Parallel Simulation of DEVS and Cell-DEVS Models on Windows-based PC Cluster Systems

Bo Feng, Qi Liu and Gabriel Wainer
Department of Systems and Computer Engineering
Carleton University Centre on Visualization and Simulation (V-Sim)
Carleton University, Ottawa, ON, Canada K1S 5B6
gwainer@sce.carleton.ca

Keywords: Discrete event simulation, DEVS, Cell-DEVS, parallel simulation, cluster systems

Abstract

The growing popularity of Networks of Workstations (NOW) in scientific computation has drawn increasing interest from the M&S community. This paper addresses the issue of parallel discrete-event simulation of DEVS and Cell-DEVS models on a Microsoft Windows-based cluster system comprising interconnected general-purpose personal computers. We present the architecture and features of PCD++Win, a parallel simulator that takes advantage of the multi-purpose graphical user interface of the DeinoMPI middleware for construction of ad-hoc PC clusters and configuration of simulation environment. This environment significantly reduces the learning curve for general users and the cost of the simulation platform. PCD++Win has been developed using a modular approach that promotes code reuse and allows for easy switching to other middleware technologies. The portability of the simulator is enhanced with multi-platform programming and compilation techniques. Moreover, it leaves open the possibility of further extensions such as Web-based distributed simulation and database-based model construction by leveraging the native support of Microsoft Visual Studio. The experiments demonstrate the capability of the new simulator, making it an ideal M&S toolkit for tapping the computational power of general-purpose desktop computers.

1. INTRODUCTION

Modeling and simulation (M&S) has become a widely used tool for tackling complex problems and supporting efficient decision-making in a broad array of domains. With the computing power and advanced software tools available today, M&S allows for cost-effective and detailed analysis of natural and artificial systems. Parallel and distributed simulation (PADS) is accepted as the technology of choice to speed up large-scale discrete-event simulation and to promote reusability and interoperability of simulation components. Traditionally, PADS is usually realized on advanced parallel computing platforms using UNIX/Linux servers or Beowulf configurations. The increasing computational power of PCs and the growing popularity of using Networks of Workstations (NOW) in scientific computation have drawn significant interest from the M&S community. Advanced PADS toolkits are being developed to unleash the relatively untapped potential of idle capacity of general-purpose desktop computers.

The Message Passing Interface (MPI) standard [1] is a technique for high-performance communications on both massively parallel machines and on workstation clusters. It has been used as the enabling middleware in various parallel and distributed simulators. However, many MPI packages only offer a list of commands with numerous options and parameters, making the configuration task intimidating for most non-experienced users. Recently, several MPI implementations with a graphical user interface (GUI) are made available on Microsoft Windows platforms, allowing for the construction of MPI-based PADS systems on commodity off-the-shelf (COTS) Windows-based PCs, promoting the adoption of cutting-edge M&S technologies by a wider community of practitioners. Among them, the DeinoMPI [2] is a freely available implementation of the MPI-2 standard for 32 and 64-bit Microsoft 2000/XP/Server 2003 platforms. PCD++Win employs DeinoMPI as the communication infrastructure due to its support for the latest version of the MPI standard and Windows operating systems, easy-to-use multi-purpose GUI tool for end users, built-in debugging and diagnostic capabilities, and small installation footprint.

The Discrete Event System Specification (DEVS) [3] is a sound formal M&S methodology for discrete-event systems. By decoupling the model and simulation concepts, the DEVS framework offers two major benefits. First, the same model can be executed on different simulators, allowing for portability and interoperability at a high level of abstraction. Moreover, the well-defined separation of concerns permits models and simulators to be independently verified and reused in later combinations with minimal re-verification. Furthermore, DEVS supports hierarchical and modular construction of models, reducing the development and testing effort. The P-DEVS formalism [4] extends the DEVS framework by eliminating the sequential execution constraints that existed in the original DEVS definition, allowing increased parallelism to be exploited in PADS systems. The Cell-DEVS formalism [5] combines Cellular Automata (CA) [6] with DEVS theory to describe n-

dimensional cell-spaces as discrete-event models, where each cell is defined as a DEVS basic model with explicit timing constructions. The synchronization mechanism used in PADS generally falls into two categories [7]: conservative protocols that strictly avoid violating the local causality constraint, and optimistic protocols that dynamically detect causality errors and provide mechanisms to recover from them at runtime. Both approaches have their strengths and weaknesses, and we followed the conservative approach in this work.

CD++ [8] is an open-source M&S environment that implements the P-DEVS and Cell-DEVS formalisms and has been used to successfully solve a variety of sophisticated problems. Over the years, CD++ has been ported to different platforms, including a standalone version called CD++Builder that runs as an Eclipse plug-in on Windows systems to support sequential simulations [9], several parallel versions, referred to as PCD++, that employ both conservative and optimistic synchronization protocols to achieve high-performance simulations on Linux cluster systems [10][11], and a distributed version called DCD++ that supports Web-based simulations over the Internet on Linux platforms [12]. The CD++ environment is further extended in this research to support parallel conservative simulation of DEVS and Cell-DEVS models on Microsoft Windows-based PC clusters using DeinoMPI. The objective is twofold. First, the resulting simulator, called *PCD++Win*, is based on COTS PC hardware running the most commonly used Windows operating system, allowing high-performance parallel simulation in a cost effective way. Secondly, by integrating PCD++Win with DeinoMPI, general users can benefit from the easy-to-use multi-purpose GUI tool in construction of flexible ad-hoc PC clusters. Furthermore, the performance of PCD++Win is compared with the conservative PCD++ running on Linux cluster systems, showing the relative merits of both approaches.

