Implementing MPI Based Portable Parallel Discrete Event Simulation Support in the OMNeT++ Framework

David Wu[†] Eric Wu[†] Johnny Lai[†] Andràs Varga[‡]
Y. Ahmet Şekercioğlu[†] Gregory K. Egan[†]
[†] *Centre for Telecommunication and Information Engineering, Monash University, Melbourne, Australia*
[‡] *Department of Telecommunications, Technical University of Budapest, Hungary*

Abstract

In this paper, we introduce our Message Passing Interface (MPI) based Object-Oriented parallel discrete event simulation framework. The framework extends the capabilities of the OMNeT++ simulation system. In conjunction with this project, our research efforts also include the development of synchronization methods suitable for architectural properties of the distributed-memory and shared-memory parallel computer systems.

We intend to harness the computational capacity of these parallel systems for simulation, and to use this framework for modeling very large scale telecommunication networks to investigate protocol performance and rare event failure scenarios.

1 Introduction

Inevitably, telecommunication networks are increasingly becoming more complex as the trend toward the integration of telephony and data networks into integrated services networks gains momentum. It is expected that these integrated services networks will include wireless and mobile environments as well as wired ones. As a consequence of the rapid development, reduced time to market, fusion of communication technologies and rapid growth of the Internet, predicting network performance, and eliminating protocol faults have become an extremely difficult task. Attempts to predict and extrapolate the network performance in small-scale experimental testbeds may yield incomplete or contradictory outcomes. Application of analytical methods is also not feasible due to the complexity of the protocol interactions, analytical intractability and size [Bagrodia et al., 1998]. For large scale analysis in both the spatial and temporal domain, accurate and detailed models using parallel simulation techniques offer a practical answer. It should be noted that, simulation is now considered as a tool of equal importance as complementary to the analytical and experimental studies for investigating and understanding the behavior of various complex systems such as climate research, evolution of solar system and modeling nuclear explosions.

As part of our ongoing research programs on analysis of protocol performance of mobile IPv6 networks, we have developed a set of OMNeT++ models for accurate simulation of IPv6 protocols [Lai et al., 2002]. We are now focusing our efforts to simulate mobile IPv6 networks in very large scale. For this purpose, we intend to use the computational capacity of APAC (http://www.vpac.org) and VPAC (http://www.apac.edu.au) supercomputing clusters.

In this paper we present our MPI (Message Passing Interface) [MPI] based portable parallel discrete event simulation framework. In a series of future articles, we will be reporting our related research on synchronization methods, efficient topology partitioning for parallel simulation, and topology generation for mobile/wireless/cellular Internet.

2 Parallel Discrete Event Simulation

Discrete event simulation of telecommunications systems is generally a computation intensive task. A single run of a wireless network model with thousands of mobile nodes may easily take several days and even weeks to obtain statistically trustworthy results even on today's computers, and many simulation studies require several simulation runs [Bagrodia et al., 1998]. Independent replicated simulation runs have been proposed to reduce the time needed for a simulation study, but this approach is often not possible (for example, one simulation run may depend on the results of earlier runs as input) or not practical. Parallel discrete event simulation (PDES) offers an attractive alternative. By distributing the simulation over several processors, it is possible to achieve a speedup compared to sequential (one-processor) simulation. Another motivation for PDES is distributing resource demand among several computers. A simulation model often exceeds the memory requirements of a single workstation. Even though distributing the model over several computers and controlling the execution with PDES algorithms may result in slower execution than on a single workstation (due to communication overhead in the PDES mechanism, a slowdown of 30 to 50% per processor reported in [Bagrodia et al., 1998]), but at least it is possible to run the model.

Discrete event simulation consists of events that are executed in order of the timestamps associated with each event. The events are executed by logical processes (LP) that simulate one or more physical processes of the physical system under simulation. An LP does not share state with any of its neighbors in the simulation. Messages are exchanged between LPs to represent event execution and also to notify changes of state. In

Proceedings 14th European Simulation Symposium
A. Verbraeck, W. Krug, eds. (c) SCS Europe BVBA, 2002

sequential simulations the events are kept in a global queue. In PDES, different events in the simulation are distributed among processors and each processor will now have to deal with messages arriving asynchronously from other processors.

