APPLICATION OF A JAVA-BASED FRAMEWORK TO PARALLEL SIMULATION OF LARGE-SCALE SYSTEMS

Ewa NIEWIADOMSKA-SZYNKIEWICZ[*,**], Maciej ŻMUDA[*]
Krzysztof MALINOWSKI[*,**]

[*] Institute of Control and Computation Engineering, Warsaw University of Technology
ul. Nowowiejska 15/19, 00–665 Warsaw, Poland
e-mail: {e-n-s, kmalinowski}@ia.pw.edu.pl, mzmuda@elka.pw.edu.pl

[**] Research and Academic Computer Network (NASK)
ul. Wąwozowa 18, 02–796 Warsaw, Poland

Large-scale systems, such as computer and telecommunication networks, complex control systems and many others, operate in inherently parallel environments. It follows that there are many opportunities to admit parallelism into both the algorithm of control implementation and simulation of the system operation considered. The paper addresses issues associated with the application of parallel discrete event simulation (PDES). We discuss the PDES terminology and methodology. Particular attention is paid to the software environment CSA&S/PV (*Complex Systems Analysis & Simulation—Parallel Version*), which provides a framework for simulation experiments performed on parallel computers. CSA&S/PV was applied to investigate several real-life problems. The case studies are presented for both computer and water networks.

Keywords: parallel computations, simulation, large-scale systems, computer systems, computer-aided system design

1. Introduction: Parallel Discrete Event Simulation

In recent years parallel processing has provided a new impetus in systems engineering. It is clear that physical systems are inherently parallel objects which lend themselves to parallel computation—this is obvious as they operate in the real world, where parallelism is a natural phenomenon of everyday life. Parallel simulations allow us to reduce the computation time of the simulation program, to execute large programs which cannot be put on a single processor and to better reflect the structure of the physical system which usually consists of several components. The role of parallel simulations is particularly and increasingly important in the field of large-scale systems, where simulations require significant execution time.

In sequential discrete event simulation all processes access the same event list. In parallel discrete event simulation (PDES) the event list is distributed over the number of processors. A parallel simulation program may be viewed as a collection of sequential simulation programs, i.e. *logical processes* (LP), each modelling a single physical process. The LPs communicate sending time-stamped messages to each other. It is important that all interactions must occur via this message-passing mechanism. We allow for the possibility that messages are not received in the order in which they were transmitted.

There are three important procedures associated with the implementation of PDES: computation decomposition and processes allocation, synchronization, and memory management.

Load balancing. The computation processes should be distributed across the processors in order to balance the load. Several strategies of problem partitioning are proposed: *domain decomposition*, where the idea is to divide the data domain into several components on which the calculations can be carried out independently, *functional decomposition*, where the idea is to divide the calculation algorithm into several modules, *dynamic decomposition* when the partitioning problem is dynamically changed as the program is executed (to achieve a more balanced workload). After decomposition the whole task can be modelled as a directed graph in which nodes represent logical processes (i.e. subtasks), and arcs (i.e. interconnections) indicate communication between the nodes. The next step is to distribute logical processes across the parallel processors, so that all processors work effectively all the time and inter-processor communications are minimized. Two basic approaches are static and dynamic allocation. *Static allocation* algorithms distribute fixed tasks (processes) over the processors for the duration of simulation. *Dynamic allocation* algorithms dynamically assign processes to processors, i.e. allow processes to migrate during the simulation. The decision about the adequate allocation technique strongly depends on the hardware platform and the characteristic features of the simulation study considered.

Synchronization. The calculation tasks require explicit schemes for synchronization. Two simulation techniques are considered (Banks, 1998; Kheir, 1996), namely a synchronous one and an asynchronous one. *Synchronous* simulation is implemented by maintaining a global clock (GVT—Global Virtual Time). The events with the smallest time-stamp are removed from the event lists of all LPs for parallel execution. The execution of these events generates new events that are requeued to the event lists. The parallelism of this technique is limited because only events with time-stamps equal to that of the global clock can be executed during an event cycle. *Asynchronous* simulation is much more effective due to its potentially high performance on a parallel platform. In asynchronous simulation each logical process maintains its own local clock (LVT—Local Virtual Time). The local times of different processes may advance asynchronously. The events arriving at the local input message queue of a logical process are executed according to the local clock and the local schedule scheme. The synchronization mechanisms fall into two categories, namely, conservative and optimistic. They differ in their approach to time management. *Conservative* schemes avoid the possibility of causality error occurrence. These protocols determine safe events, which can be executed. Classical approaches, the CMB protocol developed by Chandy, Misra and Bryant and based on *null messages*, as well as various algorithms using windows are described in the literature (Mehl, 1991; Misra, 1986; Nicol and Fujimoto, 1994). *Optimistic* schemes such as *Time Warp* and its modifications (Jefferson, 1985; 1990) allow for the occurrence of causality errors. They detect such an error and provide mechanisms for its removal. The calculations are rolled back to a consistent state by sending out antimessages. It is obvious that in order to allow rollback, all the results of the previous calculations have to be recorded. The key advantages and disadvantages of conservative and optimistic protocols are summarized in (Banks, 1998).

Memory management. While the discussion above is concerned with the minimization of simulation time, a related question is that of optimizing memory resource management. In the case of optimistic and hybrid algorithms all the reported schemes control the memory usage, but only indirectly. Another class of schemes uses memory management for “optimism control”. We can distinguish two approaches to limit memory utilization in Time Warp: passive and active schemes.

Passive techniques include infrequent and incremental state saving (Lin, 1994; Nicol and Fujimoto, 1994; Soliman, 1999). When the state vector is large and only a small part is modified in each event execution, incremental state saving may be applied—only changes in the state are recorded. An alternative approach is saving the entire state vector with the reduced frequency.

Passive techniques reduce the average memory usage but do not allow for recovering unused memory during processing. Active schemes can reclaim memory on demand. Various approaches are proposed and described in (Jefferson, 1990; Nicol and Fujimoto, 1994).

