A .NET Remoting-Based Distributed Simulation Approach for DEVS and Cell-DEVS Models

Bo Feng Gabriel Wainer
Department of Systems and Computer Engineering
Carleton University Centre on Visualization and Simulation (V-Sim)
1125 Colonel By Drive, Ottawa, ON K1S 5B6, Canada
gwainer@sce.carleton.ca

Abstract

We present a DEVS and Cell-DEVS simulation system based on Microsoft's .NET Remoting, an execution framework for Windows programs. Remoting is the process of programs interacting across different processes or machines. The numerous Remoting services in .NET provide the ability to invoke objects that exist anywhere on the network. We modified PCD++ (a parallel DEVS and Cell-DEVS simulation tool based on the Message Passing Interface - MPI) to make use of these services. The new tool, called PCD++/.NET integrates .NET Remoting into PCD++ to execute distributed DEVS and Cell-DEVS simulation. PCD++/.NET can be suitable for the model which has modest remote messages and presents tolerable performance.

1. Introduction

Modeling and simulation (M&S) has been used on a wide variety of fields ranging from economics to environment studies, from weather forecast to national defense. The Discrete Event Specification (DEVS) [1] is a modeling and simulation framework that has gained growing popularity in recent years. DEVS provides mechanisms for constructing hierarchical models in a modular manner. This allows reuse component to reduce development and testing time by separating between the model and simulation mechanism. Based on DEVS formalization, various extensions have been proposed. The P-DEVS formalism [2] increases parallelism by eliminating the serialization constraints in the original DEVS definition. The Cell-DEVS formalism [3] combines Cellular Automata [4] with DEVS to define cell spaces as discrete-event model.

With the development of DEVS, various simulation tools were created. CD++ [5] is such a tool to implement DEVS and Cell-DEVS formalism. Parallel CD++ (PCD++) employs both conservative and optimistic synchronization protocols to achieve high-performance simulations on distributed memory cluster systems [6][7] in Linux and UNIX platform. As a dominant operating system for personal computer, Microsoft's Windows has plenty of users. Developing Windows–based simulation tool allows general user to execute simulation with commodity PC machine and significantly reduce the learning curve. With the above motivation, we propose a new system, called PCD++/.NET, which execute distributed DEVS and Cell-DEVS simulation on Windows .NET framework [8].

Microsoft's .NET is an execution environment for Windows programs. It includes two main components: the common language runtime and the .NET base class library. The runtime and core parts of the base class library are specified as an open standard. This standard is called the common language infrastructure, and is published as the ECMA-335 (European Computer Manufacturers Association) [9]. In .NET, code runs in Common Language Runtime environment and has automatic memory management. Moreover, .NET provides built-in services for proxies, marshalling, network streams and remoting [10]. With these built-in services and PCD++ simulation engine, PCD++/.NET uses a set of Windows PC machines to load DEVS or Cell-DEVS models, transfer inter-process messages, and to execute simulations.

The rest of the paper is organized as follows: Section 2 introduces the P-DEVS and Cell-DEVS formalisms and CD++. Section 3 discusses the .NET Remoting techniques and gives PCD++/.NET design. Section 4 presents a performance analysis.

2. Background

The Discrete Event System Specification formalism (DEVS) is a M&S specification that aimed to study discrete event systems [11]. Based on systems theory, DEVS formalism provides a framework for defining hierarchical models in a modular way. A model described in DEVS can be either behavioral (atomic) or structural (coupled) components. A 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 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 $\delta_{ext}(s,e,X^b)$. 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.

A DEVS coupled model consists of atomic and/or other coupled models connected together. It defined as:

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

X and Y are defined as a set of input and output events respectively. D is a set of identifiers for the components of a coupled model and, for each $d \in D$, M_d is a basic 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 components are defined by the internal coupling (IC). A coupled model can be considered as a behaviorally equivalent atomic model, and thus be treated as a basic component in construction of more complicated hierarchical models.

Cell-DEVS [12] is an extension to DEVS that defines a cell space as an atomic DEVS model. The behavior of a cell space depends on the evaluation of local functions combined with explicit delay functions. A Cell-DEVS atomic model is formally defined as:

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

Each 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 c1oupled 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.

