An Empirical Evaluation of Techniques for Parallel Discrete-Event Simulation of Interconnection Networks

José Miguel[1,2], Agustín Arruabarrena[2], Ramón Beivide[3] and José A.B. Fortes[1]

[1]School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907, USA

[2]Dep. de Arquitectura y Tecnología de Computadores
UPV/EHU
Apdo. 649
20080 San Sebastián, Spain

[3]Departamento de Electrónica
Universidad de Cantabria
Av. de los Castros s/n
39005 Santander, Spain

acpmialj@si.ehu.es, acparfra@si.ehu.es, beivide@ccucvx.unican.es, fortes@ecn.purdue.edu

Abstract

Three parallel discrete-event simulators—synchronous, conservative and optimistic—implemented on an Intel Paragon multicomputer are comparatively evaluated. Parallelism is achieved by model decomposition, distributing the simulation among a set of collaborative logical processes. The three simulators differ in the way those processes synchronize to obey causal restrictions in the simulation of events. Message passing network models are used to study these simulation alternatives. A set of experiments are carried out to understand how model parameters influence simulator performance. Experimental evidence leads to the conclusion that the optimistic simulator is not a viable tool for the analysis of this kind of models. The opposite conclusion applies to the other two: if the workload assigned to each logical process is above a certain threshold then the synchronization overhead is comparatively low and the simulators perform and scale well up to a large number of processors. Performance threshold is influenced by some parameters of the simulated model (size of the network, load level and message length), as well as by the number of processors used by the simulators.

J. Miguel is currently a Visiting Scholar at the School of Electrical and Computer Engineering, Purdue University.

This work has been done under the partial support of the CICYT, Spain, under contract TIC95-0378, and the National Science Foundation, USA, under grants MIP-9500673 and CDA-9015696.

1: Introduction

Computer simulation is often required to validate, verify or complement analytical models of complex systems. Multiprocessors offer the possibility of large-scale systems simulation as long as efficient parallel processing implementations are adopted. Out of three possible approaches, this paper identifies two efficient discrete-event based parallel simulation techniques for simulation of message passing torus interconnection networks. Experiments conducted on an Intel Paragon multicomputer provide the empirical basis of the conclusions of this paper.

In [8, 9] parallel simulation techniques were analyzed using very simplistic models as benchmarks: torus networks of FCFS servers, traversed by a fixed number of jobs. Four different simulators were used: sequential event-driven, sequential time-driven, parallel conservative event-driven (with deadlock avoidance via null messages) and parallel optimistic event-driven [2, 3, 4, 7]. For the models under consideration it was found in [8, 9] that speedups up to 10 could be obtained for the conservative simulator on a network of 16 transputers. The performance of the optimistic simulator was not such good, in part due to an inefficient implementation of the algorithm.

After this preliminary evaluation performed with a toy model, a realistic model was selected to work with. In [1] a fairly complete set of measurements of the characteristics of some interconnection networks can be found, obtained with a sequential simulator. A small set of these well-known models were selected to test the actual performance of the synchronous, conservative and optimistic parallel simulators working with a useful workload.

The details of the selected model can also be found in [1]. It is basically a torus network of message routing elements (routers). Each router (Figure 1) is connected to four neighbors and to a processing element, via bi-directional channels. All the messages have the same length. The flow-control mechanism is cut-through and, for this reason, message queues are needed, which are associated to each output channel. Operation of the network is assumed to be synchronous. It takes a *cycle* to move a flit from port/queue to port/queue, in the same or in different router. This means that timestamps of events are always integer values.

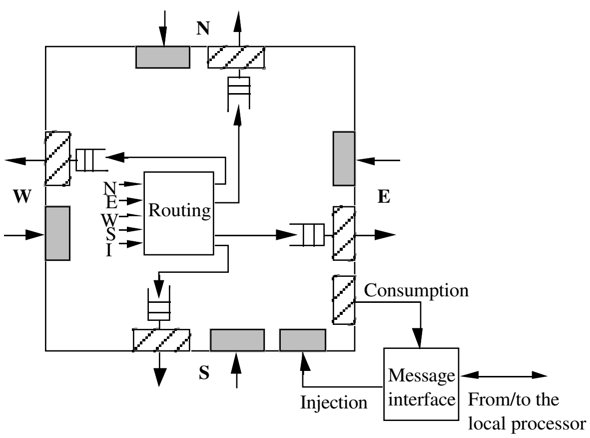


Figure 1. Model of message router.

