Parallel Simulation with OMNeT++ using the Statistical Synchronization Method

Gábor Lencse

Széchenyi István University, Department of Telecommunications
Hédervári u. 3. H-9026 Győr, Hungary
Lencse@szif.hu
http://www.hit.bme.hu/phd/lencse

Abstract. The synchronization methods for parallel discrete event simulation (PDES) are compared. The original Statistical Synchronization Method (SSM) and its time driven version (SSM-T) are described. The PDES support of OMNeT++ is explained. A case study is given about the parallel simulation with OMNeT++: the simulation of two interconnected FDDI rings executed on two processors. The presentation contains also a demonstration of this parallel simulation at the workshop.

Introduction

Parallel discrete event simulation (PDES) could be an important tool for the performance analysis of communication systems. The precise simulation of these large and complex systems would really benefit from computing power of parallel systems. However PDES is a difficult problem. Fujimoto [1] describes the well-known synchronization methods for PDES. The aim of the synchronization methods is to deal with the problem of causality.

The conservative method avoids the causality errors, by executing only *safe* events. (For a segment S_1 an event E_1 with timestamp T_1 is safe if S_1 can determine that it is impossible for it to receive an event with timestamp smaller than T_1.) Mechanisms were developed for either deadlock avoidance or deadlock detection and recovery. The main problem of the conservative method is that it can produce good speed-up only for systems that meet special criteria, otherwise the method cannot exploit the possible parallelism.

The optimistic method allows the causality errors. Then it detects and recovers from them. There are several variants of the optimistic method offering different solutions. One of the most popular is Time Warp. It performs periodic state saving. If the state-saving overhead is large it may seriously degrade the performance of the simulation. Dynamic memory allocation of the processes causes further complications.

Both of the synchronization methods mentioned above have their own limitations and require quite much support from the simulation kernel. The Statistical Synchronization Method [2] is a promising alternative. SSM does not exchange individual messages between the segments but rather the statistical characteristics of the message

flow. Actual messages are regenerated from the statistics at the receiving side. It has numerous advantages over the two other methods.

This paper first briefly describes the Statistical Synchronization Method, than the parallel simulation functionalities of OMNeT++ and finally a case study: parallel simulation of two interconnected FDDI rings.

1 The Statistical Synchronization Method

In its original form, SSM was invented by György Pongor [2] at the Lappeenranta University of Technology, Finland. It was further developed (as SSM-T) by Gábor Lencse [3] at the Technical University of Budapest, Hungary.

1.1 The Original SSM

Similarly to other parallel discrete event simulation methods, the model to be simulated – which is more or less a precise representation of a real system – is divided into segments, where the segments usually describe the behavior of functional units of the real system. The communication of the segments can be represented by sending and receiving various messages. For SSM, each segment is equipped with one or more input and output interface. The messages generated in a given segment and processed in another segment are not transmitted there but the *output interfaces* (OIF) collect statistical data of them. *The input interfaces* (IIF) generate messages for the segments according to the statistical characteristics of the messages collected by the proper output interfaces. (See *Fig. 1.)*

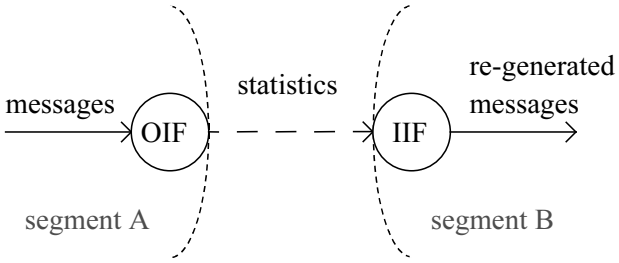

Fig. 1. An OIF – IIF pair

The segments with their input and output interfaces can be simulated separately on separate processors, giving statistically correct results. The events in one segment do not have the same effect in other segments as in the original model, so the results collected during SSM are not exact. The precision depends on the partitioning of the model, on the accuracy of statistics collection and regeneration, and on the frequency of the statistics exchange among the processors.