CD++ [5] provides a set of simulation engines to execute DEVS and Cell-DEVS models on different platforms. It decouples the modeling and simulation concepts by providing two separate frameworks. A *modeling framework* is defined as a hierarchy of classes that allow users to specify the behavior of atomic and coupled models. For each DEVS atomic models, users need to implement the various functions as required by the DEVS formalism in a C++ class, which is then incorporated into the modeling hierarchy during compilation. For DEVS coupled models and Cell-DEVS models, users can specify the coupling information and other attributes of cell spaces in a text-based configuration file using a built-in specification language.

CD++ provides a *simulation framework* that creates an executive entity for each component in the modeling hierarchy to implement the abstract simulator that is responsible for executing the simulation in line with the formalisms. These entities are specialized into two categories, namely *simulators* and *coordinators*. Simulators are associated with atomic models to trigger the output and state transition functions, while coordinators are attached to coupled models to keep track of the simulation time and to relay messages between their child simulators and parent coordinators. A special *root coordinator* keeps the global simulation time and communicates the simulated model with the surrounding environment. In order to run parallel and distributed simulations, the model is decomposed (as specified in a user-supplied partition file), and each subcomponent is executed by a separate process on a distinct processor. The simulation is carried out in a message-driven fashion. Each message represents an event with an associated timestamp that indicates the simulated virtual time of the event, and two types of messages exist:

- *content messages* include the external message (X, t) and output message (Y, t) that encode the actual data transmitted between the models,
- *control messages* include the initialization message (I, t), collect message (@, t), internal message (*, t), and done message (D, t) that are used internally by the simulator to control the simulation.

At present, there are many DEVS-based toolkits intended for parallel and distributed simulation that have been developed on different platforms using various middleware technologies. A non-comprehensive list includes:

- SOADEVS [13], a Service Oriented Architecture

(SOA) framework for test and evaluation of DEVS models. The Department of Defense (DoD) Architectural Framework (DoDAF) defines a common approach for DoD architecture description development, presentation, and integration. With the enhanced DoDAF, SOADEVS automates generation of DEVS model from DoDAF specifications can be executed and the architecture be simulated over a net-centric platform.

- DEVS/RMI [14] presents a dynamic reconfiguration mechanism that uses a run-time gathered "activity" metric to repartition a simulation model in order to improve the overall performance of a distributed simulation. DEVS/RMI uses Java RMI to achieve the synchronization of local and remote simulators and provides a software development environment for development of distributed simulation applications.
- DEVS/P2P [15] extends the DEVS modeling and simulation over a Peer-to-Peer (P2P) network system. The combination of DEVS M&S and P2P paradigm introduces a new distributed simulation approach, which is based on parallel DEVS formalism and P2P message communication protocol. DEVS/P2P consists of Autonomous Hierarchical Model Partitioning (AHMP), Autonomous Hierarchical Model Deployment (AMD), Activator, and Generic Simulator (GS). DEVS/P2P also introduces a customized new DEVS simulation protocol for the distributed simulation in which the peers solve the synchronization problem by themselves without involving a coordinator.
- DEVS/Grid [16] is a distributed M&S framework that allows DEVS M&S activities over Grid computing infrastructure. With the integration of Grid computing and DEVS M&S framework, DEVS/Grid provides cost-based hierarchical model partitioning, dynamic coupling restructuring, automatic model deployment, remote simulator activation and self-communication setup, etc components and service.
- DEVSCluster [17] is a multi-threaded, CORBA-based simulator for DEVS models that supports simulation in heterogeneous network environments.
- DEVS/HLA [18] is based on the High Level Architecture (HLA). It was used to demonstrate how an HLA-compliant DEVS environment could improve the performance of large-scale distributed modeling and simulation.
- PCD++ [6] is a parallel CD++ conservative engine that applied Master-slave coordinators to organize simulators.
- PCD++/optimal[7] presents various optimization strategies that applied to parallel CD++, including a risk-free message type-based state-saving strategy to reduce the number of states saved during the simulation, and a one log file per node strategy to improve file I/O operation.
- DCD++ [19] is web service-enabled CD++. Its goal is to interface the CD++ simulator to web service using web service wrappers, and implement distributed simulation engine using web services.
- PCD++Win [20] is integrated Windows MPI with PCD++ to execute parallel DEVS and Cell-DEVS simulation on Windows platform. It provides GUI for construction of clusters with commodity PCs and configuration of the simulation environment.

