PARSEVAL : a workbench for queueing networks parallel simulation

Philippe Mussi, Hery Rakotoarisoa

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET AUTOMATIQUE

PARSEVAL :
A Workbench for Queueing Networks Parallel Simulation

Philippe Mussi and Hery Rakotoarisoa

N˚ 2234

Avril 1994

PROGRAMME 1

Architectures parallèles,

bases de données,

réseaux et systèmes distribués

Rapport de recherche

1994

ISSN 0249-6399

PARSEVAL :
A Workbench for Queueing Networks Parallel Simulation

Philippe Mussi and Hery Rakotoarisoa[*]

Programme 1 — Architectures parallèles, bases de données, réseaux
et systèmes distribués
Projet Mistral

Rapport de recherche n˚2234 — Avril 1994 — 17 pages

Abstract: This report describes *PARSEVAL* (PARalellizing Simulations for performance EVALuation), an experimental distributed simulation workbench dedicated to queueing systems and running on a network of *Transputers* attached to a Sun workstation. It uses an implementation of the *conservative method* proposed by Chandy and Misra for the synchronization of processes.

After a brief description of the structure of *PARSEVAL*, the different parts that make it up and the implementation of the conservative mechanism, we give some considerations on its performance compared to the sequential simulator *QNAP2*. It is shown that speed-up depends on the structure of the model to be simulated and the hardware configuration used. Finally, we present current and planned developments around this workbench.

Key-words: Discrete event simulation, Distributed Simulation, Queueing networks

(Résumé : tsvp)

A version of this report has been presented at European Simulation Multiconference, Lyon (France), June 1993.

[*] {Philippe.Mussi}{Hery.Rakotoarisoa}@sophia.inria.fr

Unité de recherche INRIA Sophia-Antipolis
2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex (France)
Téléphone : (33) 93 65 77 77 – Télécopie : (33) 93 65 77 65

PARSEVAL :
Un atelier pour la simulation répartie de réseaux de files d'attente

Résumé : Ce rapport présente le système *PARSEVAL* (PARallélisation de simulations pour l'ÉVALuation de performances), un atelier expérimental de simulation répartie de réseaux de files d'attente. Ce système est implanté sur un réseau de *Transputers* attaché à une station de travail. Il utilise une variante de la méthode de synchronisation *conservatrice* proposée par Chandy et Misra.

Après une brève description de la structure de *PARSEVAL* et de son implantation, nous comparons ses performances avec celles du simulateur séquentiel *QNAP2*. Nous montrons les relations entre l'accélération obtenue, la structure du modèle simulé et la configuration matérielle choisie.

Nous présentons enfin les développements en cours et prévus autour de cet atelier expérimental.

Mots-clé : Simulation à événements discrets, Simulation répartie, Réseaux de files d'attente

1 Introduction

We are concerned with parallelizing queueing systems simulations on a distributed memory multiprocessor, particularly on a network of *Transputers*[1], in order to improve the performance. Processors communicate exclusively by a message passing mechanism on an interconnection network or on the physical links in the case of a Transputers network. Each of them simulates a part of the queueing network model, and works totally apart from the others, hence a *synchronization mechanism* is necessary to keep the whole simulation coherent.

Several synchronization techniques for distributed discrete-event simulations have been proposed in the literature [Fujimoto 1989], most of them based on either the *optimistic* paradigm or the *conservative* paradigm. The former, which includes the *time warp* method [Jefferson 1985], consists in running each process of the simulation (a processor may run several processes at the same time) without taking care of the others. When a process receives a message with timestamp less than its local clock, it will *roll back* to a simulation date where the whole system is coherent, *cancel* every side effects caused by erroneous messages already sent and *run forward* again. The later consists in stopping a process before the processing of the next event until it receives a message that ensures the correctness of this operation [Chandy and Misra 1979, Misra 1986]. The system may *deadlock* if the model under simulation includes a cycle of processes which are waiting for each other. The concept of *lookahead* was initially introduced by Chandy and Misra [Chandy and Misra 1979, Misra 1986] for synchronising processes in order to ensure the correctness of the simulation and to avoid deadlock. The lookahead is the ability of a process to predict a part of its future behaviour . In general, the lookahead is the simulation time interval from the current local clock value up to the simulation time of the next message that the process will send to another process. A *non zero* lookahead is expected for at least one process among those deadlocked in a cycle. Otherwise, the simulation time in the cycle will never progress and deadlock will never be broken. *Null messages,* which are only used for the synchronisation mechanism, carry information about *lookahead* between processes.