2. Software Environments

In order to efficiently perform simulation experiments, good software tools are needed. A present, there are two basic directions to follow when developing such software packages: the development of problem dedicated (specialized) systems (Di and Mouftah, 2002; Niewiadomska-Szynkiewicz, 2002; NS-2, 1995; OMNeT++, 1992), which are specific for a given type of processes, and the creation of general purpose (universal) systems. The advantage of a specialized system is that one can have typical algorithms for identification and control as well as process simulators built-in in the software environment. The disadvantage of such a software environment is that it has restricted use and is difficult for the user to modify when new features need to be introduced. Universal systems allow us to set up simulation experiments and to analyze different types of processes. However, one must pay for this universal applicability with having to prepare—for each particular case study—those software modules which are specific to this study.

Since parallel and distributed simulation is becoming a dominant form of model execution, the focus is on experiments carried out on parallel and distributed hardware platforms. In the last years numerous integrated environments for parallel and distributed processing have been developed (HLA, 1998). These software tools apply various techniques for synchronization and memory management, and focus on various aspects of parallel implementation. Many of them are built in Java (Kreutzer *et al.*, 1997; Nicol *et al.*, 1997).

A natural solution to consider is parallel and distributed simulation using Java, as Java offers a potential for using many different capabilities to complex simulation models. One of the advantages of Java is that threads are built directly into the language. The second is the RMI (*Remote Method Invocation*) mechanism for performing distributed calculations in computer networks. Java provides a rich assortment of classes and methods for graphical applications. The graphical interface is an important component of the simulation software. One advantage of Java over C++ is its relatively simple and direct syntax for expressing exception handling and data management. So, in practice, it is much easier to implement open architecture software tools in Java.

This paper deals with the description of a Java-based framework for parallel simulation and its application to the analysis of complex control systems.

3. CSA&S/PV—An Integrated Framework for Parallel Simulation

3.1. Description of CSA&S/PV

CSA&S/PV (*Complex Systems Analysis & Simulation—Parallel Version*) is a parallel software environment in Java for the simulation of various types of real systems. It has its origin in CSA&S, a sequential simulator written in C (Niewiadomska-Szynkiewicz *et al.*, 1995). The main idea of this system is to minimize the user's effort during the design and simulation of complex physical processes. CSA&S/PV provides a framework which allows us to perform simulations on parallel computers. It offers the graphical environment (shell) for supporting implementation of the case study considered and a library of functions providing communication between the user's applications and the system interface. CSA&S/PV manages calculations and communication between running processes and provides tools for on-line monitoring of the computed results.

An asynchronous version of simulation is applied. Each node (logical process) maintains its own local clock and event list. The local times (LVT) of different nodes may advance asynchronously. LPs can operate in two modes:

Time-Driven Mode: The increment in the LVT of a local logical process (LP) is fixed and defined during the preparatory stage. The LP is executed every defined time step (repetition time), which means that the LVT changes at regular intervals. We assume that for different LPs different repetition times may be introduced.

Event-Driven Mode: Logical processes are executed after each event occurrence. LVTs change at irregular intervals. A conservative scheme similar to the CMB algorithm is used for synchronization. The events are executed only when it is certain that no event with an earlier time-stamp can arrive. At the current time t each logical process LP$_i$ computes the minimum time LVT$_i = \min_{j\in N(i)}(t_{ij} + \tau_{ij})$, where t_{ij} is the time-stamp of the last message received from the LP$_j$ process, $N(i)$ is a set of processes transmitting data to LP$_i$ and τ_{ij} is a transmission delay from node j to i (transfer cost). Next, each LP$_i$ simulates all the events with the time-stamps less than LVT$_i$. The processes exchange messages as presented in Fig. 1. When the execution of the analysed events begins, LP$_i$ sends to all its neighbours null messages with the time-stamp LVT$_i + \Delta T_i$, where ΔT_i denotes the pending event time. It contains information about the earliest possible time of the next event execution. Null messages are used to announce the absence of messages with new data.

receiver address	flags	time stamp	transmission delay	data

Fig. 1. Contents of a message from each user application (a node of the simulated system graph).

Both the types of LPs can be executed during the same simulation experiment. All calculation processes communicate with each other via shared memory. The mechanism for parallel implementation is based on threads (see Fig. 2).

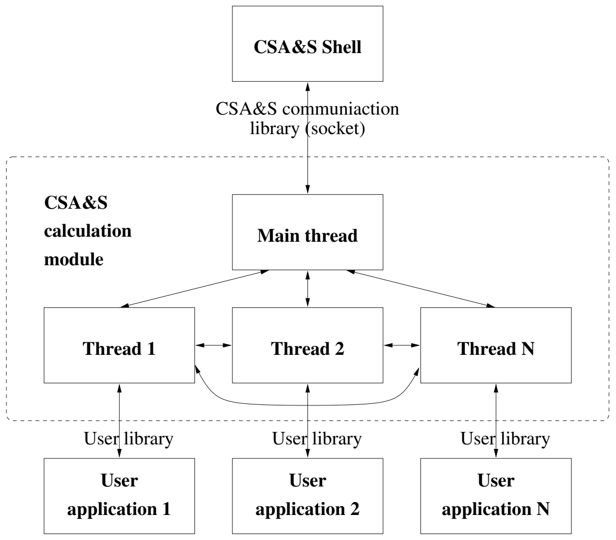

Fig. 2. Architecture of the CSA&S/PV system.

3.2. CSA&S/PV Structure

The CSA&S/PV software package consists of five components (see Fig. 2): the *shell*—the graphical interface, responsible for user-system interaction, the *calculation module (manager)*—the system kernel that manages calculations and communication between running processes, the *communication library*—the library of functions that provides communication between the graphical shell and the system kernel, the *user library*—the library of functions providing an interface between the user application and the "manager" (system kernel), the *user application*—LPs' simulators of the physical systems (developed by the user).

The interface is graphical. It was written in the Java language and may operate under MS-Windows, Windows-NT and Unix operating systems. The main component of the CSA&S/PV software package is the *manager*—the system kernel. Because the CSA&S/PV system is heterogeneous (the *shell* and the *manager* may operate under different operating systems) it was necessary to develop a library of functions that provide communication between the user interface process and the system kernel process. These processes communicate with each other via *sockets*—simple mechanisms for interprocess communication.

