A Case Study in Distributing a SystemC Model

V. Galiano[1], M. Martínez[2], H. Migallón[1], D. Pérez-Caparrós[1], and C. Quesada[1]

[1] Miguel Hernández University, Av. de la Universidad, s/n, 03202 Elche
{vgaliano,hmigallon}@umh.es, davidperez@ieee.org, carlos.quesada@graduado.umh.es

[2] Design of Systems on Silicon (DS2), C.R. Darwin, 2, 46980 Paterna, Valencia
marcos.martinez@ds2.es

Abstract. SystemC is a library that facilitates the development of Transaction Level Models (TLM). These models are composed of both hardware and software components. This library allows designing and verifying hardware system components at a high level of abstraction. This supports the development of complex systems. A real industry SystemC model usually contains a high number of functional blocks which increase its simulation run time. SystemC executes only one process at any time, even if the hardware supports execution of concurrent processes. In this paper we present a new methodology for distribution of the simulation of complex models in a parallel computing system. We apply our own approach in a real industry SystemC model of a Power Line Communication (PLC) network.

Keywords: SystemC, TLM, Distributed Systems, PDES, PLC, MPI, Serialization.

1 Introduction

Built on top of C/C++, SystemC[1] allows the full object-oriented power of the language, while providing constructs to easily describe concurrent hardware behavior. The major benefits of SystemC include architectural exploration and performance modeling of complex SoC designs, and the ability to run software on a virtual model of the hardware platform prior to the availability of Register Transfer Level (RTL) code. These benefits are enabled by the use of Transaction Level Modeling (TLM) add-on library.

As the complexity of SystemC models increases, more computational resources are required by their simulation, which means higher simulation run times. To speed the simulation run times of complex SystemC models, we propose to apply principles of Parallel Discrete Event Simulation (PDES)[2]. In this paper we present a solution to the problem of distributing the simulation of SystemC models in a parallel computing system. We demonstrate this solution by implementing it on a real industry SystemC model of a PLC network.

The rest of this paper is summarized as follows: Section 2 presents some background information on distributed SystemC simulation. We briefly discuss related work in Section 3. Section 4 describes the SystemC model that will be distributed in Section 5. Finally, this paper is concluded in Section 6.

2 Background

2.1 Parallel Discrete Event Simulation

Parallel Discrete Event Simulation (PDES), also known as Distributed Simulation, refers to execution of a single discrete event simulation program on a parallel or distributed computing system. In a discrete event simulation, model being simulated only changes its state at discrete points in simulated time. Model jumps from state to state upon the occurrence of events. Concurrent events in the simulated model are executed in a sequential manner. Subsequently, simulation of complex systems with a substantial amount of parallelism is an extremely time consuming task [3]. PDES aroused as a solution to this problem. In a PDES, simulated model is decomposed into a set of concurrent processes which are executed in a parallel computer. These processes are known as Logical Processes (LPs). LPs are essentially autonomous and independent DESs logically connected by channels, with part of the simulated system state, queues of pending events, and a local clock. All interactions between processes are modeled as time stamped event messages between LPs[2].

There are two types of algorithms that deal with the problem of synchronization between LPs in PDES systems, conservative and optimistic [2]. Conservative algorithms process in parallel only those events with the same time stamp. Parallel executions must resynchronize before any event with a greater time stamp can be processed. Optimistic algorithms execute events regardless of their time stamps, and implement mechanisms to detect and recover from any resulting causality violation [4].

2.2 System Level Communication Modeling with SystemC

SystemC library provides the implementation of many types of hardware-specific objects, such as concurrent and hierarchical modules, ports, channels, and clocks. Structural decomposition of the simulated model is specified with modules, which are the basic building blocks. The functionality of system is described in processes. Interaction between modules can be modeled using channels, interfaces and ports. Thus, a SystemC description consists of a set of interconnected modules, which are composed of processes, ports, channels and instances of other modules.

