

Logical Process based Sequential Simulation Cloning

Patrick Peschlow, Martin Geuer, Peter Martini
University of Bonn
Department of Computer Science IV
Roemerstr. 164, 53117 Bonn, Germany
{peschlow, geuer, martini}@cs.uni-bonn.de

Abstract

Discrete-event simulation is a widely used technique for the performance evaluation of systems. Recently, discrete-event simulation studies which explore alternative scenarios within a single simulation run have gained popularity. The simulation is branched at decision points, and the different branches are simulated one after another. In this paper, we show that concepts from parallel and distributed simulation can considerably speed up this type of discrete-event simulation study. We present a sequential branching mechanism, LPseq, which is based on the logical process paradigm and uses logical process cloning for efficient computation of branches. Our performance evaluation demonstrates that LPseq can achieve considerable speedup compared to the traditional branching approach.

1. Introduction

Discrete-event simulation is a very popular technique for the performance evaluation of systems. The underlying concept of discrete-event models is simple yet powerful: the system state is stored as a set of variables, and is only modified by events at discrete points in time. During the simulation, all pending events are stored in a future event list (FEL) and executed in the order of their timestamps.

The ever increasing complexity of simulation scenarios has led to the development of parallel and distributed discrete-event simulation (PADS) techniques. Compared to sequential simulation on a single processor, parallel simulation is able to speed up simulation runs by utilizing multiprocessor or multicore architectures. Distributed simulation on interconnected hosts facilitates the simulation of very large scenarios. In order to enable parallel execution, PADS partitions the system state into logical processes (LPs), which are then distributed onto the available processors or hosts. Each LP has its own FEL, executes events

related to its state partition, and forwards other events to their destination LPs. While the LPs usually operate asynchronously, they have to adhere to synchronization mechanisms in order to guarantee the correctness of the simulation. Since the 1980's, PADS is a widely researched topic, and the different methods and techniques have advanced over the years. For an introduction to PADS and a comprehensive summary of recent advances see [6, 13].

Advances in sequential discrete-event simulation, such as the development of high performance FEL data structures, are usually incorporated into PADS. In contrast to that, PADS techniques are rarely considered useful in sequential simulations, mostly due to their inherent overhead. Recently, however, it has been shown that, depending on characteristics of the simulated scenario, LP-based sequential simulations may run noticeably faster than traditional sequential simulations based on a single FEL [5, 11]. We give an overview of these research results in section 2.

In this paper, we focus on a specific, increasingly popular application area of discrete-event simulation: exploring alternative scenarios by branching simulation runs. Important applications for branching mechanisms include, e.g., comparing different queuing strategies, examining the impact of different mobility patterns, and analyzing the effects of simultaneous events. We demonstrate that an LP-based sequential simulation can take advantage of much more efficient branching techniques than those normally used in sequential simulation. Specifically, we use LP cloning mechanisms which share common computations among different branches. Our performance evaluation shows that LP-based sequential simulation cloning can achieve considerable speedup when compared to the traditional way of branching sequential simulations.

The paper is structured as follows: section 2 gives an overview of previous work on LP-based sequential simulations and LP cloning techniques. Section 3 introduces LPseq, an LP-based sequential cloning algorithm. We evaluate the performance of LPseq in section 4. Section 5 concludes the paper and discusses possible future work.

2. Current state of research

In this section, we discuss previous work on LP-based sequential simulation. Furthermore, we give an overview of cloning mechanisms for parallel and distributed simulation.

2.1. LP-based sequential simulation

Recently, the sequential use of conservative parallel synchronization mechanisms has been shown to outperform traditional sequential simulation based on a single FEL under certain circumstances. In [5], Curry et al. examine the sequential performance of a modified CMB null message protocol [3]. The sequential implementation of the CMB protocol keeps all LPs in a priority queue, and always executes the LP with the smallest next event timestamp. Null messages between LPs are replaced by direct updates of variables. It is shown that the CMB approach can achieve noticeable speedup compared to traditional sequential simulation if requirements such as high event density, large lookahead, or relatively small connectivity of LPs are met. The speedup can be attributed to increased cache locality and smaller FEL sizes due to the partitioning into LPs.

