Performance Analysis of an Optimistic Simulator for CD++

Qi Liu Gabriel Wainer
Department of Systems and Computer Engineering, Carleton University, Ottawa, Ont., Canada
{liuqi, gwainer}@sce.carleton.ca

Abstract

DEVS is a formalism to describe generic dynamic systems in a hierarchical and modular way. We present new techniques for executing DEVS and Cell-DEVS models in parallel and distributed environments based on the warped kernel. The parallel simulator PCD++ has been extended to support optimistic simulations. A non-hierarchical approach is employed to reduce the communication overhead. A two-level user-controlled state-saving mechanism is proposed to achieve efficient and flexible state saving at runtime. It is shown that optimistic PCD++ markedly outperforms other alternatives, and considerable speedups can be achieved in parallel and distributed simulations.

1. Introduction

Modeling and simulation (M&S) has become an important tool for analyzing and designing a broad array of complex systems where a mathematical analysis is intractable. As a sound formal M&S framework based on generic dynamic system concepts, the **DEVS** [1] formalism supports hierarchical and modular construction of models, allowing model reuse, reducing development and testing time. Since its first formalization, DEVS has been extended into various directions. The **Parallel DEVS** or **P-DEVS** [2] formalism is an extension that eliminates the serialization constraints. **Cell-DEVS** [3] combines Cellular Automata [4] with DEVS theory to describe n-dimensional cell spaces as discrete event models, where each cell is represented as a DEVS basic model that can be delayed using explicit timing constructions.

Parallel discrete event simulation (**PDES**) has received increasing interest as simulations become more time consuming and geographically distributed. Synchronization techniques for PDES systems generally fall into two categories: conservative approaches that strictly avoid violating causality [5], and optimistic approaches [6] that allow violations to occur, but provide mechanisms to recover from them through a process known as rollback. Usually, optimistic approaches can exploit higher degree of parallelism, whereas conservative approaches tend to be overly pessimistic and force sequential execution when it is not necessary. Moreover, conservative approaches generally rely on application-specific information to determine which events are safe to process. While optimistic algorithms can execute more efficiently if they exploit such information, they are less reliant on the application for correct execution, allowing more transparent synchronization and simplifying software development. On the other hand, optimistic algorithms may require computations with higher overhead, degrading system performance to a certain extent. The WARPED simulation kernel [7] is a configurable middleware that implements the optimistic mechanisms and various optimizations.

CD++ [8] is an M&S toolkit that implements P-DEVS and Cell-DEVS formalisms. In [9], a parallel conservative simulation engine, called as PCD++, was incorporated into CD++. It uses a centralized synchronization mechanism where the entire simulation is managed by a single root coordinator. In this work, we extend the conservative PCD++ to support optimistic simulations. While the simulator employs the same layered architecture [9], it adopts a flattened simulation structure that eliminates the need for intermediate coordinators [10]. The message-passing organization is analyzed using a high-level abstraction called wall clock time slice (WCTS). Various enhancements and optimizations are proposed and integrated into the optimistic simulator, showing that this new aproach markedly outperforms other alternatives.

2. Parallel DEVS

The **DEVS** [1] formalism provides a framework for the definition of hierarchical models in a modular way. A real system modeled using DEVS can be described as a composition of behavioral (atomic) and structural (coupled) components. The **P-DEVS** [2] formalism eliminates the restrictions that forced the original DEVS definition to sequential execution. The **Cell-DEVS** [3] formalism allows the specification of discrete event cell spaces, improving their definition by using explicit timing delays. Various DEVS-based M&S toolkits have been implemented, including:

- DEVS/CORBA [12]: a runtime infrastructure on top of CORBA to support distributed simulation of DEVS components.
- DEVS/HLA [13]: an HLA-compliant M&S environment implemented in C++ that supports high level model construction.
- DEVSCluster [14]: a CORBA-based, multi-threaded distributed simulator. It transforms a hierarchical DEVS model into a non-hierarchical one to ease synchronization.
- DEVS/Grid [15]: an M&S framework implemented using Java and Globus toolkit for Grid computing infrastructure.
- DEVS/P2P [16]: an M&S framework based on P-DEVS and P2P message communication protocol. It uses a customized DEVS simulation protocol to achieve decentralized inter-node communication.
- DEVS/RMI [17]: provides a fully dynamic reconfigurable infrastructure for handling load balancing and fault tolerance in distributed simulations. It uses the Java RMI for synchronization.