SSM has the following advantages compared to the other PDES synchronization methods:

- requires less network bandwidth
- tolerates communication delay better
- can be easily implemented
- requires less support from the simulation kernel
- may produce better speed-up

1.2 The Definition of SSM-T

In its original form, SSM was applicable for the analysis of steady state behavior of systems. In another paper [4] Dr Pongor emphasizes the advantages of the fact that the local virtual time (LVT) of the segments may be completely different, this feature may result in automatic importance sampling and super optimal speed-up.

However, an approximate synchronism of the LVTs of the segments is often desirable. For example the load of communication systems is not constant, but changes during the day according to a hat-like curve. A simulator should be able to follow this behavior. At the Department of Telecommunications, Technical University of Budapest, we further developed SSM to have this property. This version is distinguished as SSM-T, the time driven version of SSM. We also completed OMNeT++ with the necessary functionality for parallel execution.

The basic idea of SSM-T is very simple. Let the execution of the segments run independently in the majority of time facilitating good speed-up and let the LVTs of the segments meet at certain points of time ensuring an approximate synchronism.

Loose synchronization between segment A and segment B is defined formally as follows:

Let t_1, t_2, t_3, ... t_i, ... t_n be synchronization points of time. Let t_A and t_B denote the LVTs of segment A and segment B, respectively. Segment A and segment B are loosely synchronized if:

$$((t_A < t_i) \Rightarrow (t_B \leq t_i)) \wedge ((t_A > t_i) \Rightarrow (t_B \geq t_i)), \quad i = 1, 2, \ldots, n. \qquad (1)$$

The loose synchronization of the two segments means that none of them may leave any synchronization point of time (t_i) until the other one reached it.

At the t_i synchronization point of time segments A and B may exchange the statistics they have collected before t_i and they may use the new ones in the $[t_i, t_{i+1}]$ time interval. With the appropriate choice of the t_i synchronization point of time, it is ensured that the effect of a change in segment A in t_x will reach segment B earliest at t_x and latest at $t_x+\Delta t$. In the simplest case, we use loose synchronization where $t_i=i*UI$, UI being the update interval.

What does this method require from the simulation kernel? Let us consider the following example. If *segment A* wants to send a message to *segment B* at LVT t_i then segment A should ask the simulator not to let the LVT of segment B pass t_i until segment B receives a message from segment A. (For simplicity and clarity we consider only one direction of communication as the other direction can be handled in the same way. If both segments ask for synchronization for the same LVT, then the LVTs of the two segments will really meet.) As for the implementation, if segment A sends a

synchronization point request to segment B for LVT t_i than segment B schedules a self message (a special event, that will be processed by the simulation kernel itself) for LVT t_i. If the statistics package from segment A arrives to segment B at LVT $t_x<t_i$ than the message carrying the statistics is simply scheduled to t_i. If the statistics package from segment A to segment B does not arrive until t_i, segment B processes the self message and suspends processing of any further events until the statistics package arrives. Fig. 2. shows examples for both cases.

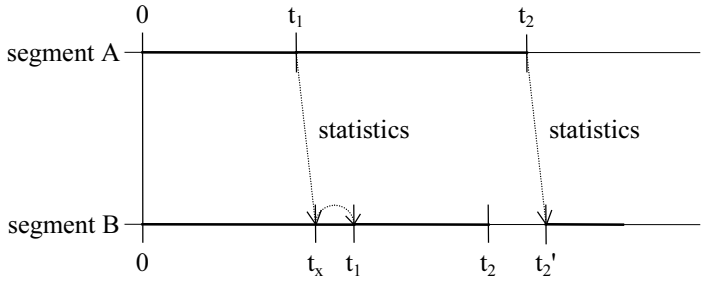

Fig. 2. The operation of SSM-T. The thin horizontal lines show the wall-clock (real) time of the processors executing the segments and the thick lines are the virtual times of the segments

1.3 Further Investigations on SSM-T

Because of space limitations we cannot cite all the results on SSM-T, instead we briefly summarize the topics covered in previous papers. All the cited papers can be downloaded from the author's web page.