3. PCD++/.NET Remoting

Remoting is a process that permits programs to interact across different processes or machines. Several remoting methods are widely used. CORBA [21] is the middleware for heterogeneous system and uses an interface description language to specify the interfaces that objects will present to the outside world. Java RMI [22] is a Java interface for performing the object equivalent of remote procedure calls and uses a proxy/stub compilation cycle. Web service [23] provided the solution to cross-platform and cross-language interoperability and can send the call to remote components via HTTP post with SOAP. In comparison to this traditional approach, .NET Remoting has no proxy/stub-compilation cycles as in Java RMI and does not have to decide up front on the encoding format of remoting requests, it can be decided in a configuration file. Moreover, .NET Remoting gives a flexible and extensible framework that allows for different transfer protocol (HTTP or TCP) and encoding formatter (SOAP or binary).

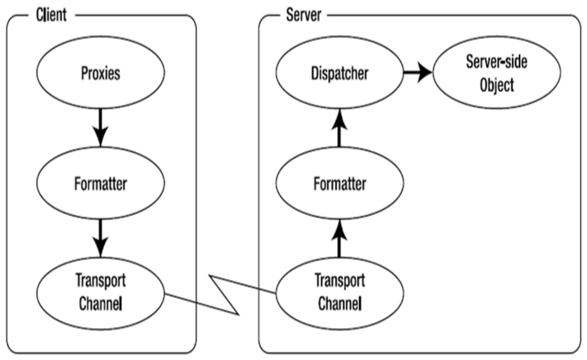


Figure 1. **.NET Remoting architecture** [24]

In Figure 1, whenever a client application holds a reference to a remote object, it will be represented by a proxy object. This proxy will allow all of the target object's instance methods to be called upon it. If a method call is placed to the proxy, it will be converted into a message, and the message will pass a serialization layer – the formatter – which converts it into a specific transfer format such as binary (or SOAP). The serialized message later reaches a transport channel, which transfers it to a remote process via a specific protocol like HTTP or TCP. On the server side, the message also passes a formatting layer, which converts the serialized format back into the original message and forwards it to the dispatcher. Finally, the dispatcher calls the target object's method and passes back the response values through all tiers. We can easily switch between implementations of the different layers without changing any source code. For example, a remoting application that's been written using a binary TCP-based protocol can be opened for third parties application using a SOAP/HTTP-based protocol by changing some lines in a configuration file to replace the .NET Remoting transport channel.

In .NET Remoting, a client can invokes a distributed object which could be in the same machine, in the same network, or located around WAN. Moreover, .NET Remoting allows a client sends a copy of message to remote object. Using this message passing facility, PCD++/.NET system is designed in Figure 2.

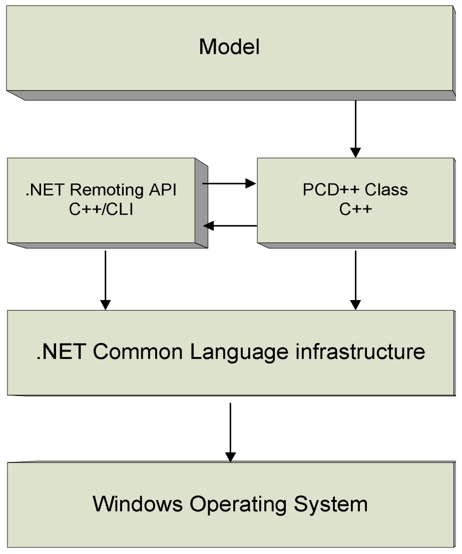

Model
.NET Remoting API C++/CLI
PCD++ Class C++
.NET Common Language infrastructure
Windows Operating System

Figure 2. **PCD++/.NET system**

In Figure 2, DEVS or Cell-DEVS models are built on top of the system. PCD++ interacts with .NET Remoting to exchange messages. The .NET Remoting API is used to build the client and server routine. Different languages can be used to support .NET Remoting API, including C++/CLI, VB.NET, C# and J# [25]. C++/CLI language is chosen to create client and server routine because C++/CLI can interoperate with native C++ code, which facilitates reusing existing software. The Common Language Infrastructure is an open specification that describes the executable code and runtime environment, and allows native C++ code of PCD++ to be targeted to the common language runtime. It does this at the source-code level by directly compiling and linking native C++ code[26]. Therefore, all code of PCD++/.NET is first compiled to the Common Intermediate Language, and then Common Language Runtime compiles Intermediate Language to machine-readable code that can be executed on the platform. Based on above PCD++/.NET system, a set of components are designed in Figure 3.

