PERFORMANCE PREDICTION OF CONSERVATIVE PARALLEL DISCRETE EVENT SIMULATION

Gábor Lencse
Széchenyi István University
Egyetem tér 1.
H-9026 Győr, Hungary
lencse@sze.hu

András Varga
Opensim Ltd.
Szőlő köz 11.
H-1032 Budapest, Hungary
andras@omnetpp.org

KEYWORDS

performance analysis, parallel discrete event simulation, conservative synchronisation method, speed-up, OMNeT++

ABSTRACT

In a previous paper, a quantitative criterion has been proposed for efficient execution of the Null Message Protocol, the best- known conservative parallel discrete event simulation (PDES) protocol. The criterion is based on a novel concept of the coupling factor, and allows one to use intuitive and easy-to-measure input parameters. The criterion can be used to assess simulation models' potential for parallel execution as well as the maximum partitioning that may still potentially yield good performance.

This paper deals with experimental verification of the proposed rule that a large coupling factor is a necessary precondition for getting a good speed-up with conservative parallel simulation. A closed queueing network is used as simulation model, and it is run on up to 24 CPU cores. It is shown that the criterion provides a quick and convenient way to determine whether a simulation model has a potential for speed-up before actually investing work in the parallelization.

INTRODUCTION

Because of the rapid development and growth of information and communication technology (ICT) systems in the last decades, the analysis of their performance is an important issue. Event-driven ***discrete event simulation*** (DES) is a powerful tool for this task. Nevertheless, the systems are often so large and complex and the number of events is so high that the execution requires unacceptably long time even using a supercomputer. Let us review first the solutions that were proposed for this problem so far.

Parallelisation can be a natural solution. However, it is not an easy task and the achievable speed-up is often limited. The reason is the algorithm of the event-driven DES. When doing ***parallel discrete event simulation*** (PDES), the model of the system is divided into partitions (called Logical Processes), and the partitions are assigned to processors that are executing them. To maintain causality, the virtual times of the partitions must be synchronised. There are three different methods for synchronisation. The first two are described in (Fujimoto 1990).

The ***conservative method*** ensures that causality is never violated. An event can be executed only if we are certain that no events with smaller timestamp exist (and also will not be generated) anywhere in the model. Unless the simulated system has a special property that the so called *lookahead* is large enough, the processors executing the partitions need to wait for each other in the majority of time, so the achievable speed-up is poor.

The ***optimistic method*** allows and detects causality errors and uses *roll-backs* to recover from them. However, there are some problems associated with roll-backs. First, the ability of the roll-backs requires extra functionality (e.g. state saving and restoration) from both the simulation kernel and the models[1]. Even though a simulator may support optimistic parallel execution, model authors may not be willing to invest the extra work in the model building that is needed to prepare the model for the optimistic method. Second, above a certain number of processors the resource consumption of the roll-backs may hinder the good speed-up and the algorithm may not scale well. Third, state saving and restoration itself has CPU and memory costs, decreasing the gain from parallel execution.

The ***statistical synchronisation method*** (SSM) has been proposed by György Pongor (Pongor 1992). SSM does not exchange individual messages between the partitions, but rather the statistical characteristics of the message flow. In its original form, SSM was applicable for the analysis of steady state behaviour of systems. It was further developed (as SSM-T) by Gábor Lencse (Lencse 1998). The method can produce excellent speed-up, but has a limited area of application (Lencse 1999). For more information about the three methods, see (Lencse 2002) and its references.

In the paper (Varga et al. 2003), the authors propose a method for assessing available parallelism in a simulation model for conservative synchronization. The method requires only a small number of parameters that can be easily measured on a sequential simulation.

[1] Assuming that C or C++, the usual languages for high-performance simulations, are used to implement the models. Models written in idiomatic C/C++ cannot be automatically save/restored by the simulation kernel.

In this paper, we intend to check the results of the aforementioned work for a higher number of processors, and also examine how different parameters of the model influence achievable speed-up.

The remainder of this paper is organised as follows: first, we briefly summarize the method for assessing the available parallelism: we describe the necessary parameters and present the method; then we describe the hardware and software environment of our experiments; next, we repeat some of the experiments of Varga et al. for higher number of processors; then we examine how the speed-up depends on some of the parameters; finally, we present our conclusions.