2.1 Synchronization Methods for Parallel Discrete Event Simulation

In PDES different events in the simulation are distributed among processors, each with their own local simulation time. Messages can now arrive asynchronously from other processors and hence synchronization is required to maintain causality. The two traditional classes of algorithms for ordering events in a parallel simulation are conservative and optimistic. Both algorithms are exact in the sense that they produce the same results as a sequential simulation would.

2.1.1 Conservative and Optimistic Synchronization

Conservative algorithms preserve the causality constraint across LPs by ensuring that a message arriving at time τ can only be processed if there are no other messages that will arrive from other LPs with a timestamp less than τ. The optimistic approach does not strictly enforce the causality constraint like the conservative approaches. The messages can potentially be processed out of order with respect to their timestamps i.e. it can process a message with timestamp of $\tau + \Delta$ and then process a newly arrived message with timestamp τ. When the algorithm detects that the causality constraint has been violated, roll back of previous computations occur, and the messages are reordered to preserve causality [Bagrodia and Takai, 2000]. Although the PDES concept was introduced two decades ago [Fujimoto, 1990], it is still not part of the everyday practice: the most commonly used simulation tools like ns2 [of Southern California] or OPNET$^{\text{TM}}$ [OPNET] have only recently added support for parallel simulation. But lack of mature tool support is not the only cause for holding back development of PDES. The other contributory causes are:

1. Difficulty in preparing models for PDES. They require manual work: models have to be partitioned, and then instrumented for parallel execution. This is extra implementation effort, which may or may not pay off during actual simulation experiments.

2. Both conservative and optimistic PDES algorithms inherently build on heavy communication and frequent synchronization between LPs, and they are very sensitive to communication latencies. For practical purposes, it is only possible to achieve speedup on shared-memory multiprocessors. On clusters, network latencies tend to ruin speedup.

3. Optimistic synchronization requires periodic state saving and the ability to restore the entire simulation to previous states. This is difficult to implement and performance can suffer in cases of excessive rollback

Issue (2) is a definite drawback, considering that clusters are more readily available than shared-memory multiprocessor systems, and are becoming the dominant form of supercomputers (see for example the Beowulf project (http://www.beowulf.org). A novel and less known approach to PDES is the Statistical Synchronization Method (SSM) [Pongor, 1992].

2.1.2 Statistical Synchronization Method

SSM works by sending statistics instead of messages. A message arrives at a segment S_a. S_a processes the message and sends only statistical data derived from the message across the link to the destination segment S_b. S_b re-generates the message from the collected statistical data.

SSM does not require instrumentation of existing models, only the addition of "*statistical interfaces*" to gather and collect the statistics. Not only is it comparably easier to implement than the traditional methods, there is potential for much greater speedup, because the method is less sensitive to network latency, due to a reduction in synchronization overhead. However the result obtained from statistical synchronization is different from conservative and optimistic synchronization. The error can be attributed to the statistical nature of the synchronization.

An extension to SSM, so-called SSM-T implemented in OMNeT++ is developed by Gabor Lencse [Lencse, 2002]. SSM-T is a time-driven version of SSM by introducing a new factor of local virtual time (LVT) in each segment. The idea of SSM-T is to let each segment run independently and let the LVTs of the segments meet at certain points of time ensuring an approximate synchronism. Lencse has demonstrated SSM's suitability for simulation of practical networks [Lencse, 1998].

3 An Overview of the OMNeT++ Simulation Framework

OMNeT++ is a C++-based discrete event simulation package developed at the Technical University of Budapest by András Varga [OMNeT++, Varga, 2001]. The primary application area of OMNeT++ is the simulation of computer networks and other distributed systems. It is open-source, free for non-profit use, and has a fairly large and active user community. It also allows the design of modular simulation models, which can be combined and reused flexibly. Additionally, OMNeT++ allows the composition of models with any granular hierarchy. It has been shown that this simulation framework is suitable for simulation of complex systems like Internet nodes and dynamics of TCP/IP protocols realistically [Kaage et al., 2001, Wehrle et al., 2001].