Three parallel simulators have been implemented and tested on the Paragon multicomputer at Purdue University: CMB (Chandy-Misra-Bryant [2,3]), TW (Time Warp [7]) and SPED (Synchronous Parallel Event Driven [11]). They have in common the fact that parallelism is exploited by model decomposition: the model to simulate is divided into several parts, and each part is assigned to a Logical Process (LP). Those LPs can run concurrently, each one simulating its part of the whole. However, in order to maintain the causal relationships among the events in the simulation, a synchronization mechanism is needed. Here is where the three simulators differ.

The SPED simulator executes events in non-decreasing timestamp order. Only those events with the same timestamp are executed concurrently. The synchronous nature of SPED comes from the fact that all the LPs share the same vision of time, and they barrier-synchronize each time the simulation clock advances.

The other two simulators, CMB and TW, are asynchronous: each LP has its own local view of time. In order to perform a simulation that is globally correct, each LP needs to obey the following rule: execute all the incoming events in non-decreasing timestamp order. This rule is not easy to follow because, after executing a sequence of events, a new one might arrive from other LP, with a timestamp smaller than that of the last executed event, thus impeding the receiver LP to obey the stated rule. A CMB simulator never allows these situations to happen. To do so, LPs block before executing an event, until it is totally safe to proceed. This kind of synchronization is known as *conservative*. A TW simulator allows erroneous situations to arise, but those are detected and a *rollback* is done to jump to an error-free point in the (simulated) past. This kind of synchronization is known as *optimistic*.

A problem with CMB simulators is that the blocking synchronization mechanism can lead to the deadlocks. For this reason, CMB has to be enhanced with a method to resolve deadlocks. A deadlock avoidance mechanism based on an interchange of null messages [3] is used for this purpose.

The rest of the paper assumes that the reader has a knowledge of the above-mentioned synchronization mechanisms. A good source of information about the topic is [4].

The paper is structured as follows. First some general characteristics of the performed experimental work are described. The actual experiments carried out for the simulators evaluation are then described, followed by specific analysis of the results for each of the simulators: conservative (CMB), optimistic (TW) and synchronous (SPED). The paper ends with some conclusions.

2: Overview of the experiments

The performed set of experiments uses the following parameters of the model:

- *Message length*: 4 and 32-*flit* (flow-control digit) messages were considered. It is assumed that a flit advances from one router to another in one cycle (of simulated time).
- *Load of the network*: amount of information generated by the processing nodes, measured from 0 (none) to 100 (theoretical maximum). The maximum (100) corresponds to the bisection bandwidth (in flits) of the network, that is, to a situation where the channels of the network bisection are continuously utilized. A given load level can be reached with many short messages or with few long messages.
- *Size of the network*: most of the experiments assume a network of 16×16 routers, but some results for other network sizes are also reported.

Other parameters such as the size of the router queues, the communication patterns and the routing strategy were fixed, although they could be easily changed (if the purpose of the study was to analyze the model, instead of the simulators). The transit queues can hold 10 messages, while the injection queue has enough room for 4 messages. A random communication pattern was used: a node can send messages to any of the other nodes, with the same probability. The (time) separation between messages generated at a given node is exponentially distributed, with a mean that is directly proportional to the message length and inversely proportional to the network size and load. The routing strategy is oblivious in order of dimension (first X, then Y), following a technique described in [1, 6] to avoid communication deadlocks. This is needed because the topology is a torus.

To perform a parallel simulation the simulated network is divided into squares of the same size, and each square is assigned to a logical process. This means that each LP simulates an aggregate of routers, instead of only one. The size of the square depends on the number of available processing elements (PEs): exactly one LP is mapped onto each PE.

No special effort is done to exploit the lookahead ability of this model, for the case of CMB [4]. It is possible to obtain lookahead when a LP simulates just a model element (a router). However, when a LP simulates an aggregate of routers the obtained lookahead is very low and the potential benefit does not pay for the effort of computing it. Fortunately, a lookahead of 1 cycle is always available, so deadlocks are conveniently avoided.

A sequential event-driven simulator was also implemented, in order to have a reference point to assess the achieved performance of the parallel simulators under different configurations and workloads. After measuring the execution time of a sequential and a parallel simulation of the same model, a speedup figure can be computed. The model simulated by the sequential and parallel programs are basically the same but, in order to make fair comparisons, some optimizations were included in the sequential version, which take advantage of the use of a single memory space.

