Conservative and Optimistic Distributed Simulation in Massively Parallel Computers: a Comparative Study

José Miguel Alonso[1], Agustín Arruabarrena Frutos[1] and Ramón Beivide Palacio[2]

[1]Dep. Arquitectura y Tecnología de Computadores UPV/EHU
Apdo. 649, 20080 San Sebastián, Spain.
Tel: +34 43 218000 Fax: +34 43 219306
E-mail: acpmialj@si.ehu.es

[2]Dep. Electrónica - Univ. de Cantabria
Av. de los Castros s/n, 39005 Santander, Spain.
Tel: +34 42 201496 Fax: +34 42 201402
E-mail: beivide@ccucvx.unican.es

Abstract

The main algorithms for sequential and parallel discrete event simulation are introduced. A set of different simulators is evaluated and compared using a transputer-based multicomputer: sequential, parallel conservative and parallel optimistic. For the proposed model, the conservative simulator shows better speedups. The obtained results show that a significant improvement on execution time can be obtained using parallel systems, but only if the model under simulation have some properties like coarse grain size and high degree of self-synchronisation.

1: Introduction

Computer simulation is considered by the computer architecture research community as an essential tool for the evaluation and design of new computer systems. Simulation is a very CPU-consuming activity if very complex systems are being developed, or if a detailed insight into the system's behaviour is desired. For this reason, current sequential machines are not fast enough if results are needed in a bounded time. We see *parallel simulation* as a means of speeding up simulations by using parallel architectures. In the next section we will review the basic ideas about parallel simulation. The reader is referred to [7] for a more detailed description of a variety of parallel simulation algorithms. We will briefly present the two main methods: conservative and optimistic. These two methods have some characteristics in common, but differ in their view of process synchronisation.

This work have been done under partial support of the CICYT, Spain, under contract TIC93-0243.

2: Parallel simulation algorithms

Usually in the literature, a *physical system* to be simulated is considered as composed of one or several *physical processes* (*PP*s). Each PP evolves independently from the others, but interacts with them by means of *messages* sent and received at discrete instants of time. The contents of a message has two fields, *event* and *timestamp*, and depend on the characteristics of the sender PP: initial state, behavioural rules and previously received messages. The *event* denotes an interaction that will happen in the future. The *timestamp* specifies when the event will occur.

A typical sequential event-driven simulator consists of a central clock and a set of data structures that represent the PPs in the model, with an additional element: the *event list*. This list stores the set of messages scheduled to be delivered *in the future* with their corresponding reception time—their timestamps. The events are processed in non-decreasing timestamp order. This fact is the key of the simulation process, because it ensures that *causality relations* are always maintained. A parallel discrete-event simulator (PDES) has to ensure the same restriction, but trying to process several events simultaneously.

In general, a PDES simulates each PP of the physical system by means of a *LP* (*logical process*). LPs do not share (conceptually) any data, and communicate only via messages. Every LP has its own clock, that represents up to what time the local simulation has progressed. The representation of time is distributed among the LPs. The simulation of a message sent from PP_i to PP_j that should be received at time *t* is done by LP_i sending a message with timestamp *t* to LP_j. This timestamp has to be strictly higher than the value of the local clock of LP_i, because it is supposed that a PP can not send messages "to the past". Every LP maintains a series of input queues where the received messages are stored, awaiting to be consumed. When a message is consumed, the local clock advances to reach the value of the message's timestamp, in the same way that is done in a sequential simulator. It has been demonstrated [14] that if each LP consumes its received messages in non-decreasing timestamp order, then the parallel simulation is correct.

2.1: Conservative approaches to PDES

Conservative methods [1, 2] take no risk when selecting the next event to process. The model has to obey two constraints: (1) the topology of the network that interconnects the LPs has to be statically defined, because each LP maintains an input queue for each of its input channels; (2) LPs have to send the messages through each output link in non-decreasing timestamp order.

