Behavior-aware Probabilistic Synchronization in Parallel Simulations and the Influence of the Simulation Model

Mirko Stoffers
Mobile Network Performance Group
RWTH Aachen University
stoffers@umic.rwth-aachen.de

ABSTRACT

Efficient event scheduling and synchronization constitutes an essential part of high-performance parallel discrete event simulation. Traditional synchronization approaches like conservative and optimistic synchronization focus on a simple scheduling paradigm based on a primitive set of rules. However, we argue that a sophisticated synchronization algorithm considering *event interactions* can remarkably improve the performance of the simulation. In this article, we discuss three different *heuristics* which analyze those dependencies online and speed up the simulation by a factor of up to 6.5. Further, we analyze the ability of this paradigm to automatically detect available parallelism by applying it to scenarios featuring different degrees of available parallelism.

1. INTRODUCTION

The goal of parallel discrete event simulation is to obtain the same results as the sequential equivalent in shorter time. This is achieved by executing *independent* events in parallel. However, if one event depends on the results computed by another event, execution order matters. In order to avoid parallel or out-of-order execution of those *dependent* events, synchronization is necessary.

Traditionally, one out of two synchronization paradigms is implemented: On the one hand, *conservative synchronization* applies strict rules to permanently ensure correct ordering of dependent events. On the other hand, overly *optimistic synchronization* allows any pair of events to be executed in parallel and detects and corrects causal violations a posteriori; this is usually done by creating checkpoints and re-running the simulation from a previously stored state. While both approaches are easy to understand and implement, and introduce little synchronization overhead, they might severely waste performance due to the following disadvantages: The strict rules in conservative synchronization might hamper fast progress due to the so called *blocked waiting problem* [16]. This occurs when a CPU waits for an event e_1 to finish, because it cannot determine whether the next event e_2 depends on e_1 or not. Optimistic synchronization suffers from high rollback costs after erroneously executing two interdependent events in parallel.

In the last decades, considerable efforts have been invested to mitigate those problems. In order to speed up conservatively synchronized simulations, lookahead maximization techniques have been applied [3, 4, 12, 13, 14]. To decrease the number of rollbacks in optimistic event execution, Turner and Xu introduced the time window approach [20]. However, those optimizations artificially limit parallel execution without considering the behavior of the simulation itself.

Pioneering efforts towards probabilistic synchronization [5, 6, 18] start analyzing the timing between events, but do not consider event dependencies. In [11], we introduce our approach to analyze both timing and scheduling dependencies. We show that this approach achieves multiple speedup over the traditional synchronization efforts.

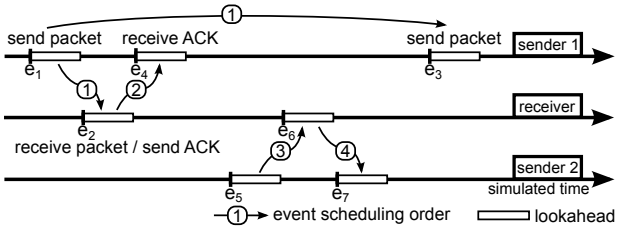
In this article, we briefly describe our approach to probabilistic synchronization, and re-introduce our three different heuristics to analyze event interactions. In a case study we show that this approach outperforms traditional synchronization, and we analyze the quality of automatic parallelism detection by varying the amount of parallelism included in the simulation model.

The remainder of this article is structured as follows: In Section 2 we describe our synchronization approach and the three heuristics. In Section 3 we introduce our evaluation methodology and discuss the results. We discuss related efforts in this research field in Section 4 before we conclude the article in Section 5.