3: Experiments with CMB

This section presents the results of seven experiments performed with CMB. Table 1 summarizes the parameters used in the experiments: the first three parameters (network size, message length and load) are related to the model. The fourth is the number of processing elements (and, thus, of LPs) used by the simulator. The obtained results, in the form of speedup curves, are plotted in Figures 2, 3 and 4.

Before analyzing the results, the purpose of these experiments is described. Experiments **1P—4P** deal with a relatively small model of 16×16 routers, with message length 4 or 32, and running on 4 or 16 PEs. The goal was to confirm some results obtained in previous experiments with a toy model [9]. In particular, the objective was to test the following hypothesis:

1 The simulator scales well, that is, the speedup with 16 PEs is better than the speedup with 4 PEs. This should be confirmed by comparing **1P** with **2P**, and **3P** with **4P**.

2 Results are better with 4-flit messages than with 32-flit messages. With the same level of load, 4-flit messages means at least 8 times more simulation events than 32-flit messages. Execution times should be larger for the sequential and CMB simulators, but the latter will have more opportunities to self-synchronize without needing many null messages, reducing the synchronization overhead. This should be confirmed by comparing **1P** with **3P** and **2P** with **4P**.

3 Performance improves for higher loads. Again, the higher the load the larger the number of events managed by the simulator, giving more synchronization opportunities. This should be confirmed by the four experiments.

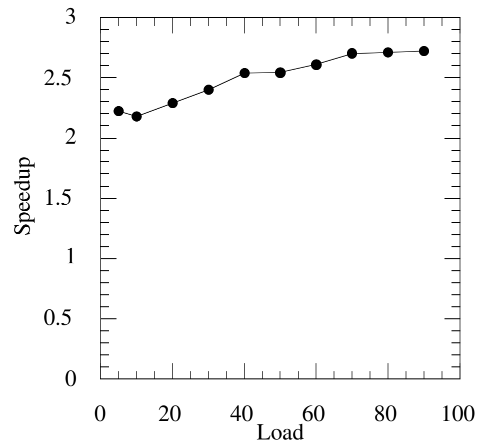
Experiments **5P** and **6P** deal with a model four times larger than that used in the previous set. The goal is to confirm that CMB performance improves with the imposed workload. In this case it is not necessary to use a synthetic workload (as was the case with the toy model), because the natural way of increasing the work performed by a LP is increasing the number of routers it simulates, and this is achieved simulating larger networks.

Experiment **7P** is a more complete scalability test. A medium-size model of 24×24 routers is simulated, using 4, 9, 16, 36 and 64 PEs.

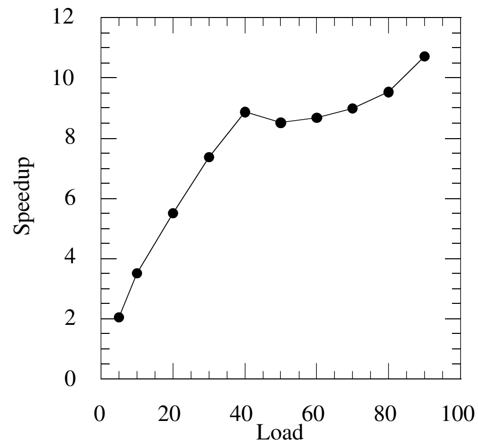
For all the experiments the simulator ran for 4000 cycles. It should be noticed that some of these experiments are not well dimensioned for the Paragon. Execution times are very short (less than 3 seconds in some cases), and a minimum variation in the run time given by the system can result in a significant variation of speedup, so trends are more significant than actual values.

	1P	2P	3P	4P	5P	6P	7P
Network size	16×16	16×16	16×16	16×16	32×32	32×32	24×24
Message length	4	4	32	32	4	32	4
Load	5—90	5—90	5—90	5—90	5—90	5—90	50
Number of PEs	4	16	4	16	16	16	4—64

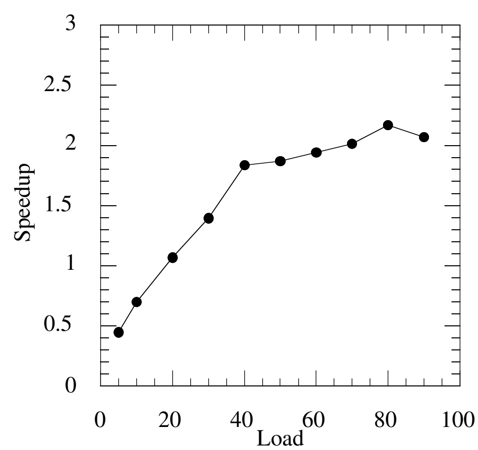
Table 1. Experiments performed with CMB-DA in the Paragon.



