Towards a user-friendly parallel simulator of discrete-event systems

***Jiří Šafařík*[1] *and* *Viliam Solčány*[2]**

K paralelnému simulátoru systémov diskrétnych udalostí

We propose an algorithm used in a user transparent parallel simulator to improve the lookahead ability of submodels by combining the lookahead values of submodel's entities. We give experimental results for a simple model indicating that using the algorithm can significantly contribute to simulation speedup.

Key words: *discrete event simulation, parallel simulation, user transparency, speed-up.*

Introduction

Typically, discrete event simulation is used in the system design for behavior analysis and performance prediction. Real time, discrete event simulation is a means of verifying a real time system in which a simulation model may interact with a surrounding environment, such as software components, hardware components or human operators (Cho, 2001). Real time, discrete event simulation can play role in the real time control of a system using its simulation model. This approach is presently used in the control of logistic systems (Verbraeck, 2000), or manufacturing systems (Min, 2002) The complexity of many systems being simulated nowadays is increasing so that the results from sequential simulation programs are often not available within a time period required in real time simulation. In such cases, there is a need to speed-up the simulation computation while retaining the level of detail of the model. This can be achieved by decomposing the model into *submodels* and using the methods developed in the research field of parallel discrete-event simulation (PDES) to execute the model in parallel (Fujimoto, 1999). However, direct application of these methods is usually too complicated. It requires that the simulationist understands the low-level details related to parallel model execution. This is really inconvenient when realizing that the simulationist wants to concentrate on the problem he/she is solving and for which the simulation is used as a tool. The inconvenience is one of the reasons that PDES has been poorly accepted in practice.

Main issue in any parallel simulator is the *synchronization* among processors, which is to avoid *causality errors* and thus ensure correct results. To tackle this issue, two approaches have evolved: *conservative* and *optimistic*. While the former strictly avoids causality errors, the latter provides means to detect and correct them. Generally, both of them require user support for parallel execution. The usage of the latter one is described in (Ghosh, 1994). We have taken the conservative approach. Here the difficulty is that the quantity known as *lookahead* must be specified for each *submodel* to allow efficient synchronization and execution speedup. Traditionally, the user has been supposed to understand the details of the synchronization and to design the submodels with lookahead.

In (Solčány, 2002) we proposed a simulator design in which the PDES internals are hidden from the simulation user. It is based on reusability of model components – entities. In large models, multiple entities are assigned to a single processor constituting a *compound submodel*. The lookahead of such submodel can be transparently improved by combining lookahead values of the individual entities. The synchronization method we use is described in section 2. The algorithm we have developed for computing the compound lookahead is presented in section 3. Preliminary experimental results are given in section 4.

The Synchronization Method

For synchronization of submodels, we use the conservative synchronous approach based on time windows. The method assumes the ability of entities to pre-send event messages. Further, it is assumed that each entity e can compute the timestamp (or lower bound) $\delta_e(t)$ of the next message e will send, provided it receives no further messages. These two actions constitute the lookahead, which is necessary for the synchronization protocol. For now, suppose that each submodel consists of a single entity. The synchronization protocol works as follows: assume that all events (in all submodels) with timestamps less than t have already been processed and that the processors are globally synchronized. Each entity e supplies the value $\delta_e(t)$ and the processors

[1] *Jiří Šafařík:* Department of Computer Science and Engineering, University of West Bohemia Plzeň, Czech Republic safarikj@kiv.zcu.cz
[2] *Viliam Solčány:* Department of Computer Science and Engineering, Slovak University of Technology, Bratislava, Slovak Republic solcany@dcs.elf.stuba.sk
(Recenzovaná a revidovaná verzia dodaná 19.11.2003)

cooperatively compute the minimum $\delta(t)=\min_{\forall e}\{\delta_e(t)\}$. The interval $[t,\delta(t))$ defines the time window within which all simulation processors may execute events independently. When all these events are processed, the window moves so that the upper edge of the current window becomes the lower edge of the next window, and a new upper edge is computed.

Compound submodels

So far, we have assumed that each submodel is constituted by a single entity. In this case the lookahead implemented in entities can be exploited directly, as described above. In large models, it may often be of advantage to reduce the number of submodels by grouping several entities into a submodel. To enable transparency of this grouping, the entities themselves have to remain unchanged. Messages sent among local entities take the form of events scheduled into the local event list. To ensure proper synchronization of the submodels, we do not need to consider these local messages when determining the upper edge of the time window. The submodels are the units which have to be synchronized, and we need to know for each submodel SM the timestamp (lower bound) $\delta_{SM}(t)$ of the next message that will be sent out of the submodel, provided it will receive no further messages. The new time window will then be determined by $\delta(t)=\min_{\forall SM}\{\delta_{SM}(t)\}$.

The Algorithm

The structure of the simulation model can be viewed as a directed graph $G(E,C)$ where E is the set of entities and C the set of channels among them. A submodel SM can be represented by a subgraph $G_{SM}(E_{SM},C_{SM})$, $C_{SM}\subset E_{SM}\times E_{SM}$.

