Proceedings of the 2001 Winter Simulation Conference
B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, eds.

BENEFITS FROM SEMI-ASYNCHRONOUS CHECKPOINTING FOR TIME WARP SIMULATIONS OF A LARGE STATE PCS MODEL

Andrea Santoro
Francesco Quaglia

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, ITALY

ABSTRACT

Checkpointing overhead is a major obstacle for the effectiveness of Time Warp parallel discrete event simulators. Semi-asynchronous checkpointing is a recent solution to tackle this obstacle for Time Warp simulations on distributed memory systems based on Myrinet. In this solution, checkpoint operations are offloaded from the host CPU and are charged to a DMA engine on board of Myrinet network cards. In this paper we report an empirical evaluation of the benefits from semi-asynchronous checkpointing for Time Warp simulations of a large state Personal Communication System (PCS) model. PCS simulation models are typically characterized by high communication locality among the LPs hosted by the same machine, therefore the hardware on board of the Myrinet cards is typically underutilized if used to support exclusively communication. We show that the execution speed of Time Warp simulations of a large state PCS model can be increased when semi-asynchronous checkpointing is adopted.

1 INTRODUCTION

Time Warp parallel discrete event simulators are based on checkpointing and rollback recovery techniques to ensure causally consistent execution of simulation events at each Logical Process (LP) (Jefferson 1985). It is widely recognized that a central factor affecting the performance of this type of simulators is the way in which checkpoint operations are executed.

Commonly, checkpoint operations are charged to the CPU and the reduction of the checkpointing overhead has been pursued by the use of checkpointing strategies based on *infrequent* or *incremental* saving of the LP state vector, see for example (Bauer and Sporrer 1993, Bellenot 1992, Fleischmann and Wilsey 1995, Lin et al. 1993, Quaglia 1999, Quaglia 2001, Ronngren and Ayani 1994, Skold and Ronngren 1996, Steinman 1993, Unger et al. 1993). These solutions pay the price of an increase in the expected rollback latency since a state to be recovered might not be available, in which case it must be reconstructed during the rollback phase. The "best suited" tradeoff is typically achieved through adequate tuning of the proper parameter(s) of the checkpointing strategy.

A completely different approach to the implementation of checkpoint operations has been recently proposed in (Quaglia and Santoro 2001) for the case of Time Warp simulation on distributed memory systems based on Myrinet. Specifically, the work in (Quaglia and Santoro 2001) presents a Checkpointing and Communication Library (CCL) that exploits data transfer potentiality offered by programmable DMA engines on board of Myrinet network cards to support not only communication but also checkpoint operations. In this way, checkpoint operations are offloaded from the CPU, thus allowing the CPU itself to perform other simulation specific operations (e.g. event list's update, event execution) while checkpointing is in progress.

On the other hand, DMA based checkpointing could suffer from data inconsistency whenever the content of a state buffer is accessed for further modifications while a checkpoint operation involving it is not yet completed. To avoid this, CCL includes also functionalities to suspend on demand the execution of the simulation program in order to wait, if needed, the completion of a pending DMA based checkpoint operation. This leads to the so called *semi-asynchronous* execution mode of checkpointing. Preliminary performance results (Quaglia and Santoro 2001, Quaglia, Santoro and Ciciani 2001) have shown that this mode is an effective solution to reduce the completion time of the simulation by reducing the delay associated with any single checkpoint operation.

However, semi-asynchronous checkpointing produces extra-utilization of the hardware on board of the Myrinet network card since that hardware is not used to support communication functionalities alone. Such an extra-utilization might harm the performance of the communication subsystem, thus possibly originating an increase in the amount of rollback (Carothers, Fujimoto and England 1994), which

would reduce the effectiveness of the Time Warp mechanism. This is likely to occur in all the circumstances in which communication functionalities fully exploit the potential of the network hardware, so that the extra-utilization due to checkpointing is likely to overload the hardware.

In this paper we investigate on the performance improvements that can be achieved when adopting semi-asynchronous checkpointing functionalities offered by CCL for Time Warp simulations of a large state Personal Communication System (PCS) model. Actually, PCS simulations exhibit high locality of communication among the LPs hosted by the same machine, thus yielding a situation in which communication operations typically under-utilize the network hardware. Therefore, PCS simulation should actually take advantage from semi-asynchronous checkpointing since no additional rollback penalty (due to excessive increase in the delivery delay of messages/antimessages at the simulation application level caused by overload on the hardware) should be paid.