3.3. User Application

The user's task is to implement simulators of the subsystems corresponding to the nodes of the graph considered. These modules may be written in Java, C or C++. As was mentioned above, the CSA&S/PV package supplies the library of functions providing the interface between the application programs and the system kernel. This allows the user to focus on the numerical part of the program only. In addition, if the functions unique to the operating system are not used by the user, the applications can be moved as needed between different computing platforms.

In general, each user's application consists of six functions: *csasInit*, the task of which is to prepare the environment for future calculations and to calculate the initial conditions, *csasExecuteArgs*, gathering data for calculations from CSA&S/PV, *csasExecute*, providing calculations (the main part of the user's application), *csasExecuteResults*, which sends the results of calculations to CSA&S/PV, *csasStore*, the task of which is to store all current calculation results after system termination (simulation can be continued), and *csasEnd*, an additional function for removing all data structures dynamically allocated during the program operation.

Communication between the user's application and the system kernel is provided by the CSA&S/PV *user library*.

3.4. Simulation under CSA&S/PV

During a simulation experiment performed under CSA&S/PV, one can distinguish two main stages: a preparatory stage and an experimental stage. At the *preparatory stage* the model and the properties of the system to be simulated are investigated. The calculation process is partitioned into several subsystems (subtasks) with respect to functionality and data requirements. The directed graph of the analysed system $G = (N, A)$ is created using the CSA&S/PV graph editor (see Fig. 9) or can be read from an XML file. The set of nodes is equal to N with node i representing the i-th subsystem $(i = 1, \dots, N)$. The presence of an arc (i, j) indicates the possibility that the i-th subsystem influences the j-th subsystem. Each node of the graph represents the program executing the tasks of the node. This program has to be prepared by the user and it must be ready to run. As far as the CSA&S/PV system is concerned, the goal of this node application program is to gather data from the connected nodes and to generate other data for the other nodes. Within the next step the user is asked to provide information related to the nodes of the graph considered. The information includes: the name of the calculation program corresponding to each node, the repetition time period (if necessary), the decision delay, i.e. the time required to execute the events in the physical application. Next the user is asked to provide some information related to all the inputs of these nodes: the name of each input, the transmission delay related to data transmission to a particular location. The currently considered graph of the simulated system may be saved into the disc file in the XML format. In this way the implemented system can be used in many future simulations.

The *experimental stage* begins when all decisions regarding the simulated system are made. The simulation time horizon is defined and the experiment starts. The programs corresponding to the nodes of the system graph are executed and the results of the calculations are displayed, Fig. 4. The user employs the monitoring and analysis of the current situation. All results may be recorded into the disc file during the experiment. There is a possibility to extend the simulation horizon if desired.

4. Practical Examples

CSA&S/PV allows for setting up simulation experiments and the analysis for different types of processes. It does not involve any restrictions regarding the size of the simulation, but such restrictions may be caused by the available computer (for PC computers the suggested number of threads is less than 100). In general, CSA&S/PV is dedicated to coarse granularity parallel implementations. The speed up of the parallel simulation with respect to the sequential approach strongly depends on the application and its decomposition.

CSA&S/PV has already proved to be very useful when performing the analysis of different control mechanisms for flood control in multireservoir systems and a preliminary analysis/tuning of routing and flow control for data networks. The presented case studies show the possible range of the discussed software system applications.

The last example is focused on the effectiveness of parallel implementations.

All numerical experiments described below were performed on a SUN HPC E10000 Starfire computer with twelve 400MHz processors running the Solaris 7 operating system.

4.1. Routing in a Data Network

The first considered case study was related to routing in data networks. There are many well known routing algorithms. They can be easily simulated under CSA&S/PV. The simple asynchronous shortest path algorithm *Adaptive-Scheme* (AS), similar to the *Bellman-Ford* (BF) one (Bertsekas and Gallager, 1992), was implemented and tested. A detailed description of this scheme can be found in (Pondarzewski *et al.*, 1999). In the case of the BF algorithm, the shortest path $D_{i,des}$ from each node i of the computer network to a destination node des is calculated and recorded in the routing table. The following iteration is executed at the i-th node:

$$D_{i,des} = \min_{j \in N(i)} [d_{ij} + D_{j,des}] \tag{1}$$

using the last estimates $D_{j,des}$ received from its neighbours $j \in N(i)$ (here $N(i)$ denotes the set of the current neighbours of node i) and the latest status and lengths of the outgoing links d_{ij} from node i. The algorithm requires that each node j transmit its latest estimate $D_{j,des}$ to all its neighbours from time to time.

The *Adaptive Scheme* similar to the Bellman-Ford algorithm calculates the minimal distances from each node to the destination node. It differs from the BF in that in the case of the AS scheme the interprocess communication and the volume of the transmitted data are minimized. The nodes do not have to transmit their current routing tables. The shortest path distances are estimated on-line using the data carried by routed messages. No assumptions are made on their initial values (the routing tables are empty in the beginning). Another difference is that instead of one length parameter d_{ij} of the outgoing link from node i to node j, we consider two values connected with sending and receiving data, i.e. $d_{ij} = costOut_i(j) + costIn_j(i)$ (here $costOut_i(j)$ denotes the cost of sending data from node i to node j and $costIn_j(i)$ is the cost of receiving data from node i by node j). We assume that the i-th node knows only its transmission cost tables $costOut_i$ and $costIn_i$. Nodes i and j exchange messages as presented in Fig. 3. Each message contains transmitted data and additional information: addresses of the sender (sen) and destination (des) nodes, the last estimate of the distance from the sen-th node to the i-th node, $D_{sen,i}$ increased by sending costs through the j-th output, $D_{sen,i}(j) = D_{sen,i} + costOut_i(j)$ and the

sender address	destination address	distance	distance	data
sen	des	$D_{sen,i}(j)$	$D_{i,des}(j)$	

Fig. 3. Contents of a message from the i-th to the j-th node.