The rest of the paper is organized as follows. Section 2 introduces the P-DEVS and Cell-DEVS formalisms and gives a brief survey on alternative DEVS-based toolkits for PADS. Section 3 discusses the techniques for parallel conservative simulation in PCD++Win. Section 4 covers the major features of the multi-purpose GUI tool provided by DeinoMPI. Section 5 presents a performance analysis. And Section 6 closes the paper with conclusion remarks.

2. BACKGROUND

2.1. P-DEVS and Cell-DEVS formalisms

In a discrete-event simulation, the system being simulated changes state only at discrete points in time, upon the occurrence of an event. Based on dynamic systems theory, the DEVS formalism [3] provides a framework for defining hierarchical models in a modular way. A system is described in DEVS as a composition of behavioral (atomic) and structural (coupled) components. The P-DEVS formalism [4] eliminates the sequential execution constraints imposed by the original DEVS definition, and provides a solid theoretical foundation for high-performance parallel and distributed discrete-event simulation. A P-DEVS atomic model is formally defined as:

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

At any given time, an atomic model is in some state $s \in S$. Without the occurrence of external events, it remains in state s for a period of time of ta(s), which is referred to as the lifetime of state s. When the lifetime expires, the atomic model outputs value $\lambda(s) \in Y$, and changes to a new state given by the internal transition function $\delta_{int}(s)$. A P-DEVS model employs a bag of inputs (X^b) to support the execution of multiple concurrent events. If one or more external events $x \in X$ occur before the expiration of ta(s), the model transfers to a state that is determined by the external transition function $\delta_{ext}(s,e,X^b)$, combining the functionality of multiple external transitions into a single one. A confluent transition function (δ_{con}) is defined to determine the next state in the case of collisions when a component receives external events at the same time of its internal transition.

The P-DEVS formalism has a well-defined concept of system modularity and component coupling to form composite models. A P-DEVS coupled model is formally defined as:

$$N = <X, Y, D, \{M_d \mid d \in D\}, EIC, EOC, IC>.$$

The sets of input and output events are defined by X and Y respectively. D is a set of indices for the components of a coupled model and, for each $d \in D$, M_d is a basic P-DEVS model (atomic or coupled). The external input coupling (EIC) specifies the connections between external and component inputs, while the external output coupling (EOC) describes the connections between component and external outputs. The connections between the components themselves are defined by the internal coupling (IC). It has been shown that the P-DEVS formalism enjoys a nice property known as closure under coupling [3], which allows a coupled model to be reduced to a behaviorally equivalent atomic model, and thus be treated as a basic component in construction of more complicated hierarchical models.

CA [6] is a theory that is capable of producing a great variety of complex behavior of systems represented as cell spaces. Traditionally, CA models are implemented on a computer using a discrete time approach. The behavior of a cell space depends on synchronous evaluation of local functions defined in the cells at discrete time intervals. To improve execution efficiency and precision of the simulated models, the Cell-DEVS formalism [5] was proposed to define n-dimensional cell spaces as discrete-event DEVS coupled models, where each cell is represented as a DEVS atomic model. Further, it defines timing constructions for each cell, allowing explicit timing specification,

asynchronous model execution, and seamless integration with other types of models. A Cell-DEVS atomic model is formally defined as:

$$C = <X, Y, I, S, \theta, N, delay, d, \delta_{int}, \delta_{ext}, \tau, \lambda, D>.$$

A cell has a modular interface (I) that is composed of a fixed number of ports; each is connected to a neighboring cell. It can input and output data (X and Y) with its neighbors as well as other models outside of the cell space. The future state of a cell is computed by the local transition function (τ) based on the cell's current state and input values. State changes in a cell are transmitted only after a delay given by the delay function (d). Each cell also has the computing apparatus (δ_{int}, δ_{ext}, and λ) as defined in DEVS atomic models. Cells are coupled by the neighborhood relationship to form a cell space, which can then be integrated with other DEVS and Cell-DEVS models. A cell space is formally defined as a Cell-DEVS coupled model:

$$GCC = <X_{list}, Y_{list}, I, X, Y, \eta, \{t_1, \ldots, t_n\}, N, C, B, Z>.$$

The cell space (C) consists of a fixed-sized n-dimensional array of cells, and the relative position between each individual cell and its surrounding neighbors is defined by the neighborhood set (N). B specifies the border of the cell space, which can be wrapped (i.e., all cells have exactly the same behavior) or non-warped (i.e., the border cells have a different behavior from others in the cell space). The translation function (Z) defines the input/output coupling between the cells.

2.2. DEVS-based toolkits for PADS

At present, there are many DEVS-based toolkits intended for PADS that have been developed on different platforms using various middleware technologies. However, few of them target COTS PC cluster systems that are readily available in most workplaces. A non-comprehensive list of existing toolkits is given as follows.