However, none of them supports optimistic simulation of Cell-DEVS models in parallel and distributed environments. In [18], a risk-free optimistic simulation algorithm is presented. In this approach, only correct outputs with the minimum global time are sent to avoid the spread of causality errors to remote processes. This mechanism is well suited for shared memory architectures, but has limitations in distributed heterogeneous environments. Optimistic PCD++ is built on top of WARPED, which provides services for defining different types of processes (*simulation objects*). Simulation objects mapped on a physical processor are grouped by an entity called as *logical process* (LP). WARPED relies on the Message Passing Interface (MPI) for both massively parallel machines and workstation clusters.

3. Optimistic simulation in CD++

PCD++ provides two loosely coupled frameworks: the modeling and simulation frameworks. The former consists of a hierarchy of classes rooted at *Model* to define the behavior of the DEVS and Cell-DEVS models; the latter defines a hierarchy of classes rooted at *Processor*, which, in turn, derives from the abstract simulation object definition in the kernel, to implement the simulation mechanisms. That is, the PCD++ processors are concrete implementations of simulation objects to realize the abstract DEVS simulators. Based on [10], optimistic PCD++ employs a flat structure with four DEVS processors: *Simulator, Flat Coordinator* (*FC*), *Node Coordinator* (*NC*), and *Root*. Introducing FC and NC eliminates the need for intermediary coordinators in the DEVS processor hierarchy. Root is no longer the global scheduler in the simulation: the simulation is managed by a set of NCs running on different machines in a decentralized manner.

Simulation is message-driven. PCD++ processors exchange messages that can be classified as *content* and *synchronization* messages. The former includes the external message (*x*, *t*) and output message (*y*, *t*), while the latter includes the initialization message (*I*, *t*), collect message (*@*, *t*), internal message (***, *t*), and done message (*D*, *t*). These messages are wrapped in kernel events and transmitted between the PCD++ processors using the functions provided by WARPED. Figure 1 shows an example of the processor structure in two machines. An LP is created on each machine, grouping PCD++ processors. Root is created only on LP0 (to start/end the simulation and perform I/O operations). NC/FC are created on each LP. FC is in charge of intra-LP communications between its child Simulators. NC is the local central controller on its LP and the end point of inter-LP communications. Simulators execute the DEVS functions defined in its atomic model.

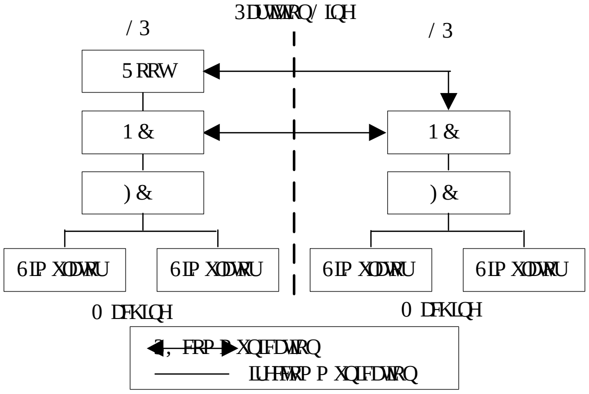


Figure 1. **Distributed processor structure**

We show a message-passing scenario using an *event precedence graph*, where a vertex (black dot) represents a message, and an edge (black arrow) represents the action of sending a message. A line with a solid arrowhead denotes a (synchronous) intra-LP message and a line with a stick arrowhead denotes an (asynchronous) inter-LP message. A lifeline (dashed line) is drawn for each PCD++ processor. Figure 2 illustrates the flow of messages on a LP with an NC, an FC, and two Simulators (S1 and S2). We do not consider out-of-order execution of messages since the rollback operations are performed automatically and transparently in the kernel.