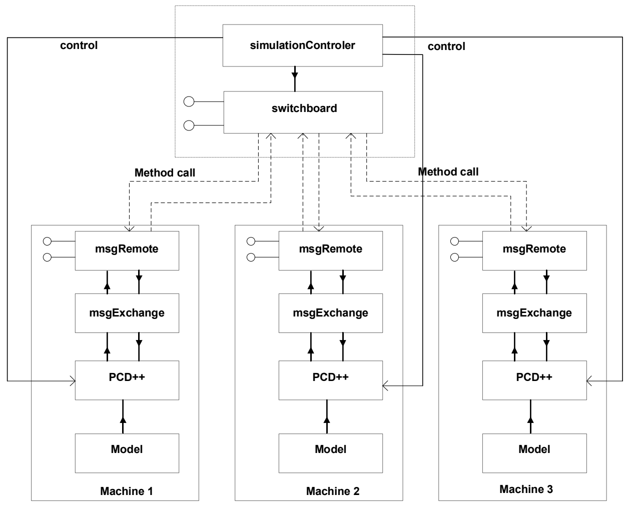


Figure 3. **PCD++/.NET components**

The components include:

- *simulationControler:* is used to control PCD++ residing in each node.
- *switchboard*: is used to conduct a coming method call to destination node.
- *msgRemote:* is used to send or receive a method call to or from the switchboard.
- *msgExchange:* is used to translate a C++/CLI message to a native C++ message and vice versa.

The above components interact according to following procedure:

1. Whenever the *simulationControler* sends a start signal, the PCD++ in each node will load the partition model and start the simulation loop.
2. If an inter-process message needs to be sent to another node, the C++ message is translated to C++/CLI message by msgExchange component. Then, the message is passed to msgRemote.
3. msgRemote invokes the sever object of the switchboard and sends message to destination node.
4. msgRemote of the destination node receives the coming message, and passes it to msgExchange.
5. msgExchange translates the coming C++/CLI message to C++ message, then pass it to PCD++.

In PCD++/.NET system, there are two very different types of remote interaction between components: objects that are referred to as ByValue object and MarshalByRefObjects. The former is serialized into a string or a binary representation and restored as a copy on the other side of the communication channel. The later uses a networked pointer to execute remote method calls on the server side. Figure 4 shows detail about how the *switchboard* and *msgRemote* work with these two types objects.

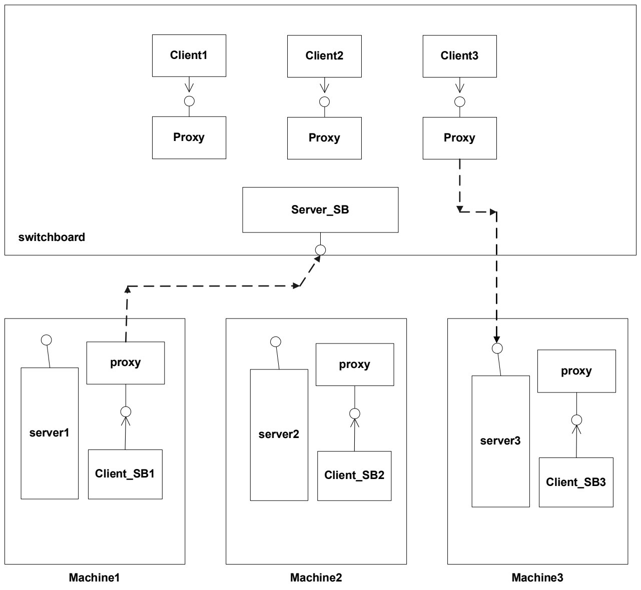


Figure 4. **PCD++/.NET method call**

In Figure 4, there has one sever and one client in each node. There has one sever and a set of clients in switchboard, where the number of the client is equal to the total number of nodes. In our example, Client_SB1, Client_SB2 and Client_SB3 are all associated with Server_SB. Client1, Client2 and Client3 are associated with server1, server2 and server3 respectively. If Machine1 needs to send a call to Machine3, Client_SB1 first registers a TCP channel and connect to Server_SB. Then, Server_SB pass the call to Client3. Finally, Client3 sends the call to server3.