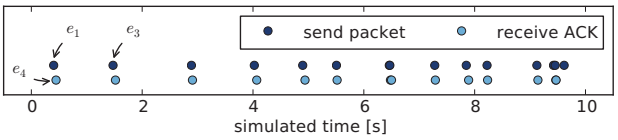
2. PROBABILISTIC SYNCHRONIZATION

We created the proof-of-concept implementation of our probabilistic synchronization scheme on top of our parallel simulation framework HORIZON [10]. HORIZON is an extension for the discrete event simulator OMNeT++ [21] to enable efficient parallel simulation on small to medium scale shared memory systems. It employs a centralized event scheduling architecture, which eases the implementation of sophisticated heuristics since all necessary information is located in the shared memory. The central scheduler continuously picks the first event from the Future Event Set (FES). By means of a simple barrier approach, it conservatively determines whether this event depends on an earlier, unfinished event. During this determination process we call the first event in the FES the *pending event* e_p since it is pending between the waiting state and the execution state. If the event is independent, the scheduler *offloads* it for parallel execution, and one of the *worker threads* handles the event.

To enable probabilistic synchronization, we modify this dependency determination process: The synchronization component first applies the strict rules of conservative synchronization. If those rules do *not* allow parallel execution, a heuristic is consulted to determine the probability of a causal violation to occur on optimistic execution. If this probability is below a predefined threshold, the probabilistic scheduler



(a) Global sequence: The senders transmit packets and the receiver acknowledges the reception.



(b) Local sequence at sender 1: After transmitting a packet the sender receives the corresponding acknowledgment.

Figure 1: Event sequences in a model consisting of two senders and a receiver.

offloads the event for parallel execution immediately.

In the following, we describe the design of each of three heuristics after discussing the general design goals and deriving the need for three different heuristics.

2.1 Design Goals

The primary goal of probabilistic synchronization is speeding up parallel discrete event simulation. Therefore, the simulation framework continuously collects data to guide a heuristic in computing probabilities to predict the computationally cheaper option: either conservative synchronization or speculative execution. In order to achieve the overall goal of determining the correct simulation results as soon as possible, we define three distinct design goals in [11]:

i) *guarantee causal correctness* of the simulation, i. e., emend inconsistencies induced by speculative event execution.
ii) *maximize the number of correct predictions* since incorrect predictions result either in blocked waiting or in a rollback.
iii) *minimize the prediction complexity* since too much heuristic overhead slows down the simulation.

Obviously, the two latter design goals conflict with each other: A sophisticated heuristic can achieve accurate results, but requires a lot of resources to come up with a decision. A simpler heuristic computes the decision faster, but will err more often.

We argue that a simulation with highly complex events requires thoughtful heuristic decisions. Even the rollback of a single event is an expensive task after a false heuristic prediction. On the other hand, a simulation with simple events cannot tolerate the overhead of a sophisticated heuristic. We therefore designed three different heuristics of different degrees of complexity and accuracy. The synthetic benchmark in [11] compares the decision quality and execution time of those heuristics.

2.2 Arrival Pattern Heuristic

The Arrival Pattern Heuristic bases on the observation that sequences of event *types* at a particular module repeat again and again. Figure 1 illustrates this for a network node that periodically transmits packets and awaits the corresponding acknowledgment. Figure 1(a) shows all events in the simulation while Figure 1(b) focuses on the event arrival pattern at "sender 1". The idea of the Arrival Pattern Heuristic is to observe this pattern (i. e., "send packet"-events are followed by "receive ACK"-events and vice versa) in order to be able to predict the type of the next event to occur at this module. This heuristic is similar to related work in probabilistic synchronization [5, 6, 18].

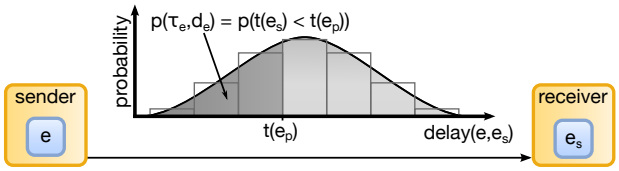


Figure 2: The PDF is sampled from the delays between an event e and its earliest successor event e_s. A histogram represents this distribution. The highlighted area denotes the probability that another event e'_s precedes the pending event e_p.