We also recall that Time Warp simulation of large state PCS models is an application for which the use of optimized checkpointing not relying on the incremental saving of the state vector is almost mandatory in practice. This is because, still due to communication locality among the LPs hosted by the same machine, the rollback pattern shows relatively infrequent, long rollbacks which could originate excessive state recovery delay for the case of incremental state saving. The experimental study we perform shows that, for simulations of a large state PCS model, semi-asynchronous checkpointing actually produces strong performance benefits as compared to the classical checkpointing mode (i.e. checkpointing charged to the CPU) used in combination with strategies relying on the infrequent recording of the whole state vector of the LP.

The remainder of this paper is structured as follows. In Section 2 we report a brief overview of CCL. The PCS simulation model we have used is described in Section 3. The performance study is presented in Section 4.

2 AN OVERVIEW OF CCL

CCL has been designed for the M2M-PCI32C Myrinet card, based on the LANai 4 chip (MYRICOM 1999), whose high level structure is reported in Figure 1. Actually, this chip is a programmable communication device consisting of: **(a)** an internal bus, namely LBUS (Local BUS); **(b)** a programmable processor connected to the LBUS, which we will refer to as LANai processor; this processor runs a control program that, for the case of CCL, is structured to support both data transfer operations between the host and the network and semi-asynchronous checkpoint operations; **(c)** a RAM bank of 1 Mbyte (LANai internal memory), connected to the LBUS, which is used for storing both data and the control program run by the LANai processor; this memory can be mapped into the memory address space of the host so that the host itself can perform read and write operations on it, which take place through a PCI bridge between the host and the LANai chip; **(d)** a packet interface between the Myrinet switch and the LANai chip, accessible by the LANai processor; **(e)** three DMA engines, used respectively for packet-interface/internal-memory transfer operations (Receive DMA), internal-memory/packet-interface transfer operations (Send DMA), internal-memory/host-memory transfer (or vice-versa) operations (EBUS DMA, namely External Bus DMA).

We note that the LANai processor cannot access host memory directly. Nonetheless, the control program run by the LANai processor can program the EBUS DMA to perform data transfer to/from that memory.

Communication functionalities provided by CCL have been implemented to fully exploit the potential offered by the hardware components on board of the chip (Quaglia, Santoro and Ciciani 2001). Specifically, messages incoming from the network are temporarily buffered into the LANai internal memory (data transfer between the packet interface and the internal memory takes place through the Receive DMA) and are then transferred into the receive queue, located onto host memory, through the EBUS DMA. This is a common choice to fast speed messaging layers for Myrinet, see for example (Pakin, Lauria and Chen 1995). Also, CCL adopts a classical optimization called "block-DMA" to transfer incoming messages from the LANai internal memory to the host memory. It allows incoming messages stored in contiguous message slots of the LANai internal memory to be transferred using a single DMA operation. Following the common design choice, any send operation issued by the application involves copying the message content directly into the LANai internal memory (this is also referred to as "zero-copy" send). Then the message is transferred onto the network through the Send DMA.

The responsibility to program the three DMA engines anytime there is the need for supporting a given data transfer operation pertains to the control program run by the LANai processor. A detailed description of this program can be found in (Quaglia, Santoro and Ciciani 2001).

2.1 Semi-Asynchronous Checkpointing Functionalities

Any semi-asynchronous checkpoint operation involves data transfer from the LP current state buffer (located onto host memory) to the stack of the checkpointed states of the LP (also located onto host memory). As shown by the directed dashed lines in Figure 1, the data transfer operation is charged to the EBUS DMA that uses the LANai internal memory as a temporary buffer. (Temporary buffering is needed since the EBUS DMA does not support direct host memory to host memory data transfer. It only supports host memory to LANai internal memory transfer or vice versa.)