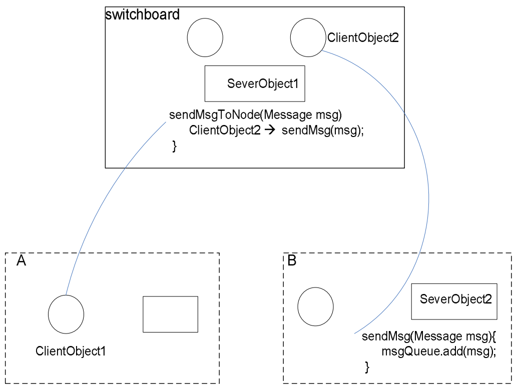


Figure 5. **Message passing**

It is worth to note that a copy of message in one node is sent to another node directly, it does not pass through switchboard. It can be illustrated with Figure 5. In Figure 5, we have node A, node B and the *switchboard*. A circle represents a client and a rectangle represents a sever. In node B, *SeverObject2* has a method *sendMsg(Message msg)*, which just adds a message to the local message queue. In the *switchboard*, *ClientObject2* associated with *SeverObject2* and can invoke the remote method *sendMsg(Message msg)*. If the node A needs send a message to node B, *ClientObject1* will invoke its remote method *sendMsgToNode(Message msg)* in *switchboard*. As shown in Figure 5, *sendMsgToNode* includes a method *sendMsg* which sends the message to node B.

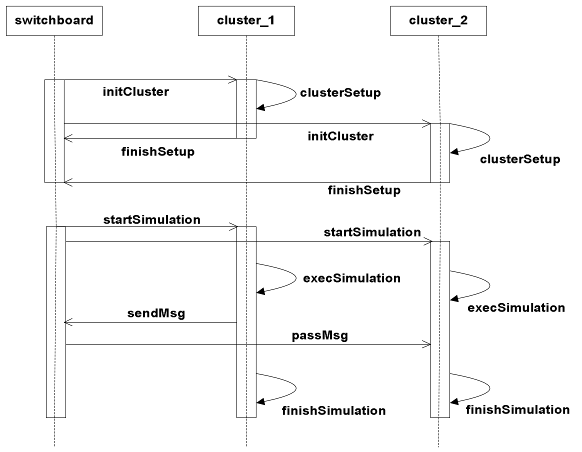


Figure 6. **PCD++/.NET sequence diagram**

From the above analysis, it can be seen that switchboard component plays an important role in the system. switchboard will create a set of client according to the number of nodes. Figure 6 shows a sequence diagram for PCD++/.NET that has two nodes. First, switchboard uses initCluster method to initialize all nodes, and then each node uses clusterSetup create server and client object. Once switchboard receives finishSetup from all nodes, it will start simulation with method call startSimulation. Each node will execute execSimulation method to enter PCD++ simulation loop, if cluster_1 needs send a message to cluster_2, sendMsg call will be forwarded to switchboard, and switchboard invokes a passMsg method to send message copy to cluster_2.

Based on above design, we can see that PCD++/.NET system has the following advantages:

- It presents a way for general user to use parallel simulation tool with commodity PC machine. That greatly reduces the learning curve and simulation cost.
- Comparing to PCD++Win, a tool that combine PCD++ with Windows MPI, PCD++/.NET provides a distributed fashion to execute DEVS and Cell-DEVS simulation across LAN or WAN.
- PCD++/.NET supports various protocols (TCP, HTTP or SMTP) and binary or SOAP formatter. That allows PCD++/.NET has more flexibility and extensibility than Web service-based simulation

tool which can only use HTTP protocol and SOAP formatter.