- DEVS/CORBA [13] is a runtime infrastructure based on CORBA middleware that supports distributed simulation of DEVS models. It can be embedded in a larger network-centric environment to provide a combination of graphical process modeling, discrete-event simulation, animation, activity-based costing, and optimization functions.
- DEVS/HLA [14] is an HLA-compliant M&S environment implemented in C++ that supports high-level model construction. It simplifies the programming effort required to establish and participate in an HLA federation.
- DEVSCluster [15] is a CORBA-based, multi-threaded distributed simulator implemented in Visual C++. It utilizes a non-hierarchical model structure to facilitate the synchronization of distributed simulations.
- DEVS/Grid [16] is an M&S framework implemented using Java and Globus for the Grid environment. It includes a set of automated simulation facilities, including cost-based hierarchical model partitioning, dynamic coupling restructuring, automatic model deployment, and naming and directory service.
- DEVS/P2P [17] is a P-DEVS based M&S framework implemented on top of Peer-to-Peer communication infrastructure. It uses a customized DEVS simulation protocol to achieve decentralized inter-node communication. Simulators are synchronized by themselves without involving a coordinator.
- DEVS/RMI [18] is a DEVS-based system that provides a dynamic and re-configurable runtime infrastructure for handling load balancing and fault tolerance in distributed simulations. It reduces the overhead associated with common middleware solutions by using the native support of Java RMI to synchronize local and remote simulators.

3. PARALLEL SIMULATION IN PCD++WIN

PCD++Win employs the layered software architecture that was originally proposed in [10], as shown in Figure 1.

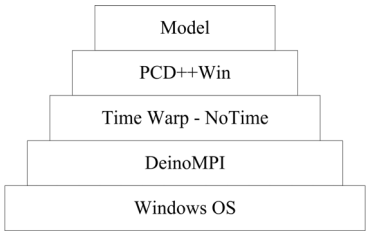

Figure 1. Layered software architecture in PCD++Win

The simulator has been built on top of the *NoTime* module of the WARPED simulation kernel [19], which is a configurable middleware that provides the abstract definition of events, states, and simulation objects, as well as MPI-based communication between the simulation objects. The NoTime module does not provide any synchronization mechanism. Instead, the parallel simulation is controlled at the PCD++Win level.

CD++ decouples the modeling and simulation concepts by providing two separate frameworks: a modeling framework that allows users to define the behavior of atomic and coupled models using a built-in specification language or C++; and a simulation framework that creates an executive entity for each component in the model hierarchy to carry out the simulation in line with the P-DEVS and Cell-DEVS formalisms. Figure 2 shows the class hierarchies of the modeling and simulation frameworks.

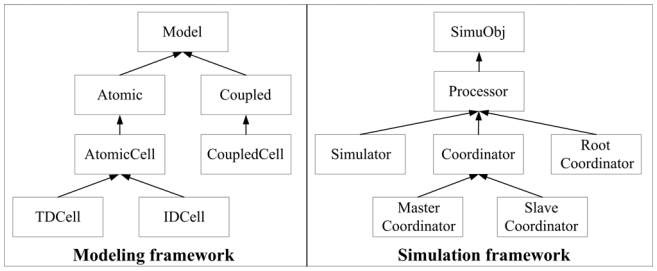

Figure 2. Modeling and simulation frameworks in CD++

Based on the abstract DEVS simulator concept [4], the simulation objects are specialized into two categories: *simulators* and *coordinators*. A simulator is associated with an atomic model to trigger the output and state transition functions, while a coordinator is created for each coupled model to keep track of the simulation time and to relay messages between its child simulators and the parent coordinator. A special root coordinator is created on node 0 (i.e., the first node in the PC cluster) to control the advance of simulation time and to communicate between the simulated model and the surrounding environment.

To reduce the communication overhead, PCD++Win employs a master/slave structure of coordinators [10]. As a result, when a coupled model is partitioned onto multiple nodes, a coordinator is created on each of them to execute the portion mapped on that specific node. The coordinator on the first node involved in the partition is the master, while all the other coordinators are slaves. The master coordinator is deemed as the immediate parent of the slaves residing on the other nodes. Figure 3 illustrates the master/slave structure on two nodes. Suppose the coupled model `C1` is partitioned onto two nodes and each portion has two atomic models. Two coordinators are created for `C1`: a master (`MasterC1`) on node0 and a slave (`SlaveC1`) on node1. The major advantage of this arrangement is that it reduces the number of inter-node MPI messages to the minimum. For example, if `A1` sends a message to `A3` and `A4` at the same time, then only one MPI message is actually transmitted between the master and slave coordinators instead of two. Due to the high cost of inter-node messaging, this structure can significantly reduce the communication overhead and improve the simulation performance.

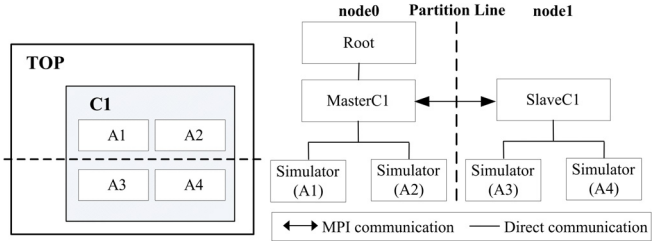

Figure 3. Master/slave coordinator structure

The simulation is carried out in a message-driven fashion. Each message has a timestamp that indicates the virtual time of the event. CD++ messages fall into two categories: *content messages* include the external message (X, t) and output message (Y, t) that encode the actual data transmitted between the models, while *control messages* include the initialization message (I, t), collect message (@, t), internal message (*, t), and done message (D, t) that are used to synchronize the simulation. The message-processing algorithms are illustrated in the following UML sequence diagrams.

