	624.27	**556.094**
	BT	0	23.31	142.42	131.00	129.16	92.86	127.71	157.00	115.95	110.42	153.06	203.00	185.56	130.662
	PEV	44719107	23008314	12145714	8524847	6714414	5628154	4903971	4386707	3998753	3697017	3455618	3258124	3093534	2954270
	NEV	0	7204	21612	36020	50428	64836	79244	93652	108060	122468	136876	151284	165692	180100
100x100x2	T	11678.90	6718.45	4140.18	3194.43	2737.51	2380.20	2308.64	2183.07	2097.28	1986.97	1856.45	1911.67	1949.74	**1779.19**
	BT	0	62.95	244.97	229.03	224.10	144.66	246.37	255.55	263.18	226.93	150.40	248.16	320.50	188.73
	PEV	179576007	91067514	46806064	32052247	24675339	20249194	17298430	15190742	13609976	12380491	11396904	10592150	9921515	9354061
	NEV	0	7204	21612	36020	50428	64836	79244	93652	108060	122468	136876	151284	165692	180100

There is a direct relation between the PEV values and the number of nodes. When more nodes are participating, the PEV values decrease because each node is given a smaller partition and thus less DEVS computations are required for each node. On the other hand, the PEV increases as the model size increases. Similar rule applies for the NEV results, where, the more nodes are engaged, the more null messages are sent back and forth. In addition, when the model is larger there are more DEVS computational cycles and in return, more synchronization phases, thus, more null messages to distribute.

The experimental results for the *Fire2* and *Watershed* models are shown in Table 4 and Table 5 respectively. Similar performance was achieved for *Fire2* compared to *Fire1*. For the various sizes, the smallest execution time was obtained at 4, 8, 10, and 14. Due to the complexity of the model, the execution time and the BT values are higher than those in *Fire1*. However, the behavior of the results can be explained in similar manner since the number of positive events and null messages as well as the synchronization cycles are the same in both models. One important note to consider is that the higher complexity of the evaluation rules causes the nodes to take longer time during each computation cycle, resulting a longer wait time for those nodes that are blocked waiting for the busy ones to distribute their new lookahead values. This behavior explains why the smallest execution time for 300x300 size was achieved on 10 nodes for *Fire2*, while the same model size had its shortest execution time on 12 nodes for *Fire1*.

In the *Watershed* model, outstanding speedups where achieved because the model is 3D and involves numerous cell updates, thus, benefited from parallel simulation very well. As shown on Table 4, the smallest execution time for the four different sizes were obtained at higher number of nodes (14, 16, 26, 26) compared to *Fire1* and *Fire2*. The BT values for various sizes showed similar behavior, where the BT starts to drop when more nodes are added and after a certain point it starts to increase due to the tradeoffs of parallel synchronization mechanism. Other factors that affected the BT results are the way the model is partitioned and how this affects the total number of communications required among the cells' neighbors. Table 3 presents the speedups for this model, where a speedup of 1.61 (on 2 nodes for the 25x25x2 size) up to 6.56 (on 26 nodes for the 100x100x2 size) is reported.

5. CONCLUSION AND FUTURE WORK

This paper presents performance evaluation of a novel conservative algorithm for DEVS and Cell-DEVS models implemented on CD++ simulation toolkit. The resulting parallel simulator, namely, CCD++ is based on CMB null message and lookahead concept and serves as the first purely conservative simulator for running large-scale Cell-DEVS models in parallel and distributed fashion. Performance analysis has been conducted to evaluate the conservative DEVS algorithm in simulating DEVS-based models. We showed that CCD++ simulator markedly improves execution times as the number of participating nodes increases. Considerable speedups were observed in our experiments, indicating the conservative simulator is well suited for simulating large and complex models. We are currently working on a thorough testing analysis by running intensive tests with larger and more complex models on both CCD++ and the purely optimistic simulator (PCD++) to provide a reference guide on whether to

use a conservative simulator or an optimistic one and under what circumstances one outperforms the other.

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

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

[10] 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.

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

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

[19] Himmelspach, J., R. Ewald, S. Leye, and A. M. Uhrmacher. "Parallel and Distributed Simulation of Parallel Devs Models".In Proceedings of the SpringSim '07, DEVS Integrative M&S Symposium, 249–256: SCS.

[20] Jafer, S.; Wainer, A. "Conservative vs. Optimistic Parallel Simulation of DEVS and Cell-DEVS: A Comparative Study". Proceedings of 2010 Summer Simulation Conference (SummerSim10), SCSC Symposium, page 342--350 - July 2010.

[21] Glinsky, E. and G. Wainer. "New parallel simulation techniques of DEVS and Cell-DEVS in CD++". In Proceedings of the 39th Annual Simulation Symposium, 2006, 244–251.

[22] 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.

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

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

[25] G. Wainer, "Applying Cell-DEVS methodology for modeling the environment". SIMULATION 82(10), 2006, pp.635-66.