The experimental distributed simulator described here uses an implementation of the conservative method with two variations :

[1] INMOS' Transputers are processors designed for parallel programming. They can easily make up a network, thanks to their four bidirectional serial links.

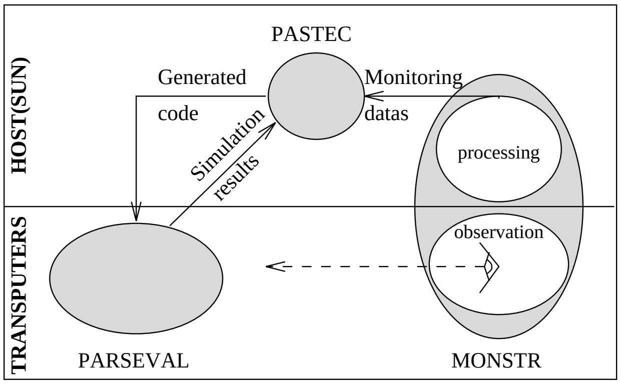

Figure 1: Global structure of PARSEVAL

- *Intelligent buffers* are used,
- Null messages are only generated when a process becomes blocked, and they are sent to each successor.

The global structure and the different parts of the PARSEVAL workbench are briefly described, followed by some considerations on its performance compared to the sequential simulator *QNAP2* [Potier and Véran 1984]. This article ends up with the description of current and planned developments around PARSEVAL.

2 Structure of the System

As shown on figure 1, the workbench is composed of three parts that are *PASTEC*, the graphical interface and simulation code generator, *PARSEVAL* itself, the simulation engine, and *MONSTR*, the monitoring system.

The queueing network model to be simulated is described by the user under *GSS*[2] upon which *PASTEC* is built. A ready-to-compile distributed simulation program expressed in *OCCAM* language is then generated by *PASTEC* for *PARSEVAL*. During the simulation run, the monitoring system *MONSTR* watches the Transputers and processes, and gathers monitoring data which is first analysed and reduced, and finally displayed using windows similar to those used by the standard X-windows tool $xload$.

3 The distributed Simulator

The distributed discrete-event simulator *PARSEVAL* is composed of a set of processes simulating the queueing model and a routing layer which takes charge of delivering messages. On each processor are placed a set of simulation processes and a message *router*. The number and the type of processes to be placed on each processor of the network must be defined by the user, in order to get a correct load balancing, both in terms of computing load and communication load. An automatic load-balancing module is under study for the future version of *PARSEVAL*.

3.1 General Purpose Processes

Besides the input-output process that allows *PARSEVAL* to keep in touch with its environment, the other service processes are *routers*. The routing network is a bidirectional ring, one direction being dedicated to customers and null messages, and the other to the monitoring messages. Its size may be defined according to the available hardware and the queueing model to be simulated. The choice of the bidirectional ring has been made to simplify the implementation and in order to reduce the perturbation introduced by the circulation of monitoring data.

[2] *GSS* has been developed as part of the ESPRIT II IMSE (Integrated Modelling System Environment) project. *The IMSE project is a collaborative research project supported by the EEC as ESPRIT project no 2143. It has been carried out by the following organizations: BNR Europe, Thomson CSF, Simulog S.A., University of Edinburgh, INRIA, IPK (Berlin), University of Dortmund, University of Pavia, SINTEF (University of Trondheim), University of Turin and University of Milan.*

3.2 The Customers Generator

It generates customers following a given probability law and sends them to the entries of the queueing network. The user has to pay special attention to the choice of the generator's parameters, to ensure that it will not send customers *too frequently* and lead the system to saturation (filling buffers)[3].