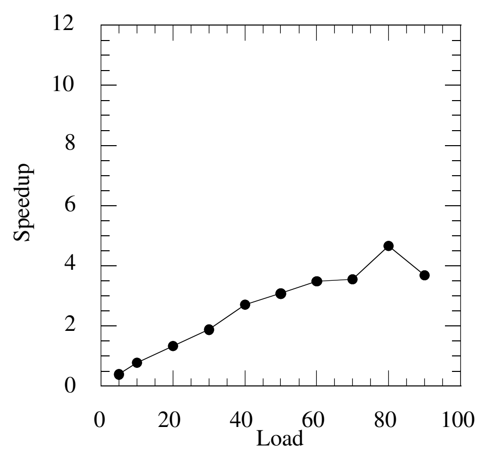
Experiment **1P**. 4-flit messages, 4 PEs



Experiment **2P**. 4-flit messages, 16 PEs



Experiment **3P**. 32-flit messages, 4 PEs



Experiment **4P**. 32-flit messages, 16 PEs

Figure 2. Results of experiments 1P—4P **with CMB. Simulation of a network of 16×16 routers.**

The results shown in Figures 2 and 3 confirm the initial hypothesis: the best performance is obtained with large, highly loaded models managing short messages. These scenarios are a challenge for sequential simulators, while CMB distributes the load quite evenly, effectively exploiting the available parallelism.

The results of experiment 7P, plotted in Figure 4 (right), shows that CMB scales very well, being the plot almost a straight line. The event density is high enough to keep all the PEs busy most of their time, even for the case of 64 PEs.

Figure 5 shows the contribution of computation and synchronization to LP execution time when running experiment **7P** with CMB. **Tsim** is the effective time performing simulations of events, plus the time needed to send non-null messages. **Trec** is the time spent receiving messages, useful or null. As the receive operation is blocking, Trec mainly accounts for the time LPs are awaiting for messages, not for the actual fact of reading a received message. **Thn** represents the time a LP spends sending null messages and checking if there is any message able to be consumed.

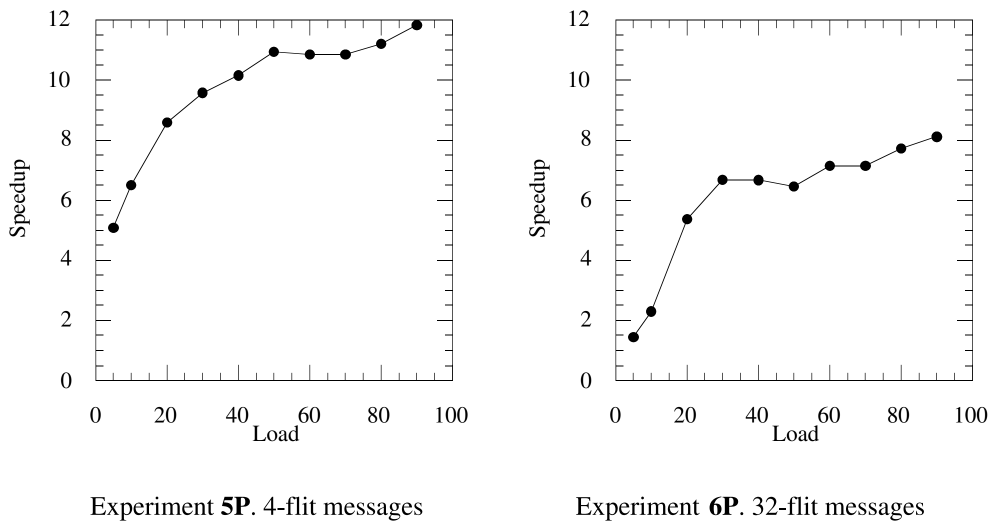


Figure 3. Results of experiments 5P and 6P with CMB. Simulation of a network of 32×32 routers in 16 PEs.