In order to store the event pattern, the learning component of the heuristic tracks at each module

i) the type τ of the event that arrived last,
ii) for every event type υ, its number of occurrences n_υ,
iii) for every pair (τ, υ), the number $n_{\tau\upsilon}$, indicating how often an event of type υ followed an event of type τ.

The predictor utilizes this knowledge to determine the probability $p_{\tau\upsilon}$ that the type $\upsilon := \upsilon_{e_p}$ of the pending event e_p occurs next after a preceding event of type τ as

$$p_{\tau\upsilon} := \begin{cases} \frac{n_\tau}{n_{\tau\upsilon}} & \text{, if } n_{\tau\upsilon} \neq 0 \\ 0 & \text{, else.} \end{cases}$$

We allow offloading an event of type υ if and only if the complementary probability $1 - p_{\tau\upsilon}$ (i. e., υ does not follow τ) is below a given threshold.

Applied to the discussed example, this yields a high probability for the next event to be a "receive ACK"-event (after the last one being a "send packet"-event), i. e., the next "send packet"-event is hold back until another "receive ACK"-event occurs. Nevertheless, this heuristic reaches its limitations as soon as situations get more complicated and arrival patterns cannot be observed anymore. For a more detailed discussion on the limitations and complexity, see [11].

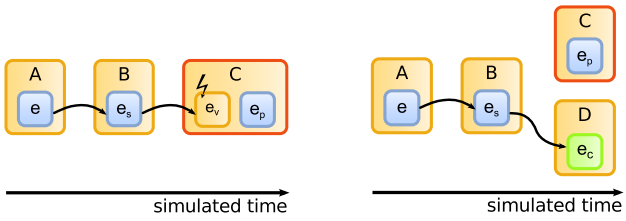
2.3 Global Order Heuristic

The Global Order Heuristic analyzes the event interactions from a global perspective, and determines the "scheduled-by" relationship among events. We observe that a causal violation occurs at module m only if:

i) two events e_1, e_2 execute speculatively in parallel, and
ii) e_1 with $t(e_1) < t(e_2)$ creates an event e_3 with $t(e_3) < t(e_2)$,

where $t(e)$ denotes the timestamp of event e.

This means, on deciding whether or not to offload a pending event e_p, the heuristic needs to determine the probability that any currently offloaded event creates a successor event e_s with $t(e_s) < t(e_p)$. Therefore the learning component tracks the minimum time difference (i. e., delay) between an event e and all events scheduled by e. Aggregating this information over all events of the same type yields the (empirical) Probability Density Function (PDF), which we store



(a) Causal violation. (b) No causal violation.

Figure 3: A causal violation can only occur if an event is scheduled to take place at the same module as the pending event e_p, i. e., causal correctness is a local property of each module.

in a histogram.

The predictor can easily derive the probability $p(\tau_e, d_e)$ that an event e of type τ_e schedules another event within a delay of $d_e := t(e_p) - t(e)$ from the histogram (see Figure 2). Aggregating this probability over the set of all offloaded events $O \subseteq E$ yields the probability for a causal violation as:

$$p_v := 1 - \prod_{e \in O} \Big(1 - p(\tau_e, d_e)\Big).$$

Again, the heuristic compares this result to a threshold to determine the offloading decision.

This heuristic is more complex than the Arrival Pattern Heuristic, but is also able to handle more complicated situations. Nevertheless, the main limitation of this heuristic is the fact that it does not consider the locality of events. Figure 3 shows two situations that are handled indifferently by the Global Order Heuristic although the situation in Figure 3(a) exhibits a causal violation while the situation in Figure 3(b) does not. To reduce complexity the Global Order Heuristic only considers the time of the next event, but not the module it occurs on. A more detailed discussion on the limitations and a formal analysis of the complexity can be found in [11].

2.4 Local Order Heuristic

Motivated by the central limitation of the Global Order Heuristic we developed a Local Order Heuristic that considers both timing and locality of events as well as the transitive scheduling relationship. Therefore, the learning component tracks not only the delays between events, but also the modules they take place on. That means, at a given module m we track for each event e and its successor events e_s

i) the target module m_s on which e_s takes place,
ii) the difference in simulated time between e and e_s.