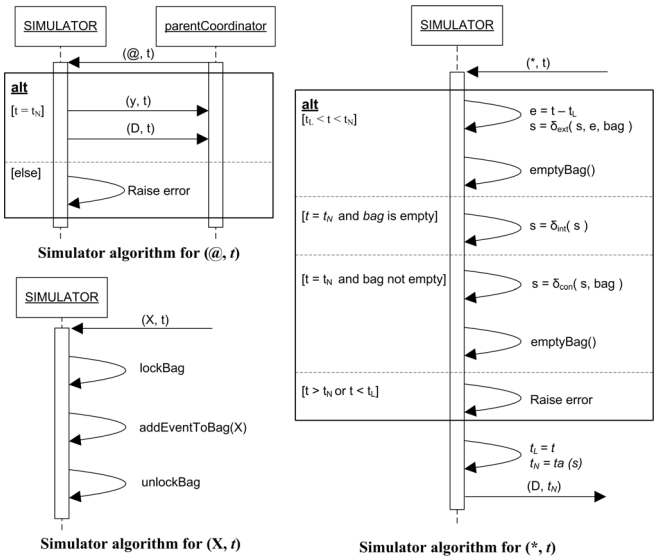

Figure 4. Message-processing algorithms (simulator)

Figure 4 shows the algorithms for simulators. When a simulator receives a (@, t) message from its parent coordinator, it executes the output function defined in the associated atomic model and sends a (Y, t) and a (D, t) to its parent. If a (X, t) is received, the message is cached in the simulator's message bag. On the other hand, the arrival of a (*, t) triggers state transitions in the atomic model based on the simulation time and current status of the message bag.

The message-processing algorithms for master coordinators are illustrated in Figure 5. A master coordinator may have three different types of child processors, including the slave coordinators on remote nodes, the local child simulators, and other lower-level master coordinators on the same node. When a (@, *t*) arrives, the master coordinator forwards the message to all imminent child processors and caches the receivers for later state transitions. It then waits for a (D, t) from each of these receivers. Afterwards, it sends a (D, t) with the updated simulation time to its parent coordinator.

Three different cases may occur upon the arrival of a (Y, t): if the message is sending to a local receiver, the master coordinator translates it into a (X, t) and forwards it to the destination; if the message targets remote receivers, the master coordinator figures out the corresponding slave coordinators, and relays the message to each of them; otherwise, the message is forwarded to the higher-level parent coordinator. The processing of a (X, t) is the same as in the simulators. The master coordinator flushes all external messages in its message bag to their destinations upon the arrival of a (*, t). It also sends a (*, t) to each child that has a scheduled internal or external state transition. After the state transitions, the master coordinator calculates the next simulation time and sends the information to its parent coordinator in a (D, t).