Accuracy of SSM is dealt with in [5]. Different statistics collection algorithms were examined concerning their resource requirements and accuracy.

The applicability criteria of SSM are given in [6] together with examples when the method can or cannot be used.

The Statistics Exchange Control Algorithm [7] is responsible for determining the appropriate virtual time for the next statistics exchange. Its task is not at all trivial, as the required accuracy gives the number of observations required for the statistics, and the statistics exchange control algorithm must meet the correct virtual time. If the prediction fails, the synchronization point is deleted and the statistics collection must be continued. The effects of the different kinds of deviations from the optimal case were examined.

2 Implementing Parallelism in OMNeT++

OMNeT++ can run parallel on different types of hosts and/or operating systems that support PVM.

2.1 Parallel Topology Description

OMNET++ parallel support includes the parallel topology description, a flexible method, that enables the user to describe the partitioning of the model. When defining a module type, a compound module (besides the formal parameter list and the submodule list) may have a *formal machine list* introduced by the keyword `machines`. When defining the inside structure of the compound module, the user can express onto which machine (from the formal machine list) he wants to place the submodules: the keyword `on` is followed by the machine. Let us see an example:

```
module Network
    machines:
        host4netA, host4netB;
    submodules:
        netA: subnetwork;
            on: host4netA;
        netB: subnetwork;
            on: host4netB;
    connections:
        netA.out --> netB.in;
        netB.out --> netA.in;
endmodule
```

When building the network topology, the simulation kernel evaluates the machine parameters and places the modules into a single segment (process) per host. Especially for testing/debugging purposes it is possible to place all the segments in separate processes onto one host, SINGLE_HOST has to be defined in the `pvmmod.cc` source file.

2.2 Communication Between the Segments

In OMNeT++, the modules communicate with each other by sending and receiving various messages. The messages may contain arbitrarily complex data structures. Basically, the user does not have to take care if the modules communicating with each other are placed into the same segment or into different ones. If communication occurs between two modules that are in different segments, OMNeT++ packs all the standard data structures – that are included in the message – into PVM messages and carries them safely to the destination. Of course there are some natural restrictions:

1. Pointers became meaningless if they are transferred from one process to another

2. The conversion of the different architectures is done interpreting data as their types are defined. For example the order of the 4 bytes of an integer is reversed if it is transferred between a little-endian and a big-endian computer. The dirty trick of accessing the 4 bytes of this integer as `char` type variables will not work well.
3. The range of the possible values of data types may differ. For example the storage size of a variable of C++ type `long` is 4 or 8 bytes on a 32 or 64 bit architecture computer.

2.3 Synchronization Between the Segments

OMNeT++ provides a very simple mechanism for the inter-segment synchronization. The user may send a `synchpoint(time)` from one segment to another. The target segment's LVT may not pass `time` (and its execution is suspended if it has no more events with less or equal timestamp than `time`) until a message from the source segment arrives. The first synchronization points are sent at the beginning of the simulation and the user must take care to send the next synchronization point always *before* he sends the expected message that deletes the actual synchronization point. This mechanism can be used for either conservative or statistical synchronization. The latter is supported by different statistics collecting classes, such as histograms, P^2, K-split [8].

3 Parallel Simulation of Two Interconnected FDDI Rings

We would like to demonstrate the parallel simulation with OMNeT++ on the example of an FDDI network. In 1996, the backbone of the Technical University of Budapest consisted of two rings: The Northern Ring was a university-wide network and consisted of 15 FDDI stations interconnected by 5 wiring concentrators. The Southern Ring was the backbone of the Faculty of Electrical Engineering and Informatics, and being a smaller ring consisted only of 7 FDDI stations. Figure 3. shows the topology of the network. The two rings are interconnected by the *bmecisco7* router. The topology of the network and the cable lengths were taken from the real system.

The load used in the simulation model came from measurements taken on the real rings. By using a protocol analyzer, the first 32 octets of all the packets were copied from the ring and the packet lengths, arrival times, as well as the addresses of the source and destination stations were stored.