A channel implements one or more interfaces. An interface consists of a set of method declarations, but does not implement these methods. A port enables a module, and hence its processes, to access a channel interface. The interface method, which is implemented in the channel, is executed in the context of the process. A port is defined in terms of an interface type, which means that the port can be used only with channels that implement that interface type [5]

[6]. Processes usually communicate with other processes through ports bound to channels by way of interfaces, or through ports bound to another type of port known as `sc_export`. `sc_export` is similar to standard ports in that the declaration syntax is defined on an interface, but this port differs in connectivity. It allows to move the channel inside the defining module, and use the port externally as though it were a channel [7].

3 Related Work

There have been several attempts to distribute SystemC simulations, but it seems that no one is definitive or used as standard. All of them, to the best of our knowledge, use conservative algorithms to maintain the consistency of the simulated model. Optimistic algorithms are harder to implement and require too many resources [2].

There are basically two strategies to distribute SystemC simulations. Some authors have tried to parallelize/distribute SystemC by directly modifying its simulation engine. The other strategy is based on wrapping communications between LPs using a self-developed communication library that extends system level modeling capabilities of SystemC. A major drawback of modifying the SystemC simulation engine is the need to provide a continuous support to follow future implementations of the SystemC standard. On the other hand, it is a more customizable approach.

[8] and [9] follow first strategy and propose to customize SystemC simulation engine. Both proposals use the Message Passing Interface (MPI)[10] standard for communication between LPs, which are wrapped in a top-level SystemC module. This strategy obtains reasonable performance results for well-balanced coarse-grained system models. LPs must be manually defined and mapped into different processes by the modeler of the simulated system. [8] avoids explicit lookahead by choosing a robust synchronization algorithm [11][12].

Other approaches, following the previously mentioned second strategy, include their own communication library that bridges LPs and synchronizes shared signals between them using explicit lookahead. [13][14][15] and [16] present solutions that avoid modifying SystemC library source code. [16] uses MPI for communication and synchronization between LPs, while [13] propose the use of TCP/IP socket communication. In [13] each LP is mapped into independent executable pieces due to its communication technique.

There are other authors working on geographically distributed SystemC simulations [17]. Communications are made over Internet protocols and middleware such as SOAP, RMI or CORBA. However, their goal is not to obtain a better performance as well as our work aims [18].

We propose a solution which is based on [13] and [16] proposals. We use a wrapper SystemC module to communicate and synchronize manually distributed LPs using the MPI standard. The following sections focus on the distribution of a real industry SystemC model by using our approach.

4 SystemC Model of a Power Line Communication Network

SystemC model that will be distributed in Section 5 is a real industry model of a PLC network. An overall view of this model is given in Figure 1. The system comprises a PLC Channel (PLCC), three traffic Flow Generators (FG), and four Endpoint PLC Nodes (EN). These modules are wrapped in a top-level module which is used as a test bench for validating the model through simulation. This module includes instantiation of all other modules that have been defined and used in the design. Each component is modeled using a complex combination of high and low levels of abstraction. The main components of the model and the communication process between them are described next.

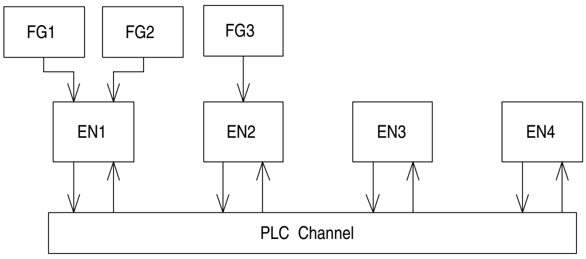

Fig. 1. SystemC model of a PLC network

4.1 Components

Flow Generators. The FG modules generate configurable network traffic according to several parameters which are set in a configuration file. The network traffic is injected into a particular EP module in the form of Ethernet frames.