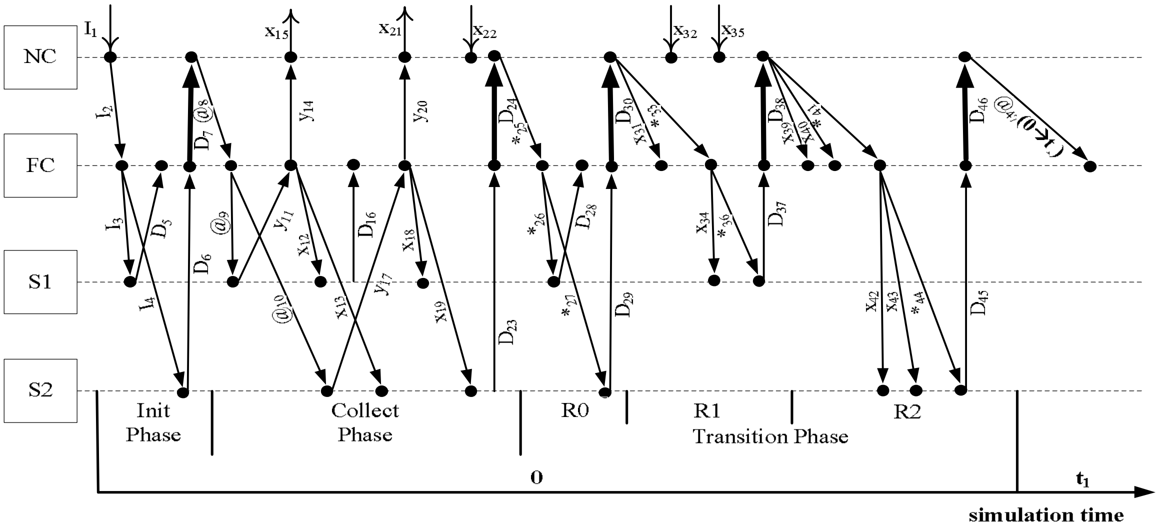
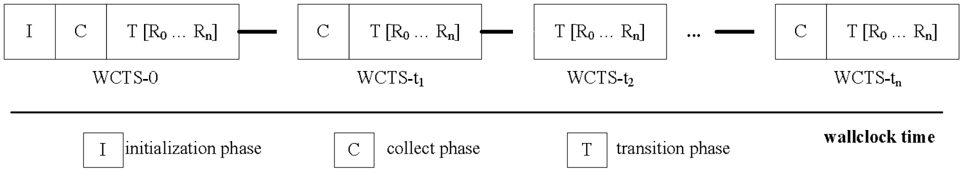


Figure 2. An example message-passing scenario on an LP

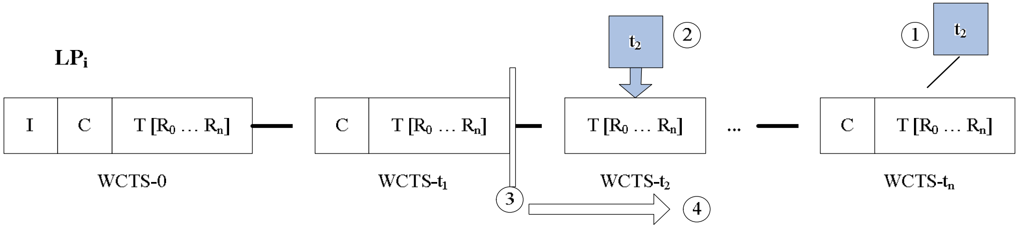
We can see that the execution of messages at any simulation time on a LP can be decomposed into at most three distinct phases: *initialization* (**I**), *collect* (**C**), and *transition* (**T**), as demarcated by done messages (bold black arrows) received by the NC. Only one initialization phase exists at time 0 ([I_1, D_7]). The collect phase at time *t* starts with a (@, *t*) sent from the NC to the FC and ends with the following (*D*, *t*) received by the NC (i.e., the collect phase at 0 comprises messages [$@_8$, D_{24}]). This phase happens if there are imminent Simulators on the LP at that time. Finally, the transition phase at simulation time *t* begins with the first (*, *t*) sent from the NC to the FC and ends at the last (*D, t*) received by the NC at time *t* (messages [$*_{25}$, D_{46}] belong to the transition phase at time 0). The transition phase is mandatory for each individual simulation time. Furthermore, a transition phase may contain multiple rounds of computations, each starts with (*x*, *t*) followed by a (*, *t*) sent from the NC to the FC and ends with a (*D*, *t*) returned to the NC (in the example, the transition phase 0 has three rounds: R_0 with messages [$*_{25}$, D_{30}], R_1 with messages [x_{31}, D_{38}], and R_2 with messages [x_{39}, D_{46}]). On each round, state transitions are performed *incrementally* with additional external messages and/or for potentially extra Simulators. Hereinafter, we will denote a transition phase of (n+1) rounds as **[R_0…R_n]**.