A necessary condition for an LP-based approach to speed up a sequential simulation is that at least one event can be executed every time an LP is executed [11]. This is not guaranteed by the CMB approach, which therefore risks to considerably slow down the simulation. In order to eliminate this shortcoming, Kiddle et al. propose the CHASE algorithm [11]. CHASE keeps all LPs in a priority queue, sorted by their next events. In a main simulation loop, CHASE always schedules the first LP in the priority queue (i.e., an LP which contains a smallest-timestamped event) for execution. Instead of null messages as used by the CMB approach, CHASE uses a global lookahead value L for all LPs. L guarantees an LP which is executed at say timestamp t that all its events up to time $t + L$ are safe. Thus, by always executing the first LP in the priority queue, CHASE ensures that at least one event is executed in every iteration of the simulation loop. The performance evaluation in [11] demonstrates that CHASE is able to speed up sequential simulations and that it avoids the performance degradation of the CMB approach in less suitable scenarios.

2.2. Cloning

The concept of cloning parallel simulations in order to efficiently explore alternative model behavior has been introduced by Hybinette and Fujimoto in the context of interactive simulation [8]. At *decision points*, created by the user, alternative actions lead to different *scenarios*. The classical method of exploring alternative scenarios consists of running independent replications from the decision point on-

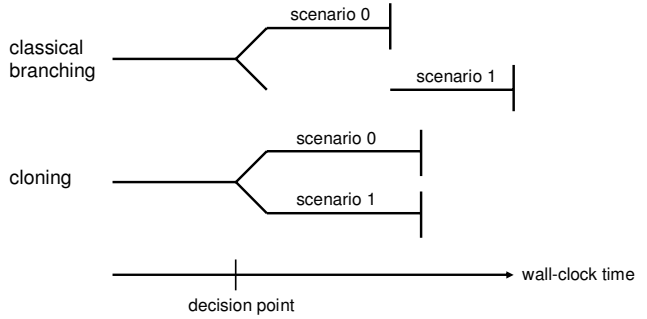


Figure 1. Cloning versus classical branching. With cloning, scenarios are computed in parallel and share common computations, which may reduce total simulation run-time.

wards. The basic idea of cloning is to speed up the computation of branches by running all scenarios in parallel so that they may share common computations (cf. figure 1).

With cloning, each LP carries an LP ID and a set of scenario IDs which identify the scenarios the LP belongs to. An LP with two or more scenario IDs is called a *shared LP*. Shared LPs enable scenarios to share computations. At a decision point, the cloning mechanism only creates clones of LPs which have different states in the resulting scenarios. Each clone receives a new scenario ID and inherits the LP ID of the cloned LP. All LPs which have identical states in the new scenarios do not create clones. Instead, the new scenario IDs are assigned to already existing clones which become shared LPs for these scenarios. As an example, assume that the simulation begins with scenario 0. Later on, at a decision point, alternative actions at LP $A\{0\}$ bring about a new scenario with ID 1. Then, LP $A\{0\}$ creates a clone $A\{1\}$. An LP $B\{0\}$, however, which is not affected by the alternative actions, becomes a shared LP $B\{0,1\}$.

Event messages sent by shared LPs contain multiple scenario IDs and may be received by several LPs. In the above example, event messages sent by LP $B\{0,1\}$ contain both scenario IDs 0 and 1, and may be received by LPs $A\{0\}$ and $A\{1\}$. Furthermore, shared LPs monitor the messages they receive. Whenever a shared LP receives different messages for different scenarios (or finds that it has not received a message for all its scenarios), it has to clone itself, too.

It has been shown that cloning can reduce simulation run-times dramatically when compared to independent replications. Furthermore, as long as enough memory is available, cloning usually scales well, especially when the differences between alternative scenarios spread slowly, or not at all. A comprehensive summary of LP cloning techniques is given in [9].

Recently, techniques which increase the performance of cloning have been proposed: *Merging* recombines cloned

LPs which have converged back to the same state again [1]. *Just-in-time-cloning* delays the creation of clones as much as possible [7]. Cloning mechanisms have also been developed for HLA-based distributed simulations. For a detailed summary see [4].

A well-known challenge in parallel and distributed simulation is the problem of simultaneous events, i.e., two or more events with the same timestamp. While simultaneous events are commonly handled by tie-breaking rules [16, 10], recent work has proposed a branching mechanism in order to analyze the effects of different execution orders of simultaneous events on simulation results. It has been shown that cloning can considerably speed up branching mechanisms for simultaneous events in parallel and distributed simulation [15].

3. Sequential Simulation Cloning

Cloning is a popular technique for exploring alternative scenarios in parallel and distributed simulation. So far, however, cloning has not been utilized in sequential simulation. In this section, we motivate the use of cloning in sequential simulation, develop an LP-based sequential cloning algorithm, and discuss its potential advantages and drawbacks.