Upon request the predictor spans the *dependency tree* (see Figure 4), i. e., for every event in the set O of offloaded events it determines the successor events and their corresponding PDFs. In order to calculate the resulting PDF over multiple hops on a path through the tree, the heuristic needs to convolve the underlying PDFs. We need to determine the probability that an event occurs on the same module as the pending event e_p occurs on. Therefore, we denote the nodes that point to the same module as e_p as *conflicting nodes*. Children of conflicting nodes are not included in the tree since a causal violation would already occur at the conflicting node.

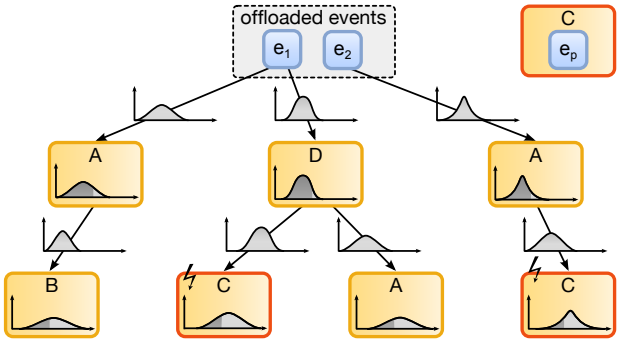


Figure 4: The heuristic constructs the successor tree to generate the prediction. A delay distribution on an edge in the tree denotes the PDF between consecutive event types. The delay distribution in a node N is constructed by adding (convoluting) the PDFs on the edges from the root to N. This reflects the cumulative delay distribution from the offloaded event to an event represented by that node. The final conflict probability is computed by aggregating the probabilities for the given delay in all conflicting nodes.

Since the tree might get arbitrarily large, we avoid spanning the whole tree, but instead build the tree while traversing it in a breadth-first manner. During that process, we calculate a lower bound and an upper bound for the probability for a causal violation to occur if e_p is offloaded. Since the PDF always shifts to the right by performing the convolution operation (no delays smaller than 0 exist), the probability always decreases. This means, we can calculate an upper bound p_u by aggregating the probabilities over all current leafs $L \subseteq V$ in the tree $T = (V, E)$:

$$p_u = 1 - \prod_{\substack{\tau \in L, e \in O, \\ \text{path}(e,\tau)}} \Big(1 - p(\tau, d_e)\Big),$$

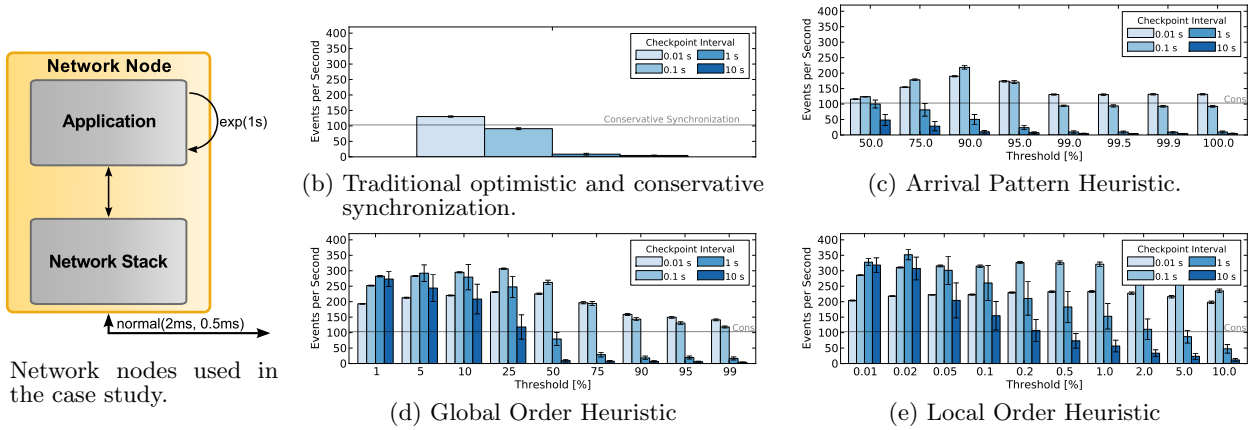
where $\text{path}(e, \tau)$ denotes the existence of a unique path in the tree from an event e in the root to the leaf node τ, and $p(\tau, d_e)$ is the convoluted probability over the path from e to τ. A lower bound can be determined analogously by considering only conflicting nodes in the leaf set. Note that a conflicting node is always a leaf since the tree traversal is stopped at a conflicting node.