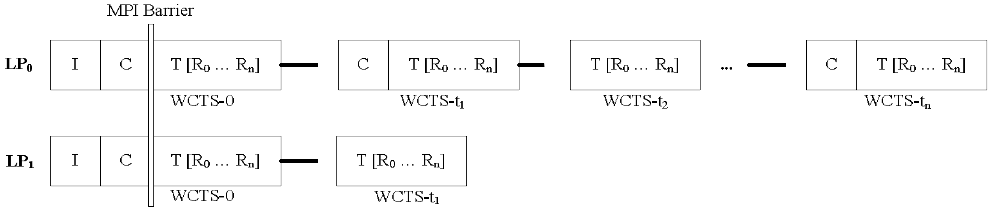
Sequential simulation on a LP can be viewed as a sequence of computation units, one for each group of simultaneous events. Each unit is performed during a timespan as measured by a physical wall clock. Such computation unit is referred to as *wall clock time slice* (WCTS). A WCTS comprising simultaneous events occurred at virtual time *t* is denoted as *WCTS-t*, and *t* is called as *the virtual time of the WCTS.*



neous events at a specific simulation time on all the PCD++ processors associated with the LP. Each WCTS-t may contain one mandatory transition phase and one optional collect phase. Several properties of the WCTS are summarized as follows:

- The simulation on a LP starts with WCTS-0, the only WCTS with all three phases.
- Wall clock time slices are linked together by messages sent from NC to FC (black arrows). When NC determines the next simulation time at the end of a WCTS, it sends out messages to be executed by FC, initiating the next WCTS on the LP.
- Completion of the simulation on a LP is marked by a WCTS sending out no linking messages, e.g. WCTS-t_n in the diagram. The whole simulation finishes only when all participating LPs have completed their corresponding parts of the simulation.
- Wall clock time slices are *atomic* computation units during rollback operations. A typical rollback scenario is shown in Figure 4.





As all outgoing inter-LP communication happens in the collect phases, messages with timestamp 0 are sent to remote LPs only in the collect phase of WCTS-0. The LPs are synchronized by a MPI Barrier at the end of this collect phase so that these messages can be received by their destinations before the simulation time advances beyond time 0. Therefore, no straggler with timestamp 0 will be received by any LP afterwards. Once the LPs exit from the barrier, they can safely continue optimistic execution. The cost of this approach is small, since the length of the synchronized execution is trivial when compared with the whole simulation.

4.2. User-controlled state-saving mechanism

In WARPED, the *copy state-saving* (CSS) strategy is implemented using state managers of type *StateManager*, which saves a process's state after executing each event, and the *periodic state-saving* (PSS) strategy is realized using state managers of type *InfreqStateManager* that only saves a process's state infrequently every a number of events. Simulator developers can choose to use either type of state managers at compile time. This rigid mechanism has two major disadvantages: (1) it ignores the fact that simulator developers may have the knowledge as to how to save states more efficiently to reduce the state-saving overhead; (2) it eliminates the possibility that different processes may use different types of state managers to fulfill their specific needs at runtime. To overcome these shortcomings, we introduced a two-level *user-controlled state-saving* (UCSS) mechanism so that simulator developers can utilize more flexible and efficient state-saving strategies. The structure of the UCSS mechanism is shown in Figure 6.

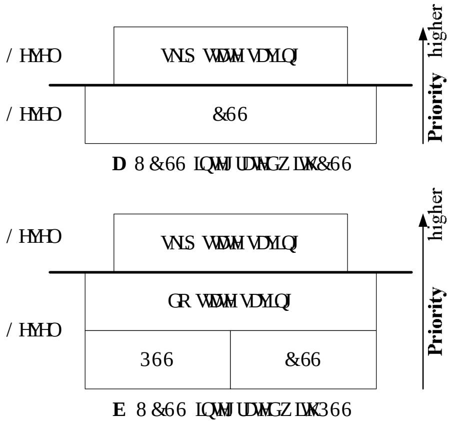


Figure 6. UCSS integrated with CSS/PSS

Therefore, a PCD++ processor can make state-saving decisions based on application-specific criteria. Further, it can dynamically switch between the CSS and PSS strategies at level 1. Thus, the UCSS mechanism virtually gives simulator developers the full power to choose the best possible combination of state-saving strategies dynamically at runtime.