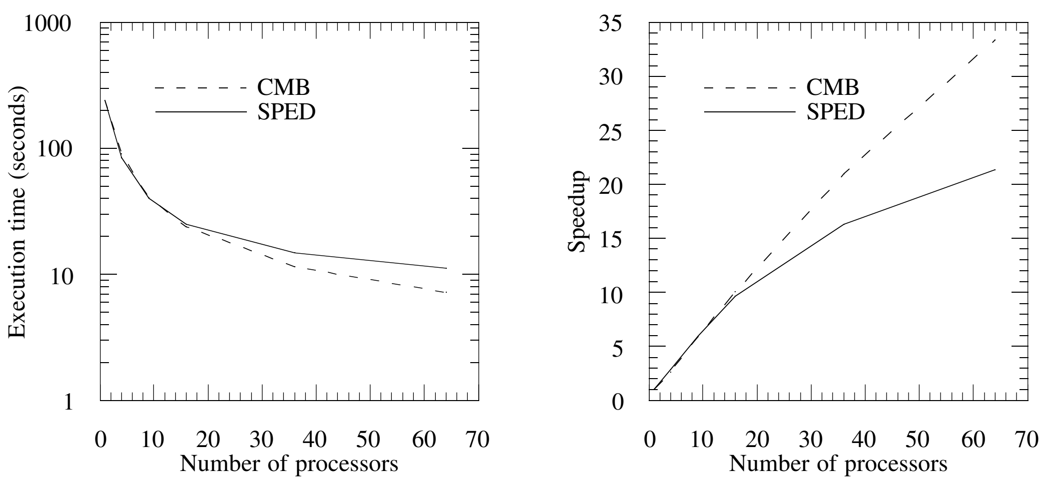


Figure 4. Execution times (left) and achieved speedups (right) of SPED and CMB running experiment 7P.

Clearly, if the model is distributed among a large number of LPs, the amount of work assigned to each LP decreases, and the synchronization demands increases. This can be seen very clearly in the Figure 5: when only 4 LPs are used, 85% of the execution time is simulation, and 15% is synchronization. When the number of LPs is increased to 64, these figures change to 55% and 45% respectively.

It is clear that if more PEs were utilized the proportion of synchronization time would increase, resulting in a performance degradation. However, for a big enough problem, it pays to use all the available PEs.

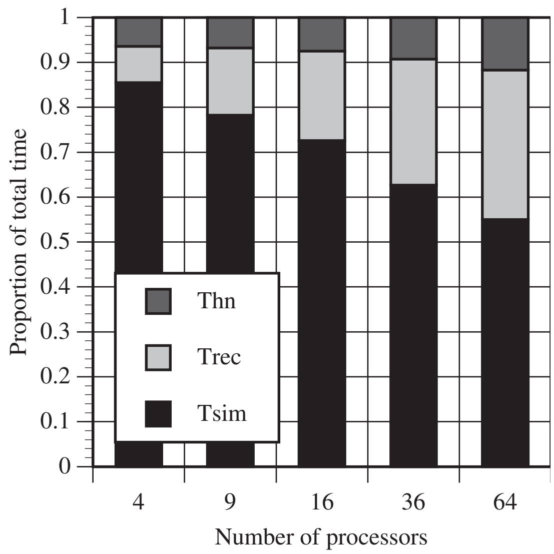


Figure 5. Distribution of the execution time of the LPs in CMB among simulation and synchronization.

4: Experiments with TW

Preliminary tests done with an implementation of TW using *conservative time windows* gave very poor performance figures, much worse than it was expected from [9]. This was (partly) caused by the large size of the state to be saved before executing each event: the state of the LP is the combination of all the state variables that represent the collection of routers assigned to the LP, plus a small number of variables for statistics gathering. This is excessive, because an event only modifies the state of one router, plus the statistics.

This fact led to the implementation of an optimization known as *incremental state saving*. Before executing an event, only the part of the state that is affected by that event is saved, making a copy of the full state only every *N* events. The interval *N* can be a static value or, as is the case for our implementation, dynamically adjusted. This optimization doubled execution speed, but performance remains inferior to that of CMB. In fact, for all the performed tests, the sequential simulator *was always much faster* that TW, independently of the number of PEs.