3.3 The One-Queue Simulator

It simulates a queue with its servers. Allowed queue types are first come first served queues (FCFS), preemptive last come first served queues (LCFS), preemptive queues scheduled according to the priorities of customers, queues with *quantum* time slices, and processor sharing queues (PS) where all customers are served immediately and share the servers. All queues are supposed to be infinite queues[4].

3.3.1 Intelligent buffers

Two extra processes are attached to the one-queue-simulator, the *receiver* at the entry and the *sender* at the output. They act as *intelligent* buffers limiting the number of circulating messages in the routing layer. Useless null messages (which are those immediately followed by another one or by a customer sent to the same destination) are eliminated. This is a variation from the initial conservative method described in [Chandy and Misra 1979] and [Misra 1986]. A message sent to the one-queue simulator is the one wearing the minimal timestamp.

3.3.2 Event generation and processing

Four event types are considered in the system. arr represents a customer's arrival, ss the beginning of a customer service, se the end of a customer service and qe the end of a *quantum* time-slice. Events are put in the event-list in an increasing date order. The event at the head is the one currently processed. An arr is created when receiving a customer from the receiver. Processing a qe, a se or a $preemption$ creates a ss. According to the scheduling type of the queue and the number of available

[3] Let us remark that such a behaviour is not specific to distributed discrete-event simulators. Simulation of unstable queueing networks does not really make sense.

[4] queue sizes are obviously limited by the available memory

servers, the processing of an arr may also generate a ss. A se or a qe is generated after the processing of a ss.

A special event named *non_valid_arr* is generated when the one-queue simulator receives a null message from the receiver. It means that no customer in the receiver buffers is safe to be simulated. The second variation from the initial method is that the one-queue simulator sends a null message to its successors only at the beginning of an idle period. The following algorithm shows our implementation of the conservative synchronisation method. This operating algorithm of a one-queue simulator in *PARSEVAL* assumes that the receiver sends messages in an increasing timestamp order. The receiver sends no message at all if the timestamp of the null message it may send is not greater than the previous one.

```
While not simulation end
  wait for a message from the receiver
  If the message is a customer
  then
    generate an $arr$
  else (the message is a null message)
    generate a non_valid_arr
  endif
  Do
    If the first event is not a non_valid_arr
    then
      process it
    else
      process the lookahead
      If prediction date $>$ date of the last null message sent
      then
        send the prediction null message
      endif
    endif
  until the first event is an arr or a non_valid_arr
  process all events wearing the same timestamp as
  the $arr$ or the non_valid_arr
end while
```

Let us note that the *prediction* is the minimal date (worn by the null message) at which the one-queue simulator will send a customer, while the *lookahead* is the time interval between the current simulation time when the null message is sent and the prediction itself.

3.3.3 Null messages and lookahead

The lookahead is an important performance parameter in a distributed simulation. The "bigger" it is, the more efficient the simulation is, although the lookahead must be comparable with the time interval between two messages in the neighbourhood of the process to have a very high performance [Wagner and Lazowska 1989]. The ratio of the time interval between two messages to the lookahead, called the *lookahead ratio* [Fujimoto 1988] is a good indication of expected performance. A low lookahead ratio indicates a good lookahead .

As long as the one-queue simulator is not blocked waiting for a message from its receiver, it will send no null message. It is indeed of no use to send a prediction when a customer can be sent immediately. The lookahead is calculated and the prediction null message sent, only while processing the *non_valid_arr*. In the current version of *PARSEVAL*, the same prediction (*non_valid_arr*'s date increased by the lookahead) is sent to all the successors. Most of the lookahead computations in *PARSEVAL* are based on the *future list* technique proposed by Nicol [Nicol 1988]. However, the destination of a customer is not drawn in advance.