4.3. Message type-based state saving

During rollbacks, the state of a PCD++ processor is always restored to the *last* state saved at the end of a WCTS with virtual time strictly less than the present rollback time. Hence, it is sufficient for a processor to save its state only after processing the last event in each WCTS for rollback. The state-saving operation can be safely skipped after executing all the other events. The last event in a WCTS is processed at the end of R_n in the transition phase. Although the actual number of rounds in a transition phase cannot be determined, we can identify the *type* of the messages executed by a given processor. For NC and FC, it must be a (*D*, *t*), and for the Simulators, it should be a (*, *t*). Therefore, PCD++ processors need to save states only after processing these particular types of messages. Since Root only processes output messages, it still saves state for each event. We call the resultant state-saving strategy as *message type-based state-saving* (MTSS). Considering that there are a large number of messages executed in each WCTS, and that they are dominated by external and output messages, MTSS can significantly reduce the number of states saved during the simulation. Further, the rollback overhead is reduced as well because fewer states need to be removed from the state queues during rollback operations. MTSS is risk-free in the sense that there is no penalty for saving fewer states.

4.5. One log file per node

Previously, one log file is created for each PCD++ processor to log the received messages in a human readable format. Depending on the size of the model, this can consume many file descriptors. In addition, creating these files and transferring data to them constitute a large operational overhead, especially when the files are accessed via a Network File System (NFS) during the simulation. To reduce the overhead of file I/O operations, a new optimization strategy, called as *one log file per node*, is implemented. Only one log file is created for the NC on each node. The NC's file queue is shared among all the processors on that node. Messages received by the NC itself are logged directly in the NC's file queue, while the other processors on that node must first get a reference to the local NC (which can be done in constant time) and then log their received messages into the NC's file queue.

5. Experimental results

Our experiments were conducted on a HP PROLIANT DL Server, a cluster of 32 compute nodes (dual 3.2GHz Intel Xeon processors, 1GB PC2100 266MHz DDR RAM) running Linux WS 2.4.21 interconnected through Gigabit Ethernet and communicating over MPICH 1.2.6. The Cell-DEVS models tested in our experiments include a model for forest fire propagation [19] based on Rothermel's mathematical definition [20] and a 3-D watershed model representing a hydrology system [19]. The following simulation results are averages over 10 independent runs. We use two different speedups in our analysis: the *overall speedup* (i.e., the total execution time as perceived by the users) and the *algorithm speedup* (i.e. without considering the simulation bootstrap time) that is used to assess the performance gain attributed to the parallel algorithms alone.

5.1. Effect of one log file per node

The performance improvement derived from the one log file per node strategy is tested using the fire propagation model of 900 cells arranged in a 30×30 mesh.

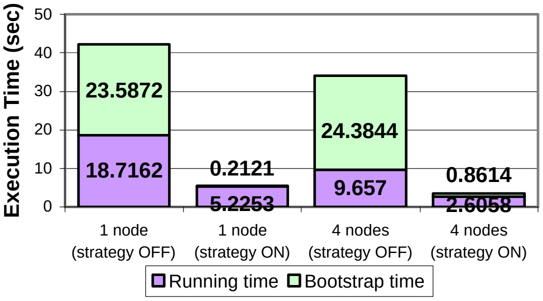


Figure 7. One log file per node: 1 and 4 nodes

The bootstrap time is even greater than the actual running time. This clearly indicates that the bootstrap operation is a bottleneck in the simulation. When the strategy is turned on, the bootstrap time is reduced by 99.1% on 1 node and by 96.47% on 4 nodes. Further, the running time is decreased by 72.08% on 1 node and by 73.02% on 4 nodes due to more efficient communication, I/O, and rollback operations.