3.1. Discussion of classical branching

Whenever a decision point is reached during a simulation run, the simulation is branched in order to examine the different alternatives. In sequential simulation, the standard branching technique is to compute all different branches independently as successive replications, one after another. As an example, consider a branching mechanism for the analysis of simultaneous events, as presented in [2]: when the FEL is headed by simultaneous events, they are executed in all possible orders. If two or more execution orders are found to lead to different system states, a decision point has been reached. One of the states is selected for further simulation, while the other states are stored to memory or hard disk. When the end of a branch is reached, i.e., the termination condition of the simulation is fulfilled, one of the previously saved states is restored and the simulation of the corresponding branch continues. This procedure is repeated until all branches have been completed.

With regard to efficiency, this classical branching scheme has two potential disadvantages: first of all, if only a part of the simulated system is affected by alternative behavior at decision points, there will be many duplicate computations among the different branches. Second, the whole set of simulation modules is treated as a single, global simulation state. At each decision point, this potentially large system state has to be copied and stored. Furthermore, it may have to be restored several times during the simulation.

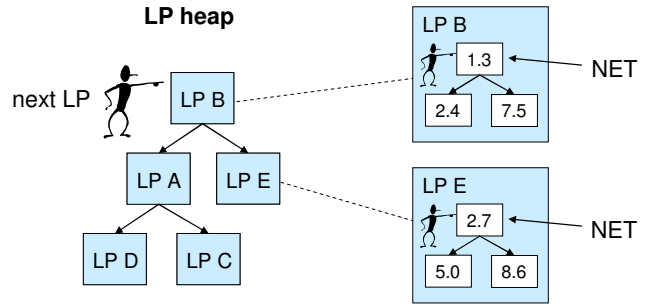


Figure 2. The LP heap used by LPseq.

These shortcomings of sequential simulation branching make it worthwhile to consider alternative approaches. Interestingly, the logical process paradigm from parallel simulation offers a potential solution: with LPs, sequential simulation can take advantage of LP cloning techniques which enable simulation branches to share computations (cf. section 2.2). Shared computations may include event executions but also further cloning at later decision points [15]. Unlike classical branching, cloning only copies parts of the system state when it is absolutely necessary. Thus, LP-based sequential simulation has a huge potential to outperform traditional sequential simulation if alternative scenarios are to be explored. In the following, we define a simple LP-based sequential simulation algorithm. Based on this algorithm, we develop a sequential cloning algorithm.

3.2. The LPseq algorithm

Just like parallel and distributed simulation, an LP-based sequential simulation requires a synchronization mechanism in order to avoid causality errors, i.e., event executions in the wrong order. We define a simple LP-based sequential algorithm, *LPseq*, which stores all LPs in a priority queue data structure, the *LP heap*. The LP heap keeps the LPs sorted by their *next event times*. The next event time (NET) of an LP is defined as the smallest timestamp of all events in its FEL. Figure 2 shows an example LP heap with five LPs, and the FELs of LP B and LP E.

The simulation runs in a main loop, which always schedules the first LP on the LP heap for an *execution session*. Within its execution session, the LP computes a safe time interval and executes all safe events in its FEL in increasing timestamp order. For the safe time calculation, the LP may utilize any available lookahead information. The loop exits when the termination condition of the simulation is fulfilled.

LPseq is similar to the CHASE algorithm described in section 2.1. However, in contrast to CHASE, which uses explicit channel data structures between the LPs, LPseq adds all created events immediately to the FELs of their destination LPs. While CHASE attempts to keep FELs small this

way, we refrain from the use of channels for two reasons. First, in addition to the replication of system states, cloning would also involve the creation of channel structures whenever a decision point leads to new scenarios. We found this to lead to a very high memory demand. Second, the channel-based approach relies on an assumption commonly used in conservative parallel simulations, that all events sent on the same channel have increasing timestamps. Without this assumption, LPseq does not impose such restrictions on the simulated scenario and can therefore be used for arbitrary sequential simulations.

LPseq enables cloning in sequential simulation, but also has a potential drawback: even though the LP heap achieves synchronization between the LPs by relatively simple means, it may nevertheless cause noticeable overhead. Therefore, a cloning mechanism only pays off if its performance benefits outweigh this overhead. In some cases, however, this is not possible. Assume that a simulation run does not contain any decision points at all (which may not be known in advance). Then, cloning is not required, yet the LP synchronization overhead remains. Similarly, if LPs are cloned but the differences between alternative scenarios spread rapidly among them, many (or all) shared LPs also have to clone themselves very soon. Again, the amount of shared computations will be small and the LP synchronization overhead remains.