This topic was identified as being of importance in the parallel simulation of large systems.

ASSESSING AVAILABLE PARALLELISM

The paper (Varga et. al. 2003) uses the notations *ev* for events, *sec* for real world time in seconds and *simsec* for simulated time (model time) in seconds. The paper uses the following quantities for the assessing of available parallelism:

- *P performance* represents the number of events processed per second (*ev/sec*). *P* depends on the performance of the hardware and the amount of computation required for processing an event. *P* is independent of the size of the model.

- *E event density* is the number of events that occur per simulated second (*ev/simsec*). *E* depends on the model only, and not on the hardware and software environment used to execute the model. *E* is determined by the size, the detail level and also the nature of the simulated system.

- *R relative speed* measures the simulation time advancement per second (*simsec/sec*).
  Note that $R = P/E$.

- *L lookahead* is measured in simulated seconds (*simsec*). When simulating telecommunication networks and using link delays as lookahead, *L* is typically in the *microsimsec–millisimsec* range.

- *τ latency* (sec) is the latency of sending a message from one Logical Process (LP) to another. This value is usually in the µs-ms range, and is largely determined by the hardware and software on which the simulation runs.

- *λ coupling factor* can be calculated as the ratio of *LE* and *τP:*

$$\lambda = \frac{L \cdot E}{\tau \cdot P} \qquad (1)$$

The paper (Varga et. al. 2003) states that the chance of the good speed-up of the PDES using the *conservative synchronisation method* can be predicted on the basis of the magnitude of *λ*. The work supports its theoretical result by experiments performed on clusters of 2 or 4 (single core) CPUs. The aim of this paper is to verify those results by experiments performed on larger clusters up to 12 dual core CPUs (that is, 24 cores).

HARDWARE AND SOFTWARE ENVIRONMENT

For our experiments, we used a cluster of 12 PCs with *AMD Athlon 64 X2 Dual Core 4200+* processor, *2*1GB DDR2 667MHz* (dual channel) RAM and *NVIDIA nForce® 500 SLI™ MCP built-in Gigabit Ethernet* NIC. The hosts were interconnected by a *3Com 2948-SFP* Gigabit Ethernet switch.

The hosts were running Debian Squeeze GNU/Linux operating system and LAM/MPI 7.2.1 cluster software.

The communication latency between the hosts (the PCs of the cluster) over MPI was about *25µs.*

We have created user accounts called "mpi" on all the hosts. Its home directory was stored on a *SUN Fire X4200 M2* NFS server connected by the same Gigabit Ethernet switch.

We used private IP addresses, and there was no other load on the cluster and NFS server during the experiments.

We used OMNeT++ version 4.0p1.

THE SIMULATION MODEL

For the experiments, we used the Parallel CQN (Closed Queueing Network) simulation sample program of OMNeT++ – the same model that the original work used. This model consists of *M* tandem queues where each tandem consists of a switch and *k* single-server queues with exponential service times (Figure 1).

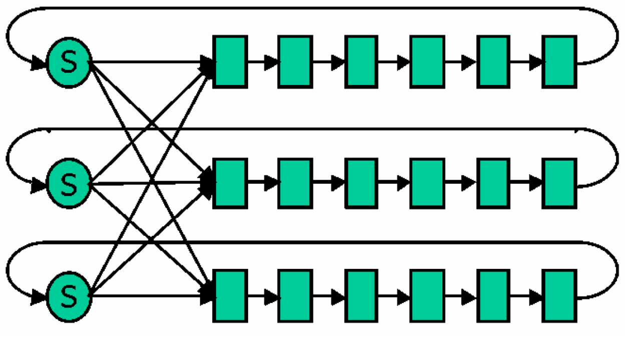

Figure 1. *M*=3 Tandem Queues with *k*=6 Single Server Queues in Each Tandem Queue

The last queues are looped back to their switches. Each switch randomly chooses the first queue of one of the tandems as destination, using uniform distribution. The queues and switches are connected with links that have nonzero propagation delays. The OMNeT++ model for CQN wraps tandems into compound modules.