Simulated models are composed of hierarchically nested modules. In OMNeT++, there are two types of modules: simple and compound modules. Simple modules form the lowest hierarchy level and implement the activity of a module, and they can arbitrarily be combined to form compound modules. Modules communicate with message passing. Messages can be sent either through connections that span between modules, or directly to their destination modules. The user defines the structure of the model (the modules and their interconnection) by using the topology description language (NED) of OMNeT++ [Varga, 1997].

Simple modules are implemented in C++, using the simulation kernel system calls and the simulation class library. For each simple module, it is possible to choose between process-style and protocol-style (state machine) modeling. Therefore, different parts of computing and communication systems can be programmed in their natural way and connected easily. The simulation class library provides a well-defined application programmer's interface (API) to the most common simulation tasks, including: random number generation; queues, arrays and other containers; messages; topology exploration and routing; module creation and destruction; dynamic topologies; statistics; density estimation (including histograms, P2 and k-split [Varga and Fakhamzadeh, 1997]); output data recording. The object-oriented approach allows the flexible extension of the base classes provided in the simulation kernel.

Model components are compiled and linked with the simulation library, and one of the user interface libraries to form an executable program. One user interface library is optimized for command-line and batch-oriented execution, while the other employs a graphical user interface (GUI) that can be used to trace and debug the simulation.

3.1 OMNeT++ Kernel Support for Implementing Parallel Simulation

OMNeT++'s simulation kernel allows messages to traverse across segments. A message can contain arbitrarily complex data structures; these are transferred transparently, even between hosts of different architectures. The simulation kernel provides a simple synchronization mechanism (*syncpoints*, available through the `syncpoint()` call) that can preserve causality when sending messages between segments. Syncpoints correspond to null messages found in the literature.

To illustrate the concept of syncpoints, consider two segments A and B. Segment A must know in advance when it will send the next message to segment B and announce it with the `syncpoint()` call. The simulation kernel sends the syncpoint to segment B. When segment B's model time reaches the specified time, segment B's simulation kernel blocks execution until the promised message arrives from A. Then the simulation continues, typically but not necessarily with the message that has just been received from A.

In the reverse case when A is ahead of B, A's message arrived at B before it has reached the syncpoint. In this case, there is no problem and the syncpoint is just an unnecessary precaution. B just inserts the message in its future event set, clears the syncpoint and continues execution.

Message sending and syncpoints enable one to implement conservative PDES and also SSM. The simulation class library contains objects that explicitly support the implementation of models using Statistical Synchronization.

OMNeT++ supports flexible partitioning of the model. In the NED language, by using machine parameters you can specify logical hosts for different modules at any level of the module hierarchy of the network. You map logical hosts to physical ones in the ini file; if you map several logical hosts into the same physical machine, they will be merged into a single OMNeT++ process.

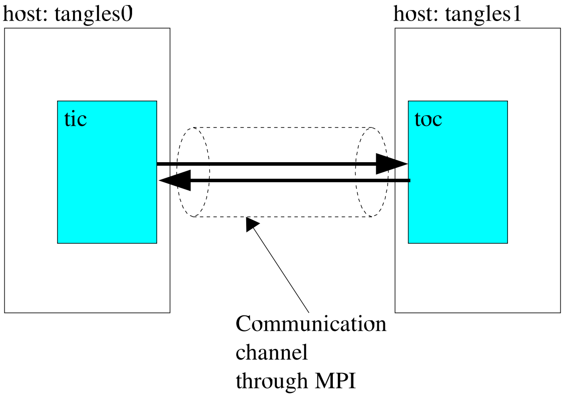

Figure 1: A very simple parallel simulation: `tic` runs on host `tangles0` and `toc` runs on host `tangles1`. The simulated communication links use MPI. MPI messages are sent and received through TCP/IP over an Ethernet switch connecting the hosts.

4 Design of the Parallelized OMNeT++ by using Message Passing Interface (MPI)

OMNeT++ has support for parallelization through an abstract interface, that is used by the simulation kernel to send and receive messages between segments (partitions of a simulation model) when in distributed mode. The interface implementation allows the use of an arbitrary parallel programming library. By using this design feature, we have added support for MPI alongside existing support for PVM (Parallel Virtual Machine).