expected distance from the current i-th node to the destination des node decreased by sending costs through the j-th output $D_{i,des}(j) = D_{i,des} - costOut_i(j)$. The current estimates of the shortest distances from node i to all other nodes are recorded in a two-dimensional routing table $RT_i(N(i), N-1)$, where $N(i)$ denotes the number of the neighbours of node i. The following algorithm is executed at each node j, as a result of a new message from node i:

$$D_{sen,j} = D_{sen,i}(j) + costIn_j(i), \tag{2}$$

$$D_{j,des} = D_{i,des}(j) - costIn_j(i), \tag{3}$$

The routing table is updated. Two cases are recognized:

- If $D_{j,des} \geq \min_{k \in N(j)} RT_j(k, des)$ (the current shortest distance to node des is less than or equal to the actual estimate), then

  ```
  if $D_{sen,j} < RT_j(i, sen)$ then
      $RT_j(i, sen) = D_{sen,j}$
  else if $D_{sen,i} > RT_j(i, sen)$ then
      $RT_j(i, sen) = RT_j(i, sen) + w$.
  ```

  In the case when $D_{sen,j} < RT_j(i, sen)$, the currently calculated shortest path from node sen to node i is recorded in the routing table. Otherwise, the value of $RT_j(i, sen)$ is increased by a small value w depending on the difference between the real and expected costs. In this case the values in the routing table increase in small steps. Such an approach is proposed because of the mistakes that may occur in the computer network.

- If $D_{j,des} < \min_{k \in N(j)} RT_j(k, des)$ (the current estimate of the shortest distance to node des is greater than the actual estimate), then the current value of the expected cost to des is calculated:

  $$D_{j,des} = \min_{k \in N(j)} RT_j(k, des) + costIn_j(i)$$

  and sent back to node j.

This is a basic version of the proposed algorithm. It must then be tested and (if needed) modified prior to the implementation. The contents of a message, as presented in Fig. 3, would be possible in IPv6. In the case of IPv4 the expected values of the examined distances $D_{sen,i}$ and $D_{i,des}$ must be sent as additional messages. The main question is how often they should be sent.

A hypothetical computer network model consisting of 24 nodes was simulated. All tests were performed under the following assumptions: each node could generate a limited number of initial messages M, the destination of each generated message was randomly chosen from $N-1$ nodes, and the horizon considered was equal to 1000 time units.

The network was implemented in the CSA&S/PV system using 24 units representing the nodes of the network and one additional global unit for the presentation of the results. The trajectories presented in Fig. 4 show the number of all transmitted messages and the costs (time delay) of their transmission at the time instant considered. In general, we can distinguish two phases: the adaptation phase and the working phase (see Fig. 4). It can be observed that the costs of data transmission are quite high in the first phase. The length of the adaptation phase depends on the number of the messages M generated by each node: the smaller M, the longer the adaptation phase. The costs trajectory is quite smooth in the second phase, even in the case of some modifications of $costIn$ tables (see Fig. 4, time instant 260). The fluctuations are higher in the case of a smaller number of the initial messages M (see Table 1).

Table 1. Possible transmission delays of messages.

M (initial messages)	adaptation phase	working phase
5	360–600	50–200
10	195–280	30–68

We did not observe a serious speed up of calculations in the case of the parallel implementation. This example and the next one presented below are fine granularity parallel applications. A potential reduction in the computation time should be observed after increasing the size of the systems considered.

4.2. Optimization Network Flow Control

The second examined case study was related to the optimization approach to flow control in communication networks. The asynchronous link algorithm for the pricing of network services based on the Price Method was implemented and tested. A detailed description of this method together with the discussion of its convergence can be found in (Low and Lapsey, 1999).

Consider a network consisting of a set $L = \{1,\dots,L_n\}$ of unidirectional links of capacities c_l, $l \in L$ and a set $S = \{1,\dots,S_m\}$ of traffic sources. Each source is defined by the quadruple $(L(s), U_s(x_s), x_{s_{\min}}, x_{s_{\max}})$, where x_s denotes the transmission rate, $U_s(x_s)$ stands for the source utility function defined over the interval $X_s = [x_{s_{\min}}, x_{s_{\max}}] \subseteq \mathbb{R}_+$, $x_{s_{\min}}$ and $x_{s_{\max}}$ are minimum and maximum transmission rates, respectively. For each link l, let $S(l)$ be the set of the sources that use l, so $l \in L(s)$ if and only if $s \in S(l)$. The objective is to maximize the aggregate source utility over their transmission rates, so the flow optimization problem can be formulated as follows:

$$\max_{x_s \in X_s} \sum_s U_s(x_s), \qquad \sum_{s \in S(l)} x_s \le c_l, \quad l \in L_n. \qquad (4)$$

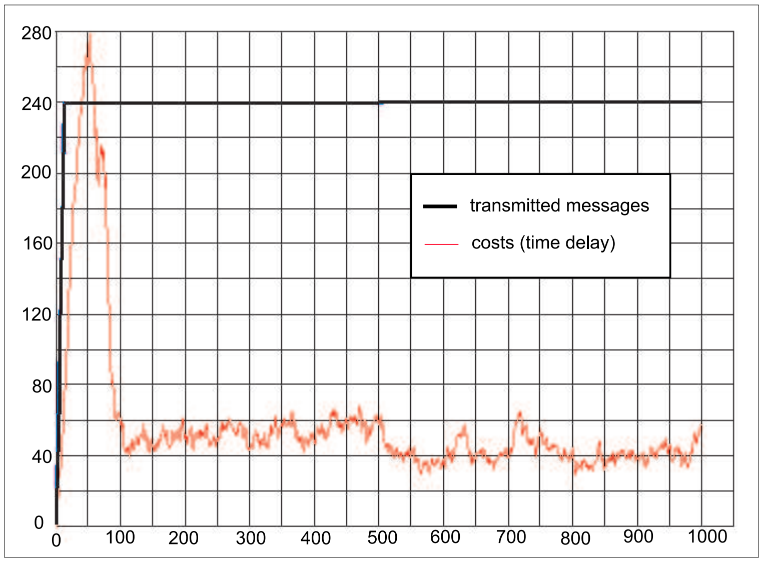

Fig. 4. Transmitted messages and time delays (each node generates 10 initial messages).