To run the model in parallel, we assign tandems to different LPs (Figure 2.). Lookahead is provided by delays on the marked links.

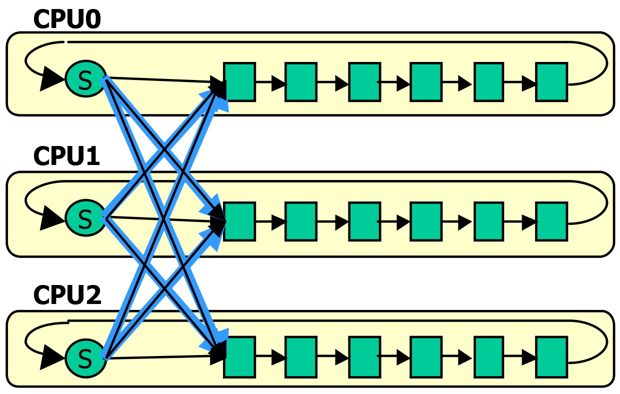

Figure 2. Partitioning the CQN Model

As for the parameters of the model, we have used the preset values shipped with the model. We chose configuration B, the one that promised good speed-up. The main parameters were: M=24 tandem queues, k=50 queues in each tandem queue, exponential service time of the queues with expected value of 10 seconds. As for the delay between the tandem queues, first we used the preset value of L=100 seconds, and later we used this parameter to achieve the values of λ, which we were interested in.

THE EXPERIMENTS

First, we ran a single processor simulation with L=100s to measure the E and P variables necessary for the calculation of λ. It is a very convenient feature of the OMNeT++ simulator that it displays these variables both in its GUI and command line user interfaces. We used the latter for the measurement, because at the time of the parallel simulation we needed to use the command line environment to facilitate convenient experimenting. For the value of λ, we got:

$$\lambda = \frac{L \cdot E}{\tau \cdot P} = \frac{100 \cdot 156}{25 \cdot 10^{-6} \cdot 250000} \approx 2500 \qquad (2)$$

This is an appropriate value to expect good speed-up according to (Varga et. al. 2003). The value of λ decreases with the number of LPs. If we use N number of LPs, then:

$$\lambda_N = \frac{\lambda}{N} \qquad (3)$$

We performed a series of experiments for the following values of N: 1, 2, 4, 6, 8, 12 and 24. N=1 means that there was only a single LP. From 2 to 24, we included N/2 number of dual core hosts in the MPI virtual machine, each running 2 LP's. Each LP contained M/N tandem queues. (E.g. N=6 means that there were 6 LP's executed by the 6 cores of 3 hosts, and each LP contained 4 tandem queues.)

With a few exceptions, we executed every experiment n=11 times, and calculated average and standard deviation for the t_i execution times.

Initial Results

Our first results are shown in Table 1.

Table 1. Execution Time and Speed-up in the Function of the Number of Logical Processes with L=100s Lookahead

number of LP's	*1*	*2*	*4*	*6*	*8*	*12*	*24*
avg. ex. time [sec]	516.55	291.27	140.00	94.09	72.27	50.18	33.91
exec time std. dev.	5.47	11.31	7.46	1.14	1.62	0.60	0.30
speed-up	1.00	1.77	3.69	5.49	7.15	10.29	15.23
relative speed-up	1.00	0.89	0.92	0.91	0.89	0.86	0.63

Note that except the case of N=24 LPs, the value of λ_N is always higher than 200, and there is an excellent speed-up. λ_{24}=104 corresponds to an acceptable but significantly lower speed-up (the relative speed-up is only 0.63).

To examine the effect of λ being one order of magnitude smaller, we carried out the same series of experiments for L=10s, too.

Table 2. Execution Time and Speed-up in the Function of the Number of Logical Processes with L=10s Lookahead

number of LP's	*1*	*2*	*4*	*6*	*8*	*12*	*24*
avg. ex. time [sec]	523.09	307.64	157.73	114.64	99.27	91.45	119.27
exec time std.	11.78	20.72	1.19	.9633	2.57	0.69	0.79
speed-up	1.00	1.70	3.32	4.56	5.27	5.72	4.39
relative speed-up	1.00	0.85	0.83	0.76	0.66	0.48	0.18

Table 2 shows the results for an L lookahead 10 times smaller than in Table 1. The changed lookahead also means that the values of λ_N are also approximately[2] 10 times smaller. For N=4 LPs, the calculated value of λ_4 =2500/10/4=62.5 still results in a 3.32 times speed-up and a relative speed-up of 0.83. For the values of N=6, 8 and 12, the relative speed-up is visibly degrading, and for N=24 it becomes poor: 0.18. (The execution time *increases* if we use 24 LPs instead of 12 LPs.)