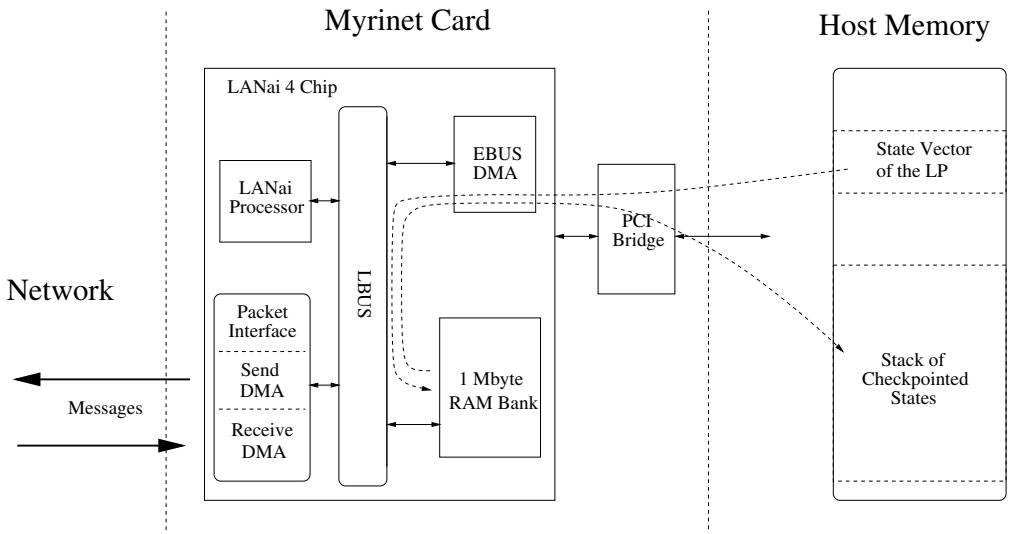

Figure 1: High Level Structure of the M2M-PCI32C Card and Data Transfer Associated with Semi-Asynchronous Checkpointing

In addition to the classical API for usage of communication functionalities at the application level, the following API is provided with CCL for usage of semi-asynchronous checkpointing functionalities:

- `semi_asynch_ckpt(int LP_id, time_type simulation_clock)`, where `LP_id` is the identifier of the LP whose state vector needs to be checkpointed, and `simulation_clock` is the value of the current simulation time seen by that LP. Any checkpoint operation issued at the application level through this function actually means requesting the LANai processor to program the EBUS DMA for the data transfer associated with checkpointing.
- `ckpt_wait()`. This function supports the "on demand suspension" of the simulation program mentioned in the Introduction, which is used for data consistency maintenance in case the simulation execution requires access to a state vector which is currently being checkpointed through the EBUS DMA. Invocation of this function suspends the execution of the simulation application until a semi-asynchronous checkpoint operation, if any, is still in progress. If there is no pending checkpoint operation, `ckpt_wait()` returns immediately.

Note that data transfer associated with checkpointing uses the EBUS DMA, the LBUS, the internal memory and the PCI bridge, therefore semi-asynchronous checkpointing causes extra-load on these components (we use the term "extra" since these hardware components are used also for data transfer operations associated with communication). As pointed out in the Introduction, we expect that the extra-load does not produce performance degradation of the communication subsystem anytime it will not result in an overload on these components.

3 THE PCS SIMULATION MODEL

A PCS is a system that provides communication services to *mobile units*. In our simulation model the service area is partitioned into cells, each of which is modeled by a distinct LP. Each cell represents a receiver/transmitter having either some fixed number of channels allocated to it (Fixed-Channel-Assignment, namely FCA) or a number of channels dynamically assigned to it (Dynamic-Channel-Assignment, namely DCA). In this paper we consider an FCA model with 50 channels per cell. The model is *call-initiated* (Carothers, Fujimoto and Lin 1995) since it only simulates the behavior of a mobile unit during conversation (i.e. the movement of a mobile unit is not tracked unless the unit itself is in conversation). Therefore, the model is organized around two entities, namely cells and calls.

Call requests arrive to each cell according to an exponential distribution (Boukerche et al. 1999, Carothers, Fujimoto and Lin 1995, Carothers, Bauer and Pearce 2000) with inter-arrival time 20 seconds. All the calls initiated within a given cell are originated by the LP associated with that cell, therefore no external call generator is used. There are three main types of events, namely hand-off (due to mobile unit cell switch), call termination and call arrival. When a call arrives at a cell, channel availability must be determined. If all channels are busy, the incoming call is simply counted as a "block". If at least one channel is available, then channel assignment for the new call takes place. A call termination simply involves the release of the associated channel and statistics update.

