# AKAROA-2: EXPLOITING NETWORK COMPUTING BY DISTRIBUTING STOCHASTIC SIMULATION

G. C. Ewing, K. Pawlikowski and D. McNickle†
Department of Computer Science and †Management
University of Canterbury
Christchurch
New Zealand
Email: {greg,krys}@cosc.canterbury.ac.nz
don@mgmt.canterbury.ac.nz

## KEYWORDS

Stochastic Simulation, Distributed and Parallel simulation, Networked Computing, Multiple Replications in Parallel

## ABSTRACT

Research in distributed and parallel simulation has been almost entirely focused on partitioning the model of the system being studied and simulating the parts on different processors. The main challenge of this technique concerns the logical synchronisation of these interdependent simulated subprocesses as they autonomously evolve in time.

There is another approach to distributed stochastic simulation – known as Multiple Replications In Parallel, or MRIP – that has attracted much less attention, despite being able to take full advantage of the distributed computing power of multiple networked processors without suffering from inherent problems of the former approach.

In MRIP, the computers of the network run independent replications of the whole simulation process, generating statistically equivalent streams of simulation output data. These data streams are fed to a global data analyser responsible for analysis of the final results and for stopping the simulation when the results reach a satisfactory accuracy.

In this paper we discuss AKAROA-2, the latest version of a fully automated simulation tool designed for running distributed stochastic simulations in MRIP scenario in a local area network environment. It was designed within the AKAROA project, at the University of Canterbury in Christchurch, New Zealand. We focus on programming issues associated with the design of AKAROA-2, as well as on the main advantages and limitations of it as simulation tool.

## 1 INTRODUCTION

Over the last decade discrete-event simulation has become perhaps the most common tool used by engineers of various disciplines of science and engineering for studying and evaluating performance of various systems and processes. This is the result of broad proliferation of powerful and cheap computers, and significant achievements in software technology. There is easy access to various user-friendly simulation packages in which traditional discrete-event simulation modeling is supported by various techniques adopted from artificial intelligence. This climate has fostered a popular impression that simulation is mainly an exercise in computer programming.

Traditionally, the main research activity in the area of stochastic simulation has been focused on developing methods for concurrently executing loosely-coupled parts of large simulation models on multi-processor computers, or multiple computers of a network. In the context of stochastic simulation this approach is known as Single Replication in Parallel, or SRIP (Pawlikowski et al. 1994), since each simulated subprocess evolves in time only once. The main challenge of this approach is to find a robust and efficient method of synchronization between processes simulated on different processors, since the evolution of each sub-model of the original model often depends on events occuring in other sub-models. Sophisticated techniques have been proposed to solve this and related problems, surveyed for example in (Fujimoto 1990; Bagrodia 1996; Hellekalek 1998). In addition to efficiently managing the execution of large partitioned simulation models, this approach can also offer reasonable speedup of simulation, provided that a given simulation model is sufficiently decomposable. Unfortunately, this feature is not frequently observed in practice, thus the efficiency of this kind of distributed simulation is strongly model-dependent (Wagner and Lazowska 1989), and has its well known limitation; see for example (Bagrodia 1996).

An inherent problem of quantitative stochastic simulation, conducted for assessing performance quality of simulated systems, is the issue of credibility of the final results. Since any such simulation can be regarded as a (simulated) statistical experiment, it is necessary to generate a sufficient amount of output data, and to apply appropriate methods of analysis, to ensure that the final results have acceptable precision.

There is general agreement that the only practical way to ensure statistical credibility of the results is by sequential analysis of simulation output data (Heidelberger and Welch 1983; Law and Kelton 1991).

In sequential analysis, the simulation is continued as long as the number of collected observations, i.e. the sample size, is not sufficient for reducing statistical errors of the results below an acceptable threshold. Sequential analysis of correlated observations, for example for assessing precision of steady-state performance measures, can require quite elaborate techniques and most of research in this area has been so far focused on analysis of steady-state means and percentiles; see for example (Pawlikowski 1990) and (Raatikainen 1990; Lee et al. 1999).