If the feasible set is nonempty and the performance function is strictly concave, then the unique maximizer $\hat{x}$ exists (Low and Lapsey, 1999).

The optimization problem (4) can be solved by the Price Method (the dual method using price coordination, cf. (Findeisen *et al.*, 1980)) in parallel or distributed environments.

Define the Lagrange function of (4):

$$L(x, \lambda) = \sum_s U_s(x_s) - \sum_l \lambda_l \left(\sum_{s \in S(l)} x_s - c_l \right)$$

$$= \sum_s \left(U_s(x_s) - x_s \sum_{l \in L(s)} \lambda_l \right) + \sum_l \lambda_l c_l, \quad (5)$$

where $\lambda_l \geq 0$, i.e. the Lagrange multipliers associated with capacity constraints denote the link prices.

We can formulate the local (source) and coordinator level optimization problems:

LPs: $s = 1, \dots, S_m$, for given λ_l find a maximum with respect to x_s of the local performance index

$$\max_{x_s \in X_s} \left[L_s(x_s, \lambda) = U_s(x_s) - x_s \sum_{l \in L(s)} \lambda_l \right]. \quad (6)$$

CP: For the results of the LPs find a minimum with respect to λ_l of the coordinator performance index

$$\min_{\lambda_l \geq 0,\, l=1,\dots,L_n} \left[\varphi(\lambda) = \sum_s L_s(\hat{x}_s, \lambda^s) + \sum_l \lambda_l c_l \right], \quad (7)$$

where $\lambda^s = \sum_{l \in L(s)} \lambda_l$.

Synchronous and asynchronous distributed algorithms for computing prices were proposed by Low and Lapsley. In the synchronous version the l-th link price at the iteration instant $k+1$ is calculated as follows:

$$\lambda_l(k+1) = \left[\lambda_l(k) - \gamma \frac{\partial \varphi(\lambda(k))}{\partial \lambda_l} \right]_+$$

$$= \left[\lambda_l(k) + \gamma \left(\sum_{s \in S(l)} \hat{x}_s(k) - c_l \right) \right]_+, \quad (8)$$

where $[y]_+ = \max(y, 0)$ and γ is a sufficiently small step size.

Thus, in the approach defined by (8) all sources receive, at a given time instant k, prices $\lambda_l(k)$, compute the respective source prices $\lambda^s(k)$ and calculate optimal source rates $\hat{x}_s(k)$ solving LP problems. The obtained values of the source rates $\hat{x}_s(k)$ are then sent to the links, and the new link prices $\lambda_l(k+1)$ are computed according to (8).

In the case of an asynchronous approach both sources and link algorithms use the weighted averages of the past values of the link prices and the locally optimal source rates. So, the l-th link price at the iteration instant $k+1$ is calculated according to (8) assuming $\hat{x}_s(k) = \sum_{k'=k-k_0}^{k} a_{ls}(k', k) x_s(k')$ with $\sum_{k'=k-k_0}^{k} a_{ls}(k', k) = 1$, for all k, l and $s \in S(l)$; k_0 denotes the length of past window taken into account. Furthermore, the s-th source rate at time $k+1$ is calculated solving LPs, assuming $\lambda^s(k) = \sum_{l \in L(s)} \sum_{k'=k-k_0}^{k} b_{ls}(k', k) \lambda_l(k')$ with $\sum_{k'=k-k_0}^{k} b_{ls}(k', k) = 1$, for all k, s and $l \in L(s)$.

The algorithm was applied to flow control in an experimental computer network, as presented in Fig. 5. It consists of nine nodes: three sources, three routers, three destination nodes and eight bidirectional links. The maximal capacity of the links *Router1–Router2* and *Router2–Router3* was equal to 290. The capacity of other links was unlimited. The network was implemented in the CSA&S/PV system using nine calculation processes. The processes exchanged messages as presented in Fig. 1, containing adequate data: link prices—messages from routers and source rates—messages from sources. All the calculation processes corresponding to the nodes in Fig. 5 could communicate and update their controls asynchronously at different time instants, with different frequencies and transmission delays. The utility functions U_s of the sources were set to $\alpha_s \log(1 + x_s)$, with $\alpha_s = 10^4$ for all sources. Only the last received rate $x_s(\tau)$ for $\tau \in k - k_0, \dots, k$ was used to estimate the locally optimal source rates and the link prices. Each source transmitted data for a total of 120 000 time units; *source 1* started transmission at time 0, *source 2* at time 40 000, *source 3* at time 80 000. The whole simulation horizon was equal to 240 000 time units. The goal was to test the convergence of the algorithm with respect to the value of the step size in (8) and transmission delays in the network. Several experiments were performed taking into account different values of the step size $\gamma = \{$1E–3, 1E–4, 1E–5$\}$,

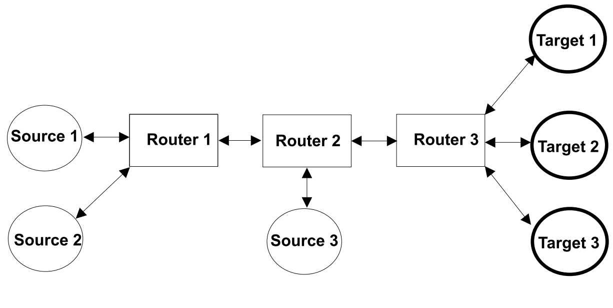

Fig. 5. Analysed IP network under the CSA&S/PV system.

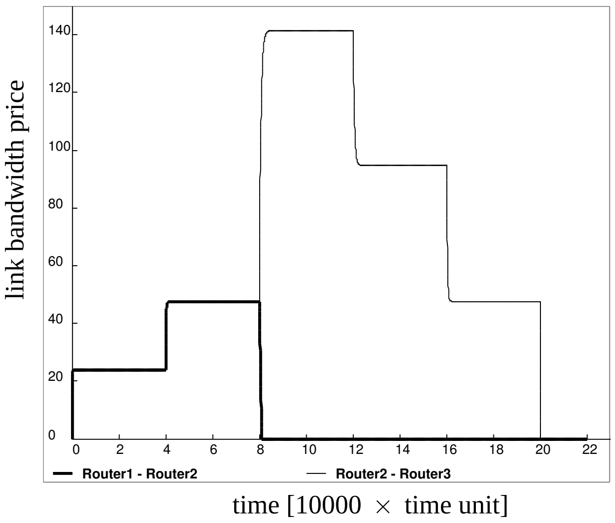

Fig. 6. Link prices for γ =1E–3 and $\tau_D = 1$.