Finally, an implication of LP-based simulation is that the simulation modules have to be allocated to LPs at the beginning of the simulation. However, while this usually has to be done carefully in parallel simulation in order to avoid load imbalances, this danger does not exist in sequential simulation. Therefore, a simple solution is to use a 1:1-mapping between modules and LPs. Alternatively, if there is a-priori knowledge about the occurrence of decision points, the user may choose a specific allocation of modules to LPs in order to maximize the performance of cloning.

3.3. Cloning with LPseq

The basic cloning scheme used by LPseq is identical to the cloning mechanism described in section 2.2: at a decision point, new clones are created for LPs which show alternative behavior. All other LPs become shared LPs for the new scenarios. Shared LPs always monitor the scenario IDs of event messages they receive. If an LP is to execute an event e , but then detects it has not received e for all its scenarios, it creates new clones of itself accordingly.

The sequential environment allows for an efficient implementation of cloning. Since there is only a single thread of execution, events need not carry scenario IDs as in parallel simulation. Instead, new events may be scheduled at their receivers immediately, with their scenario IDs handed over as additional method parameters.

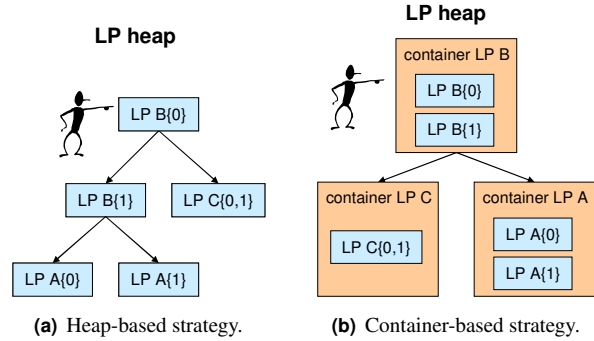


Figure 3. Strategies for LP management.

Every time a clone is created, it has to be included into the LP scheduling mechanism so that it may execute events. This leads to an important design decision between two possible strategies.

Heap-based strategy: new clones are simply added to the LP heap. This requires only small modifications to the structure of LPseq. A drawback of this approach, however, is that clones with identical LP ID (but different scenario IDs) are also synchronized by the LP heap, although they will never send events to each other. Thus, with a large number of clones, the LP heap will likely spend much more effort on sorting LPs than required for the correctness of the simulation.

Container-based strategy: the original LPs are used as *container LPs* for all their clones, i.e., a container LP manages all clones with the same LP ID. This approach aims to keep the LP heap (and thus the sorting overhead compared to traditional sequential simulation) as small as possible. Nevertheless, this means that additional work has to be done inside the container LPs. First of all, a container LP has to compute its NET (as the sorting criterion for the LP heap, cf. section 3.2) by taking into account all its clones. A minimum NET value has to be stored, and updated by the clones every time they execute or receive events. Furthermore, during its execution session, a container LP has to iterate over all its clones and let them execute their safe events. This, however, can be implemented as a simple unsorted iteration, since there is no need to synchronize different clones of the same LP.

Figure 3 summarizes the two LP management strategies. It is worth noting that, theoretically, the container-based strategy has a significant potential advantage with regard to the event density: from the point of view of the LP heap, the number of events executed in each LP execution session can be expected to be much larger than with the heap-based strategy. We have included both strategies into LPseq and compare their performance in the next section.

4. Experiments and Results

In this section, we evaluate the performance of LPseq cloning in comparison to classical branching. We have implemented LPseq and the LP management strategies introduced in section 3.3 for the discrete-event simulator MOOSE [12], which already contains a sequential branching mechanism. In the following, we refer to the heap-based strategy as “CL (heap)” and to the container-based strategy as “CL (container)”. Additionally, we have implemented a classical branching scheme based on LPseq, referred to as “BR”, to examine the overhead caused by the LP heap separately from the performance of the cloning mechanism.

4.1. Simulation setup

It is well known from previous research that the performance of cloning mainly depends on the amount of shared computations (cf. section 2.2). Therefore, we base our evaluation on two different types of scenarios: in *structured* scenarios, the LPs are divided into groups which exchange events much more frequently. This represents systems with an inherent spatial decomposition, e.g., cellular networks, manufacturing lines, or air traffic simulations. In contrast to that, *dynamic* scenarios do not impose any restriction on the communication between LPs. This conforms to highly dynamic systems such as wireless mobile networks.

We represent both scenario types by a variant of the synthetic workload model described in [14]. The model consists of nodes which schedule events at each other. When executing an event with timestamp t , a node schedules a newly created event at another, randomly chosen node. The timestamp of the new event is randomly chosen from the interval $[t + d_{\min}, t + d_{\max}]$, where d_{\min} and d_{\max} are constants set at the beginning of the simulation. Each node is mapped to its own LP. For structured scenarios, we divide the set of nodes into disjunct groups, where nodes only schedule events at other nodes of the same group. For dynamic scenarios, we do not specify groups.