Vacationing Jobs and their Influence on λ

Given that our simulation model is relatively simple, it would be tempting to think that its performance characteristics (P and E) do not depend on the L lookahead, and so λ is directly proportional to L. However, this assumption is incorrect due to the phenomenon of "vacationing jobs", explained below. The consequence is that λ must be calculated independently from P and E measured for each value of L.

[2] But not exactly, see explanation later.

If we compare the execution times of the sequential simulations we can observe a slight difference: it is 523s and 517s for the lookahead values of 10s and 100s, respectively. The difference is even more radical for *L*=1000s; then the execution time is 416s. The explanation is that as *L* increases, a higher proportion of the jobs will be "buffered" in the long-delay links among the tandems, that is, they are effectively removed from the queueing inside the tandems. This phenomenon results in a lower load for higher values of *L*. Let us refer to these jobs as "vacationing" jobs.

Our model is a closed queueing network, so the number of the jobs in the system is constant; in our model this number is always 2400, as there are 24 tandems, 50 queues in all tandems, and initially 2 jobs are placed in each queue. When the switch makes a decision about a job, with the probability of 23/24≈95.8% the job will be sent to a different tandem, that is, the job will be vacationing for *L* simulated seconds before it arrives at the first queue of the destination tandem. A tandem consists of *k*=50 queues, and each of them has an exponential service time with a mean of 10s and an output latency of 1s (modelled as link delay between adjacent queues). As a rough estimation, we can say that at the beginning of the simulation (when the jobs are distributed evenly and there are no vacationing jobs) the complete travel of a job through the tandem will last:

$$50*(10s+10s+1s)=1050s \quad (4)$$

This is an estimated service time of the tandem, valid only at the beginning of the simulation. For *L*=100s and 1000s, the vacationing time is comparable to the roughly estimated service time of the tandem. As a very rough[3] approximation, we can say that 10% of the jobs are vacationing for the *L*=100s lookahead. Because of the capabilities of the OMNeT++ simulator, the number of vacationing jobs can be easily determined. The jobs are represented by messages in OMNeT++, and the messages representing vacationing jobs are residing in the Future Event Set (FES). The number of the events in the FES is displayed in the GUI of the simulator, and even the contents of the FES can be interactively examined during the simulation. We have found that on average, 245 messages representing jobs (≈10%) were present in the FES in the case of the sequential simulation with *L*=100s lookahead.

The consequence of vacationing jobs is that the value of λ cannot be calculated for the different lookahead values from the parameters that we measured with *L*=100s lookahead, but the parameters should be measured for all the different lookahead values.

[3] The vacationing events are missing from the tandems, so the service time of the tandems will be less, than it was in the beginning of the simulation.

We measured the value of λ for the different values of *L* to see how much the difference is. Now, we did not use samples from the *P* and *E* values displayed by the simulator during the experiments (as we did it for the initial estimation of λ), rather made complete runs and used the measured full event number (*ev*) and execution time (*sec*) for the calculation of *P* and *E*. The elapsed virtual time was always 10 days, that is, 864000 *simsec*. Table 3. shows the measured values, and the λ values calculated (incorrectly) using linear extrapolation from the *L*=100s case. It is apparent that in the *L*=1000s case there is a significant difference between the measured and linearly extrapolated λ values.