4. Experimentation results

This section presents a performance analysis and compares PCD++/.NET with PCD++Win which uses Windows MPI to transfer inter-process messages. The hardware is a group of desktop workstations (Intel Core 2 Duo Processor E6400 @ 2.13 GHz, 2 GB DDR2-Synch DRAM) connected through a 100 Mbps local area network (LAN). Microsoft Windows XP Professional, .NET framework 2.0 and Windows MPI DeinoMPI 1.1.0 [27] are installed in each machine. Both PCD++/.NET and PCD++Win use PCD++ conservative simulation engine and adopt master/slave coordinators [6] to organize simulators that is illustrated as Figure 7.

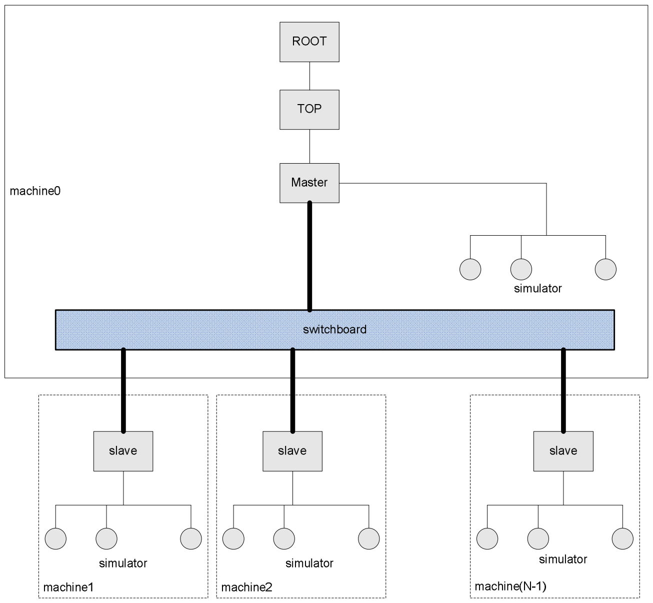


Figure 7. **PCD++/.NET simulators**

In Figure 7, the wide line represents the connecting with remoting routine and thin line represents the direct connecting in same Application Domain. Switchboard and Master coordinator are put together in machine0, but they belong to different Application Domain, so they connect with remoting routine.

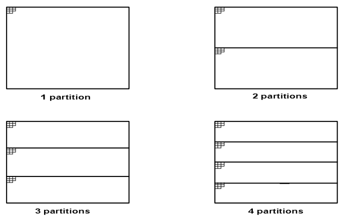
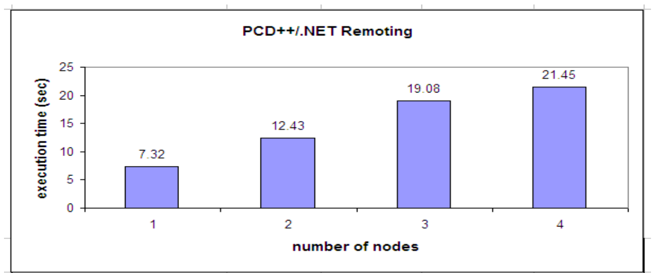


Figure 8. **Model partition**

The first testing model is watershed [28] that represents the water flow and accumulations depending on the characteristics of different vertical layers: air, vegetation, surface waters, soil, ground water, and bedrock. A simple partition strategy is used in the model testing as shown in Figure 8. It evenly divides the cell space into horizontal rectangles and each partition is run by one PC workstation. The model was coded as a 15*15*2 3D Cell-DEVS cell space and is testing by PCD++/.NET. The experiment result is shown in Figure 9.



	1 node	2 nodes	3 nodes	4 nodes
Local Messages (%)	100	78.87	72.76	64.79
Remote Messages (%)	0	21.13	27.24	35.21
Total Messages	65876	67102	68901	71245

Figure 9. **Experiment result of watershed model**

As the result to show, with the computing nodes increasing from 1 to 4, the execution time increases 2.9 times. This is due to the increasing remote messages between the nodes (remote messages is 35.21% with 4 nodes). The watershed model is a 3D cell space and uses a neighborhood consisting of 10 cells at both layers. Because of its complex rule definitions, the model requires high computing power to carry out the simulation and needs high bandwidth for the neighborhood's communication. PCD++/.NET can not offer such high bandwidth for remote communication. As a result, the performance is pulled down.