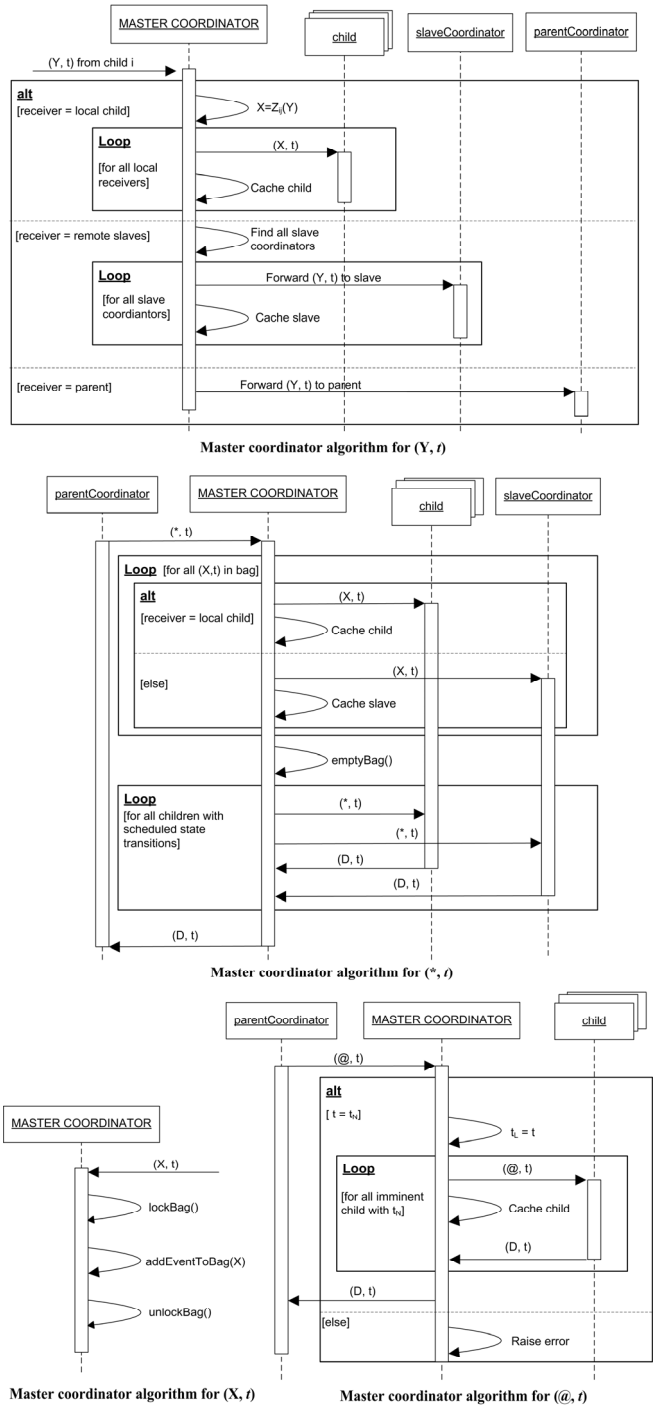

Figure 5. Message-processing algorithms (master)

The slave coordinator handles (@, t), (X, t), and (*, t) messages in the same manner as the master coordinator. However, they differ in one aspect: whenever a (Y, t) has to be sent to a remote receiver, the slave coordinator will forward it to its parent master coordinator. In this master/slave structure, a slave coordinator can have only two types of child processors, namely the local child simulators and lower-level master coordinators (i.e., it will not have other slave coordinators as descendants). Figure 6 shows the slave coordinator algorithm for (Y, t).

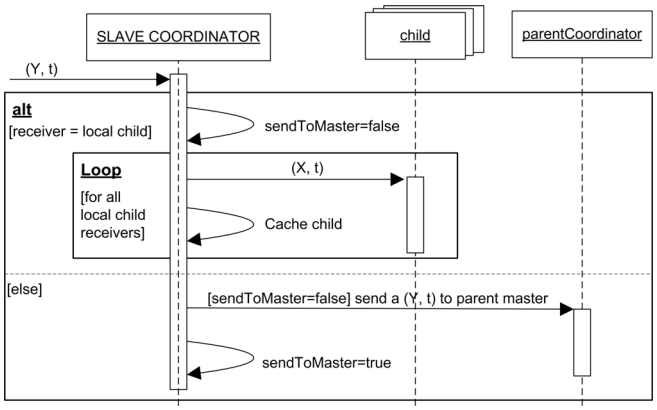

Figure 6. Message-processing algorithms (slave)

When a (Y, t) arrives, the slave coordinator transforms the message into a (X, t) and sends the resulting (X, t) to the local child receivers. If the (Y, t) targets remote receivers on other nodes, the slave coordinator simply forwards the message to its parent master coordinator, which in turn will send the message to other slave coordinators if necessary. Notice that only one (Y, t) is forwarded to the master coordinator, as guaranteed by the `sendToMaster` flag.

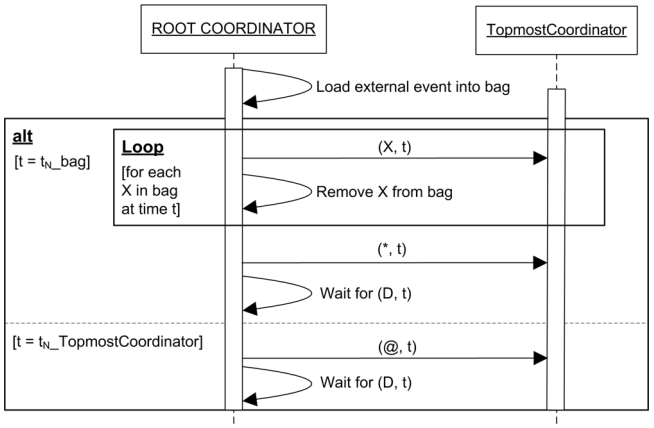

Figure 7. Root algorithm for simulation control

The root coordinator is a special processor that controls the whole simulation and handles events exchanged between the simulated model and the environment. Figure 7 shows the root coordinator algorithm for controlling the simulation, where it sends (*, t) and (@, t) alternatively with potential external events as (X, t) messages to the top-level master coordinator to drive the simulation forward.

In this project, several techniques were used to enhance the modularity, portability and capability of the PCD++Win simulator on Windows platforms. Following is a summary of the major portion of our research effort.

1. Modular software development using dynamic-link libraries (DLL): a DLL contains the implementation of a

shared library that allows for modular software development and promotes code reuse. We compiled the NoTime Warped kernel as a separate DLL module, which is dynamically linked to the simulator at runtime. This approach reduces the memory footprint of PCD++Win and makes it possible to easily switch to other middleware technologies in future developments.

2. Multi-platform compilation with preprocessor macros. Although PCD++Win is intended for Windows-based PC clusters, the code was written to be executed on Linux OS as well. This is achieved by defining preprocessor macros in the source code and using conditional compilation to generate appropriate versions for different platforms, increasing the portability of the toolkit.

3. Porting code from GCC (GNU Compiler Collection) to Microsoft Visual Studio: PCD++Win was developed based on the conservative simulator PCD++ [10], which uses GCC on Linux systems. We ported the source code to Microsoft Visual Studio that has native support for Web service and SQL database access based on the .NET Framework. As a result, it is now convenient to further extend PCD++Win into Web services to carry out Web-based distributed simulations, and to access a shared database of predefined model components that would greatly facilitate the model development process.