4.1 Parallel Simulation Example

We will now illustrate the organization of parallel simulation system through a very simple example. We have done the development on the CTIE's (Center for Telecommunications and Information Engineering, http://www.ctie.monash.edu.au) Linux cluster (which consists of 8 dual-CPU 1 GHz Pentium-III processors connected via a 100 Mb/s Ethernet switch). We have installed the LAM-MPI [LAM-MPI] system on our cluster.

In order to keep the explanation as short as possible, we choose a network of two simulated end-systems ("`tic`" and "`toc`") which send packets to each other via two unidirectional communication channels (See Figure 1). This simulation configuration is described by a text file (a *ned* or *network description* file in OMNeT++ jargon) that identifies the network's nodes and links between them:

```
network TangleNet :
```

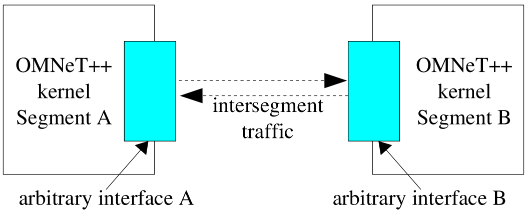

Figure 2: Sending/receiving functions are invoked by the simulation kernel and handled by an arbitrary parallel interface.

```
 TicToc_net on:
  processor1, processor2;
endnetwork

module TicToc_net
  machines:
    processor1, processor2;
    submodules:
      tic: tictoc on: processor1;
      toc: tictoc on: processor2;
    connections:
      tic.out --> toc.in;
      tic.in  <-- toc.out;
endmodule
```

The *ned* file mainly performs two important tasks: (1) provides the network topology, the modules used, and (2) distribution of the topology to the “virtual” processors for execution.

One way of providing the mapping between the virtual processors and real processors can be through a *hostfile*. In this file, the names of the hosts that will participate in the computations are written. For the above example, the hostfile we use contains names of two hosts: `tangles0` and `tangles1`. The packet sources `tic` and `toc` are derived from the simple OMNeT++ module `tictoc` (their dynamic behavior is identical). IN OMNeT++, simple modules are written in C++ by extending the base simulation classes. For details and C++ code of the `tictoc` module, [Şekercioğlu, 2002] can be referred.

Since OMNeT++ supports unlimited levels of nesting of simple modules into compound modules, the partitions of a topology to be distributed into separate computational nodes can be as complex as one would want.

Implementation of PDES Support by using Message Passing Interface (MPI)

OMNeT++ provides the inter-segment communication functionality as an abstract interface (See Figure 2). This interface provides a predefined set of abstract functions for initialization, termination, network setup, and sending/receiving of messages and abstract data types between the segments. The MPI interface simply overrides the abstract functions of the parallel interface to use the functionality provided by MPI.

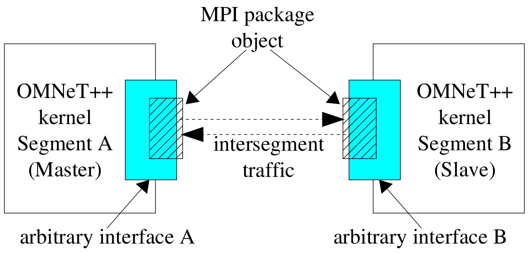

Figure 3: The structure of the MPI - Interface is very similar to that shown in Figure 2, with the exception of additional MPI Package Object.

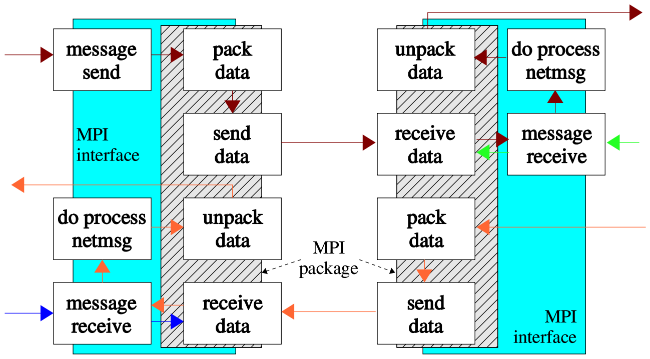

Figure 4: Conceptual message flows for communicating segments in the `TicToc` example.