The natural segmentation of the network is to place each ring into its own segment. The bmecisco7 router is a part of both rings and is cut into two in the middle. The statistical interfaces are inserted between the two rings, each segment has one input and one output interface, as the data flow between the segments is bi-directional. The NeD description of the network looks as follows (the parameters were removed to save space):

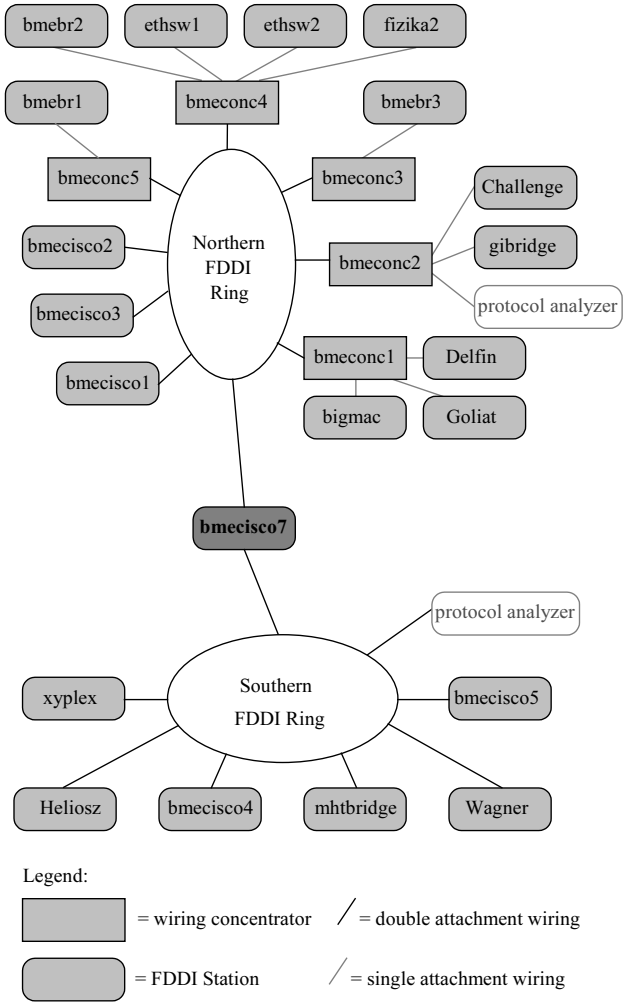

Fig. 3. The FDDI backbone of the Technical University of Budapest in 1996

```
module TUB_SSM
  machines:
    NR_host, SR_host;
  submodules:
    NRing: TUBNRing;
      on: NR_host;
    SRing: TUBSRing;
      on: SR_host;
    NRing_oif: SSM_OIF_type like SSM_OIF;
      on: NR_host;
    NRing_iif: SSM_IIF;
```

```
      on: NR_host;
    SRing_oif: SSM_OIF_type like SSM_OIF;
      on: SR_host;
    SRing_iif: SSM_IIF;
      on: SR_host;
  connections:
    NRing.out --> NRing_oif.in;
    NRing_oif.out --> delay 0.05 us --> SRing_iif.in;
    SRing_iif.out --> SRing.in;
    SRing.out --> SRing_oif.in;
    SRing_oif.out --> delay 0.05 us --> NRing_iif.in;
    NRing_iif.out --> NRing.in;
endmodule
```