Let $PRED_e$ and $SUCC_e$ denote the set of predecessors and successors of entity e, respectively. Let $E_{out} \subset E_{SM}$ be the set of *output entities* which can send messages to other submodels. Let $\chi_e(t)$ denote the timestamp of the next message sent by e after t regardless of whether e receives further messages or not. Let $\sigma_e(t)$ denote the timestamp increment of the next output message sent by e after time t with respect to a previously received input message. The algorithm is similar to the Dijkstra's shortest path algorithm. It has been proven that all entities stored in SOL have correct χ values. (Solčány, 2002).

Input: subgraph G_{SM}, values $\delta_e(t) \geq t$ and $\sigma_e(t) \geq 0$, $\forall e \in E_{SM}$

Output: time $\delta_{SM}(t)$

Data structures:

- priority queue Q of entities. The queue is sorted in ascending order according to a key – timestamp $\chi_x(t)$ of the message sent by entity x. Operations on Q are:
  - `insert(x)` inserts element x into the proper position,
  - `get_min()` returns the element with minimum key and removes it from Q,
  - `sort(x)` moves element x to the proper position after its key has changed.
- the solution set SOL for storing entities that have already been processed.

Algorithm:

```
(1)  SOL = ∅;
(2)  foreach (e ∈ E_SM) { χ_e(t) = δ_e(t); Q.insert(e); }
(3)  while (∃ x ∈ E_out such that x ∉ SOL) {
(4)      e = Q.get_min();
(5)      foreach (x ∈ SUCC_e)
(6)          if (χ_x(t) > χ_e(t) + σ_x(t))
(7)              { χ_x(t) = χ_e(t) + σ_x(t); Q.sort(x); }
(8)      SOL = SOL ∪ {e}; }
(9)  δ_SM(t) = min_{∀e∈Eout}(χ_e(t));
```

Experimental results

We carried out simulations of the model depicted in Figure 1. It consists of 3 entities partitioned into two submodels SM1 and SM2. The source S generates transactions with exponentially distributed interarrival times with mean varying so that the traffic intensity at queue Q varies between 0.1 and 0.9. Queue Q has exponentially distributed service times. Each simulation is run twice, with time windows being computed once using the algorithm and another time without it – involving each individual entity. We count the number of time windows

(NTW) established during the simulation run in either case. The ratio of the two values is shown in Figure 2. As can be seen, the number NTW (i.e. the cost of interaction between submodels) can be substantially reduced using the algorithm.

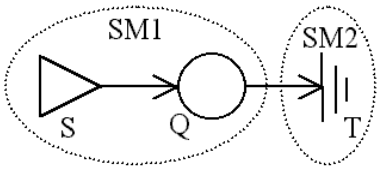

Fig.1. Simulation Model.

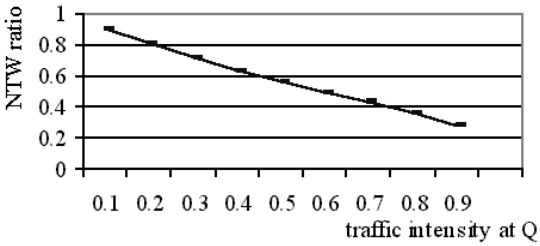

Fig.2. Experimental results.

Conclusion and further work

We presented an algorithm used in the user-transparent parallel simulator to compute the lookahead of compound submodels along with promising experimental results. Further we intend to carry out more experiments and study the impact of the algorithm on the speedup as well as to investigate more deeply the issue of lookahead cumulation.

Acknowledgement

This work was supported by the Ministry of Education of Czech Republic project No. MSM-35200005 – Information Systems and Technologies, and by the Slovak Grant Agency under grant No. VG 1/0161/03.

References

CHO, M.S. and KIM, T.G.: Real Time Simulation Framework for RT-DEVS Models. In Trans. Society for Modeling and Simulation International. Vol. 18, No. 4, 2001. pp. 203-215.

FUJIMOTO, R. M.: Parallel and Distributed Simulation Systems. *John Wiley & Sons*. 1999. ISBN: 0471183830

GHOSH, K., FUJIMOTO, R.M. and SCHWAN, K.: A Parallel, Optimistic, Real Time Simulator, In *ACM SIGSIM Simulation Digest*, Vol. 24, No. 1, 1994. pp. 24-32.

MIN, B. et al.: Integration of RT Control Simulation to a Virtual Manufacturing Environment. In *Journal of Advanced Manufacturing Systems*. Vol. 1, No. 1, 2002. pp 67-87.

SOLČÁNY, V. and ŠAFAŘÍK, J.: The lookahead in a user-transparent parallel simulator. In 16th Workshop Parallel & Distributed Simulation, Washington D.C., USA, 2002. pp. 11-16.

VERBRAECK, A. et al.: Simulation as a Real Time Logistic Control System: AGV Control with Simple++. In The New Simulation in Production and Logistics – Prospects, Views and Attitudes. Mertins, K., Rabe, M. (eds.). Berlin, Germany, 2000. pp. 245-255.