The algorithm for an individual LP can be found in [2] or in [14]. Its main characteristic is that it takes no risk when selecting a message to be consumed. It always consumes *safe* messages. If that safety is not guaranteed, the LP blocks, awaiting for incoming messages. The drawback of this synchronisation mechanism is that it can lead to a deadlock between a group of LPs. Those deadlocks have to be coped with, and that is done by means of an interchange of *null messages*. Null messages are a serious overhead in the simulation. Its number strongly influences the achieved speedup. A characteristic of the model, callel *lookahead* [3], influences this number: the higher the lookahead, the smaller the number of null messages, so the better the speedup.

2.2: Optimistic approaches to PDES

Some limitations found in consevative methods led to the development of a new method, called Time Warp [8, 9], with a more *optimistic* way of synchronisation. While conservative methods avoid completely any causality conflict, optimistic methods *allow* them to occur. Messages are consumed as fast as possible in each LP. When a causality error is detected—a new message is received with timestamp smaller than the one of the last processed message—a *rollback* process is done to jump to the past, just before the erroneous computation.

While a LP is processing erroneous messages, its status may change and new messages can be generated. A rollback process has to jump to the past, to a state free of causality errors (which means that previous values of the state of a LP have to be stored) and the messages eventually sent have to be cancelled (which means that a LP has to remember all the messages previoulsy sent). The cancellation of erroneous messages is made sending *anti-messages*. When a LP sends a message, it saves a negative copy of it—the anti-message—in an *output queue*. When a rollback is done, the LP sends an anti-message for every erroneous message sent before. If a LP receives an anti-message before the corresponding positive copy has been processed, both copies annihilate mutually, and no additional action is required. On the other hand, if the positive copy has been already consumed then a causality error occurs and a rollback procedure is needed.

From the previous description of the algorithm, it could be deduced that the data structures grow unboundedly as the simulation progresses. All the information about the past is stored, just in case a return to the past were forced. This storage has to be limited, otherwise the LPs would exhaust their assigned memory space. A practical optimistic simulation is implemented as a series of LPs that perform local activities plus a *global control* mechanism. This global control computes regularly the value of the *global virtual time* (*GVT*), an indicator of the global situation of the simulation. The GVT is computed as the minimum timestamp among all the messages in all the LPs awaiting to be consumed. Once computed, the GVT is distributed to all the LPs, that use it to destroy messages, anti-messages and copies of the state of the process that will never be used (as the LP will never roll back to a time before the GVT).

3: Description of the experiments

In [10] the reader can find a complete description of the computer system and the model used to evaluate our simulators. We did our experiments on a Transputer-based multicomputer, using up to 17 processors. The selected model consists of a toroidal network of FCFS servers, traversed by a fixed number of jobs that visit the servers to be processed (see Fig. 1; the real model is 12x12). This kind of model or similar ones have also been used in works like [4, 11].

Some of the parameters of the model that can be varied to evaluate the behaviour of the simulator are:

- Initial load (C) of the network. A value of the load $C=n$ means that when the simulation starts each queue has n jobs waiting to be served.
- Distribution of the *service time*. A biased distribution has been considered with mean M and minimum L. As was done in [4], L is used as the value of the lookahead. M has been fixed to 9.5 in the experimets reported here.
- The number of *processors*—transputers—used.

Using this model, an important aspect to be taken into account is that simulating the consumption of a job in the server takes a negligible amount of time and CPU effort: it is only needed to increment some counters and to move small amounts of data. Another point to be considered is the high communication time needed to pass a message between two Transputers, because the switching mechanism is store-and-forward, with all message management done in software. These considerations are specially relevant when evaluating the parallel simulator: the computation-communication ratio is extremely unbalanced towards the communication.