Direct application of sequential estimation is hindered in practice by the fact that, even in the case of moderately complex models, it can require very long simulation runs. A simple solution to this problem is to run stochastic sequential simulation in parallel, on multiple processors acting as identical simulation engines, cooperating in generation of statistically identical observations and submitting them to a global analyser for statistical analysis. This approach to distributed stochastic simulation is known as Multiple Replications In Parallel (MRIP) (Pawlikowski et al. 1994). Note that SRIP and MRIP are not two competing alternative techniques for distributed simulation, since both could be applied at the same time. If a cluster of processors is used in SRIP simulation, and such clusters are replicated according to MRIP, then the final speedup should be the product of speedups achievable when using SRIP and MRIP separately.

The first implementations of the MRIP scenario in simulation packages were independently reported by research teams from the Purdue University in USA (Rego and Sunderam 1992; Sunderam and Rego 1991) and the University of Canterbury in New Zealand (Pawlikowski and Yau 1992; Yau and Pawlikowski 1993) which, respectively, designed EcliPse and Akaroa. In this paper we discuss Akaroa-2, the latest version of our fully automated simulation tool designed for running distributed stochastic simulations in local area networks using MRIP. We focus on programming issues associated with the design of Akaroa-2, and also discuss its advantages and limitations.

# 2 USER INTERFACE

Akaroa-2 is a re-designed and improved version of the original Akaroa (described in (Yau and Pawlikowski 1993), and referred to here as Akaroa-1). Like Akaroa-1, Akaroa-2 is written in object-oriented C++ and runs on multiple Unix workstations connected by a local area network. It offers fully automated parallel execution of ordinary simulation programs for (automated) analysis of mean values[1], and automated stopping of simulation when the final results reach the required precision. It can be applied to both terminating and steady-state sequential stochastic simulation.

Akaroa-2 is designed mainly for use with simulation programs written in C or C++, but can be easily adapted to work with other languages and systems. It has been used, for example, with Pascal programs and Ptolemy models.

## Programming Interface

Ease of adapting existing simulation programs was one of the main design objectives of both Akaroa-1 and Akaroa-2. Any simulation program which produces a stream of observations, and is written in C or C++ or can be linked with a C++ library, can be converted to run under Akaroa-2 by adding to the existing code as little as one procedure call per analysed performance measure. In the simplest case, at the point where the program generates an observation, the observation is passed to Akaroa-2 by making the call

```
AkObservation(value)
```

The AkObservation routine is part of the Akaroa-2 library with which the simulation program is linked.

If more than one parameter (performance measure) is being studied, an additional call has to be made before the simulation begins:

```
AkDeclareParameters(n)
```

where $n$ is the number of parameters that are to be analysed. Then, when an observation used in analysis of the parameter $i$ is generated, $i$ is passed as an extra argument of AkObservation:

```
AkObservation(i, value)
```

The Akaroa-2 system will stop the simulation itself when the required precision of the estimated parameter(s) has been reached.

## Shell Command Interface

Once the simulation program is compiled, the same executable may be used in two different ways: *stand-alone mode* and *MRIP mode*. The primary purpose of stand-alone mode is for debugging the simulation program. Debugging techniques such as writing diagnostic output and using a source-level debugger are much easier to apply in this mode than they are in MRIP mode.

The *akrun* program is used to start a simulation in MRIP mode. An example of the use of *akrun* is:

```
% akrun -n 3 mm1 0.5
```

This command specifies that a simulation program called *mm1* is to be run (with an argument of 0.5) using 3 simulation engines.[2] The user is informed of the progress of the simulation by a transcript such as the following:

[1] Work is being done to add sequential analysis of proportions and quantiles to Akaroa-2 (Lee et al. 1999; Lee et al. 1999a).

[2] In this example, *mm1* a program performs steady-state simulation of an M/M/1 queueing system with a specified traffic intensity.