FCFS: If *no server is available*, the prediction is the date of the first se in the event list. Otherwise, N service durations are drawn for the N next customers that will be served by the N servers available. Those future services drawn beforehand are stored in the future list. The date of the service end, namely the current date increased by the drawn service time, is computed for each of those customers. The prediction is the earlier date among the first se in the event list and the service ends.

LCFS: There is a *preemption* if no server is available when a new customer arrives. The prediction is here the minimum of the first se in the list and the current date increased by the minimal service duration.

PRIORITY: The arriving customer is directly served by an available server if any. Otherwise, when a customer with a priority p_{arr} arrives, there is a *preemption* if one of those currently served has a priority level strictly lower than p_{arr}. Let us make the following assumptions to show the calculation of the lookahead.

- The priority p of a an arriving customer ranges from 1 to P (the highest).
- There are S servers in the station and AS of them are available when computing lookahead. Service durations for the next AS arriving customers are drawn beforehand.
- pl_{ser} equals the lowest priority of customers currently served if $AS = 0$. Otherwise, $pl_{ser} = 1$.
- n_p is the number of customers currently served and having the priority p. For $p = 1$, n_p includes the AS potential customers if $AS \neq 0$.

The total number of service durations to be drawn beforehand (the minimal size of the future list when the lookahead is to be processed) including the AS durations above is $N = AS + \sum_{p=pl_{ser}+1}^{P}(S - n_p - \sum_{q=p+1}^{P} n_q)$. The prediction is the earliest date among the potential departure dates corresponding to those N services and the date of the first se in the event-list.

If P is large, the processing of the lookahead becomes time-consuming and the required memory may be huge.

In addition, the prediction is computed in the same way as in the LCFS case when the priority level is an integer ranging from 1 to ∞.

QUANTUM: If *no server is available*, the simplest solution is that the prediction is the date of the first se or qe in the event list. Really an optimal prediction might be computed by running the simulation fictively as far as possible, taking into account the currently served customers, those in the queue and those that may arrive. This solution is an efficient one for the successors but not for the null message sender because the longer the queue is, the more expensive the processing is.

If N *servers are available*, a service time is drawn beforehand for each of the N next arriving customers. Both described solutions are valid in this case, but the drawn service times must be taken into account for both of them.

PS (Processor sharing): If *no customer is currently served*, the prediction is the current simulation time increased by the minimal service time. Otherwise, the prediction is

- the earlier between the first se and $\{$*the current simulation time* $+$ *((the minimal service time* $*$ *(1* $+$ *the number of customers currently served))*$\}$, if the smallest among all the customers remaining service times is greater than the minimal service time,

- the first se in the event list otherwise.

4 PARSEVAL Performance

The simulation results given by *PARSEVAL* have been validated by comparison with the discrete-event simulation module of the commercial package *QNAP2* [Potier and Véran 1984]. Various types of service laws (constant, uniform, exponential, hyper-exponential, Erlang and Cox) and the above scheduling policies have been compared to those of *QNAP2* and showed the correctness of the results. The statistical module [Pawlikowski 1990] not being implemented yet, the accuracy of the measurements could not be assessed.

The performance depends essentially on the four following points:

- A one-queue simulator is a compilation unit that cannot be distributed on more than one processor. The optimal load balancing is then realized by putting one one-queue simulator on each processor, if enough processors are available.
- each CPU does not handle the communications on physical links[5]. This task is dedicated to the link communication manager (a separate device).
- Idle times of the one-queue simulators depend on the repartition of the circulating messages in physical time.
- The transmission delay of a message depends strongly on the respective place of sender and receiver.

Communications, although less expensive than processing load (internal communications require very few CPU resources and external communications are handled by link managers), introduce an extra processor load (routers) and idle periods in the processors' activity (locking periods for one-queue simulators). More thoroughly, the duration of the idle periods and the time spent waiting for communications depend on :

- the Transputers network configuration,

[5] except a memory access each 4th byte.

- the model (number of queues, links and cycles, scheduling , service laws and number of servers of each queue),
- the mapping of processes onto processors.