All simulations were run on a 3GHz Intel Pentium 4 computer with 1GB of RAM, Ubuntu linux 6.06, kernel version 2.6.15-28-686. We use the computer exclusively for the simulations, in order to minimize the influence of background load. We run multiple replications of all simulations, and all figures include 95% confidence intervals (which, however, are very small throughout).

In the following, we examine small scenarios with 50 LPs and larger scenarios with 150 LPs. In structured scenarios, we divide the LPs into five groups of equal size. We create simulation branches at three decision points, always at the same point in simulation time. We run one simulation series with 8 branches and another one with 216 branches.

In order to guarantee a fair evaluation with respect to tra-

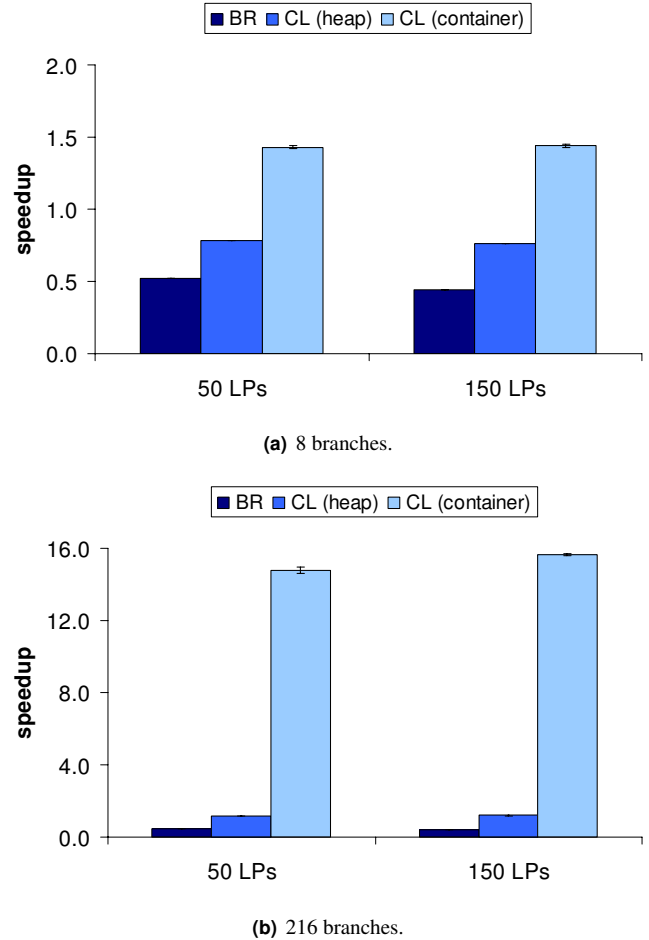


Figure 4. Speedup (structured scenarios).

ditional sequential simulation (in the following referred to as “Seq”), we make sure that the LP structure of LPseq already causes noticeable overhead. We use a low event density of three events per node, and a small lookahead which only enables LPs to execute a single event at a time. Reference simulations without decision points have confirmed that these settings result in the desired effect: in all scenarios, the speedup achieved by LPseq compared to Seq is at most 0.5. Thus, without branching, it takes LPseq at least twice as long to complete the simulation.

4.2. Structured scenarios

Figure 4 shows the performance results for LPseq in structured scenarios. First of all, we discuss the LP-based classical branching scheme (“BR”), in order to evaluate the LP management overhead without the use of cloning mechanisms. According to both Figure 4(a) and Figure 4(b), “BR” results in a speedup of about 0.5, which conforms to the reference simulations without decision points (cf. section 4.1).

Thus, the overhead caused by the LP heap in comparison to Seq stays the same with additional branches.

As Figure 4(a) shows, heap-based cloning runs faster than LP-based branching, but still slower than Seq. Container-based cloning, however, achieves a noticeable speedup of about 1.5 compared to Seq. These results are almost identical for 50 and 150 LPs. In the scenario with 216 branches (see Figure 4(b)), the difference between heap-based and container-based cloning is remarkable. Although heap-based cloning achieves a speedup of about 1.2 compared to Seq, this is negligible compared to the speedup of 14.8 (for 50 LPs) resp. 15.7 (for 150 LPs) achieved by container-based cloning. Speaking in absolute values, this means that, on average, container-based cloning is already finished with the computation of all 216 branches while Seq branching is still simulating the 14th branch. We draw two conclusions for structured scenarios:

1. Container-based cloning is clearly preferable to heap-based cloning. Heap-based cloning is simple, but increases the main bottleneck of LPseq, the LP heap. In contrast to that, container-based cloning keeps the LP heap small and does not synchronize clones of the same LP. Although a container LP still has to iterate over its clones internally, many of them may have safe events scheduled for execution. Therefore, an increasing number of clones of a container LP also increases the expected number of events that are executed within its execution session.
2. In scenarios with many branches, container-based cloning is able to achieve remarkable speedup compared to traditional sequential branching. Also, in scenarios with only few branches, it is preferable to Seq branching because of the large amount of shared computations among the different branches.