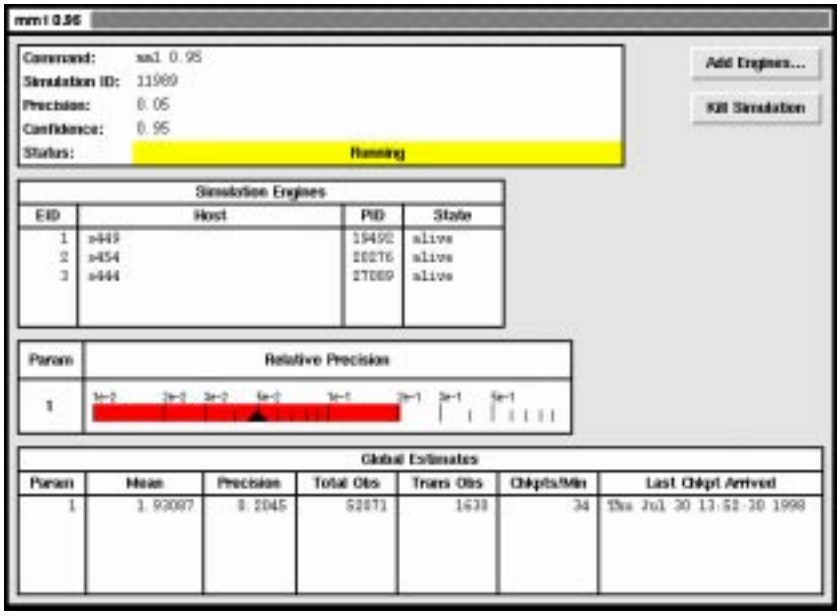


Figure 1: Akgui window showing the status of a simulation being executed.

```
Simulation ID = 10152
Simulation engine started: host = host1
Simulation engine started: host = host2
Simulation engine started: host = host3
Par Estimate Delta  Conf Var       Count Trans
1   0.2054   0.0096 0.95 2.141e-05 13236 828
```

The first few lines confirm that three simulation engines have been launched on the hosts named *host1*, *host2* and *host3*. As the simulation runs, Akaroa-2 performs sequential steady-state analysis of the data based on the methods described in (Pawlikowski 1990), adapted to MRIP; see (Pawlikowski et al. 1994).

At the end, the final results are displayed. *Estimate* and *Delta* are the final estimates of the parameter analysed and the half-width of its confidence interval; *Var* is the variance of the estimate; *Count* is the total number of observations submitted for analysis by all three simulation engines; and *Trans* is the total number of observations discarded during the transient phase (before analysis of steady-state was initiated).

The shell-command interface to Akaroa-2 also provides facilities for specifying options such as the desired precision and confidence level of results, selecting different analysis methods, enquiring about the status of running simulations, and adding more engines to an existing simulation; see (Ewing et al. 1998).

### Graphical User Interface

A graphical user interface, *akgui*, has been recently added as an alternative to the shell command interface. The main akgui window shows a list of hosts available for running simulation engines and a list of currently running simulations. New simulations may be started by entering parameters into a dialog box, and existing simulations may be opened up to show their progress.

Figure 1 shows a simulation progress window. The bar graph dynamically displays the relative precision of each performance measure, showing its convergence to the requested precision (which in this case is 0.05, at a confidence level 0.95). The same information, along with the current estimate of each performance measure, is presented numerically in the *Global Estimates* table.

## 3 ARCHITECTURE

The main components of Akaroa-2 are the *akmaster*, the *akslaves*, *akrun* and the *simulation engines*. The relationships between these components are shown in Figure 2. Each bold-outlined box represents one Unix process, and the connecting lines represent TCP/IP stream connections.

The *akmaster* process coordinates the activity of all other processes initiated by Akaroa-2. It launches new simulations, maintains state information about running simulations, performs global analysis of the data produced by simulation engines, and makes simulation stopping decisions.

*Akslave* processes run on hosts which are to run simulation engines. The sole function of the akslave is to launch simulation engine(s) on its host as directed by the akmaster. The akslave processes have been introduced because other methods of launching remote processes under Unix (for example, by using *rsh*) tend to be slow and unreliable.

The *akrun* program, as already mentioned, is used to initiate a simulation. It first contacts the akmaster process, obtaining its host name and port number from a file left by the akmaster in the user's home directory. For each simulation engine requested, the akmaster chooses a host from among

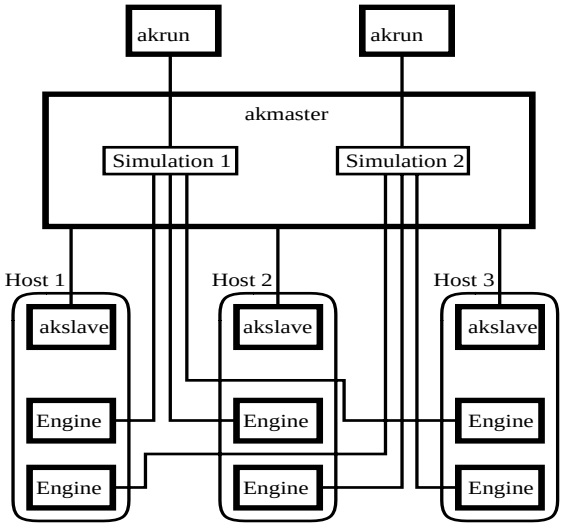


Figure 2: Akaroa-2 architecture, with two simulations in progress, each using three engines on separate network hosts.

those hosts on the LAN which are running akslave processes. It instructs the akslave on that host to launch an instance of the user's simulation program, passing on any specified arguments. The first time the simulation program calls one of the Akaroa-2 library routines, the simulation engine opens a connection to the akmaster process and identifies the simulation to which it belongs, so that the akmaster can associate the connection with the appropriate simulation data structure.