From this particular experience, it is concluded that TW is not a suitable approach to the study of the particular kind of model presented here. Execution monitoring revealed that almost all the events executed in a speculative way have to be undone. The result is that the simulator looses most of its time doing these “management” activities:

- Storing state copies and antimessages.
- Traversing the input queue to locate messages to annihilate, or to find the right place to insert a message belonging to the past.
- Coast-forwarding to reconstruct appropriate state versions.
- Sending and managing antimessages.
- Computing the global virtual time.

In the list of sources of overhead, insertion of future events and extraction of events to simulate, have not been included, because those activities have to be done in any simulator. However, as many of those insertions and extractions are speculative and have to be undone an done again, they also contribute to the overall overhead.

It is interesting to observe that, despite the good deal of effort devoted to the implementation of the TW simulator in a distributed memory environment, its performance remains discouragingly low. This seems to be an exception to the observations by Fujimoto in [5], one of the most comprehensive evaluations of TW: “*thus far, the experience of researchers (with TW) has been that (situations where most of the time is spent executing incorrect computations and rolling them back) are seldom encountered in practice, and, when discovered, usually points to a correctable flaw in the implementation rather than any fundamental flaw in the algorithm*”. It should be mentioned that these observations were made in the context of a shared memory implementation of TW.

As also pointed out in [5], state saving overhead seems to be the root cause of this poor performance. Fujimoto concludes that “*State-saving overhead limits the effectiveness of TW to applications where the amount of computation required to process an event can be made significantly larger that the cost of saving a state vector.*” This is not the case for this particular application: the cost of executing an event is very low, while the state size is huge. Even when incremental state saving is used the amount of data moved for state saving is several times larger than the data moved to execute an event.

It can be concluded that TW is not a viable approach for the kind of parallel computer and the domain of models presented in this study. This must be said without implying that TW is not viable in any context: many references can be found in the literature that report successful application of this synchronization method, in different hardware platforms and with different problem domains.

5: Experiments with SPED

After performing a series of experiments with the SPED simulator in the Paragon, it was observed that it behaves very much like CMB, with only a significant difference: SPED degrades sooner when the load of the LPs is reduced. Figure 4 (left) shows the execution time of these two simulators when running experiment **7P** (see Table 1). The scale is logarithmic to better appreciate the differences. Figure 4 (right) shows the speedups achieved by both simulators as a function of the number of PEs. For small number of PEs, SPED performs slightly better than CMB. However, when the number of PEs increases (and, therefore, the workload of each LP decreases) CMB performs better than SPED.

Figure 6 shows the contribution of computation and synchronization to LP execution time when running experiment **7P** with SPED. **Tsim** is the time devoted to simulate events and send messages. **Trec** is the time spent awaiting and receiving messages. **Tbar** is the time spent barrier-synchronizing. The Figure is self-explanatory: the problem size of experiment **7P** is large enough to keep 4 PEs busy most of the time but, when distributed along 64 nodes, each node needs most of its time for synchronization (60%), performing useful computation only a 40% of the time.

An experiment like **7P**, but simulating a network of 90×90 nodes, has also been done with SPED and CMB. The results indicate that, for this case, SPED performs slightly better than CMB, even using 100 Paragon PEs, and that both simulators scale equally well. The reason is that for this size of model the workload is large enough to keep all the LPs busy, performing useful computation most of the time and minimizing the (relative) effort devoted to synchronization.

The number of barriers executed by the LPs was observed to be 4000 in all the cases, that is, as many barriers as simulation cycles. The reason for this is simple: the density of events is big enough to have always at least one event per cycle. Only for some particular cases with small models and very low loads (less than 5) the number of barriers was found to be less than the number of cycles.

A good characteristic of this simulator is that obtains reasonable speedups using a very simple algorithm, without the complicated memory management strategies of TW, and without some the restrictions of CMB (no cycles with zero timestamp increment).

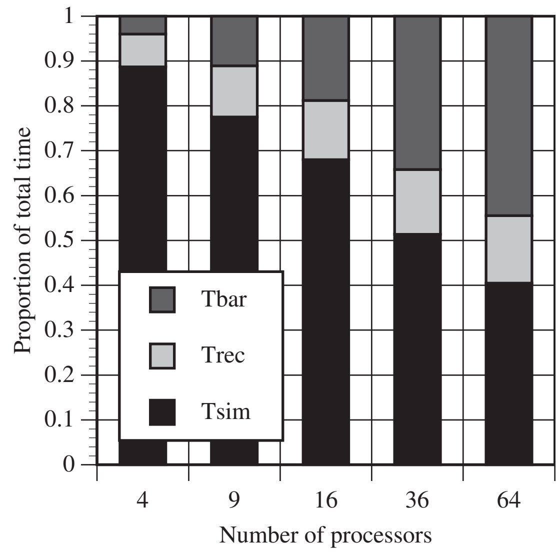


Figure 6. Distribution of the execution time of the LPs in SPED among simulation and synchronization.