4. PARALLEL SIMULATION WITH DEINOMPI

By using the DeinoMPI GUI tool [2], users can easily configure an ad-hoc PC cluster to carry out parallel simulations with PCD++Win. This section explains how to set up the cluster environment and monitor the simulation progress. Figure 8 shows the GUI for cluster configuration.

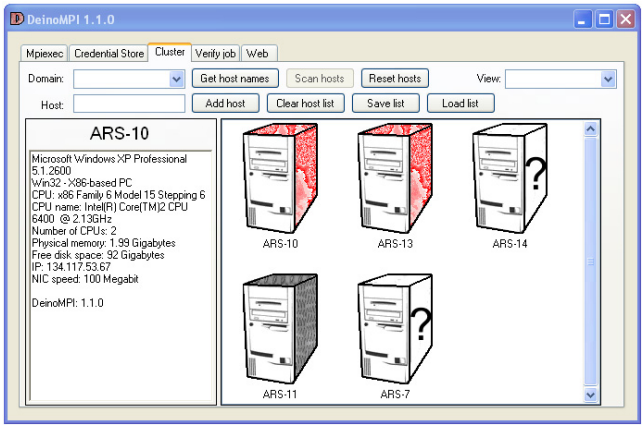

Figure 8. Cluster configuration in DeinoMPI

Computers on a network can be added to the panel either by scanning the whole network or by specifying their host names. The tool can automatically check the machines and present their status to the user. For instance, Figure 8 shows that ARS-10 and ARS-13 are available nodes that have all necessary software to carry out parallel simulations, while ARS-11 lacks the DeinoMPI package and hence cannot be involved in the cluster. A question mark means the condition of the node is not yet fully known. Detailed information about each node is also probed and presented, including CPU speed, memory size, disk space, network connectivity and so on. With this information, the user can then select appropriate nodes to form an ad-hoc cluster.

The main execution window is illustrated in Figure 9. Users can specify the simulation parameters using the GUI tool and dynamically change the nodes involved in the cluster. The simulation-related information is shown in the window underneath.

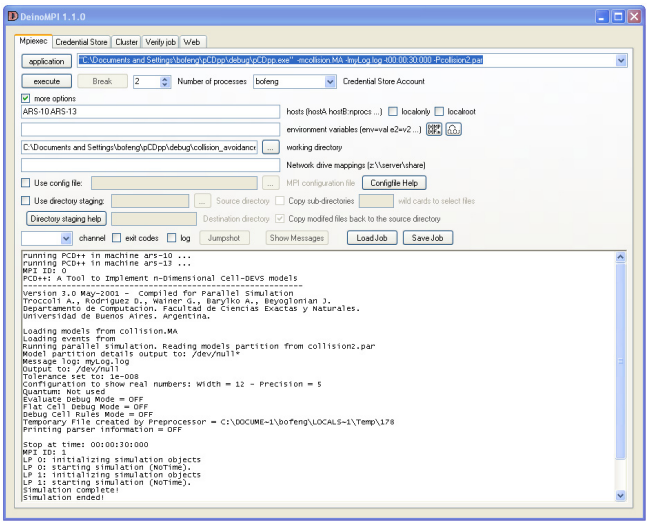

Figure 9. Main execution window in DeinoMPI

If errors happen in the simulation, users can diagnose the error condition by using the job verification tab. As shown in Figure 10, the tool gives a detailed description of each job and the possible causes of the failure.

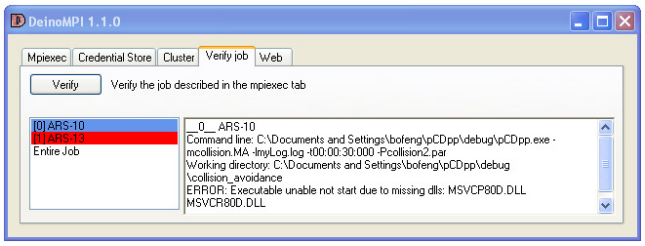

Figure 10. Job verification tool in DeinoMPI

As we can see, PCD++Win provides a user-friendly environment for conducting parallel simulations by leveraging the easy-to-use GUI tool of DeinoMPI. Any Windows-based PCs interconnected via a network can be used to form a cluster platform for parallel simulations that are otherwise not possible on a single machine, making advanced simulation technologies available not only to users in traditional office environment equipped with wired desktop PCs, but also to practitioners on the move working on laptops connected by ad-hoc wireless networks.

5. EXPERIMENTATION RESULTS

This section presents a preliminary performance analysis that compares PCD++Win with the conservative PCD++ [10]. As these two simulators actually target different platforms and computing environment, we should stress that the data presented here are obtained on their intended typical platforms, and only serves as a general indicator of the simulation performance. Before moving to the experimental results, it is important to highlight the key differences existed in the computing environment to eliminate any potential bias in judging the performance data. The differences are described as follows.