As soon as both boundaries are smaller (larger) than the threshold, we conclude that the sought probability is smaller (larger) than the threshold, such that we can derive the offloading decision.

This approach determines highly accurate predictions, but building the tree and convoluting the histograms comes with high computational complexity. For a detailed complexity analysis and a discussion on implemented optimizations we refer to [11].

3. EVALUATION

In [11] we describe our synthetic benchmark analyzing the quality and efficiency of the heuristics. Further, a case study shows the overall runtime performance achieved by applying the heuristics. In this article, to ease the application of theoretical analyses, we simplify the model underlying the



(a) Network nodes used in the case study.

(b) Traditional optimistic and conservative synchronization.

(c) Arrival Pattern Heuristic.

(d) Global Order Heuristic

(e) Local Order Heuristic

Figure 5: Performance of the traditional synchronization techniques and our heuristics in the simplified case study.

case study and show that the results are similar. We then analyze the degree of parallelism included in the model and compare this to the degree of parallelism detected by the Local Order Heuristic. We show that the heuristic is able to detect an increasing degree of parallelism and speeds up the simulation.

We performed all measurements on a dedicated simulation server, equipped with two six-core AMD Opteron 2431 CPUs and 32 GB of RAM. The server runs our Horizon-based proof-of-concept implementation of the heuristics on OMNeT++ [21] on a 64-bit Ubuntu 10.04 LTS server OS. For every data point we ran 30 independent simulations and computed the 99 % confidence intervals.

3.1 Simplified Case Study

We base our case study on the model we used in the case study in [11]. This is a wireless mesh network consisting of 57 nodes focusing on the detailed simulation of a wireless OFDM channel. In this scenario, the channel performs the complex OFDM fading calculations. Since this model abstracts from many network stack details, the main task of the other network layers is to forward the packets from layer to layer. To aid theoretical analyses, we merge the three lower layer modules into one network stack module performing all network based operations. This results in one application module and one network module per node as depicted in Figure 5(a).

We train the heuristics once for every configuration of threshold and checkpoint interval until 40,000 samples of each distribution have been collected. We then run the steady state 30 times in order to compute reliable values for the simulation performance. The results are qualitatively similar to the results presented in [11]. However, we observe that the simplified model is better suited for optimistic synchronization than the model consisting of more layers. We attribute this to the fact that the high amount of fast processed events increases the risk of rollbacks.

We do not discuss the results in close detail, but refer to the detailed analysis of the similar case study in [11]. All in all, from Figure 5(b) to Figure 5(e) we observe that all heuristics outperform the conservative synchronization, and that the Local Order Heuristic reaches the highest performance of all described heuristics if configured appropriately. This is the expected result since the simulation contains highly complex fading calculations such that a sophisticated heuristic has a high potential for speeding up the simulation.

However, we see that the Local Order Heuristic achieves a speedup of only 3.5 although we have 12 cores available. In the following, we analyze the underlying simulation model in closer detail in order to investigate if

a) a more sophisticated heuristic could perform better, or
b) the model does not allow the execution of more events in parallel.