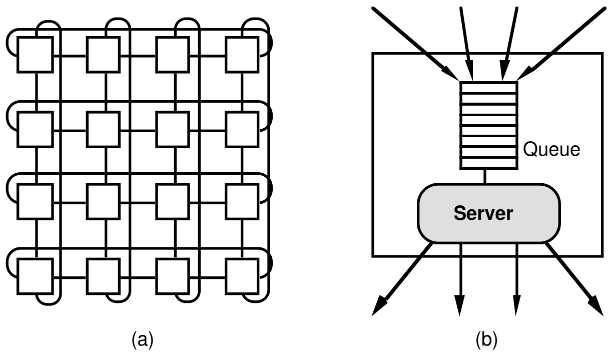

Figure 1. (a) Toroidal network simulated in the experiments. (b) Detail of a node

To evaluate what would happen if the ratio were not so bad, another parameter has been included in the simulators: *synthetic workload*. This is an artificial CPU effort (in the form of a null loop "**for** i **from** 1 **to** synthetic_workload **do** nothing") included in the point of the simulator where the processing of a job is simulated. Incrementing this value (action that could represent an increase in the grain size of the simulation) we bias the program towards a more balanced computation-communication ratio and then the simulation should be more effective.

3.1: Experiments with the conservative simulator

An exhaustive report of the results of the simulation runs can be found in other works of our group [10]. In Table 2 we sumarise some of those results, for a set of interesting experiments. The parameters for each one of the experiments (**A** through **E**) are those of Table 1. It should be remarked that results with 1 processor correspond to simulation runs made with a sequential simulator, instead of using the parallel simulator in only one processor. For each experiment, the left-hand column shows the execution time, and the right-hand column the obtained speedup, relative to the execution time of the sequential simulator.

A first set of experiments, varying *L*, *C* and the number of processors, was made to assess the influence of these three parameters in the global performance of the simulators. The results of three of those experiments are shown in columns **A**, **B** and **C** in Table 2. The main conclusions that can be drawn out are (1) speedup scales well with the number of Transputers, (2) the load *C* has the greatest impact in the simulation speed and (3) the lookahead is specially relevant with low loads.

Experiment	C	L	W
A	1	1	0
B	6	6	0
C	10	10	0
D	1	1	10000
E	6	1	10000

Table 1: Input parameters for the experiments reported on Table 2 and Table 3.

The obtained speedups are not really bad, specially if the consideration about communication-computation ratio is taken into account. In fact, most of the execution time is due to the cost of the simulation algorithm, mainly the synchronisation time of the LPs (that includes the interprocess communication by means of messages), whereas the time to simulate an event is negligible. We made another set of experiments inserting synthetic CPU workload when a job is processed (*W*). With a reasonably high value of *W*, the simulations exhibit very satisfactory speedups. The parallel simulator is able in this case to distribute the synthetic workload among the processors, and thus available parallelism is conveniently exploited. Additionally, the communication-computation ratio becomes more balanced.

Our experiments confirm the results reported in the literature [4, 7, 11]. It could be concluded that, in general, the conservative approach to parallel simulation does not exhibit a brilliant performance. This is certainly true in many cases, but some models can successfully be simulated with this kind of simulators: (1) models that require small synchronisation effort to the LPs, that is, models with a high degree of communication, or models with a low connectivity between LPs; (2) models with high lookahead, that allows the simulation to progress quickly when the LPs do not have events to process; (3) models where the simulation of an event requires a high computation effort, which is distributed among the processors. A "coarse grain" distribution of model components over logical processes will give more efficient parallel simulations. In summary, the performance of conservative simulation schemes is highly dependent on the intrinsic characteristics of the model.

3.2: Experiments with the optimistic simulator

After developing the conservative simulator, an implementation of Time Warp was also done, trying to maintain the maximum level of similarities between the two simulators in order to make fair comparisons.

We found out that the optimistic simulator was terribly greedy in memory demands. Many experiments were aborted because the memory assigned to a LP was exhausted: the LPs data structures grew too fast, and the fossil collection mechanism (made possible by means of a global control mechanism to calculate and distribute the GVT) did not free much memory to be reused. In addition to that, the message passing subsystem also failed because of lack of space to store messages in its journey from a LP to another. After analysing the evolution of the programs, we discovered two reasons for this behaviour:

(1) The characteristics of the model. The toroidal network used in the experiments exhibits a high degree of (local) communication. The result is that an erroneous computation in a LP spreads quickly to the neighbouring LPs, making them advance in an erroneous way. What is worse, a feedback effect is produced when, after erroneously processing a job, it is sent to another LP that also processes it and sends it back to the original LP, where the chain starts again. In the proposed model, this situation is highly probable. The result is that, firstly, a big deal of memory space is required to store anti-messages and copies of the state and, secondly, whenever a causal error is detected, an avalanche of anti-messages is sent, producing the overflow effect in the message passing system.