1. PCD++Win executes on a group of desktop workstations (Intel Core 2 Duo Processor E6400 @ 2.13 GHz, 2 GB DDR2-Synch DRAM) running Microsoft Windows XP Professional connected through a *shared* local area network (LAN) and communicating with DeinoMPI 1.1.0, while PCD++ executes on a HP PROLIANT DL Server, a cluster of 32 compute nodes (Intel Xeon Dual Processor @ 3.2 GHz, 1 GB PC2100 266 MHz DDR RAM) running Linux WS 2.4.21 interconnected via a *dedicated* Gigabit Ethernet and communicating over MPICH 1.2.6. Although the differences in CPU speed, memory size, network type (dedicated or shared), MPI middleware and operating system might have a non-negligible effect on the performance data, the setting is not materially biased in favor of any of these two simulators.

2. In the case of PCD++Win, the log data generated during the simulation are stored to the local file system on each workstation. On the other hand, the data are transferred over the network to a NFS (Network File System) when PCD++ is executed on the Linux cluster. Since the file I/O operations are much faster in the former case, this would give a boost to the performance of PCD++Win.

Three Cell-DEVS models were tested in our experiment, including a 20×20 fire propagation model and a 15×15×2 watershed model that are described in [20], as well as a 20×20 collision avoidance model that simulates the behavior of moving robots trying to steer clear of obstacles in their way [21]. The execution time and speedup shown in Figure 11 is averaged over ten simulation runs for each model.

As we can see, PCD++Win achieves comparable performance to the conservative PCD++ in terms of execution time and speedups. In some cases, PCD++Win attains an improved execution time (by as much as 88.06%) for the models evaluated in our experiments, mainly due to the much more efficient local file I/O operations. Comparing the speedups obtained by the simulators, the data suggests that PCD++Win generally surpasses its counterpart on a relatively small number of computers. When the simulation involves greater number of nodes, the communication overhead becomes the primary bottleneck that eventually hinders the scalability of PCD++Win running on a shared network.

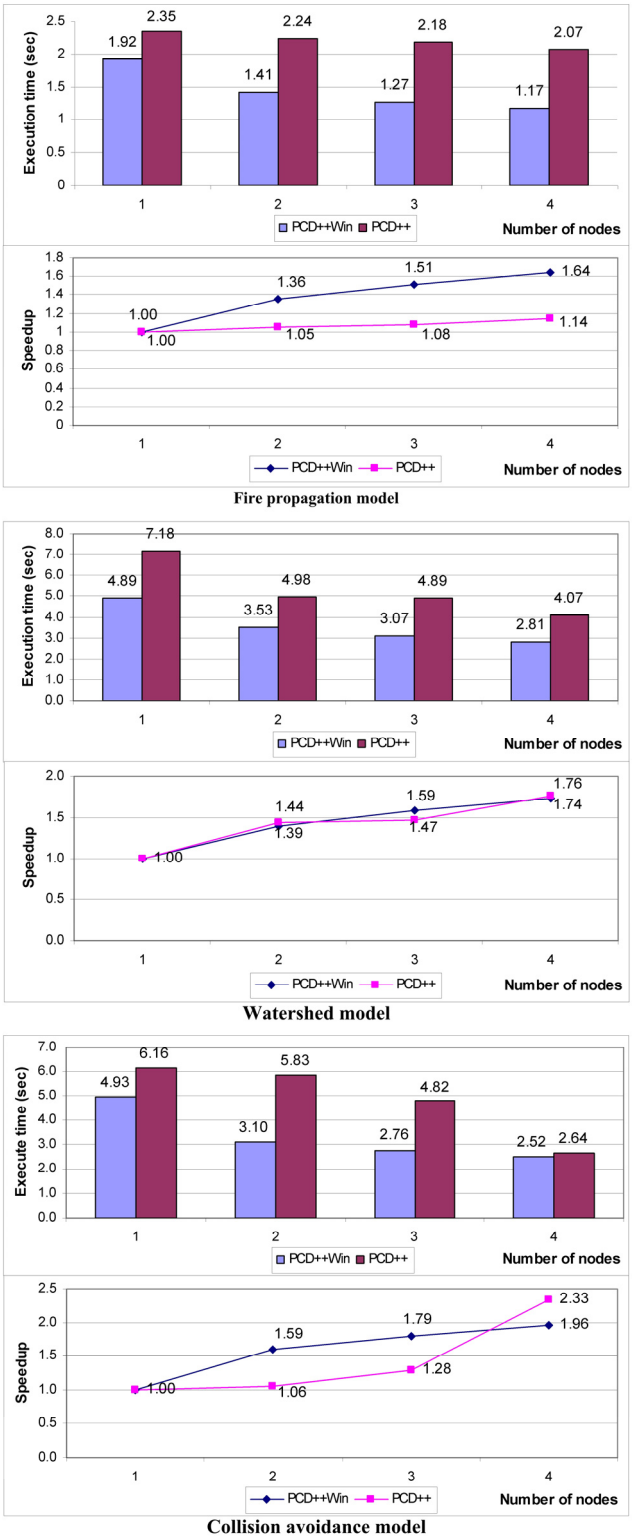

Figure 11. Execution results of different Cell-DEVS models

6. CONCLUSION

This paper addresses the issue of parallel simulation of DEVS and Cell-DEVS models on Microsoft Windows-based cluster systems. A parallel simulator, PCD++Win, has been developed as a new simulation engine for the CD++ environment. PCD++Win is integrated with the open-source DeinoMPI middleware that provides an easy-to-use GUI for construction of ad-hoc clusters from commodity PCs and configuration of the simulation environment, significantly reducing the learning curve for general users and the simulation cost. It has been developed using a highly modular approach that promotes code reuse and allows for easy switching to other middleware technologies in future development. The portability of the simulator is enhanced with multi-platform programming and compilation techniques. Moreover, it leaves open the possibility of further extensions such as Web-based distributed simulation and database-based model construction by leveraging the native support of Microsoft Visual Studio. The experiments demonstrate the capability of the new simulator, making it an alternative M&S toolkit for tapping the computational power of general-purpose desktop computers.