3.2 Parallelism Analysis

We first take an analytical approach to investigating the parallelism available in the model. The network consists of 57 nodes, and every packet is broadcast to the set of neighboring nodes. As a consequence of the broadcast, every receiver needs to calculate the fading coefficients. This results in a series of events that can be processed in parallel. Since the modules of the concurrent events are physically close together, we denote this as *local parallelism*. Further, we observe that two events can be processed in parallel if they belong to independent data streams, e. g., one communication at the "west" end, and one at the "east" end of the network. As opposed to the local parallelism, we call this *global parallelism*. Note that two independent data streams that can be executed in parallel are still referred to as global parallelism if they are close together in the network.

Calculating the available local parallelism can be easily performed by analyzing the node degree of the nodes in the network graph. We compute the theoretically achievable speedup by exploiting all available local parallelism (and no global parallelism) on a varying number of worker threads, and compare this to the speedup achieved by the Local Order Heuristic with a threshold of 0.01 % and a checkpoint interval of 1 s (see Figure 6).

We conclude from the plot that the heuristic is able to (almost) completely exploit local parallelism, but beyond that, it only exploits a small amount of global parallelism. This might have two causes: Either the heuristic is not smart enough to detect global parallelism, or the model simply does not provide enough global parallelism to allow significant speedup. We investigate this in the following.

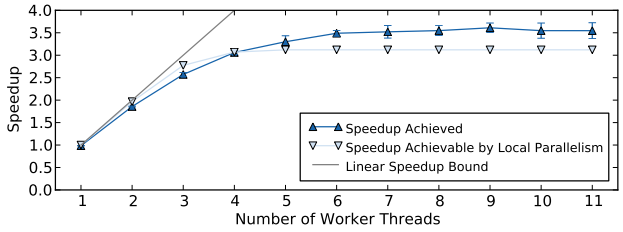


Figure 6: Comparison of the measured speedup achieved by the Local Order Heuristic and the theoretically possible speedup by (only) exploiting local parallelism.

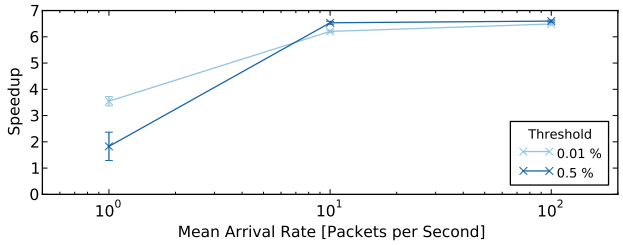


Figure 7: Speedup achieved by the Local Order Heuristic on varying the degree of global parallelism with two different threshold configurations.

3.3 Increasing Available Parallelism

A theoretic approach to analyzing the amount of global parallelism available in the simulation model is more complicated than for the local parallelism since, in this case, receivers, transmitters, and routes need to be considered. Instead, we take a practical approach to investigating whether the heuristic is able to detect global parallelism or not. While the amount of local parallelism remains constant (it depends only on the node degree), the amount of global parallelism grows when increasing the number of concurrent data streams. In the case of an average delay of 1 s between two packet generation events (as in the case study so far), the first multi-hop transmission from source to destination and back finishes with high probability before the next one starts. However, if we decrease the inter-arrival time of the packet-generation-events, we increase the probability of two data streams running in parallel in the physical network.

We claim that a smart heuristic must be able to detect this increase of global parallelism. Figure 7 depicts the speedup achieved by the Local Order Heuristic for arrival rates of 1 packet/s to 100 packets/s. We observe an increase of the speedup from about 3 up to 6.5. Since it is not possible to achieve a speedup of 6.5 only by exploiting local parallelism, we conclude that our heuristic is indeed able to detect and exploit both local and global parallelism if both are present in the model.

We further observe that the optimal threshold for the heuristic shifts to a higher value. We attribute this to the fact that more available parallelism allows for more optimistic heuristic setups.

4. RELATED WORK

Although there were extensive research efforts in the past to optimize parallel simulations, it is still an ongoing research topic. Perumalla created an elaborate survey on the traditional techniques of conservative and optimistic synchronization as well as recent advances [16]. Due to space constraints we do not include such a detailed analysis, but shortly describe the most relevant advances.