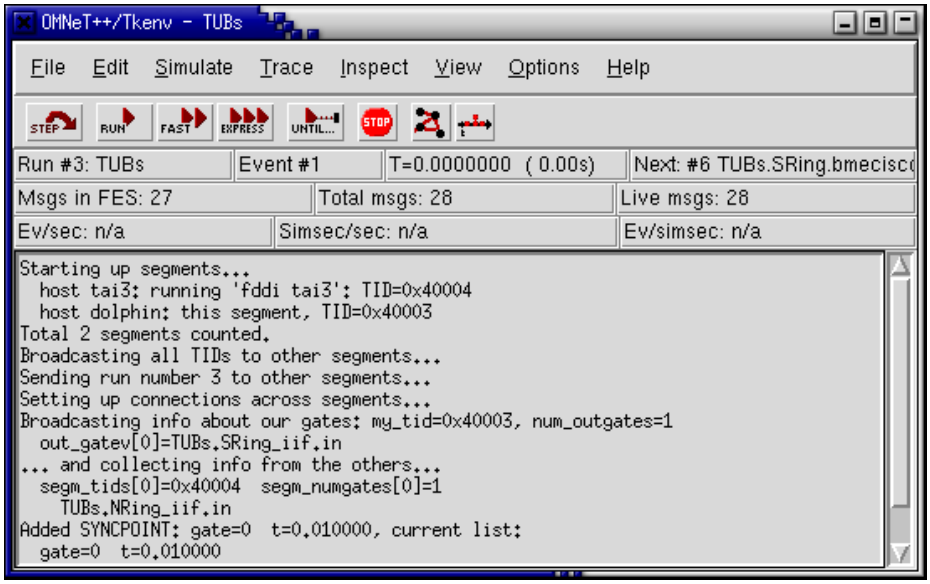

Fig. 4. The main window of OMNeT++

The demonstration of the parallel simulation runs on two notebooks, interconnected via 10BaseT Ethernet. The PVM version is 3.4.4, OMNeT++ the currently available 2.1 distribution [9]. The complete FDDI model can be found among the samples. The OMNeT++ manual gives a detailed description about how to configure PVM, and about getting the FDDI model run parallel. Despite the fact, that the FDDI simulation was written in 1997, before the public release of OMNeT++ the code still compiles and even works. (Thanks to András Varga, who always updated it when the simulator was changed.) For those, who would like to repeat the experiment: the above statements are true only if SINGLE_HOST is *not* defined. If it is defined the

2.1 distribution `src/sim/pvm/pvmmod.cc` file will not compile, but needs about 10 minutes hacking… Bugfix is planned to be provided soon.

4 Summary

We have shown, that SSM-T is a good solution for PDES synchronization. All the necessary functionality is already included in OMNeT++. So now we are looking for volunteers, who test the method for real life applications.

References

1. Fujimoto, R. M.: "Parallel Discrete Event Simulation". *Communications of the ACM* 33, (1990.) no 10, pp. 31-53
2. Pongor, Gy.: "Statistical Synchronization: a Different Approach of Parallel Discrete Event Simulation". *Proceedings of the 1992 European Simulation Symposium (ESS'92)* (Dresden, Germany. Nov. 5-8) SCS Europe, pp. 125-129.
3. Lencse, G.: "Efficient Parallel Simulation with the Statistical Synchronization Method" *Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation (CNDS'98)* (San Diego, CA. Jan. 11-14). SCS International, pp. 3-8.
4. Pongor, Gy.: "Multiple Virtual Times in Parallel Discrete Event Simulation". *Proceedings of the Parallel Processing Workshop* (Technical Univ. of Budapest, Budapest, Hungary, Febr. 10-11, 1994.)
5. Lencse, G.: "Statistics Collection for the Statistical Synchronisation Method" *Proceedings of the 1998 European Simulation Symposium (ESS'98)* (Nottingham, UK. Oct. 26-28). SCS Europe, pp. 46-51.
6. Lencse, G.: "Applicability Criteria of the Statistical Synchronization Method" *Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation (CNDS'99)* (San Francisco, CA. Jan. 17-20). SCS International, pp. 159-164.
7. Lencse, G.: "Design Criterion for the Statistics Exchange Control Algorithm used in the Statistical Synchronization Method" *Proceedings of the Advanced Simulation Technologies Conference (ASTC 1999)* part of the 32nd Annual Simulation Symposium (San Diego, CA. April 11-15) SCS International, pp. 138-144.
8. Varga, A.: "K-split – On-Line Density Estimation for Simulation Result Collection". *Proceedings of the 1998 European Simulation Symposium (ESS 98)* (Nottingham, UK. Oct. 26-28,.). SCS Europe, 41-45.
9. Varga, A.: "OMNeT++ Discrete Event Simulation System" (2001) http://www.hit.bme.hu/phd/vargaa/omnetpp.htm