4.3. Dynamic scenarios

Figure 5 shows the speedup achieved by LPseq in dynamic scenarios. Just as in structured scenarios, the speedup of almost 0.5 with “BR” shows that the relative overhead caused by the LP heap is similar to the reference simulations. The performance of the cloning mechanisms, however, is completely different compared to the structured scenarios. First of all, heap-based cloning is slower than LP-based branching in all dynamic scenarios. Furthermore, in the scenarios with 216 branches, there is a strong decline in speedup, with values of approx. 0.1 compared to Seq branching. This is because, in dynamic scenarios, the differences between branches spread rapidly among the shared LPs, which results in a large number of clones very soon. With 216 branches, the LP heap will contain many thousands of LPs. Still, just as in structured scenarios, container-based cloning

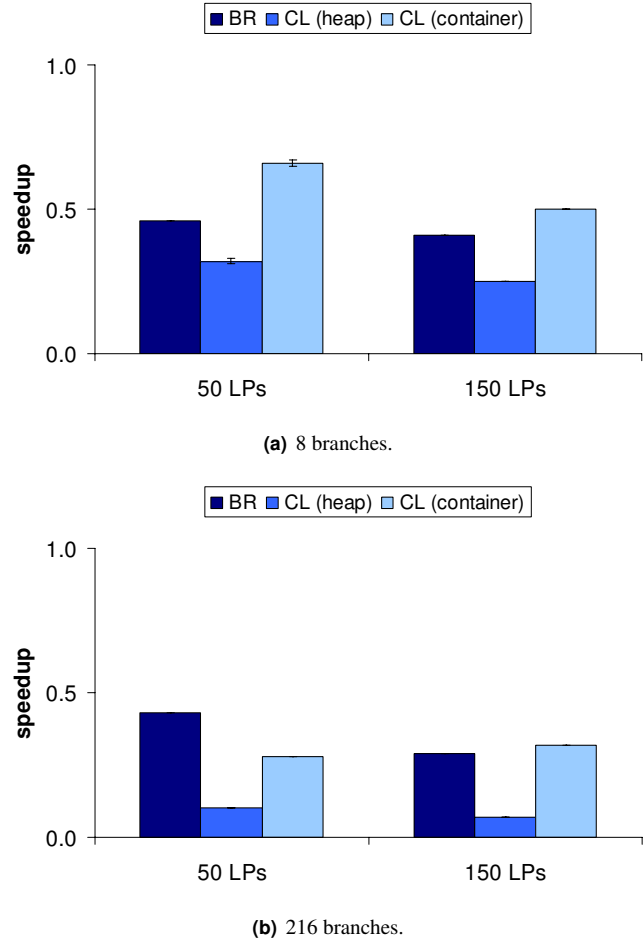


Figure 5. Speedup (dynamic scenarios).

achieves higher speedup than heap-based cloning in all simulations. However, even though container-based cloning does not suffer a comparable decrease in performance, it runs only slightly faster than “BR” in most scenarios and is even slower in scenarios with 50 LPs and 216 branches. This confirms that the main advantage of cloning is nullified by the lack of common computations.

Most importantly, none of the LP-based mechanisms is able to outperform Seq. We therefore draw the following conclusion for the dynamic scenarios examined here: LPseq is not able to achieve speedup compared to traditional sequential simulation branching. Instead, it may lead to considerably larger simulation run-times.

4.4. Discussion

In the preceding sections, we have evaluated the LPseq cloning algorithm in structured as well as dynamic scenarios. It is important to note, however, that the performance evaluation so far does not take into account the possible

overhead induced by decision points. Decision points at predefined instants in time, as in our performance evaluation, do not cause any additional overhead apart from setting up the different branches. However, as described in section 2.2, decision points may also arise dynamically, depending on conditions which have to be checked during the simulation and may therefore cause overhead. One example is the analysis of simultaneous events.