One can act only on the mapping of the processes to reduce this lost time. A given mapping generates an amount of null messages which can either minimise the locking times or compromise the overall system performancer. If processes communicate within the same processor, a large number of null messages is not necessarily wrong. Otherwise, a strong synchronisation can fastly decrease the performance if processors are weakly loaded.

The different types of processes (according to the model configuration and parameters) must be evaluated and efficient load-balancing strategies must be used in order to find the optimal mapping and, then, minimise the number of null messages. Nevertheless, let us note that the parallelisation of the simulation can be efficient for some models and not for others. This is due to the inherent parallelism in the structure of the model [Wagner and Lazowska 1989].

4.1 Comparison To QNAP2

QNAP2 simulation performance appears to be roughly similar on a SUN3/60 workstation and on a single T800 Transputer. However, QNAP2's Transputer version has not yet been completely validated, so we have chosen to base our comparisons on the commercial Sun version.

Let us consider four workshop models, in order to give some ideas of the kind of performance that we can get on *real world* models. Figure 11 shows four examples of workshop models: $wshop1$ exhibits a tree structure, $wshop3$ is linear, $wshop4$ is based on a uni-directional ring, $wshop2$ has a complex structure based on a line of rings.

Simulation times in PARSEVAL for these four models are plotted, versus the number of processors, on figures 12 to 15. For each case, optimal mappings of processes to processors have been chosen. In $wshop1$, each queue has roughly 80 customers to serve, in $wshop2$ 3500 customers, in $wshop3$ 100, in $wshop4$ 1200. QNAP2 simulation times are respectively 3.6, 197.0, 7.0 and 16.6 seconds.

Let us remark that, for loop-free models (models 1 and 3), PARSEVAL beats QNAP2, even with only one processor. As soon as loops are introduced (models 2