Table 3. The Values of λ in the Function of the Lookahead (measured vs. calculated from the initial estimation)

L [simsec]	**0.1**	**1**	**10**	**100**	**1000**
# events	138122606	138091806	137816386	134885378	102957082
exec. t.*[sec]*	524.18	521.36	523.09	516.54	415.73
P [ev/sec]	263502.24	264868.43	263465.92	261132.49	247653.72
E [ev/simsec]	159.86	159.83	159.51	156.12	119.16
meas.'d λ	**2.43**	**24.14**	**242.17**	**2391.39**	**19246.76**
λ_0*L/L_0	2.50	25.00	250.00	2500.00	25000.00

The Effect of the Magnitude of λ on the Available Speed-up

To fully explore the effect of the magnitude of λ on the available speed-up, we conducted a series of experiments for other values of *L: L*=100ms, 1s, 10s, 100s and 1000s. Figures 3-7 show the results.

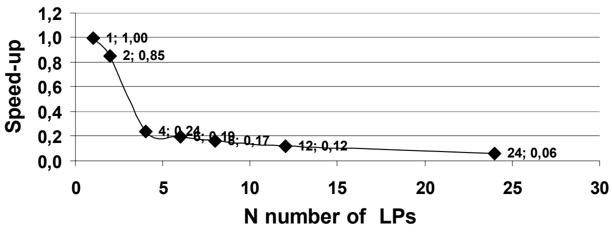

Figure 3. Speed-up in the Function of *N* for *L*=0.1s

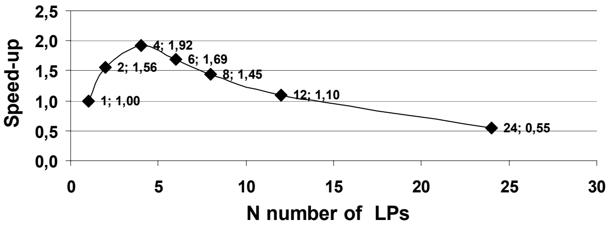

Figure 4. Speed-up in the Function of *N* for *L*=1s

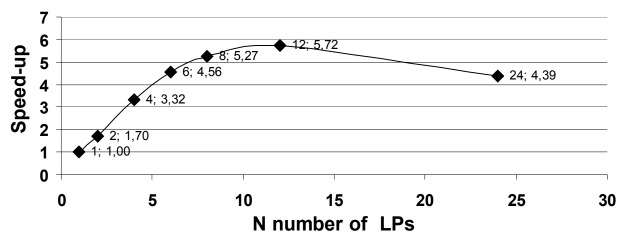

Figure 5. Speed-up in the Function of *N* for *L*=10s

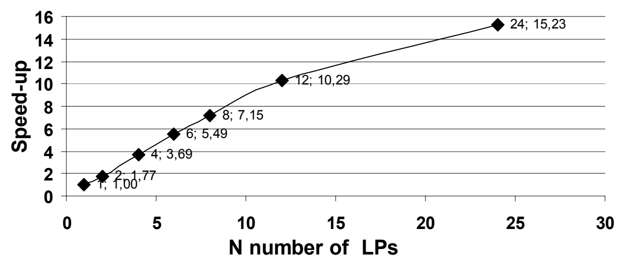

Figure 6. Speed-up in the Function of *N* for *L*=100s

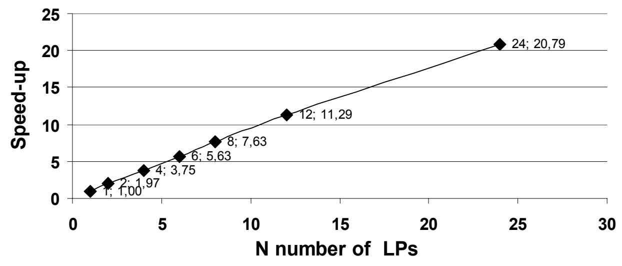

Figure 7. Speed-up in the Function of *N* for *L*=1000s

Results in Figures 3-7 have been summarized in Figure 8, and speed-ups have been translated to relative speed-up (speed-up divided by the number of LPs) for better comparison.