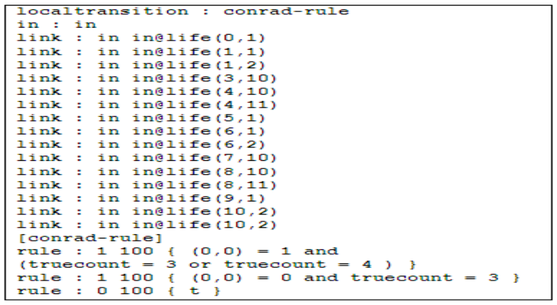
The next model we investigated is the Life model [29]. The Life was created by John Conway in 70's. It uses a 2D grid cells which can be either alive or dead. A new cell is born when it has exactly 3 neighbors; an existing cell survives if it has 2 or 3 neighbors. In all other cases, the cell dies. Life model can be code with specification language and model file are shown as Figure 10 and Figure 11.

```
[top]
components : life
in : in
out : out
link : out@life out
link : in in@life

[life]
type : cell
width : 12
height : 12
delay : inertial
defaultDelayTime : 100
border : nowrapped
neighbors : life(-1,-1) life(-1,0) life(-1,1)
neighbors : life(0,-1)  life(0,0)  life(0,1)
neighbors : life(1,-1)  life(1,0)  life(1,1)
```

Figure 10. **First part of Life model**

As Figure 10 shown, the input and output ports are add to the model. This is because that the Root coordinator advances the clock of the simulation and the simulation continues until at least one of the following conditions holds: there are no more events/messages scheduled by any of the processors, or the simulation clock reaches the maximum execution time as provided by the user. Therefore, input events keep simulation running until maximum execution time is reached. As Figure 11 shown, a set of input ports link to model and an event file can be used to input events at any time point.



```
localtransition : conrad-rule
in : in
link : in in@life(0,1)
link : in in@life(1,1)
link : in in@life(1,2)
link : in in@life(3,10)
link : in in@life(4,10)
link : in in@life(4,11)
link : in in@life(5,1)
link : in in@life(6,1)
link : in in@life(6,2)
link : in in@life(7,10)
link : in in@life(8,10)
link : in in@life(8,11)
link : in in@life(9,1)
link : in in@life(10,2)
link : in in@life(10,2)
[conrad-rule]
rule : 1 100 { (0,0) = 1 and
(truecount = 3 or truecount = 4 ) }
rule : 1 100 { (0,0) = 0 and truecount = 3 }
rule : 0 100 { t }
```

Figure 11. **Second part of Life model**

For the Life model, we use a non-wrapped border and 12*12 cell space. The "border" parameter allows define the neighborhood for the cell in border and a "non-wrapped border" may reduce remote communication of cells. The partition strategy also evenly divides the cell space into horizontal rectangles (Figure 8). The experiment results of the "Life" model are shown in Figure 12 and Table 1.

Figure 12 shows that PCD++/.NET achieves comparable performance to the PCD++Win for the "Life" model. This is primarily due to the model is 2D cell space and has simple rule definition and modest remote messages between the nodes (7.91% in 4 nodes), so it does not need high bandwidth for the communication between nodes.

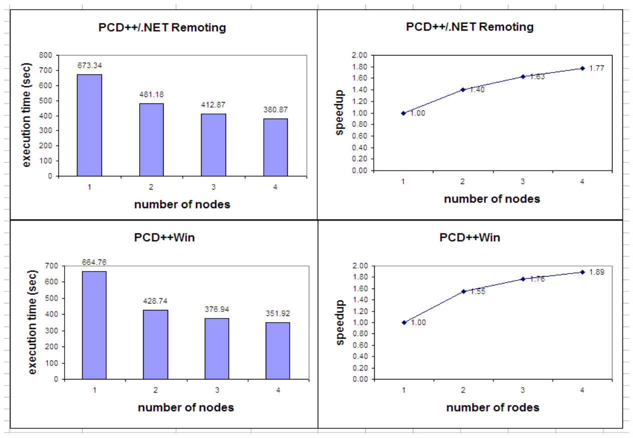


Figure 12. **Execution time of Life model**

In Table 1, it can be seen that the ratio of remote message of Life model has greatly decreased comparing with watershed model (Figure 9). Therefore, the partition strategy and model behavior is very suitable for the communication bandwidth of PCD++/.NET and leads to a better performance.