Hand-off takes place each time a mobile unit currently involved in conversation moves from one cell to another.

In our model there are two distinct classes of mobile units. Both of them are characterized by a residence time within a cell which follows an exponential distribution, with mean 5 minutes (fast movement units) and 40 minutes (slow movement units), respectively. The average holding time for each call associated with both fast and slow movement units is 2 minutes. When a call arrives at a cell, the type (slow or fast) of the mobile unit associated with the incoming call is selected from an uniform distribution, therefore any call is equally likely to be destined to a fast or a slow movement mobile unit.

When a hand-off occurs between adjacent cells, the hand-off event at the cell left by the mobile simply involves the release of the channel. Instead, the hand-off event at the destination cell checks for channel availability. If there is no available channel, then the call is simply cut off (dropped), otherwise an available channel is assigned to the call. Hand-off events for destination cells are not pre-computed, i.e. they are scheduled only upon the occurrence of the hand-off event at the cell left by the mobile unit.

The state vector of any LP records statistics, information about busy channels and, for each channel, information about features of the mobile unit involved in the ongoing call (e.g. scheduled call termination time, call initiation time, class of the mobile unit etc.), if any. As a result, the size of the state vector is about 2Kbytes, thus giving rise to a large state model. (Depending on the abstraction level, PCS models might exhibit smaller LP state granularity (Carothers, Fujimoto and Lin 1995, Carothers, Bauer and Pearce 2000). This might happen when information maintained in the LP state vector are almost exclusively related to the current state of the channels within the associated cell, or when the number of channels per cell is small.)

We have simulated a PCS in which each cell is hexagonal, therefore all the cells, except bordering cells of the coverage area, have six neighbor cells. The model size, in terms of number of cells, has been fixed at 64.

4 THE EXPERIMENTAL STUDY

4.1 The Hardware/Software Architecture and the Simulation Engine

The hardware/software architecture we have used to test the effectiveness of semi-asynchronous checkpointing for the previously described PCS model is a cluster of 8 PCs Pentium II 300 MHz running LINUX (kernel version 2.0.32) which are equipped with 128 Mbytes RAM, 512 Kbytes second level cache and M2M-PCI32C Myrinet cards. The experiments have been performed using the CCL based Time Warp simulation engine described in (Quaglia and Santoro 2001), whose main loop is structured as shown below (for sake of simplicity GVT calculation and "fossil collection" for memory recovery are not reported. Also, rollback relies on aggressive antimessage sending (Gafni 1985), i.e. antimessages are sent as soon as the LP rolls back):

```
 1. pending_LP = no_LP;
 2. while(not end)
 3.   <receive messages>
 4.   LP = schedule_next();
 5.   if (LP = pending_LP)
 6.         ckpt_wait();
 7.         pending_LP = no_LP;
 8.   if (rollback required for LP) rollback();
 9.   next_event_execution();
10.   if (checkpoint required for LP)
11.         ckpt_wait();
12.         pending_LP = LP;
13.         semi_asynch_ckpt();
14.   <send messages>
```

This loop structure exhibits two distinct invocations of the function `ckpt_wait()`. The invocation in line 6. takes place only if the currently scheduled LP is associated with a pending semi-asynchronous checkpoint operation. This allows data consistency maintenance into the stack of checkpointed state vectors of that LP. The invocation in line 11. actually leads to suspension of the simulation application only in case a new checkpoint operation must be issued while the last issued one is still in progress. This is required since CCL does not support concurrent execution or queuing of multiple EBUS DMA based checkpoint operations.

Even distribution of the 64 LPs of the simulation model among the 8 machines has been adopted, with an obvious mapping of LPs to machines (i.e. a mapping assigning to any machine a specific group of adjacent cells).

4.2 Reference Checkpointing Technique

As a reference checkpointing technique to evaluate the benefits from semi-asynchronous checkpointing we have selected the Periodic State Saving (PSS) strategy relying on classical checkpointing charged to the CPU. In our implementation, a `memcpy()` call is issued to copy the content of the current state vector of the LP into the stack of checkpoints of the same LP. For CPU based checkpointing, we have located the stack and the current state buffer of the LP into reserved pages of physical memory that will never be swapped out. This is done in order to ensure fairness in the comparison since in semi-asynchronous checkpointing both the stack and the current state buffer of each LP are located into unswappable pages of physical memory. This is because DMA based data transfer works only with physical memory addresses.