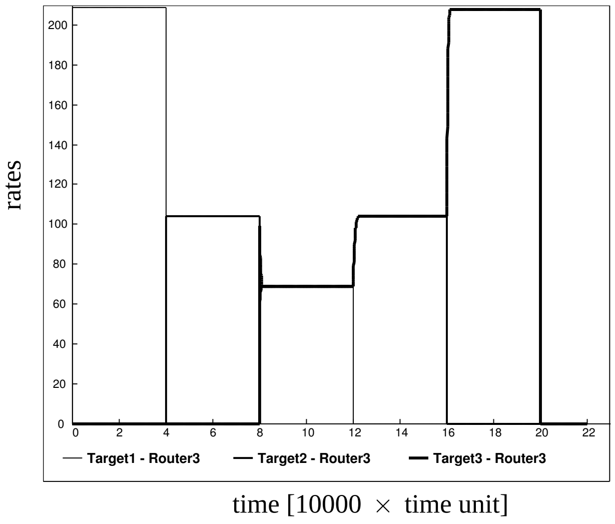

Fig. 7. Source rates for γ =1E–3 and $\tau_D = 1$.

Table 2. Percentage of rejected packages for different values of γ and τ_D.

	τ_D = 1	τ_D = 10	τ_D = 100
$\gamma = 1$E–3	14.59	15.62	98.41
$\gamma = 1$E–4	63.54	63.55	65.51
$\gamma = 1$E–5	92.22	92.23	92.28

and different transmission delays $\tau_D = \{1,\ 10,\ 100\}$ expressed in time units. It was assumed that the transmission was delayed for all links. The results are presented in Table 2 and Figs. 6 and 7. Table 2 contains the percentage of the rejected data packets with respect to all packets passed during the experiment. Figures 6 and 7 show respectively the link prices and the source rates. We can observe that the source rates adjusted dynamically as new sources started or stopped transmitting. As expected, the number of rejected packages increased for longer transmission delays. For $\tau_D = 100$ and the step size $\gamma = 1$E–3 the algorithm was not convergent to the optimum (see Tab. 2). After decreasing the step size a better solution was achieved. On the other hand, decreasing γ made it longer for the algorithm (8) to arrive at proper price values. In the case of the very small value ($\gamma = 1$E–5) the algorithm seemed to track the optimum but the solution was not reached. The presented results show that the examined pricing algorithm for flow control is very sensitive to the value of the step size. The estimation of the proper γ may involve many problems especially in the case of a huge network traffic.

4.3. Flood Control in a Multireservoir System

This case study is related to a hierarchical control structure for flood operation in the Upper Vistula river-basin system in the southern part of Poland. Three retention reservoirs, located on the Soła, Raba and Dunajec rivers, were considered. The optimal release problem was defined as the problem of minimizing the flood damages related to the peak flows at the measurement points in the whole river system. The hierarchical control mechanism (HDM) for reservoirs management was investigated. This mechanism is based on the application of the repetitive optimization of the outflow trajectories, using predicted inflows (Niewiadomska-Szynkiewicz *et al.*, 1996; Niewiadomska-Szynkiewicz, 2002). It incorporates two decision levels, as presented in Fig. 8: the upper level with the control centre (coordinator) and the local level formed by the operators of the reservoirs. The local decision rules are designed in such a way that a central authority, the coordinator, may adjust them in the process of periodic coordination so as to achieve the coordination of reservoirs in minimizing the global damages. Hence, the decision problem of the i-th local reservoir operator ($i = 1, 2, 3$) at time t_l is as follows:

$$\min_{u_i} \left[q_i\big(u_i(\cdot), a_i\big) = \max_{t\in[t_l,t_f]} \big(u_i(t)\alpha_i(t)\big) \right], \qquad (9)$$

where $[t_l, t_f]$ denotes the local level optimization horizon, q_i is the local cost function, and the a_i's mean parameters specified by the coordinator. The vector a_i of coordinating parameters for the i-th reservoir is related to the weighting function $\alpha_i(\cdot)$ defined as follows: $\alpha_i(t) = 1 + (c_i - 1) \cdot 1(t - T_i^\star)$, i.e. $\alpha_i(t) = 1$ for $t \in [t_l, T_i^\star)$ and $\alpha_i(t) = c$ for $t \in [T_i^\star, t_f]$.

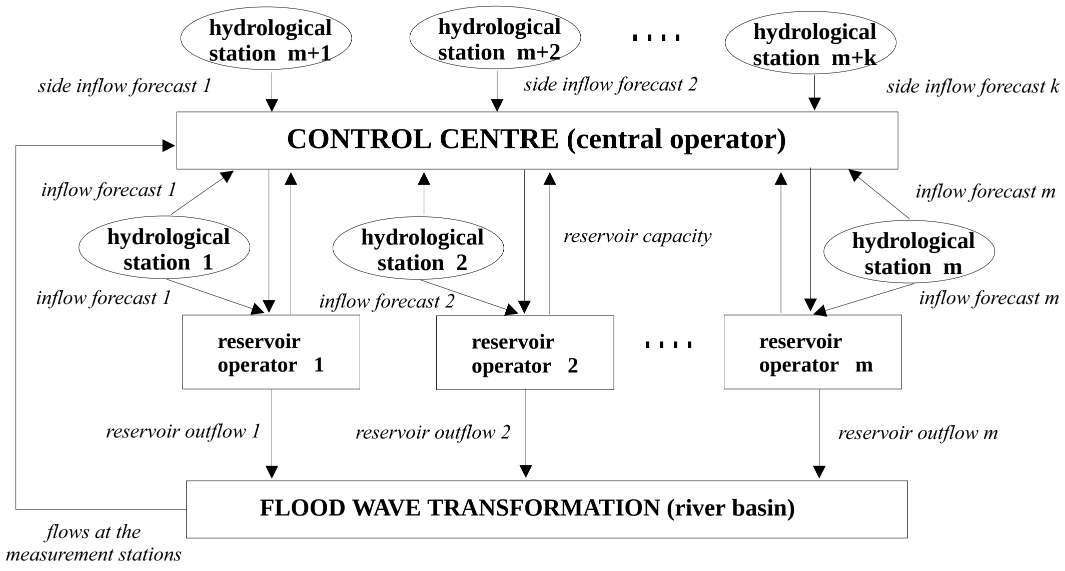

Fig. 8. Flood control in the Vistula reservoir system.