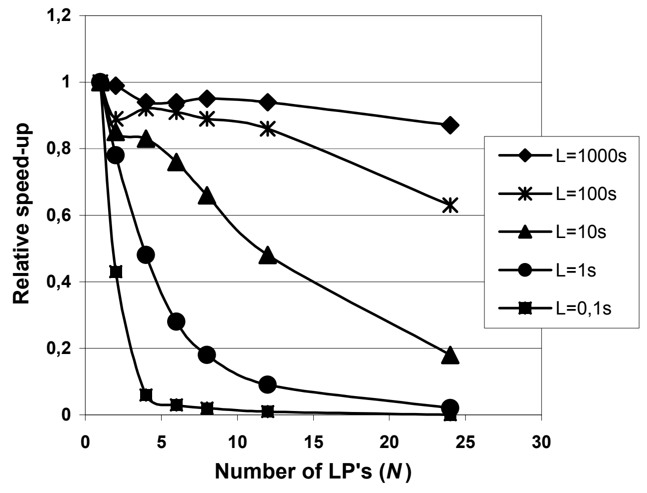

Figure 8. Relative Speed-up in the Function of *N* for *L*=0.1s, 1s, 10s, 100s, 1000s Lookahead

Figure 8 shows the relative speed-up in the function of *N* for these values of *L*. The nearly constant 1 value of the relative speed-up for *L*=1000s means a near-linear speed-up. This is in accordance with the value of λ_{24}=19246.76/24≈802 (taken from Table 3). As for the other extremity, there is no speed-up even for two processors for *L*=0.1s – this is also complies with our expectations as λ_2=2.43/2≈1.21. Halfway (on the logarithmic scale) between them, there is a certain speed-up for *L*=10s, see Figure 5. Here, the measured value of λ_4≈60.68 is practically equal to the previously used estimated one (62.5). For λ_8≈30.34, there is an acceptable speed-up of 5.27, and probably this is the most economical working point of the system, as the further increase of the number of LPs (and CPU cores) does not significantly improve the performance.

Figure 4 shows the speed-up for *L* = 1s lookahead. Here, the highest speed-up of the value 1.92 can be observed when there are 4 LPs executed by 4 cores. Here sequential λ≈24.14 results in λ_4≈6. Now, we are on the theoretical boundaries: λ_{24}≈1 and the simulation in 24 LPs requires nearly twice longer execution time than the sequential simulation.

Figure 9 shows the relative speed-up in the function a λ_N. The data series of a given color belong to a given value of *L*. Note that as λ_N is decreasing in the function of *N*, if we consider the values of *N* in ascending order: (2, 4, 6, 8, 12 and 24) then we need to look at the data series from the right to left.

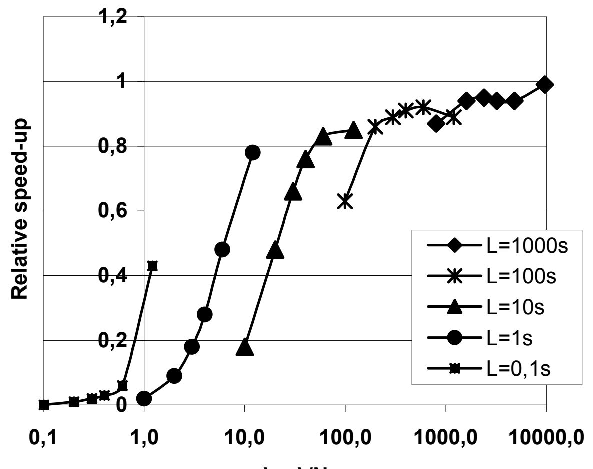

Figure 9. Relative Speed-up in the Function of λ_N for Different Values of *L*, and for *N*=24, 12, 8, 6, 4, 2

On the basis of our experimental results summarized in Figure 9, we can confirm the findings of (Varga et. al. 2003) that:

- If λ is in the order of a couple times 100 or higher, then we may expect good speed-up. It may be nearly linear (that is the relative seed-up is close to 1) even for higher number of LPs (we checked it until 24) if λ_N (that is λ divided by the number of LPs) is also at least in the order of a couple of hundreds.

- If λ is in the order of a couple of times 10, we may experience fair speed-up for a number of processors until λ_N (that is, λ divided by the number of LPs) also remains in this order (or at least λ_N is higher than a couple times 1).

- No speed-up is possible if λ or λ_N (that is, λ divided by the number of LPs) is less than 1.