Actually for CPU based checkpointing we have used the previously described simulation engine with a minor modification consisting in substituting the invocation of the function `semi_async_ckpt()` with the invocation of a proper `memcpy()` function. Also, code lines related to

the management of the function `ckpt_wait()` have been obviously removed.

We have selected PSS since simulations of the previously described PCS model with even distribution of the LPs among the machines and with no other active user load typically reaches a steady state of the rollback behavior. Therefore, observing the performance achieved by PSS while varying the checkpoint period χ (evaluated in terms of number of executed events between two consecutive checkpoint operations) is likely to point out the best performance achievable with any (adaptive) strategy just relying on periodic checkpoints.

We recall that recently infrequent checkpointing strategies that do not rely on periodic checkpoints have been proposed, see for example (Quaglia 2001). These strategies exhibit the potential for performance improvement over strategies based on periodic checkpoints. However the performance gain for the case of PCS models is typically limited (Quaglia 2001) due to the particular structure of the rollback pattern that, as already pointed out, shows long, relatively infrequent rollbacks. Therefore we do not include data related to those techniques in the analysis. The particular nature of the rollback pattern is also the reason why checkpointing based on incremental saving of portions of the state vector is typically unused for PCS models, so we do not consider incremental methods in the analysis.

Finally, we study the behavior of semi-asynchronous checkpointing in combination with PSS while varying the checkpoint period χ. This will allow us to point out possible differences in the effects of the checkpoint period itself for the case of DMA based and CPU based checkpointing.

4.3 Observed Parameters

We report measures related to the following parameters:

(i) The efficiency, that is the ratio between the number of committed events and the total number of executed events (committed plus rolled back). This parameter allows us to evaluate whether semi-asynchronous checkpointing produces relevant variations in the amount of rollback of the simulation, which might be an indication of a strong increase in the message delivery delay (Carothers, Fujimoto and England 1994) caused by possible overload on the hardware of the Myrinet network cards.

(ii) The frequency of application suspension (for semi-asynchronous checkpointing only) due to invocations of the function `ckpt_wait()` in lines 6. and 11. of the previously presented simulation engine. This parameter is computed as the ratio between the number of times the application is really suspended and the total number of invocations of the `ckpt_wait()` function.

(iii) The event rate, that is the number of committed events per sec. This parameter indicates how fast the simulation execution is, therefore it is representative of the final performance perceived.

We report the average observed values of previous parameters, computed over 10 runs that were all done with different seeds for the random number generation. At least 5×10^6 committed events were simulated in each run.

4.4 Results

The results are reported in Figure 2, Figure 3 and Figure 4. By the plots related to the efficiency, we note no relevant difference between the values obtained with CPU based checkpointing and those obtained with semi-asynchronous checkpointing. As expected, this is an indication that communication latency seems not to suffer from extra-load on the network cards due to semi-asynchronous checkpointing. Specifically, if relevant interference on the delivery delay had been produced, we should have noted a decrease in the efficiency due to an increase in the amount of rollback, since delaying the delivery of messages/antimessages might mean higher likelihood of incorrect computation (Carothers, Fujimoto and England 1994).

A second interesting point relates to the frequency of application suspension due to invocations of the function `ckpt_wait()` when semi-asynchronous checkpointing is adopted. The plot shows that when the checkpoint period χ is increased from 1 to 4, the suspension frequency shows a reduction and then assumes a stable value. This is one of the reasons why the event rate produced by semi-asynchronous checkpointing shows a steep growth while χ is increased up to 4.

As respect to the event rate, the plots indicate that semi-asynchronous checkpointing allows an acceleration of the simulation model execution. In particular, the best event rate of semi-asynchronous checkpointing is about 63×10^3 events per sec., while the best event rate with CPU based checkpointing is about 58×10^3 events per sec., thus semi-asynchronous checkpointing provides a performance gain in the order of 9%.