The goal of the control centre is to calculate optimal values of the parameters a in the sense of minimizing the damages in the whole river basin:

$$\min_{a \in A} J(Q_{[t_c,t_f]}),$$

$$Q(t) = F\left(Q(t_c),\ \hat{u}_{[t_c,t]}(a),\ \overline{d}\,^{t_c}_{[t_c,t]}\right), \qquad (10)$$

where $[t_c, t_f]$ denotes the control centre optimization horizon ($t_c \leq t_l$), $Q(t)$ is the vector of flows at the measurement points, $Q(t_c)$ denotes the vector of real flows measured at time t_c, $\overline{d}\,^{t_c}$ stands for the vector of forecasts of all the inflows calculated at time t_c, $\hat{u}$ is the vector of optimal outflows from the reservoirs (associated with the vector of parameters a), and $J(Q_{[t_c,t_f]})$ denotes a performance (loss) function. In each iteration of the optimization process, the value of $J(\cdot)$ is computed based on the simulation of the lower decision level (reservoir operators) and the flow transformation in the whole river basin.

The presented control structure was implemented under CSA&S/PV, cf. Fig. 9. The whole system was decomposed into several subsystems (processes) associated with the nodes in Fig. 8: a control centre, (coordination parameters calculation), reservoir operators (releases calculation), hydrological stations (inflow forecasts computing), and rivers (flow transformation). Simulations were performed for a set of historical data. The results obtained for this control system were compared with the centralized decision mechanism (CDM), where decisions about all outflows are made by the central operator, with the case of autonomous control of each reservoir LDM, based on the local decision mechanism and the traditional control rules TR—the instructions that have been used in operational flood control in Poland so far. In the simulation study the inflow forecasts were calculated based on two different models provided by the Institute of Meteorology and Water Management, denoted by CFM and WFM. The reductions of the flood damages with respect to the uncontrolled flood wave are presented in Fig. 10. A detailed description of multiple experiments performed for a set of historical hydrograms of major flood events that

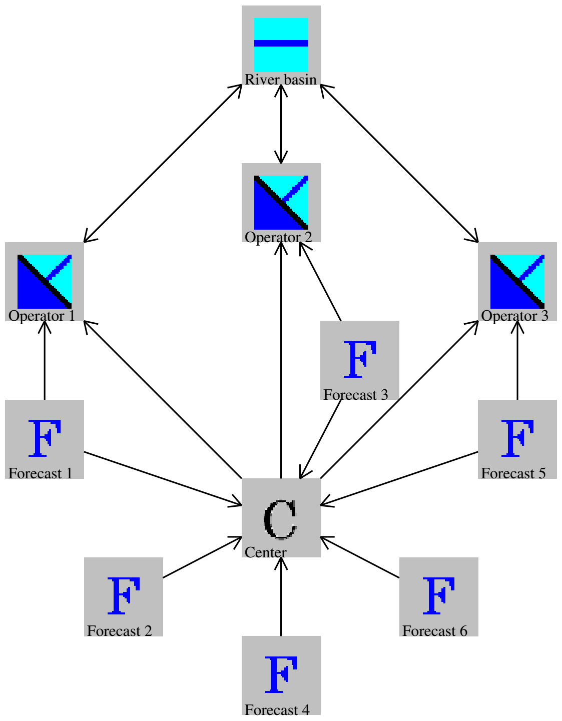

Fig. 9. Implementation of HDM control under CSA&S/PV.

occurred between the years 1960 and 1974 can be found in (Niewiadomska-Szynkiewicz, 2003).

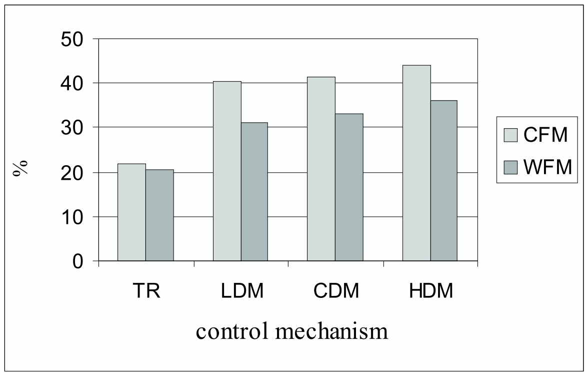

Fig. 10. Average reduction of the global damages with respect to the uncontrolled flood (the *Vistula river system*).

In this example the sequential part of the application—control centre simulation—strongly influences the calculation time. Because of this, the acceleration factor with respect to the sequential simulation is about 2. A way to speed up the calculations is to apply a parallel optimization method to solve the central dispatcher decision problem. It can be developed using Java threads.

4.4. Parallel Global Optimization

The last presented application under CSA&S/PV was the the global optimization problem. The goal was to calculate the minimum of the following test function (the so-called Acley function):

$$f(x) = -20\exp\left(-0.2\sqrt{\frac{1}{100}\sum_{i=1}^{100} x_i^2}\right) - \exp\left(\frac{1}{100}\sum_{i=1}^{100}\cos(2\pi x_i)\right) + 20 \quad (11)$$

subject to the constraints $-30 \leq x_i \leq 30$, $i = 1,\dots,100$. There is one global minimum, which occurs at the point $x = (0,\dots,0)$.

The co-evolution algorithm described in (Michalewicz, 1994) was used to solve the optimization problem. This technique is easily adaptable to parallel environments. In the described implementation several instances of the evolution algorithm were executed, each being represented by a CSA&S/PV calculation unit. From time to time the units interchanged a few randomly chosen elements from their current population with four neighbours.