Endpoint Nodes. EN modules communicate with other EN modules by sending and/or receiving Packet Data Units (PDUs) through the PLC Channel. Each EN module implements a protocol stack (see Figure 2), which divides the network architecture into five layers. These protocol layers are from top to bottom: Bridge, Convergence, Logical Link Control (LLC), Media Access Control (MAC), and Physical Layer (Phy).

In the sending process, Bridge Layer of the EN module receives through an external interface the Ethernet frames generated by the FG module. Nodes encapsulate these frames downwards to lower layer protocols. The resulting Protocol Packet Data Units (PPDUs) are broadcast over the PLC channel. The action of receiving comprises the opposite operation of reversing the encapsulating process.

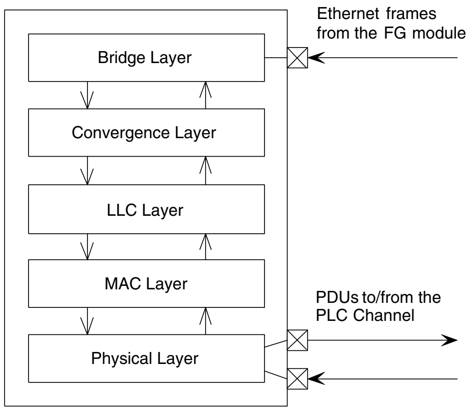
Ethernet frames from the FG module
Bridge Layer
Convergence Layer
LLC Layer
MAC Layer
PDUs to/from the PLC Channel
Physical Layer

Fig. 2. Protocol stack implemented in EN module

PLC Channel. The physical link is modeled in a separate module. This module acts as a communication channel amongst the EN modules. It is modeled like a First-In First-Out (FIFO) queue of Physical Layer PDUs, which is a common data structure to manage data flows.

4.2 Communication between Endpoint Nodes and PLC Channel

Depending on the network traffic source-destination, PLC network model uses two different implementations of SystemC communication.

– Endpoint nodes send PPDUs to PLC Channel through a `sc_export` bound to a port of the PLC Channel. This type of communication allows to move the implementation of interfaces inside the PLCC module (see Section 2.1).
– Network traffic from PPLC module to EN modules is done through a SystemC channel. A PLCC module port is bound to the same SystemC channel as the EN module port. In this case, interfaces are implemented by the SystemC channel.

5 Distributed Model

Following the principles of PDES, we have manually split the PLC network model described in the previous section into two LPs (see Figure 3). The proposed partition separates the EN and FG modules from the PLC Channel.

We propose two solutions to deal with the distribution of the two types of SystemC communication implementations that are used in the sequential model (see Section 4.2).

As mentioned in Section 4.2, if one module is connected to another one through a `sc_export`, interfaces are implemented only in the second module. Both modules, sender and receiver, need to use these interfaces. In the distributed model,

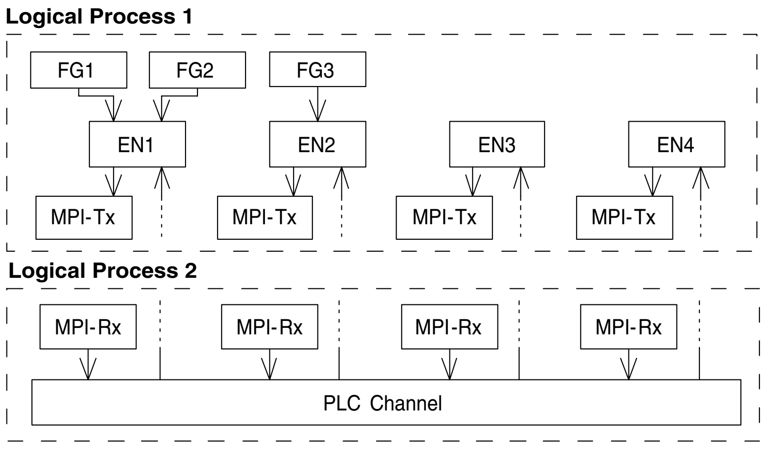

Fig. 3. Distributed model of a PLC network

sender and receiver are mapped into different LPs. To communicate these modules we have developed two new SystemC modules, `MPITx` and `MPIRx`. `MPITx` receives PPDUs from EN module and send them to `MPIRx` using the MPI standard, which is located in Logical Process 2 as it can be seen in Figure 3. `MPIRx` is connected to the PLCC module through a `sc_export` (see Figure 4). We have used Boost.MPI and Boost.Serialization libraries[19] for MPI communication and serialization of PPDUs.