Following the principles of sequential stochastic simulation (Pawlikowski 1990), each engine performs sequential analysis of its own data to form a *local estimate* of each performance measure. At more or less regularly determined *checkpoints*, the engine sends its local estimates to the akmaster process, where the local estimates of each performance measure from all engines are combined to give a set of *global estimates*.

Whenever a new global estimate is calculated, the relative half-width of its confidence interval at the requested confidence level is computed, and compared with the requested precision. When the precision of all analysed performance measures becomes satisfactory, the akmaster terminates all the simulation engines, and sends the final global estimates to the akrun process, which in turn reports them to the user.

### Interprocess Communication.

Akaroa-1 used UDP/IP datagrams to communicate between processes. Since the UDP protocol does not guarantee reliable packet delivery, Akaroa-1 spent much effort attempting to deal with issues of packet loss and duplication. The system tended to be unreliable and difficult to manage. If a process failed to respond within an arbitrary timeout, it was hard to tell whether it had died or was simply taking longer than usual to respond due to host and/or network loading.

In Akaroa-2, all interprocess communication is via TCP/IP stream connections, which provide reliable, sequenced, non-duplicated delivery of messages. This has greatly simplified the communication subsystems of Akaroa-2, and made it much more dependable. The only disadvantage is a limit under some versions of Unix on the total number of processes that can participate in a single Akaroa session. For Solaris this limit is about 1000, which should not be a problem in practice.

### Random Numbers.

In MRIP, each simulation engine must use pseudo-random numbers (PRNs) independent from those used by other engines. Consequently, in Akaroa-2, random number generation is not left to the user's simulation program. Instead, the akmaster process is given full control over PRNs used by different simulation engines.

Currently Akaroa-2 uses 25 multiplicative congruential generators with modulus $M = 2^{31} - 1$ whose multipliers are taken from the top of the list of over 200 generators recommended in (Fishman and Moore III 1986), plus another 25 whose multipliers are the inverses (mod $M$) of the first ones. The akmaster process concatenates these 50 sequences into one sequence with a total length of about $10^{11}$ numbers. This is the limit on the total length of a simulation that can be executed under Akaroa-2.

Since it would be very inefficient for a simulation engine to have to communicate with the akmaster process every time it wanted a PRN, the akmaster allocates *blocks* of PRNs to simulation engines. The first time an engine requests a PRN, it receives a tuple $(k, i, n)$ representing a segment of the total sequence of numbers of length $n$, beginning at number $i$ of the sequence generated by multiplier $A_k$. The engine has its own local generic PRN generator and a copy of the table of multipliers. It initialises the generator by setting $x = A_k^i$ (mod $M$) and generates numbers locally until all $n$ assigned numbers have been used, whereupon it requests a new block from the akmaster.

## 4 PERFORMANCE EVALUATION OF AKAROA-2

Credibity of the final results of performance evaluation studies based on stochastic simulation and conducted with a help of a simulation tool should be the main issue addressed by designers of any tool of practical significance. One cannot achieve this without providing statistically sound source(s) of randomness and ensuring that the applied methods of analysis of simulation output data are able to produce correct results.

Within the framework of MRIP, different methods of sequential statistical analysis can be applied to the simulation output data. To test the accuracy of these methods, we have proposed a special methodology for coverage analysis (Pawlikowski et al. 1998). Since analysis of output data during steady-state simulation is more challenging than from terminating simulation, let us focused at the former.