4.5. Case study: simultaneous events

In the following, we present a case study about branching in the context of analyzing simultaneous events. During the simulation, FELs are monitored for the occurrence of simultaneous events. Whenever simultaneous events are detected, their possible execution orders are analyzed. If they lead to different states, branches are created accordingly.

Now consider two simultaneous events e_1 and e_2 which are to be handled by two different simulation modules. With traditional sequential simulation based on a single FEL, the execution orders of e_1 and e_2 would be analyzed as described above. However, unless the two modules share state variables, the execution orders of such events will always lead to identical states. This may result in a lot of unnecessary overhead. Here, an LP-based approach is potentially advantageous: e_1 and e_2 would be scheduled at different LPs, i.e., in different FELs, and thus not even considered as simultaneous.

Another important aspect is that, even in dynamic scenarios, it is well possible that the effects of simultaneous events do not spread among many LPs. As an example, consider a communication network. Assume that a TCP timeout event and the reception of the corresponding TCP acknowledgment happen simultaneously. Then, only one execution order of these events causes the TCP sender to time out and issue a retransmission. Since the resulting internal state of the node is different for both execution orders, two branches are created. However, if the node has further packets to send, its behavior as observed by many other nodes in the network may well stay the same. In this situation, LPseq cloning may prove very effective.

We examined this situation in wireless network simulations based on IEEE 802.11 Wireless LAN. The network consists of 100 nodes, which send data packets to randomly chosen destinations. Each node is mapped to its own LP. We differentiate between two types of scenarios: in the “no spread” scenarios, the execution orders of two simultaneous events lead to different internal state at a node which does, however, not affect the other nodes in the network (as in the TCP example above). In contrast to that, in the “spread” scenarios, we assume that the node experiencing the timeout does not have further packets to send. With one execution order, it simply stops sending packets. Thus, the effects of

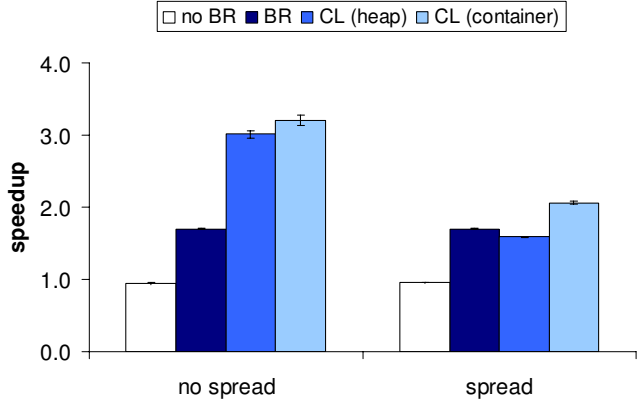


Figure 6. Speedup (network scenario).

the simultaneous events spread to all other nodes not much later than the decision point.

Figure 6 shows the speedup achieved by the different LPseq algorithms when compared to Seq branching. For reference, we include the results of simulation runs in which the analysis of simultaneous events was deactivated (“no BR”). LPseq achieves a speedup of 0.95 in the “no BR” case and thus is slightly slower than Seq. This can be attributed to the overhead caused by the LP heap. However, with “BR”, the use of LPseq already pays off: it achieves a speedup of 1.7 compared to Seq. This is a huge difference to the results in sections 4.2 and 4.3, where LPseq branching was noticeably slower than Seq. The reason for this considerable speedup in the wireless network scenario is that the explicit spatial decomposition into LPs strongly reduces the number of detected simultaneous events. In contrast to that, due to the single FEL, Seq also examines simultaneous events handled by different nodes (and thus causes “false positives”). Although our simulations use an optimized version of Seq which only analyzes simultaneous events if their handling modules differ, the results show that this still causes noticeable overhead. Note that the performance gain by LPseq branching (“BR”) is identical in the “spread” and “no spread” scenarios, since it immediately replicates the complete system state.

As expected, the performance of the cloning mechanisms is different in the two scenarios. In the “no spread” scenarios, both the heap-based and the container-based algorithm far exceed the speedup gained by “BR”. Since almost all computations can be shared from the decision point onwards, the speedup of 3.0 resp. 3.2 is almost twice as large as with LPseq branching. Note that the speedup achieved by “CL (heap)” is only slightly smaller than the speedup of “CL (container)”. Since there are only two different simulation branches, the heap-based approach causes only small additional LP management overhead. In the

“spread” scenarios, there are only very few computations shared between the two branches. Therefore, the potential for achieving additional speedup compared to “BR” is small. Container-based LPseq cloning runs only slightly faster than “BR”, while the heap-based cloning scheme even experiences a small decline in speedup due to the LP management overhead.