As a last remark we would like to bring to the reader's attention, semi-asynchronous checkpointing does not provide unacceptable performance even in case the state vector of the LP is saved at each event execution (i.e. even when the checkpoint period χ is set to one). Specifically, the provided performance is, at worst, 7% lower than the best performance achievable with CPU based checkpointing while varying χ. Instead, the performance provided by CPU based checkpointing with χ set to one results definitely worse. This is a relevant result for semi-asynchronous checkpointing since

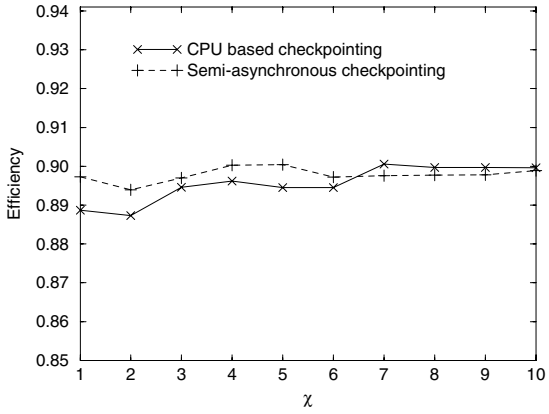

Figure 2: Efficiency vs the Checkpoint Period χ

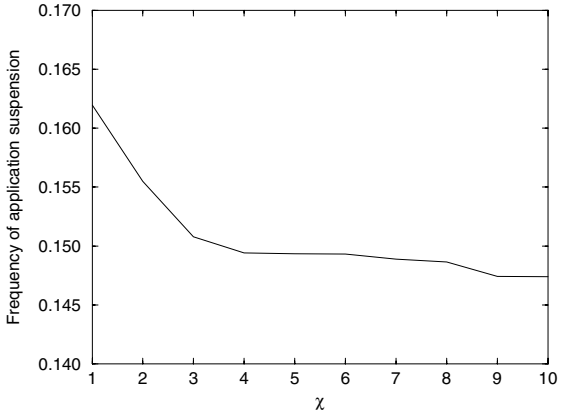

Figure 3: Frequency of Application Suspension vs the Checkpoint Period χ

it is an indication that semi-asynchronous checkpointing could even be used in combination with no optimized infrequent state saving strategy while still providing adequate performance. Transparency at the simulator programmer's level can clearly take advantage from this feature.

5 SUMMARY

In this paper we have reported an experimental study on the performance that can be achieved when adopting semi-asynchronous checkpointing (i.e. CPU offloaded checkpointing) for Time Warp simulations of a large state PCS model on a distributed memory system based on Myrinet. The study has been carried out using a Checkpointing and Communication Library (CCL) recently presented to support semi-asynchronous checkpointing on that type of systems.

The study points out the performance benefits from semi-asynchronous checkpointing over classical CPU based checkpointing combined with optimized checkpointing strategies based on infrequent saving of the LP state vector. The results show that using semi-asynchronous checkpointing in combination with those strategies yields a relevant acceleration of the simulation model execution. They also point out that semi-asynchronous checkpointing exhibits adequate (but sub-optimal) performance even when the state vector of the LP is saved at each event execution. If optimal performance is not mandatory, then semi-asynchronous checkpointing can be adopted in combination with no optimized infrequent strategy for saving the LP state vector, which would allow better transparency at the level of the PCS simulator programmer.

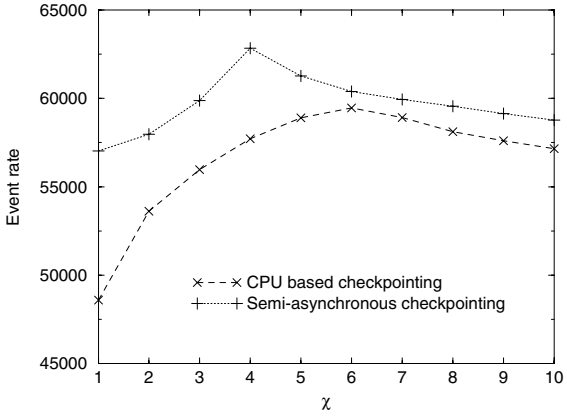

Figure 4: Event Rate vs the Checkpoint Period χ

REFERENCES

Bauer, H., and C. Sporrer. 1993. Reducing rollback overhead in Time Warp based distributed simulation with optimized incremental state saving. In *Proceedings of the 26th Annual Simulation Symposium*, 12-20.

Bellenot, S. 1992. State skipping performance with the Time Warp operating system. In *Proceedings of the 6th Workshop on Parallel and Distributed Simulation*, 33-42.