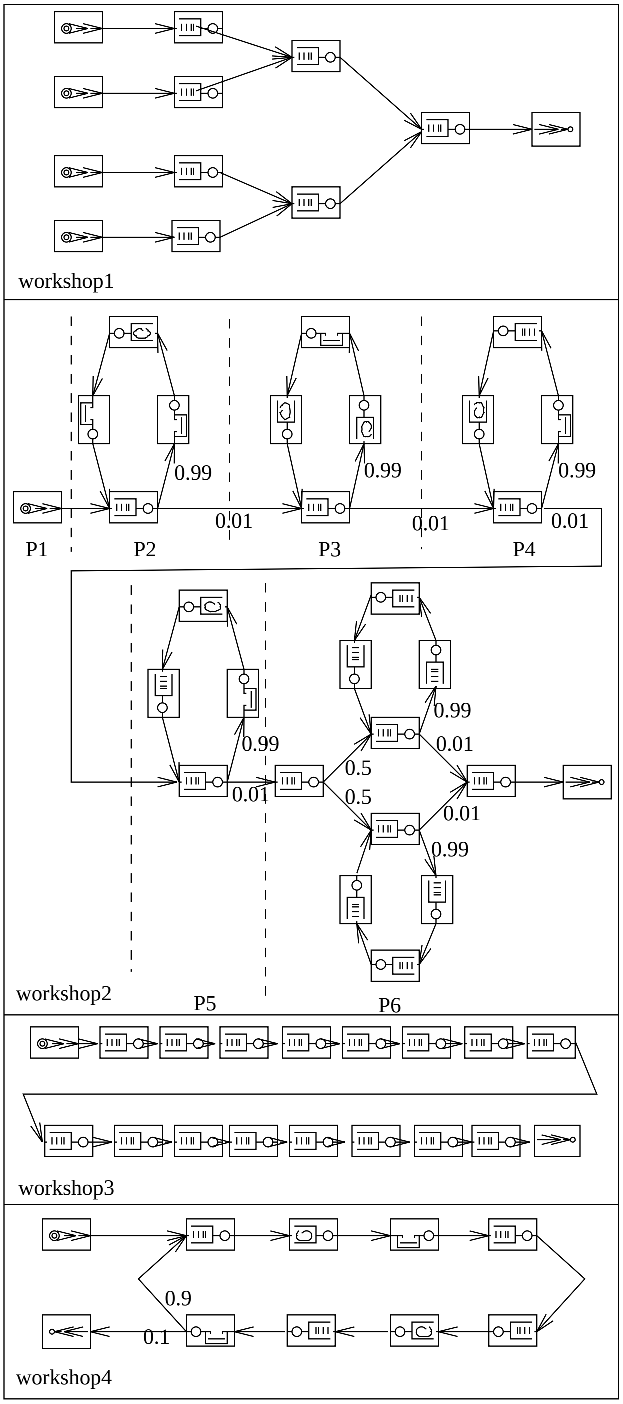

Figure 2: Four workshop models

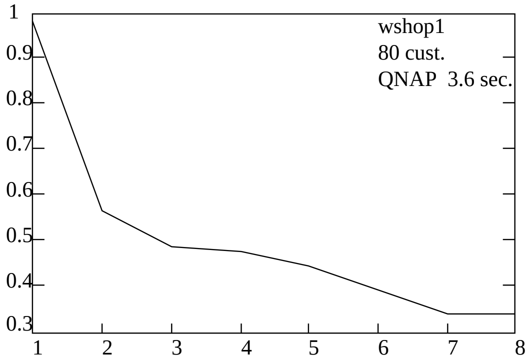

Figure 3: PARSEVAL simulation time for $wshop1$ vs. number of processors

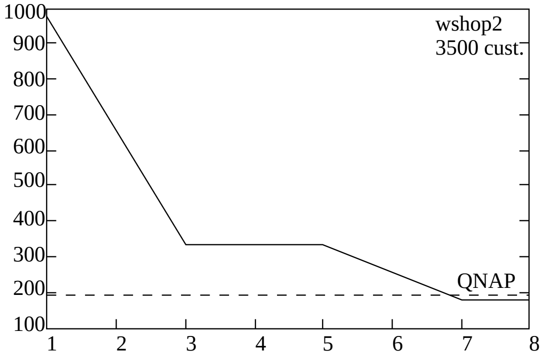

Figure 4: PARSEVAL simulation time for $wshop2$ vs. number of processors

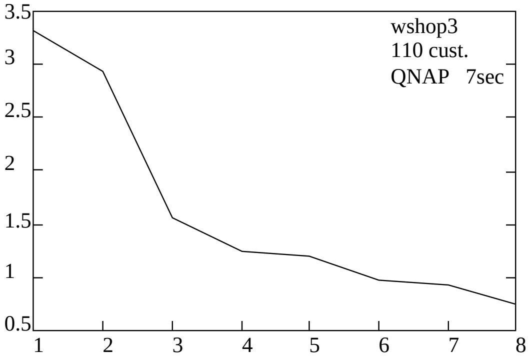

Figure 5: PARSEVAL simulation time for $wshop3$ vs. number of processors

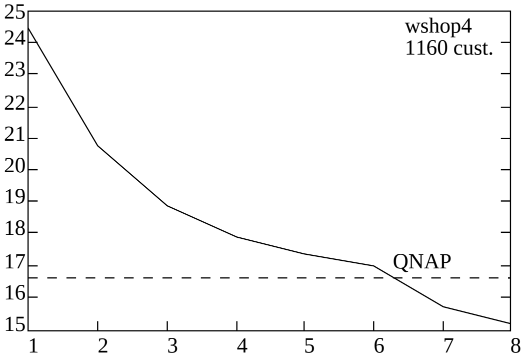

Figure 6: PARSEVAL simulation time for $wshop4$ vs. number of processors

and 4), the synchronisation overhead is no longer negligible, and PARSEVAL needs 6 processors to reach QNAP2 performance. For $wshop1$ and $wshop3$, *PARSEVAL* outperforms QNAP2 by a factor of 3.6 and 2 respectively, using a single processor. It is due to the fact that, on one hand *PARSEVAL* was designed for transputers, so that it uses its specific execution environment (hardware and micro-scheduler) for the simulation of a given model. On the other hand, the simulation of the same model by QNAP2 runs sequentially on a SUN workstation as well as on a transputer because its initial code was dedicated to sequential hardware. Furthermore, statistical operations done in QNAP2 are not yet implemented in *PARSEVAL*.