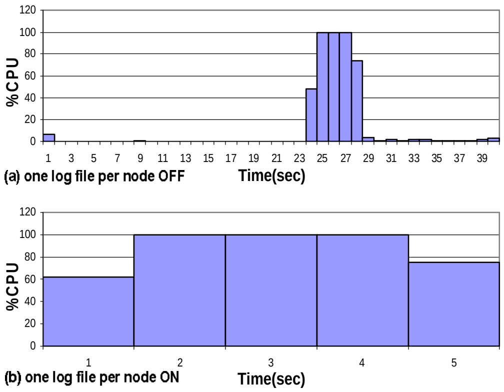


Figure 8. CPU use: 1 logfile per node, 1 node

The CPU usage monitored in our experiments also suggests that the file I/O operation is a major barrier in the bootstrap phase. As shown in Figure 8, the CPU is utilized much more efficiently with the one log file per node strategy. A similar pattern was observed in simulations running on multiple nodes.

5.2. MTSS

The same fire propagation model is used to test the effect of MTSS strategy. The model was executed on 1 and 4 nodes with and without the MTSS strategy.

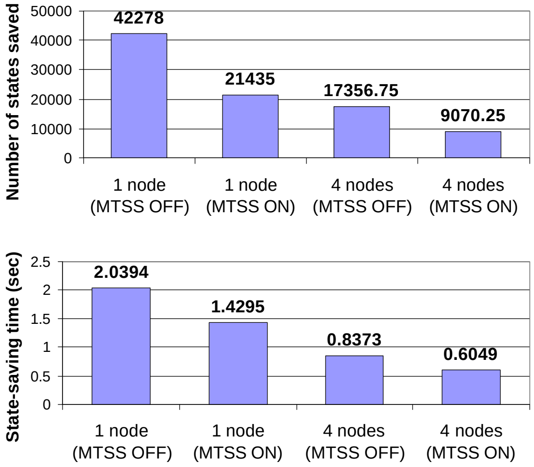


Figure 9. Number of saved states

Due to the MTSS strategy, the number of states saved during the simulation is reduced by 49.29% and 47.74% on 1 and 4 nodes respectively. Accordingly, the time spent on state-saving operations is decreased by 29.9% and 38.18%. The state-saving time declines more steeply on 4 nodes due to the distributed management of the state queues.

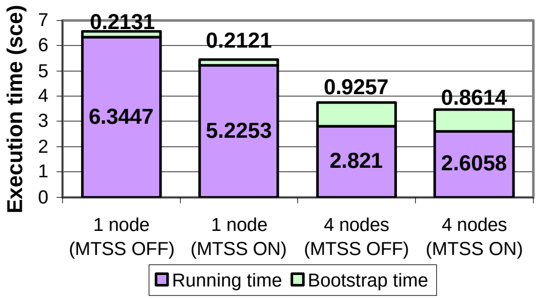


Figure 10. Running and bootstrap time

The corresponding running and bootstrap times are shown in Figure 10. While the bootstrap time remains nearly unchanged, the actual running time is reduced by 17.64% and 7.63% on 1 and 4 nodes respectively because fewer states are saved in the state queues and, potentially, removed from the queues during rollbacks.

Figure 11 shows the time-weighted average and maximum memory consumption with and without the strategy on 1 and 4 nodes. The average memory consumption declines by 26% in both cases, while the peak memory consumption decreases by 25.13% and 27.44% on 1 and 4 nodes respectively.

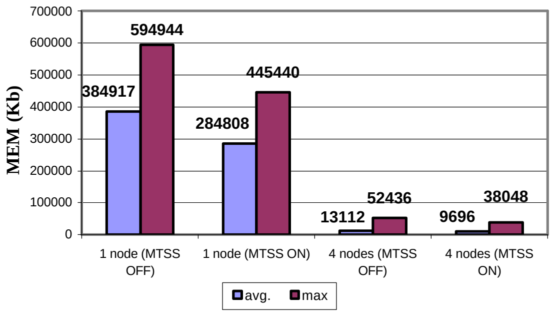


Figure 11. Memory consumption

5.3. Performance of the PCD++ toolkit

The key metrics for evaluating the performance of the PCD++ simulator are the execution time and speedup. Both the one log file per node and MTSS strategies were applied to the simulator in the following experiments. For all the Cell-DEVS models, a simple partition strategy was used that evenly divides the cell space into horizontal rectangles. First, the fire propagation model was tested using different sizes of cell spaces: 20×20 (400 cells), 25×25 (625 cells), 30×30 (900 cells) and 35×35 (1225 cells). The total execution time and running time of the fire model with different sizes and executed on 1 up to 4 nodes are listed in Table 1.