6: Conclusions

In this paper a comparative experimental evaluation of three parallel discrete-event simulators for the study a model of a message passing network has been presented. The knowledge obtained from previous experiments with a toy model has been applied, when possible, to this, more realistic, model.

In general, preliminary ideas on which characteristics of the simulated model help to improve the performance of the conservative parallel simulator have been confirmed. In order to take advantage of CMB, a model with a high degree of internal communication is needed, which allows processes to remain synchronized without needing null messages. In the model used in this study, this happens when network size is large, load is high and messages are short. These three parameters have the strongest influence in the performance of the simulator. It is interesting to observe that, conveniently, the best scenario for the CMB simulator is the worst for the sequential event-driven simulator: speedups are best precisely when they are most needed, and are poor only in cases where simulation runs are very short in the sequential but also in the parallel version.

The experience with optimistic synchronization has been quite discouraging. The performance of the tested TW implementation is very poor, even after introducing a series of optimizations. The conclusion is that TW is not the right tool for the kind of study presented here.

Finally, the experience with SPED, the synchronous parallel simulator, has been very positive. It performs worse than CMB only for very lowly loaded systems, being as efficient as CMB (or even more efficient) for medium and highly loaded systems. It has the additional advantage of a more simple implementation, and can be used in environments were CMB is not applicable because of the presence of loops with zero timestamp increment.

Acknowledgments

Many thanks to the personnel at Purdue University Computing Center, Research Computing Division, for their technical support using the Paragon.

References

[1] Arruabarrena A. 1993. *Análisis y evaluación de sistemas de interconexión para procesadores masivamente paralelos*. PhD Dissertation, Dept. of Computer Architecture and Technology, University of the Basque Country.

[2] Bryant, R.E. 1977. *Simulation of packet communications architecture computer systems*. MIT-LCS-TR-188, Massachusetts Institute of Technology.

[3] Chandy, K.M. and Misra, J. 1979. "distributed simulation: a case study in design and verification of distributed programs" *IEEE Transactions on Software Engineering SE-5*, no. 5 (Sept.), 440—452.

[4] Fujimoto, R.M. 1990. "Parallel discrete event simulation". *Comm. of the ACM 33*, no. 10 (Oct.) 31—53.

[5] Fujimoto, R.M. 1990 "Optimistic approaches to parallel discrete event simulation". *Trans. of The Society for Comp. Simulation*, Vol. 7, No. 2, 153—191.

[6] Izu, C. 1994. *Análisis, evaluación y aportaciones al diseño de arquitecturas para encaminadores de mensajes*. PhD Dissertation, Dept. of Computer Architecture and Technology, University of the Basque Country.

[7] Jefferson, D.R. 1985. "Virtual time". *ACM Trans. on Programming Languages and Systems 7*, no. 3 (July) 405—425.

[8] Miguel, J.; Arruabarrena A. and Beivide, R. 1993. "Conservative parallel discrete event simulation in a transputer-based multicomputer". In *Transputer Applications and Systems '93*. R. Grebe et al. (Eds.) IOS Press, 636—650.

[9] Miguel, J.; Arruabarrena A. and Beivide, R. 1994. "Conservative and optimistic distributed simulation in massively parallel computers: a comparative study." In *Proc. of the Conference on Massively Parallel Computing Systems MPCS'94* (Ischia, Italy, Sept.), 528—532.

[10] Miguel, J.; Arruabarrena, A. and Beivide, R. 1995. "Parallel simulation of message routing networks". In *Proc. 3rd. Euromicro Workshop on Parallel and Distributed Processing PDP'95* (San Remo, Italy, Jan.), 138—145.

[11] Soulé, L. 1992. Parallel logic simulation: an evaluation of centralized-time and distributed-time algorithms. PhD dissertation. Stanford Univ. Technical Report CSL-TR-92-527.