(2) The Transputer built-in scheduler. Many studies of optimistic simulators have been done using ad-hoc microkernels to schedule the execution of a number of LPs in a smaller number of processors. Those schedulers prioritize the LPs with minimum local time [5, 6]. With this approach, the greedyness of the optimistic LPs is strongly limited: erroneous computation does not go too far and, consequently, anti-messages are sent less often and in smaller number. In our simulator, instead of designing a new scheduler, we used the transputer built-in one, whose main characteristic is its attempt to be fair and distribute the CPU evenly among all the LPs. Now we are planning to redesign the system using a special purpose software scheduler.

To solve these problems we adopted a proposal of Sokol et al. [12]: we added a certain degree of "conservadurism" into the LPs. With a purely optimistic scheme, the LPs advance unboundly (in fact, as just mentioned, they advance too fast). The conservadurism is imposed by limiting the ability to go into the future: LPs are allowed to advance to a ceirtan degree, calculated as the value of the GVT plus a *time window size*. If it tries to go beyond, it is (temporally) blocked. The window size is dinamically changed: if the LP is successful in its advance (that is, if it does not force rollbacks), the window size is increased. Otherwise, the window is decremented. A minimum window size is guaranteed, to have always a degree of optimism.

We did a series of experiments to evaluate our optimistic simulator with the time window mechanism. Some of those are compiled in Table 3. As many authors reported that the optimistic simulator's performace is much better than the conservative's [6], we were quite dissapointed: in most of the cases, the conservative simulator runs faster than the optimistic.

Only in the worst case for the conservative simulator (low load and low lookahead) the optimistic simulator is faster. We studied the reasons of this behaviour, and we found (as other researchers did before [13]) that the memory management in the LPs is absolutely critical, up to the point that, in most cases, the overhead of managing the data structures is higher that the effort of synchronising processes by means of blocking and null messages. Additionally, although null messages are not needed, anti-messages are, and its number is not negligible, especially when the optimistic advance of the LPs is not bounded.

An important characteristic of the optimistic simulator is that the execution times are less sensitive to parameters as load and, particularly, lookahead. That means that the user does not have to worry about the best way to exploit the lookahead of the model. The level of abstraction that is offered is higher. While a conservative simulator has to be done ad-hoc for a class of models, if a good performance is desired, the optimistic one is more general, and can be applied (with similar success) to a variety of models [15, 16].

From the experiments shown in Table 3, it can also be concluded that the speedup scales with the number of processors. In relation with the effect of the synthetic workload in the performance (experiments **D** and **E**), a big value of *W* has important effect on the speedup, but this effect is not as dramatic as it was in the conservative case. The reason is that in the conservative simulator a busy-wait loop is done just whenever it is needed, and never has to be undone. On the other hand, with the optimistic simulator, many jobs are processed in a speculative way, and its effect have to be undone later. That means that, when the grain size is big, the effect of erroneous computations is also a serious drawback: it could be better to wait and consume a message just when it has to be consumed, than consume it and later undo its effects. As in experiment **E** the simulator has more jobs to process, the probability of making a mistake is bigger, so it takes more time than experiment **D**.

4: Conclusions

The current simulation techniques are fundamentally sequential, optimized to get the maximum performance from a single processor. Unfortunately, these methods can not be easily adapted for its use in multicomputer systems. For this reason, new, parallel simulation methods have been developed: conservative and optimistic.

Our experiments with the conservative simulator confirms the results of other authors: high values of load, lookahead and synthetic workload make possible good performance. Nevertheless, the simulator has to be hand made to exploit the characteristics of the model. The optimistic simulator is more general, offering a higher level interface to the user, but its performance is, in our experiments, less brilliant than we expected. The memory management tasks involved into the Time Warp mechanism wastes most of the CPU time. The selected model, the computer system used as workbench, and our particular implementation allies to make the performance even worse.