MPI Interface Implementation

Due to the tedious requirement of maintaining buffers for MPI transmissions, we have implemented the MPI interface with an MPI package class. This class handles all packing/unpacking and sending/receiving functions as well as those required for buffer maintenance (i.e., updating buffer pointers, setting buffer sizes, specifying package sizes etc.)

The MPI package encapsulates the low level MPI calls and aims to reduce redundant calls to MPI where possible e.g. when requesting a new buffer for communication, reuse an existing buffer if it does not exceed its capacity.

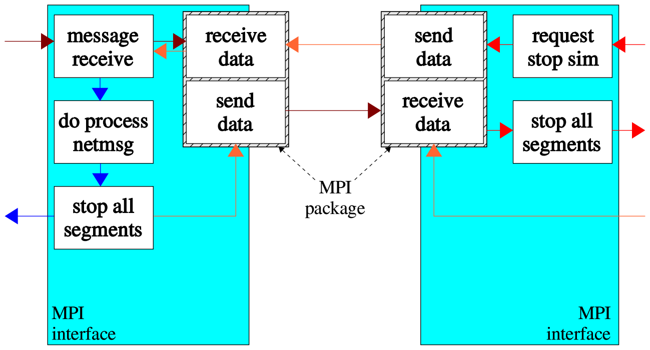

Figure 5: Message flows during simulation termination.

To illustrate the structure of the MPI Interface implementation we revisit the example simulation presented in Section 4.1, where we have two modules “Tic” and “Toc”, on separate segments. Tic initiates the simulation by sending a message to Toc, who then returns a message to Tic and vice-versa until the simulation time limit is reached. Assume that Tic is sitting on the Master segment and Toc is sitting on the Slave segment and that all network initializations and start-up sequences have been completed. Times of interest then begin from $t = 0$ seconds. The sequence of events can be summarized as follows (see also Figures 4 and 5 for message flows):

1. Tic initiates a message send to Toc.
2. Toc blocks and waits for a message from Tic to arrive. This is implemented via a blocking receive function in MPI (the blocking receive is always called in favor of non-blocking receives when the message queue is empty. Since there is only one message moving through this simulation, blocking receive is used. When the message queue is not empty, non-blocking receives are performed).
3. Once the message is received at Toc, it is processed by a `do_process_netmsg()` function inside the simulation kernel. This function is responsible for identifying the type of message according to its MPI message tag.
4. `do_process_netmsgs()` is more or less just a large switch statement handling the different message types. If the message received is a simulation packet, then unpack the contents of the message and hand it over to the actual module.
5. The simulation kernel in the Toc segment then performs some processing and checks if the module has messages to transmit to other modules. In this case Toc just echoes back the message so the simulation kernel determines that the destination module is on another segment and reroutes a normal send to a a network send which is provided by the MPI Package Object and performs the MPI library function call `MPI_Send()`.
6. The last two steps are repeated for n number of iterations. Once the simulation time limit is reached either by the master or the slave, a call to `Stop_All_Segments()` or `Request_Stop_All_Segments()` are called respectively. In the case for `Stop_All_Segments()` call, the master simply sends a termination message to all the slaves. This message contains a text message to be displayed at the console and also carries the appropriate MPI message tag for simulation termination. Otherwise, for the `Request_Stop_All_Segments()` case, a message from the slave is sent to the master first before the previous step is carried out by the master. This message also contains text message to be displayed as well as an appropriate MPI message tag.

Implementation of SSM in MPI

We are now implementing the SSM-T synchronization method using the MPI interface of OMNeT++. Been the simplest of all the synchronisation schemes to implement, SSM does not require insertions of syncpoints between segment boundaries that change as we experiment with different partitioning configurations when attempting to find the optimal one.

Later, we will implement the conservative approach and compare the performance between the two and the amount of error SSM introduces. We aim to find the best variables to be used in SSM to minimize the statistical error and then increase the number of nodes in a full scale simulation of a randomly generated internetwork topology.