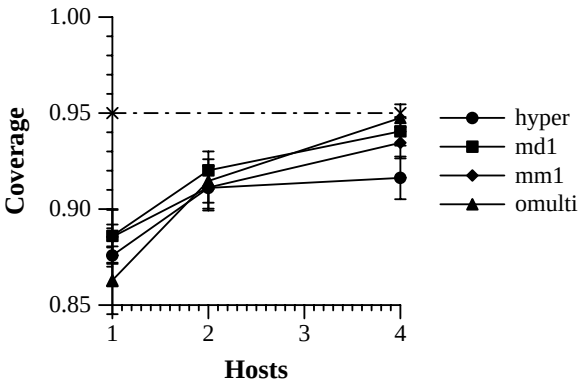


Figure 3: Coverage of SA/HW (steady-state analysis of the mean delay) applied to four models: M/M/1, M/D/1, $M/H_2/1$ ("hyper") and an open queueing network modelling a CPU with two disks ("omulti").

Our main results of analysis of the sequential methods proposed for steady-state simulation have been published in (McNickle et al. 1996; Ewing 1997; Lee et al. 1998; Lee et al. 1999). They showed an interesting phenomenon: in MRIP, the quality of sequential methods of data analysis *improves* with the degree of parallelisation. As an example, Figure 3 shows the results of coverage analysis obtained for a sequential method of analysis of steady-state mean values known as SA/HW, in its MRIP version (Pawlikowski et al. 1994). One can clearly see that the quality of this method improves as the number of parallel simulation engines increases.

These encouraging coverage results also verify the selection of PRN generators used in Akaroa-2, although we are aware of their limitations (see (Hellekalek 1998)) and are searching for better generators.

The main reason for using MRIP is its expected speedup. The (average) speedup $S$ can be defined as the ratio of the (average) run length of simulation executed on a single processor and the (average) run length of simulation on one of $P$ processors participating in MRIP simulation. In both cases the run length can be measured by the number of observations needed to be collected for stopping the simulation with the required level of precison of the final results. Assuming a fine granularity of run length (i.e. a small distance between consecutive checkpoints), the maximum speedup obtainable when running an MRIP simulation on $P$ computers is governed by a truncated Amdahl's law (Pawlikowski and McNickle 1998), which states that

$$S = \begin{cases} 1/(s+(1-s)/P), & \text{if } P \le N(1-s)/n_1 \\ N(1-s)/n_1, & \text{otherwise.} \end{cases} \quad (1)$$

where $s$ represents the (average) fraction of the simulation run length which cannot be parallelised; $N$ is the mean total number of observations needed to reach the required precision of results; and $n_1$ is the (average) number of observations needed to reach the first checkpoint.

As discussed in (Pawlikowski et al. 1998), in the case of terminating simulation $s = 0$, and the speedup is equal to the number of simulation engines used, as long as $P \le (N - n_1)/n$. Depending on the method of simulation dataoutput analysis, in steady-state simulation $s \ge 0$, so the speedup can be less than linear.

All proven methods of steady-state analysis that are currently implemented in Akaroa-2 (or planned to be implemented (Lee et al. 1999)) depend crucially on discarding of all observations that do not represent steady-state, thus for all of them $s > 0$. When $P \ge (N - n_1)/n$, no further speedup is possible, since none of the engines can reach more than one checkpoint before the simulation is stopped.

Figure 4 shows speedup achieved in Akaroa-2 as the number of simulation engines increases. These results are from a simulation for estimating the mean delay in an M/M/1 queueing system loaded at 0.9, using the SA/HW method of analysis.

The dotted line represents the maximum speedup as given by Equation 1. On the basis of our experiments, in this case $N$ is about 2,000,000, $n_1$ is about 500 and $s$ is about 0.025%. Thus, Equation 1 says that the maximum theoretically possible speedup of the simulation reported in Figure 4, if an unlimited number of processors were available, would be about 4000.

There are two sources for the shortfall in actual speedup shown in Figure 4. One is the checkpoint spacing, which determines the granularity of the run lengths and causes deviations from the theoretical maximum speedup predicted by Equation 1. Some improvement in speedup could be obtained by taking checkpoints more often, but some methods of simulation output data analysis, including SA/HW, impose limits on how far the checkpoint spacing can be reduced before incurring large performance penalties and/or compromising the accuracy of the analysis. Overcoming this problem is one of our current research areas.

The other source, more significant in this example, is the fact that the simulation results produced by the single- and multi-processor cases are not of the same quality. When more processors are used, SA/HW produces better estimates of the precision of the results, leading to better coverage. However, this comes at the expense of longer overall run lengths, and therefore some loss of speedup. This tradeoff should be borne in mind when judging the benefits of using MRIP.