CONCLUSIONS

In this paper we have experimentally verified that a coupling factor of $\lambda >> 1$ is a necessary precondition of getting a good speed-up with conservative parallel simulation. We have used a closed queueing network as simulation model, and run it on up to 24 CPU cores. The results confirm that with our model, a $\lambda_N = \lambda / N$ (N being the number of LPs) value near or below 1 practically prohibits good parallel performance. The 10..100 range of λ_N can provide an acceptable speed-up, and there is a high chance for a good speed-up if λ_N is above that range.

One has to keep in mind that λ only helps to determine whether a given lookahead is enough for a given simulation model and software/hardware environment, so it cannot guarantee good speed-up. Speed-up can still be adversely affected by other factors such as excessive message traffic between LPs as a result of partitioning.

However, the criterion for λ provides a quick and convenient way to determine whether it makes sense to experiment with parallelizing a particular simulation model or not, before actually investing work in the parallelization.

We conclude that the recommended methods are worth further studying.

ACKNOWLEDGEMENT

The authors of this paper would like to express their thanks to Éva Szabó, who conducted the measurements used in this paper.

REFERENCES

Fujimoto, R. M. 1990. Parallel Discrete Event Simulation. *Communications of the ACM* 33 no. 10, 31-53

Lencse, G. 1998. "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, 3-8.

Lencse, G. 1999. "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, 159-164.

Lencse, G. 2002. "Parallel Simulation with OMNeT++ using the Statistical Synchronization Method" *Proceedings of the 2nd International OMNeT++ Workshop* (Jan. 8-9, 2002, Technical University Berlin, Berlin, Germany) 24-32.

Pongor, Gy. 1992. "Statistical Synchronization: a Different Approach of Parallel Discrete Event Simulation". *Proceedings of the 1992 European Simulation Symposium (ESS'92)* (Nov. 5-8, 1992, The Blockhaus, Dresden, Germany.) SCS Europe, 125-129.

Pongor, Gy. 1992. "Statistical Synchronization: a Different Approach of Parallel Discrete Event Simulation". *Proceedings of the 1992 European Simulation Symposium (ESS'92)* (Nov. 5-8, 1992, The Blockhaus, Dresden, Germany.) SCS Europe, 125-129.

Varga, A., Y. A. Sekercioglu and G. K. Egan. 2003. "A practical efficiency criterion for the null message algorithm". *Proceedings of the European Simulation Symposium (ESS 2003)*, (Oct. 26-29, 2003, Delft, The Netherlands.) SCS International, 81-92.

Sekercioglu, Y. A., Varga, A. and Egan, G. K. Egan, 2003. "Parallel Simulation Made Easy with OMNeT++". *Proceedings of the European Simulation Symposium (ESS 2003)*, Oct. 26-29, 2003, Delft, The Netherlands.

Varga, A. and Hornig, R., 2008. "An overview of the OMNeT++ simulation environment", *Simutools '08: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*. March 7, 2008, Marseille, France.

AUTHOR BIOGRAPHY

GÁBOR LENCSE received his M.Sc. in electrical engineering and computer systems at the Technical University of Budapest in 1994, and his Ph.D. in 2001 from the same university. His area of research is simulation methodology and parallel discrete event simulation. His academic interests include the acceleration of the simulation of information and communication technology systems. Since 1997, he has been affiliated with the Széchenyi István University in Győr, where he lectures on computer networks, networking protocols and Linux as Associate Professor. He is a founding member of the Multidisciplinary Doctoral School of Engineering, Modelling and Development of Infrastructural Systems at the Széchenyi István University. He performs R&D in the field of the simulation of communication systems for the Elassys Consulting Ltd. since 1998. Dr Lencse has been working part time at the Budapest University of Technology and Economics (the former Technical University of Budapest) since 2005, where he teaches computer architectures.

ANDRÁS VARGA received his M.Sc. in electrical engineering and computer systems from the Technical University of Budapest in 1994. Between 1994 and 1998 he pursued PhD studies at the same university, but went to the industry before completing the PhD. Between 2003 and 2005, he spent altogether more than a year at CTIE, Monash University, Melbourne as a visiting research fellow, where he was performing research on parallel simulation of large telecommunication networks, and working towards the PhD degree. His field of interest is the simulation of telecommunication networks, and he is the architect and main developer of the widely used OMNeT++ network simulation software. Currently he works at OpenSim Ltd. that he founded for commercializing the OMNeT++ software.