References

[1] Bryant R. E. 77, *Simulation of Packet Communications Architecture Computer Systems*. MIT-LCS-TR-188, Massachusetts Institute of Technology

[2] Chandy K. M. and Misra J. 79, *Distributed Simulation: A Case Study in the Design and Verification of Distributed Programs*. IEEE Trans. Soft. Eng. 5-5 440-452

[3] Fujimoto R. M. 88, *Lookahead in Parallel Discrete Event Simulation*. Proc. Int. Conf. on Parallel Processing pp. 34-41

[4] Fujimoto R. M. 89, *Performance Measurements of Distributed Simulation Strategies*. Transactions of SCS 6-2 pp. 89-132

[5] Fujimoto R. M. 89, *Time Warp on a Shared Memory Multiprocessor*. 1989 Int. Conf. on Paralllel Processing pp. III-242-249

[6] Fujimoto R. M. 90, *Optimistic Approaches to Parallel Discrete Event Simulation*. Transactions of the SCS 7-2 pp. 153-191

[7] Fujimoto R. M. 90, *Parallel Discrete Event Simulation*. Comm. ACM 33-10 pp. 31-53

[8] Jefferson D. R. 85, *Virtual Time*. ACM TOPLAS 7-3 pp. 404-425

[9] Jefferson D. R., Beckman B., et al. 87, *The Time Warp Operating System*. Operating Systems Review 21-5 pp. 77-93

[10] Miguel J., Arruabarrena A., Beivide R 93, *Conservative Parallel Discrete Event Simulation in a Transputer-based Multicomputer.* Transputer Applications and Systems'93, Vol. 1, pp. 651-659. IOS Press.

[11] Reed D. A., Malony A. D. and McCredie B. D. 88, *Parallel Discrete Event Simulation using Shared Memory*. IEEE Trans. Soft. Eng. 14-4 pp. 541-553

[12] Sokol L. S., Stucky B. K., Hwang V. S. 89, *MTW: A Control Mechanism for Parallel Discrete Simulation*. 1989 Int. Conf. on Paralllel Processing pp. III-250-254

[13] Tallieu F., Verboven F. 1991, *Using Time Warp for Computer Network Simulation on Transputers*. Proc. 24th Annual Simulation Symposium, pp. 112-117.

[14] Wagner D. B. 89, *Conservative Parallel Discrete-Event Simulation: Principles and Practice*. Ph. D. dissertation, Univ. of Washington

[15] Wieland F., et al. 89, *The Performance of a Distributed Combat Simulation with the Time Warp Operating System*. Concurrency: Practice and Experience 1-1 pp. 35-50

[16] Wieland F., Jefferson D. 89, *Case Studies in Serial and Parallel Simulation*. 1989 Int. Conf. on Parallel Processing pp. III-255-262

Proc.	A		B		C		D		E	
	Time	SU	Time	SU	Time	SU	Time	SU	Time	SU
16	2'30"	0.29	0'46"	2.52	0'44"	3.39	4'16"	5.32	4'02"	9.56
9	3'32"	0.20	1'03"	1.84	0'58"	2.57	-	-	-	-
4	8'10"	0.09	2'38"	0.73	2'26"	1.02	-	-	-	-
1	0'43"	1	1'56"	1	2'29"	1	22'41"	1	38'33"	1

Table 2: Execution times and speedups for a series of experiments done with a conservative parallel event driven simulator.

Proc.	A		B		C		D		E	
	Time	SU	Time	SU	Time	SU	Time	SU	Time	SU
16	1'48"	0.40	2'50"	0.68	2'40"	0.93	6'00"	3.78	11'51"	3.25
9	2'12"	0.32	4'00"	0.48	3'22"	0.74	-	-	-	-
4	3'40"	0.20	5'43"	0.34	5'02"	0.49	-	-	-	-
1	0'43"	1	1'56"	1	2'29"	1	22'41"	1	38'33"	1

Table 3: Execution times and speedups for a series of experiments done with an optimistic parallel event driven simulator.