# 5 CONCLUSIONS

In this paper we discussed the main issues of distributed stochastic simulation using the MRIP approach, focusing on aspects of its implementation in Akaroa-2, the latest version of a simulation package developed in the Department of Computer Science at the University of Canterbury in Christchurch, New Zealand. Without requiring the use of any parallel programming techniques, Akaroa-2 automatically distributes simulation models over a number of computers linked by a local area network, and controls the simulation run length so as to

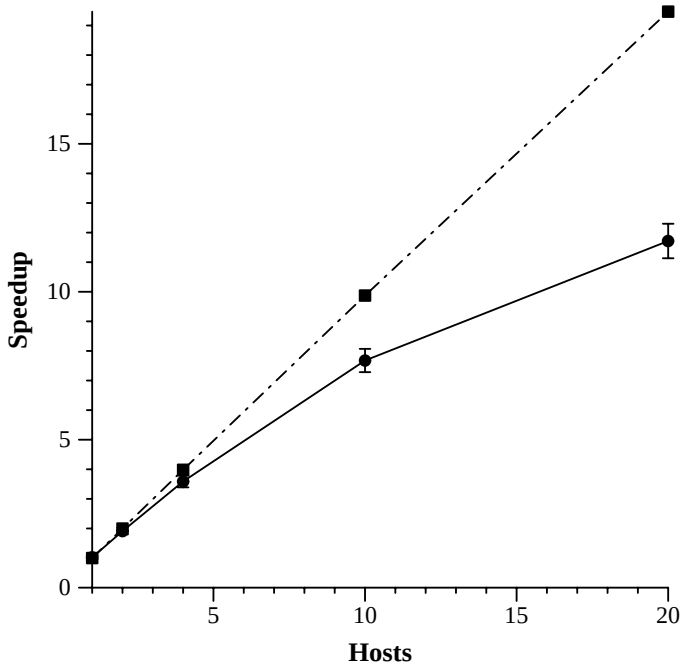


Figure 4: Theoretical (dotted) and actual (solid) speedup as a function of the number of simulation engines.

produce final results having a specified precision.

From the user's point of view, distributed stochastic simulation using MRIP appears to be a very attractive application of network computing. It makes good use of the distributed computing power of processors linked by a local area network, significantly speeding up simulation experiments on dynamic stochastic systems regardless of the internal structure of their models. This is done in a way which is transparent for users.

It is important to realise that MRIP is not an alternative to traditional methods of distributed simulation, but that the two can complement each other.

While successful implementation of SRIP crucially depends on finding a solution to the problem of synchronisation between simulated sub-processes, the main challenge in the case of MRIP lies in the area of statistics. Very little is known about the properties of linear combinations of estimators used in analysis of output data generated by multiple simulation engines.

Research activities in the Akaroa project are continuing. They are currently aimed at increasing the functionality of Akaroa-2. We are also investigating the possibility of applying MRIP-like techniques in other areas of scientific computation besides stochastic simulation.

# ACKNOWLEDGEMENT

Partial financial support for this research has been provided by the University of Canterbury (Grant U6301).

# REFERENCES

Bagrodia, R. L. 1996. "Perils and Pitfalls of Parallel Discrete Event Simulation." In *Proceedings of 1994 Winter Simulation Conference WSC'94*, IEEE Press, 136-143.

Ewing, G.; D. McNickle; and K. Pawlikowski. 1997. "Multiple Replications in Parallel: Distributed Generation of Data for Speeding up Quantitative Stochastic Simulation." In *Proceedings of 15th IMACS World Congress on Scientific Computation, Modelling and Applied Mathematics*, Berlin (Aug.), 379-402.

Ewing, G.; K. Pawlikowski; and D. McNickle. 1998. *Akaroa 2: User's Manual*. Technical Report TR-COSC 07/98, Department of Computer Science, University of Canterbury, Christchurch, New Zealand.

Fishman, G. S. and L. R. Moore III. 1986. "An Exhaustive Analysis of Multiplicative Congruential Random Number Generators with Modulus $M = 2^{31} - 1$." *SIAM J. Sci. Stat. Comput.*, 7: 24-45.

Fujimoto, R. 1990. "Parallel Discrete Event Simulation." *Comms. of the ACM*, 33 (Oct.): 30-60.

Heidelberger, P. and P. Welch. 1983. "Simulation Run Length Control in the Presence of Initial Transient." *Oper. Res.*, 31: 1109-1144.