In the second case, communication between two modules is implemented by binding sender and receiver ports to the same SystemC channel. In the distributed model, sender and receiver ports are bound to two instances of the same type of SystemC channel. We have implemented MPI primitives in the interfaces used by these ports to send and receive data to/from the channel (see

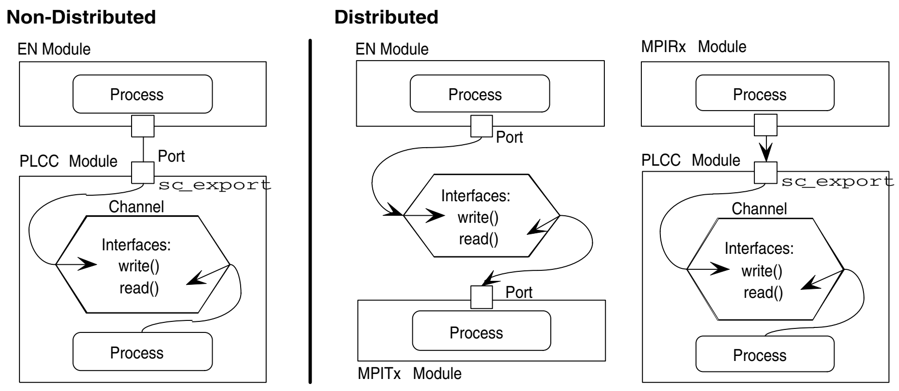

Fig. 4. Distribution of sc_export based communication between modules

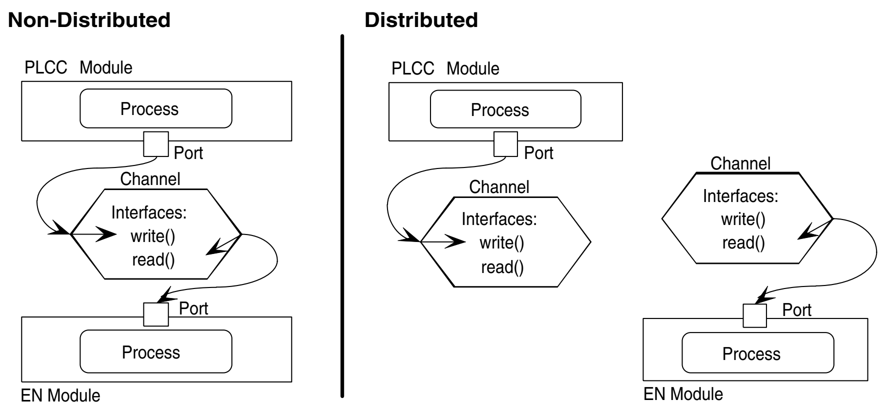

Fig. 5. Distribution of channel based communication between modules

Figure 5). This solution can be applied only if the channel does not implement any data buffer that should be shared for the communication to work.

6 Conclusion

In this paper, we present a new approach to distribute the simulation of complex systems modeled with SystemC in a parallel computing system. We use this approach to distribute a real industry SystemC model of a Power Line Communication (PLC) network.

The previously mentioned PLC network model is composed by several nodes which communicate with each other through a PLC channel. To distribute this model, we split the system into Logical Processes (LPs) which exchange complex Packet Data Units (PDUs). These PDUs are built following a set of network protocols (Ethernet, MAC, etc.). To implement the communication of structured data types amongst LPs we used the MPI standard and serialization techniques.