Concluding Remarks

In this paper, we presented the general overview of our Message Passing Interface (MPI) based Object-Oriented parallel discrete event simulation framework. We have initiated this project for simulating and modeling very large scale telecommunication networks on shared-memory and distributed-memory parallel processing systems. Currently, we focus our attention in the following three research projects:

- Automatic generation of realistic network topologies and *ned* files.
- Optimal topology partitioning for minimizing the communication among the distributed network segments.
- Selection of the best synchronization method suitable for the architecture of the underlying parallel computing hardware.

We will be reporting the progress of our work in future papers.

Acknowledgment

This work is partially supported through a Victorian Partnership for Advanced Computing (www.vpac.org) expertise grant.

References

R. L. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, and H. Y. Song. Parsec: A parallel simulation environment for complex systems. *IEEE Computer*, pages 77–85, October 1998.

R. L. Bagrodia and M. Takai. Performance evaluation of conservative algorithms in parallel simulation languages. *IEEE Transactions on Parallel and Distributed Systems*, 11(4):395–414, 2000. URL citeseer.nj.nec.com/bagrodia98performance.html.

Y. A. Şekercioğlu. Introduction to network simulation using OMNeT++. Technical Report CTIE-TR-2002-004, Centre for Telecommunications and Information Engineering, Monash University, 2002.

R. M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, October 1990.

U. Kaage, V. Kahmann, and F. Jondral. An OMNeT++ TCP model. In *Proceedings of the European Simulation Multiconference (ESM'2001)* Soc [2001].

J. Lai, E. Wu, A. Varga, Y. A. Şekercioğlu, and G. K. Egan. A simulation suite for accurate modeling of IPv6 protocols. In *Proceedings of the 2nd International OMNeT++ Workshop*, pages 2–22, Berlin, Germany, January 2002.

LAM-MPI. LAM (Local Area Multicomputer): MPI programming environment and development system for heterogeneous computers on a network. URL reference: http://www.lam-mpi.org.

G. Lencse. Efficient parallel simulation with the Statistical Synchronization Method. In *Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation (CNDS'98)*, pages 3–8, San Diego, CA, USA, January 1998.

G. Lencse. Parallel simulation with OMNeT++ using the Statistical Synchronisation Method. In *Proceedings of the 2nd International OMNeT++ Workshop*, pages 24–32, Berlin, Germany, Jan 2002.

MPI. MPI: A message-passing interface standard. *International Journal of Supercomputer Applications*, 8(3/4):165–414, 1994. Message Passing Interface Forum.

The University of Southern California. The Network Simulator - ns-2. URL reference: http://www.isi.edu/nsnam/ns/. URL http://www.isi.edu/nsnam/ns/.

OMNeT++. OMNeT++ object-oriented discrete event simulation system. URL reference: http://www.hit.bme.hu/phd/vargaa/omnetpp.htm, 1996.

OPNET. OPNET Modeler. URL reference: http://www.opnet.com. OPNET Technologies.

G. Pongor. Statistical synchronization: a different approach for parallel discrete event simulation. In *Proceedings of the 4th European Simulation Symposium (ESS'92)*, pages 125–129, Dresden, Germany, November 1992. The Society for Modeling and Simulation International (SCS).

Proceedings of the European Simulation Multiconference (ESM'2001), Prague, Czech Republic, June 2001. The Society for Modeling and Simulation International (SCS).

A. Varga. *OMNeT++ User Manual*. Department of Telecommunications, Technical University of Budapest, 1997. URL reference: ftp://ftp.hit.bme.hu/sys/anonftp/omnetpp/doc/usman.pdf.

A. Varga. The OMNeT++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference (ESM'2001)* Soc [2001].

A. Varga and B. Fakhamzadeh. The K-Split algorithm for the PDF approximation of multi-dimensional empirical distributions without storing observations. In *Proceedings of the 9th European Simulation Symposium (ESS'97)*, pages 94–98, Passau, Germany, October 1997. The Society for Modeling and Simulation International (SCS).

K. Wehrle, J. Reber, and V. Kahmann. A simulation suite for internet nodes with the ability to integrate arbitrary quality of service behavior. In *Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS'2001)*, Phoenix, Arizona, USA, January 2001.