References

[1] MPI Forum. 2003. *MPI: A Message-Passing Interface Standard*. http://www.mpi-forum.org/docs/docs.html.

[2] Deino Software. 2006. *DeinoMPI – User Manual, Version 1.1.0*. http://mpi.deino.net/manual.htm.

[3] Zeigler, B.P.; H. Praehofer, and T. G. Kim. 2000. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems.* Academic Press, San Diego, CA.

[4] Chow, A.C., and B.P. Zeigler. 1994. "Parallel DEVS: A Parallel, Hierarchical, Modular Modeling Formalism". In Proceedings of the 26th Winter Simulation Conference, Orlando, FL, 716-722.

[5] Wainer, G., and N. Giambiasi. 2002. "N-dimensional Cell-DEVS Models". Discrete Event Dynamic Systems, 12, no. 2, (April): 135-157.

[6] Wolfram, S. 2002. *A New Kind of Science.* Wolfram Media Inc., Champaign, IL.

[7] Fujimoto, R.M. 2000. *Parallel and distributed simulation systems*. John Wiley & Sons, New York, NY.

[8] Wainer, G. 2002. "CD++: A Toolkit to Develop DEVS Models". Software: Practice and Experience, 32, no. 13, (September): 1261-1306.

[9] Chidisiuc, C. and G. Wainer. 2007. "CD++Builder: An Eclipse-based IDE for DEVS Modeling". In Proceedings of the 2007 DEVS Integrative M&S Symposium (DEVS'07), Norfolk, VA.

[10] Troccoli, A. and G. Wainer. 2003. "Implementing parallel Cell-DEVS". In Proceedings of the 36th IEEE Annual Simulation Symposium (ANSS'03), Orlando, FL, 273-280.

[11] Liu, Q. and G. Wainer. 2007. "Performance Analysis of an Optimistic Simulator for CD++". In Proceedings of the 40th IEEE Annual Simulation Symposium (ANSS'07), Norfolk, VA, 123-132.

[12] Madhoun, R. and G. Wainer. 2007. "Studying the Impact of Web-Services Implementation of Distributed Simulation of DEVS and Cell-DEVS Models". In Proceedings of the 2007 DEVS Integrative M&S Symposium (DEVS'07), Norfolk, VA.

[13] Zeigler, B.P., D. Kim, and S. Buckley. 1999. "Distributed Supply Chain Simulation in a DEVS/CORBA Execution Environment". In Proceedings of the 31st Winter Simulation Conference, Phoenix, AZ, 1333-1340.

[14] Zeigler, B.P., and H.S. Sarjoughian. 1999. "Support for Hierarchical Modular Component-based Model Construction in DEVS/HLA". In Proceedings of the 1999 Spring Simulation Interoperability Workshop, Orlando, FL.

[15] Kim, K. and W. Kang. 2004. "CORBA-based, Multi-threaded Distributed Simulation of Hierarchical DEVS Models: Transforming Model Structure into a Non-hierarchical One". In Proceedings of the International Conference on Computational Science and Its Applications (ICCSA 2004), Assisi, Italy, LNCS 3046: 167-176.

[16] Seo, C., S. Park, B. Kim, S. Cheon, and B.P. Zeigler. 2004. "Implementation of Distributed High-performance DEVS Simulation Framework in the Grid Computing Environment". In Proceedings of the 2004 Advanced Simulation Technologies Conference – High-Performance Computing Symposium (ASTC'04), Arlington, VA.

[17] Cheon, S., C. Seo, S. Park, and B.P. Zeigler. 2004. "Design and Implementation of Distributed DEVS Simulation in a Peer to Peer Network System". In Proceedings of the 2004 Advanced Simulation Technologies Conference – Design, Analysis, and Simulation of Distributed Systems (ASTC'04), Arlington, VA.

[18] Zhang, M., B.P. Zeigler, and P. Hammonds. 2006. "DEVS/RMI – An Auto-adaptive and Reconfigurable Distributed Simulation Environment for Engineering Studies". In Proceedings of the 2006 DEVS Integrative M&S Symposium (DEVS'06), Huntsville, AL.

[19] 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 2nd International Symposium: Computing in Object-oriented Parallel Environments (ISCOPE 98), Santa Fe, NM, LNCS 1505: 13-23.

[20] Wainer, G. 2006. "Applying Cell-DEVS Methodology for Modeling the Environment". SIMULATION: Transactions of the Society for Modeling and Simulation International, 82, no. 10, 635-660.

[21] Wainer, G. 2008. *Discrete-Event Modeling and Simulation: A Practitioner's Approach*. Taylor and Francis. To Appear.