Table 1. Execution/running times for the fire model

Total Execution Time (sec)				
No.nodes	**20×20**	**25×25**	**30×30**	**35×35**
1	2.0733	3.2949	5.0442	7.8702
2	1.9719	2.7959	3.5232	4.7138
3	1.8787	2.5237	3.1573	3.9667
4	1.9254	2.6091	3.0922	3.8138
Running Time (sec)				
No.nodes	**20×20**	**25×25**	**30×30**	**35×35**
1	1.9515	3.1273	4.3566	7.6428
2	1.4232	2.1225	2.8838	3.9952
3	1.3574	1.8953	2.5237	3.2959
4	1.4296	1.8656	2.3314	3.0224

For any given number of nodes, the execution time always increases as the size of the model goes up. Moreover, the execution time rises less steeply when more nodes are used to do the simulation. For example, as the model size increases from 400 to 1225 cells, the execution time ascends sharply by nearly 280% (from 2.0733 to 7.8702 seconds) on 1 node, whereas it merely rises by 98% (from 1.9254 to 3.8138 seconds) on 4 nodes. On the other hand, for a fixed model size, the execution time tends to, but not always, decrease when more nodes are utilized. However, when the number of nodes increases further, the downward trend in execution time is reversed. When a model, especially a small one, is partitioned onto more and more nodes, the increasing overhead involved in inter-LP communication and potential rollbacks may eventually degrade the performance. Hence, a trade-off between the benefits of higher degree of parallelism and the concomitant overhead costs needs to be reached. We can also find that better performance can be achieved on a larger number of nodes as the model size increases. The shortest execution time is achieved on 3 nodes for the 20×20 and 25×25 models, while it is obtained on 4 nodes for the other two larger models.

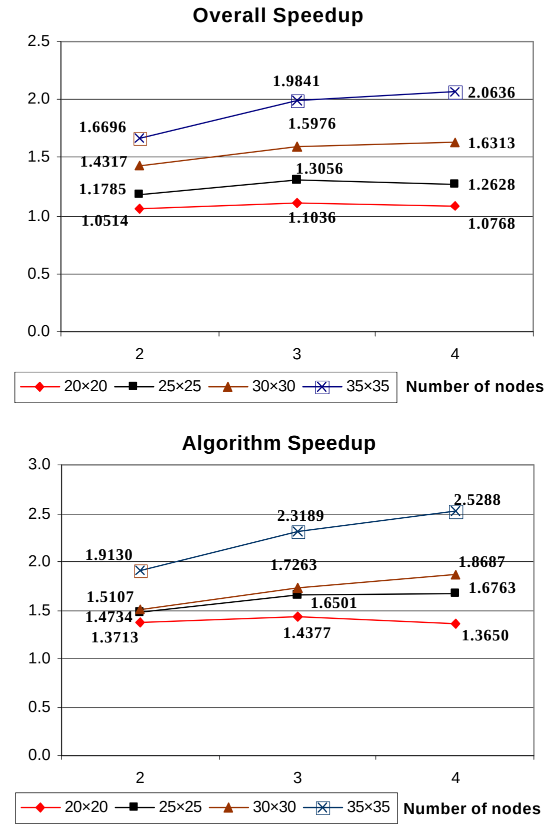


Figure 12. Overall/algorithm speedups

Using the execution and running times, we can calculate the overall and algorithm speedups, as shown in Figure 12. The algorithm speedup is always higher than its counterpart overall speedup, an evidence showing that the Time Warp optimistic algorithms are major contributors to the overall performance improvement.

A more computation-intensive 3-D watershed model of size 15×15×2 (450 cells) was tested to evaluate the performance of PCD++ for simulating models of complex physical system. Table 2 shows the resulting total execution and running times. The best performance is achieved on 5 nodes with execution and running time of 6.1538 and 5.6743 seconds respectively. The speedups are illustrated in Figure 14. The best overall and algorithm speedups are 2.7306 and 2.9373 respectively, higher than those obtained with the fire models.