4.6. Summary

Our performance evaluation has shown that the container-based LPseq cloning algorithm is clearly preferable to the heap-based cloning algorithm. Furthermore, compared to traditional sequential simulation branching, the use of container-based LPseq cloning is extremely beneficial in structured scenarios where the effects of decision points do not spread among many LPs (cf. section 4.2). In contrast to that, LPseq shows poor performance in dynamic scenarios, if predefined decision points are used (cf. section 4.3). However, as shown by the case study on simultaneous events, LPseq cloning may also achieve considerable speedup in dynamic scenarios if a different branching criterion is used. We therefore conclude that LPseq should be used with careful consideration of both the concrete scenario at hand and the expected overhead induced by decision points.

5. Conclusions and future work

In this paper, we have motivated and developed an LP-based cloning algorithm, LPseq, for sequential discrete-event simulation. Although LPs are used almost exclusively in parallel and distributed simulation, we have demonstrated that LP cloning can be used effectively for exploring alternative scenarios within sequential simulation runs.

Our performance evaluation has shown that the use of LPseq can lead to considerable speedup in structured scenarios. In dynamic scenarios, however, we cannot draw a definite conclusion. While the performance of LPseq is poor in scenarios with predefined decision points, it achieves noticeable speedup in the context of analyzing simultaneous events because the LP partitioning strongly reduces the number of “false positives”.

In future work, we are going to increase the robustness of LPseq in dynamic scenarios. We plan to examine combinations of traditional sequential simulation and LPseq. One idea is to use a single future event list for all LPs as long as no decision point is encountered during the simulation. At a decision point, however, the algorithm may switch to LP cloning. Similarly, it may pay off to switch back to classical branching at some point in the simulation, e.g., when a large fraction of LPs has eventually been cloned and there are only few shared computations left.

References

- [1] A. Agarwal and M. Hybinette. Merging parallel simulation programs. In *Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation (PADS '05)*, 2005.
- [2] C. Barz, R. Göppfarth, P. Martini, and A. Wenzel. A new framework for the analysis of simultaneous events. In *Proceedings of the 2003 Summer Computer Simulation Conference (SCSC '03)*, 2003.
- [3] K. M. Chandy and J. Misra. Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM*, 24(4):198–206, 1981.
- [4] D. Chen, S. J. Turner, W. Cai, B. P. Gan, and M. Y. H. Low. Algorithms for HLA-based distributed simulation cloning. *ACM Transactions on Modeling and Computer Simulation*, 15(4):316–345, 2005.
- [5] R. Curry, C. Kiddle, R. Simmonds, and B. Unger. Sequential performance of asynchronous conservative PDES algorithms. In *Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation (PADS '05)*, 2005.
- [6] R. M. Fujimoto. *Parallel and Distributed Simulation Systems*. Wiley Interscience, 2000.
- [7] M. Hybinette. Just-in-time cloning. In *Proceedings of the 18th Workshop on Parallel and Distributed Simulation (PADS '04)*, 2004.
- [8] M. Hybinette and R. Fujimoto. Cloning: A novel method for interactive parallel simulation. In *Proceedings of the 29th Conference on Winter Simulation (WSC '97)*, 1997.
- [9] M. Hybinette and R. M. Fujimoto. Cloning parallel simulations. *ACM Transactions on Modeling and Computer Simulation*, 11(4):378–407, 2001.
- [10] V. Jha and R. Bagrodia. Simultaneous events and lookahead in simulation protocols. *ACM Transactions on Modeling and Computer Simulation*, 10(3):241–267, 2000.
- [11] C. Kiddle, R. Simmonds, and B. Unger. Channel based sequential simulation. In *Proceedings of the 37th Conference on Winter Simulation (WSC '05)*, 2005.
- [12] MOOSE - Module-based Object-Oriented Simulation Environment. <http://web.cs.uni-bonn.de/IV/MOOSE/>.
- [13] K. S. Perumalla. Parallel and distributed simulation: Traditional techniques and recent advances. In *Proceedings of the 37th Conference on Winter Simulation (WSC '06)*, 2006.
- [14] P. Peschlow, T. Honecker, and P. Martini. A flexible dynamic partitioning algorithm for optimistic distributed simulation. In *Proceedings of the 21st Workshop on Principles of Advanced and Distributed Simulation (PADS '07)*, 2007.
- [15] P. Peschlow and P. Martini. Efficient analysis of simultaneous events in distributed simulation. In *Proceedings of the 11th IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT '07)*, 2007.
- [16] R. Rönngrén and M. Liljenstam. On event ordering in parallel discrete event simulation. In *Proceedings of the 13th Workshop on Parallel and Distributed Simulation (PADS '99)*, 1999.