Table 1. **Message number of Life model**

Game of Life model				
	1 node	2 nodes	3 nodes	4 nodes
Local Messages (%)	100	97.58	94.37	92.09
Remote Messages (%)	0	2.42	5.63	7.91
Total Messages	8895321	8896763	8897981	8898813

In Figure 13, we show a comparison between PCD++/.NET and PCD++Win for Life model executing on 1-4 nodes. The results demonstrate that PCD++/.NET is slower than PCD++Win between 1.3% and 12.2%. This is because that In PCD++/.NET, the source code is compiled to Intermediate Language (IL), and then Intermediate Language is compiled to machine code in the runtime and interacts with operating system. PCD++Win is based on MPI which uses Abstract Device Interface (ADI) [30] to interact on operating system. ADI is a set of functions which represent abstraction of communication device operations modes and protocols. The ADI binary code interacts with operating system directly in the runtime. That leads PCD++/.NET is slower than PCD++Win.

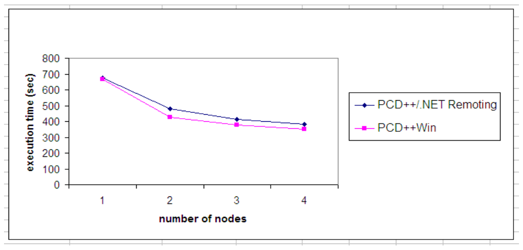


Figure 13. **Result comparison**

5. Conclusion

.NET Remoting is a technology that programs and software components can interact across application domains, processes, and machine boundaries. This allows applications to take advantage of remote resources in a networked environment. In this paper, PCD++/.NET framework is presented. It integrates .NET Remoting with PCD++ to execute distributed simulation for DEVS and Cell-DEVS model. PCD++/.NET provides a way for general user to use parallel simulation tool with commodity PC windows machine. That reduces the learning curve and simulation cost. Moreover, it presents a distributed fashion to execute simulation across network and supports various protocols and formatter. Finally, PCD++/.NET could has a fast communication speed by using TCP channel and binary formatter; therefore, it could reach a tolerable performance for some testing models.

Reference

[1] 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.

[2] 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.

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

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

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

[6] 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.

[7] 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.

[8] http://www.microsoft.com/net/

[9] http://www.ecma-international.org/publications/standards/Ecma-335.htm

[10] http://msdn.microsoft.com/en-us/library/ms978411.aspx

[11] Zeigler, B. "Theory of modeling and simulation". First Edition, Wiley. 1976

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

[13] S. Mittal, J.L. Risco. DEVSML: automating DEVS Execution over SOA towards Transparent Simulators. In Special Session on DEVS Collaborative Execution and Systems Modeling over SOA, DEVS integrative M&S Symposium DEVS'07, March 2007

[14] 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.

[15] 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.

[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] 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.

[18] 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.

[19] Madhoun, R; Wainer, G. "Performance analysis of Web-Based CD++". In Proceedings of DEVS Symposium 2007, Norfolk, VA. 2007

[20] Feng, B; Liu, Q; Wainer, G. "Parallel simulation of DEVS and Cell-DEVS models on Windows-based PC cluster systems". In Proceedings of SCS High Performance Computing and Simulation Symposium (HPCS 2008). Ottawa, Canada.

[21] http://www.corba.org/

[22] http://java.sun.com/docs/books/tutorial/rmi/index.html

[23] http://www.w3.org/

[24] Rammer, I. And Szpuszta, M. 2005. "Advanced .NET Remoting", Second Edition, ISDN: 1-59059-417-7

[25] http://www.dotnetlanguages.net/DNL/Resources.aspx

[26] Heege, M. 2007, "Expert C++/CLI: .NET for Visual C++ Programmers", ISBN-13: 978-1-59059-756-9, ISBN-10: 1-59059-756-7

[27] http://mpi.deino.net/

[28] Moon, Y.; Zeigler, B.; Ball, G; Guertin, D. P. "DEVS representation of spatially distributed systems: validity, complexity reduction", IEEE Transactions on Systems, Man and Cybernetics. PP, 288-296. 1996

[29] http://www.sce.carleton.ca/faculty/wainer/wbgraf/samplesmain_1.htm

[30] http://www-unix.mcs.anl.gov/mpi/mpich1/papers/mpicharticle/node22.html