For $wshop2$, additional processors are useless, due to the ring communication structure of PARSEVAL (the optimal mapping is to put one P_i sub-model on each processor). However, with a smarter communication network, better speed-ups could be achieved, in particular by splitting the P_6 submodel between different processors.

5 Conclusion and Perspectives

We have described the distributed simulation workbench built around the distributed simulation engine PARSEVAL. This graphical workbench appeared to be useful for a quick testing of various implementation strategies, allowing to easily describe complex queueing networks and to study the performance effects of different mappings.

However, the current version of this workbench exhibited several drawbacks, essentially in terms of simulation performance. Hence, various additional developments have been initiated or are planned for the next version of *PARSEVAL* which will be written in *C* and run on the multi- transputer machine *MEIKO CS*:

- use of a better communication kernel (DeBruijn networks for example) and its optimisation.
- automatic sub-optimal initial mapping and dynamic mapping
- further reduction of the number of null messages
- optimal use CPUs' idle times (blocking intervals), for beforehand random drawings, statistical operations etc…

- integration of intrinsic parallelism evaluation modules

In addition, various extensions to PARSEVAL description language are under study, in order to be able to simulate queueing networks with synchronisation between queues.

References

[Chandy and Misra 1979] Chandy, K.M. and J. Misra. 1979. “Distributed Simulation : A Case Study in Design and Verification of Distributed Programs”. *IEEE Transactions on Software Engineering*, Vol.SE-5, n.5 (Sep)

[Fujimoto 1988] Fujimoto, R.M. 1988. “Lookahead in Parallel Discrete-Event Simulation”. In *Proceedings of the 1988 International Conference on Parallel Processing*.

[Fujimoto 1989] Fujimoto, R.M. 1989. “Parallel Discrete-Event Simulation”. In *Proceedings of the 1989 Winter Simulation Conference*.

[Jefferson 1985] Jefferson, D.R. 1985. “ Virtual Time”. *ACM Transactions on Programming languages and Systems*, Vol.7, n.3 (Jul)

[Misra 1986] Misra, J. 1986. “Distributed Discrete-event Simulation”. *ACM Computing Surveys*, Vol.18, n.1 (Mar)

[Mussi and Rakotoarisoa 1991] Mussi, Ph. and Rakotoarisoa, H. 1991. “PARSEVAL: PARallélisation sur réseaux de Transputers de Simulations pour l’EVALuation de performances”. Rapport technique n.131 INRIA Sophia Antipolis (Jul).

[Nicol 1988] Nicol, D.M. 1988. “Parallel Discrete-Event Simulation of FCFS Stochastic Queueing Networks”. *Parallel Programming : Experiences with Applications, Langages and Systems*, ACM SIGPLAN Notices, 23 (Sep): pp.124-137.

[Pawlikowski 1990] Pawlikowski, K. 1990. “Steady-State Simulation of Queueing Processes: A Survey of Problems and Solutions”. *ACM Computing Surveys*, Vol. 22, No. 2 (Jun).

[Potier and Véran 1984] D. Potier and M. Véran. 1984. “QNAP2 : A Portable Environment for Queueing Systems Modelling”. Rapport de recherche n.314 INRIA Rocquencourt (Jun).

[Wagner and Lazowska 1989] Wagner, D.B. and E.D. Lazowska. 1989. “Parallel Simulation of Queueing Networks: Limitations and Potentials”. *Performance Evaluation Review*, Vol. 17 #1, May 1989.

Unité de recherche INRIA Lorraine, Technôpole de Nancy-Brabois, Campus scientifique,
615 rue de Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399