Boukerche, A., S. K. Das, A. Fabbri and O. Yildz. 1999. Exploiting model independence for parallel PCS network simulation. In *Proceedings of the 13th Workshop on Parallel and Distributed Simulation*, 166-173.

Carothers, C. D., R. Fujimoto and P. England. 1994. Effect of communication overheads on Time Warp performance: an experimental study. In *Proceedings of the 8th Workshop on Parallel and Distributed Simulation*, 118-125.

Carothers, C. D., R. Fujimoto and Y. B. Lin. 1995. A case study in simulating PCS networks using Time Warp. In *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*, 87-94.

Carothers, C. D., D. Bauer and S. Pearce. 2000. ROSS: a high performance modular Time Warp system. In *Proceedings of the 14th Workshop on Parallel and Distributed Simulation*, 53-60.

Fleischmann, J., and P.A. Wilsey. 1995. Comparative analysis of periodic state saving techniques in Time Warp simulators. In *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*, 50-58.

Gafni, A. 1985. Space management and cancellation mechanisms for Time Warp. Technical Report TR-85-341, University of Southern California, Los Angeles, California, USA.

Jefferson, D. 1985. Virtual time. *ACM Transactions on Programming Languages and Systems* 7(3):404-425.

Lin, Y.B., B.R. Preiss, W.M. Loucks and E.D. Lazowska. 1993. Selecting the checkpoint interval in Time Warp simulation. In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation*, 3-10.

MYRICOM. 1999. *LANai 4, Draft*.

Pakin, S., M. Lauria and A. Chen. 1995. High performance messaging on workstations: Illinois Fast Messages (FM). In *Proceedings of the 1995 Supercomputing Conference*.

Quaglia, F. 1999. Combining periodic and probabilistic checkpointing in optimistic simulation. In *Proceedings of the 13th Workshop on Parallel and Distributed Simulation*, 109-116.

Quaglia, F. 2001. A cost model for selecting checkpoint positions in Time Warp parallel simulation, *IEEE Transactions on Parallel and Distributed Systems* 12(4):346-362.

Quaglia, F., and A. Santoro. 2001. Semi-asynchronous checkpointing for optimistic simulation on a myrinet based NOW. In *Proceedings of the 15th Workshop on Parallel and Distributed Simulation*, 56-63.

Quaglia, F., A. Santoro and B. Ciciani. 2001. Tuning of the checkpointing and communication library for optimistic simulation on myrinet based NOWs. In *Proceedings of the 9th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*.

Ronngren, R., and R. Ayani. 1994. Adaptive checkpointing in Time Warp. In *Proceedings of the 8th Workshop on Parallel and Distributed Simulation*, 110-117.

Skold, S., and R. Ronngren. 1996. Event sensitive state saving in Time Warp parallel discrete event simulations. In *Proceedings of the 1996 Winter Simulation Conference*.

Steinman, J. 1993. Incremental state saving in SPEEDES using C plus plus. In *Proceedings of the 1993 Winter Simulation Conference*, 687-696.

Unger, B.W., J. Cleary, A. Covington and D. West. 1993. External state management system for optimistic parallel simulation. In *Proceedings of the 1993 Winter Simulation Conference*, 750-755.

AUTHOR BIOGRAPHIES

ANDREA SANTORO is a Ph.D. student at the University of Rome "La Sapienza". He received the Laurea degree in electronic engineering in 1999 from the same institution. His research interests are in parallel simulation and computer networks. His email address is <santoro@dis.uniroma1.it> and his web page is <http://www.dis.uniroma1.it/~santoro>.

FRANCESCO QUAGLIA received the Laurea degree in electronic engineering in 1995 and the Ph.D. degree in computer engineering in 1999 from the University of Rome "La Sapienza". From summer 1999 to summer 2000 he held an appointment at CNR ("Consiglio Nazionale delle Ricerche" - Italy). Currently he is an assistant professor at the University of Rome "La Sapienza". His research interests include parallel discrete event simulation, parallel computing, fault-tolerant programming and performance evaluation of software/hardware systems. He regularly serves as a referee for several international conferences and journals. He has served on the program committee of the 14th and 15th editions of the Workshop on Parallel and Distributed Simulation (PADS), and has been invited to serve as a program co-chair for the 16th edition of the same conference. His email and web addresses are <quaglia@dis.uniroma1.it> and <www.dis.uniroma1.it/~quaglia>.