Hellekalek, P. 1998. "Don't Trust Parallel Monte Carlo." In *Proceedings of 12th Workshop on Parallel and Distributed Simulation, PADS'98*, Banff, Canada (May), 82-89.

Law, A. M. and W. D. Kelton. 1991. *Simulation Modeling and Analysis.* McGraw-Hill.

Lee, J.-S. R.; D. McNickle; and K.Pawlikowski. 1999. "Confidence Interval Estimators for Coverage Analysis in Sequential Steady-

state Simulation." In *Proceedings of the Twenty Second Australasian Computer Science Conference*, Auckland, New Zealand, 21 (Jan.): 87-98.

Lee, J.-S. R.; D. McNickle; and K.Pawlikowski. 1999a. "Quantile Estimation in Sequential Steady-State Simulation". In *Proceedings of the European Simulation Multiconference ESM'99*, (Warsaw, Poland, June). SCS.

McNickle, D.; K. Pawlikowski; and G. Ewing. 1996. "Experimental Evaluation of Confidence Intervals in Sequential Steady-State Simulation." In *Proceedings of the Winter Simulation Conference WSC'96*, San Diego (Dec.), 1996 IEEE Press, 382-389.

Pawlikowski, K. 1990. "Steady-State Simulation of Queueing Processes: A Survey of Problems and Solutions." *ACM Computing Surveys*, 2: 123-170.

Pawlikowski, K. and V. Yau. 1992. "An Automatic Partitioning, Runtime Control and Output Analysis Methodology for Massively Parallel Simulations." In *Proceedings of the European Simulation Symp. ESS'92*, Dresden, SCS, 135-139.

Pawlikowski, K.; V. Yau; and D. McNickle. 1994. "Distributed Stochastic Discrete-Event Simulation in Parallel Times Streams". In *Proceedings of the Winter Simulation Conference WSC'94*, IEEE Press, 723-730.

Pawlikowski, K.; G. Ewing; and D. McNickle. 1998. "Performance Evaluation of Industrial Processes in Computer Network Environments." In *Proceedings of European Conference on Concurrent Engineering ECEC'98*, Erlangen, Germany (Apr.), 129-135.

Pawlikowski, K.; D. McNickle; and G. Ewing. 1998. "Coverage of Confidence Intervals in Sequential Steady-State Simulation." *Simulation Practice and Theory*, 6: 255-267.

Pawlikowski, K. and D. McNickle. 1998. "Distributed Stochastic Simulation and Amdalh's Law." Submitted.

Raatikainen, K. E. E. 1990. "Sequential Procedure for Simultaneous Estimation of Several Percentiles." *Transactions on Society for Computer Simulation*, 1: 21-44.

Rego, V. J. and V. S. Sunderam. 1992. "Experiments in Concurrent Stochastic Simulation: the EcliPSe Paradigm." *Journal of Parallel and Distributed Computing*, 14: 66-84.

Sunderam, V. S. and V. J. Rego. 1991. "EcliPse: A System for High Performance Concurrent Simulation." *Software-Practise and Experience*, 11: 1189-1219.

Wagner, D. B. and E.Lazowska. 1989. "Parallel Simulation of Queueing Networks: Limitations and Potentials." *Performance Evaluation Review*, 17: 146-155.

Yau, V. and K. Pawlikowski. 1993. "AKAROA: a Package for Automatic Generation and Process Control of Parallel Stochastic Simulation." In *Proceedings of 16th Australian Computer Science Conference*, Australian Computer Science Comms., 71-82.

# BIOGRAPHIES

GREG EWING is a research associate in the Department of Computer Science at the University of Canterbury, in Christchurch, New Zeland. He has completed a Ph. D. on geographical informaiton systems. His research interests include programming languages, 3D graphics, graphical user interfaces and parallel processing.

KRZYSZTOF PAWLIKOWSKI is an Associate Professor (Reader) in Computer Science at University of Canterbury, in Christchurch, New Zealand. He received his PhD degree in Computer Engineering from the Technical University of Gdansk, Poland. The author of over 80 research papers and four books. His research interests include stochastic simulation, distributed processing, ATM, optical and wireless telecommunication networks, and teletraffic modelling. Senior member of IEEE.

DON MCNICKLE is a Senior Lecturer in Management Science on the Department of Management, University of Canterbury, in Christchurch, New Zealand. His research interests include queueing theory and statistical aspects of simulation. He received a PhD in Mathematics from the University of Auckland, New Zealand. Member of ORSA.