Not many researchers have dealt with the simulation of distributed SystemC models since the Open SystemC Initiative was announced in 1999. There have been several SystemC parallelization attempts, but it seems there is no one definitive or used as standard. We propose a novel solution to the problem of distributing the simulation of real industry SystemC models. The distribution of a SystemC realistic model encourages us to continue working on this solution. Our future work will be focused on implementing a communication library that could be used in a wider range of distributed SystemC models.

Acknowledgements

This work has been partially funded by the Ministry of Industry, Tourism and Trade of Spain under project number TSI-020100-2008-215.

References

1. IEEE Computer Society: IEEE Standard SystemC Language Reference Manual (2006), http://standards.ieee.org/getieee/1666/index.html
2. Fujimoto, R.: Parallel and Distribution Simulation Systems. John Wiley & Sons, Inc., New York (1999)
3. Livny, M.: A study of parallelism in distributed simulation. In: Proceedings of the SCS Multiconference on Distributed Simulation, pp. 94–98 (1985)
4. Bhargava, B., Lian, S.R.: Independent checkpointing and concurrent rollback for recovery in distributed systems-an optimistic approach. In: Seventh Symposium on Reliable Distributed Systems, 1988. Proceedings, pp. 3–12 (October 1988)
5. Pasricha, S.: Transaction level modeling of soc with systemc 2.0. In: Synopsys Users Group Conference (2002)
6. Panda, P.: Systemc: a modeling platform supporting multiple design abstractions. In: ISSS 2001: Proceedings of the 14th international symposium on Systems synthesis, pp. 75–80. ACM, New York (2001)
7. Black, D.C., Donovan, J.: SystemC: From the ground up. Eklectic Ally (2005)
8. Cox, D.: Ritsim: Distributed systemc simulation. Master's thesis, Rochester Institute of Technology (2005), http://hdl.handle.net/1850/1014
9. Chopard, B., Combes, P., Zory, J.: A parallel version of the osci systemc kernel. In: Alexandrov, V.N., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2006. LNCS, vol. 3994, pp. 653–660. Springer, Heidelberg (2006)
10. Dongarra, J., Huss-Lederman, S., Otto, S., Snir, M., Walkel, D.: The message passing interface (mpi) standard (1998), http://www-unix.mcs.anl.gov/mpi
11. Bagrodia, R., Takai, M.: Performance evaluation of conservative algorithms in parallel simulation languages. IEEE Trans. Parallel Distrib. Systems 11(4), 395–411 (2000)
12. Chandy, K., Sherman, R.: The conditional-event approach to distributed simulation. In: Proceedings of the SCS Multiconference on Distributed Simulation. Society for Computer Simulation International, vol. 21, pp. 93–99 (1998)
13. Trams, M.: Conservative distributed discrete event simulation with systemc using explicit lookahead. Technical report, Digital Force (2004), http://www.digital-force.net
14. Trams, M.: A first mature revision of a synchronization library for distributed rtl simulation in systemc. Technical report, Digital Force (2004)
15. Trams, M.: User manual for distributed systemc synchronization library rev. 1.1.1. Technical report, Digital Force (2005)
16. Hamabe, M.: Systemc with mpi for clustering simulation, www5a.biglobe.ne.jp/~hamabe/SystemC
17. Meftali, S., Dziri, A., Charest, L., Marquet, P., Dekeyser, J.: Soap based distributed simulation environment for system-on-chip (soc) design. In: Forum on Specification and Design Languages, FDL 2005 (2005)
18. Galiano, V., Pérez-Caparrós, D., Palomino, J.A., Migallón, H., Martínez, M.: Speeding up in distributed systemc simulations. Advances in Soft Computing 50/2009(4), 24–28 (2008)
19. Dawes, B., Rivera, R.: Boost c++ libraries, http://www.boost.org/