Several series of experiments considering different numbers of units were performed. When the global optimum was reached with the accuracy 0.1, the algorithm stopped. The goal was to present the effectiveness of the parallel CSA&S/PV implementation. Table 3 shows the speed up of calculations with respect to the number of logical processes (the CSA&S/PV units).

Table 3. Speed up of the calculations performed under CSA&S/PV versus the number of logical processes.

LPs number	1	2	3	8	12	16	24	30	36	64
simulation time [s]	517	264	177	110	91	82	75	71	67	65

Similarly to the previous examples, all tests were performed on a SUN computer with 12 processors. It should be pointed out that the time of calculations strongly depends on the application partitioning. In some cases the problem decomposition into m logical processes, where $m > p$ (p is the number of the available processors), may speed up the calculations with respect to the decomposition with $m = p$.

5. Conclusion

All the presented applications demonstrate the effectiveness and efficiency of the CSA&S/PV system. The parallel, asynchronous simulation adopted in the package allows us to perform fast simulation of large-scale systems. The user is able to perform an analysis of given system behaviour in various conditions and operating systems without writing separate user applications for each case. He or she can influence the simulation process, record and browse through all the results of calculations. As a final observation, we can point that general-purpose parallel software environments should be developed to allow programmers to focus on the numerical algorithm without worrying additionally about functions for calculation synchronization and memory management.

Acknowledgements

This work was supported by the Research and Academic Computer Network (NASK) and the Polish State Committee for Scientific Research (KBN) Grant No. 7 T11A 022 20.

References

Banks J. (Ed.) (1998): *Handbook of Simulation.* — New York: Wiley.

Bertsekas D. and Gallager R. (1992): *Data Networks*. — New Jersey: Pentice-Hall.

Di Z. and Mouftah H.T. (2002): *QUIPS-II: A simulation tool for the design and performance evaluation of diffserv-based networks*. — Comput. Comm., Vol. 25, No. 1, pp. 1125–1131.

Findeisen W., Bailey F.N., Brdyś M., Malinowski K. and Woźniak A. (1980): *Control and Coordination in Hierarchical Systems*. — London: Wiley.

HLA (1998) (High Level Architecture). Available at `http://www.dmso.mil/public/transition/hla/`

Jefferson D.R. (1985): *Virtual time*. — ACM Trans. Program. Lang. Syst., Vol. 7, No. 3, pp. 404–425.

Jefferson D.R. (1990): *Virtual time II: Storage management in distributed simulation*. — Proc. 9th Ann. ACM Symp. *Principles of Distributed Computing*, New York, USA, pp. 75–89.

Kheir N.A. (Ed.) (1996): *Systems Modeling and Computer Simulation*. — New York: Marcel Dekker.

Kreutzer W., Hopkins J. and van Mierlo M. (1997): *SimJava—A framework for modeling queueing networks in Java*. — Proc. 1997 *Winter Simulation Conf.*, Atlanta, pp. 483–488.

Lin Y.B. (1994): *Memory management algorithms for parallel simulation*. — Inf. Sci., Vol. 77, No. 1, pp. 119–140.

Low S. and Lapsey D.E. (1999): *Optimization flow control I: Basic algorithm and convergence*. — IEEE/ACM Trans. Networking, Vol. 7, No. 6, pp. 861–874.

Mehl H. (1991): *Speedup of conservative distributed discrete-event simulation methods by speculative computing*. — Adv. Parall. Distrib. Simul., SCS Simul. Ser., Vol. 23, No. 1, pp. 163–166.

Michalewicz Z. (1994): *Genetic Algorithms + Data Structures = Evolution Programs*. — Berlin-Heidenberg: Springer.

Misra J. (1986): *Distributed discrete-event simulation*. — Comput. Surveys, Vol. 18, No. 1, pp. 39–65.

Nicol D.M. and Fujimoto R. (1994): *Parallel simulation today*. — Ann. Oper. Res., Vol. 53, pp. 249–285.

Nicol D.M., Johnson M., Yoshimura A. and Goldsby M. (1997): *Performance modeling of the IDES framework*. — Proc. Workshop *Parallel and Distributed Simulation*, Lockenhaus, Austria, pp. 38–45.

Niewiadomska-Szynkiewicz E., Pośnik P., Bolek P. and Malinowski K. (1995): *Software environment for complex systems analysis and simulation*. — Prep. IFAC/IFORS/IMACS Symp. *Large Scale Systems: Theory and Applications*, London, pp. 147–152.

Niewiadomska-Szynkiewicz E., Karbowski A. and Malinowski K. (1996): *Predictive methods for real time control of flood operation of a multireservoir system—Methodology and comparative study*. — Water Res. Res., Vol. 32, No. 9, pp. 2885–2895.

Niewiadomska-Szynkiewicz E. (2002): *Software environment for simulation of flood control in multiple-reservoir systems*. — Proc. 5th Int. Conf. *Hydro-Science and Engineering, ICHE 2002*, Warsaw, Poland, pp. 2885–2895.

Niewiadomska-Szynkiewicz E. (2003): *Computer-based analysis and design of control mechanisms for flood operation in multireservoir systems*, In: Modelling and Control of Floods (J. Napiórkowski, Ed.). — Publications of the Institute of Geophysics, Polish Academy of Sciences, E–3 (365), pp. 97–117.

NS-2 (1995) (network simulator). — Available at `http://www.isi.edu/nsnam/ns/ns-documentation.html`

OMNeT++ (1992) (Objective Modular Network Testbed in C++). — Available at `http://www.hit.bme.hu/phd/vargaa/omnetpp.htm`

Pondarzewski A., Niewiadomska-Szynkiewicz E. and Żmuda M. (1999): *Software environment for distributed computing and simulation; user guide and applications*. — Tech. Rep. Inst. Contr. Eng., Warsaw University of Technology, No. 99–55 (in Polish).

Soliman H.M. (1999): *On the selection of the state saving strategy in time warp parallel simulation*. — Trans. Soc. Comp. Simul., Vol. 16, No. 1, pp. 32–36.

Received: 10 February 2003
Revised: 25 July 2003