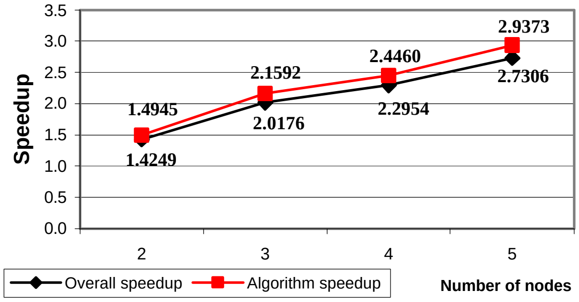


Figure 13. 15×15×2 watershed model

6. Conclusion

We tackled the problem of executing DEVS and Cell-DEVS models in parallel and distributed environments based on the Time Warp synchronization protocol. The algorithms for the PCD++ processors and Cell-DEVS models with transport and inertial delays were redesigned to address the need of distributed optimistic simulation. The simulation process on each LP was abstracted using the notion of WCTS, which greatly simplifies the task of analyzing the complex message exchanges between the PCD++ processors involved in the simulation. A two-level UCSS mechanism was proposed so that simulator developers can utilize more flexible and efficient state-saving techniques during the simulation. Mechanisms were provided to handle various issues in optimistic simulations such as rollbacks at virtual time 0 and messaging anomalies. Several optimization strategies were implemented in the optimistic PCD++ such as the MTSS strategy and the one log file per node strategy. We showed that optimistic PCD++ simulator markedly outperforms the conservative one in all testing scenarios. Considerable speedups were observed in our experiments, indicating the simulator is well-suited for simulating large and complex models.

7. References

[1] Zeigler, B.; Kim, T.; Praehofer, H. “Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems”. Academic Press. 2000.

[2] Chow, A. C.; Zeigler, B. “Parallel DEVS: A parallel, hierarchical, modular modeling formalism”. Proc. of Winter Computer Simulation Conference. Orlando. USA. 1994.

[3] Wainer, G.; Giambiasi, N. “N-dimensional Cell-DEVS models”. Discrete Event Dynamic Systems. Springer Netherlands. ISSN 0924-6703. Vol. 12. No. 2. 2002.

[4] Wolfram, S. “Theory and applications of cellular automata”. Vol. 1. Advances Series on Complex Systems. World Scientific. Singapore. 1986.

[5] Fujimoto, R. M. “Parallel and Distributed Simulation Systems”. Wiley-Interscience publication. 2000.

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

[7] Radhakrishnan, R.; Martin, D. E.; Chetlur, M.; Rao, D. M.; Wilsey, P.A. “An Object-Oriented Time Warp Simulation Kernel”. Proceedings of the International Symposium on Computing in Object-Oriented Parallel Environments (ISCOPE’98). Vol. LNCS 1505. 1998.

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

[9] Troccoli, A.; Wainer, G. “Implementing Parallel Cell-DEVS”. Proceedings of the 36th Annual Simulation Symposium (ANSS’03). IEEE. 2003.

[10] Glinsky, E; Wainer, G. “New parallel simulation techniques of DEVS and Cell-DEVS in CD++”. Proceedings of the 39th Annual Simulation Symposium. 2006.

[11] Wainer, G. “Improved cellular models with Parallel Cell-DEVS”. Transactions of the Society for Computer Simulation International. Vol. 17, No. 2, pp. 73-88. 2000.

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

[13] Zeigler, B.; Sarjoughian H. S. “Support for hierarchical modular component-based model construction in DEVS/HLA”. Simulation Interoperability Workshop. 1999.

[14] 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. Assisi, Italy. 2004.

[15] Seo C.; Park, S.; Kim, B.; Cheon, S.; Zeigler, B. “Implementation of distributed high-performance DEVS simulation framework in the Grid computing environment”. Advanced Simulation Technologies Conference (ASTC). Arlington, VA. USA. 2004.

[16] 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.

[17] Zhang, M.; Zeigler, B.; Hammonds, P. “DEVS/RMI – An auto-adaptive and reconfigurable distributed simulation environment for engineering studies”. DEVS Integrative M&S Symposium. Huntsville, Alabama, USA. 2006.

[18] Nutaro, J. “Risk-free optimistic simulation of DEVS models”. Military Government and Aerospace Simulation Symposium. 2004.

[19] Ameghino, J.; Troccoli, A.; Wainer, G. “Models of complex physical systems using Cell-DEVS”. The 34th IEEE/SCS Annual Simulation Symposium. 2001.

[20] 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.