Probabilistic Synchronization.

Probabilistic synchronization has been considered in prior approaches. The pioneering effort originates from Ferscha et al. [5, 6]. Like our Arrival Pattern Heuristic, this approach analyzes the arrival pattern of events. However, as opposed to our heuristic, they apply statistical methods on the arrival times, neglecting the different types of events. Based on these observations the heuristic predicts the time stamp of the next incoming event. Since this heuristic is also limited to local observations, it cannot accurately predict the next event in more complicated situations, like the Arrival Pattern Heuristic.

A similar approach was proposed by Som et al. [18]. This approach utilizes PDFs to sample the time differences between events. Nevertheless, they still ignore event types, such that they inherit the drawbacks from the approach by Ferscha et al.

Optimizing Traditional Approaches.

Orthogonal approaches focus on optimizing either conservative or optimistic synchronization. Since the main challenge of conservative synchronization is the blocked waiting problem, dynamic lookahead extraction was proposed to reduce the effects of this drawback. Several research papers discuss different techniques in this area. For instance, Cota et al. [4] analyze the internal behavior of a simulation model component to compute a lower bound on the time of the next event being generated. The advances of Meyer et al. [13, 14] analyze the data flow between components from a manually provided graph to compute longer, but still conservative lookaheads. Other approaches deal with domain specific lookahead extraction, for example in the wireless domain (Liu et al. [12]), or the domain of multi-processor systems (Chung et al. [3]).

In the area of optimistic synchronization, approaches to reduce the rollback costs were proposed by the research community. For example, Carothers et al. [2] introduced reverse computation as a scheme to avoid state saving, but instead undo events by additional routines to roll back to a previous state. An efficient checkpoint-and-rollback scheme was introduced by Toccaceli et al. [19].

Combined Synchronization.

While probabilistic synchronization dynamically combines conservative and optimistic synchronization based on a heuristic result, previous approaches had already combined those techniques statically. The pioneering efforts were performed in the early 90s [1, 9, 17]. More recent frameworks allowing the use of combined synchronization are µsik [15] and the High Level Architecture (HLA) [7]. An orthogonal approach to combining optimistic and conservative synchronization is relaxed synchronization [8] relaxing the constraint of causal correctness and allowing causal inconsistencies as long as two events being executed out-of-order are not too far apart in simulated time. This approach bases on the idea that also in reality it can often not be guaranteed that two events close together in time happen in the specified order. How-

ever, while this approach might allow fast execution, it does not guarantee equality to the results computed by sequential execution and defeats reproducibility of simulation results.

5. CONCLUSION

In this article, we described our probabilistic synchronization scheme for parallel discrete event simulations. In [11] we already showed that a sophisticated heuristic can speed up a parallel simulation by a factor of 3.2. This article further analyzes the underlying model and the degree of parallelism available in the model. We showed that our heuristic automatically exploits more parallelism if the model supports this. In a model with increased degree of inherent parallelism we can achieve a speedup factor of 6.5. However, a sophisticated heuristic always introduces a lot of overhead, such that not every model will perform fast with every heuristic.

All in all, we conclude that automatic analysis of event interactions and interdependencies can remarkably speed up parallel discrete event simulations.

Acknowledgments

This article bases on my Master Thesis at RWTH Aachen University, Mobile Network Performance Group. I acknowledge my supervisors Georg Kunz and Prof. James Gross for valuable contributions and support. This research was also supported by the DFG Cluster of Excellence on Ultra High-Speed Mobile Information and Communication (UMIC), German Research Foundation grant DFG EXC 89.

6. REFERENCES

[1] R. Bagrodia and W.-T. Liao. Maisie: A Language for the Design of Efficient Discrete-Event Simulations. *IEEE Trans. on Software Engineering*, 20(4):225–238, 1994.

[2] C. D. Carothers, K. S. Perumalla, and R. M. Fujimoto. Efficient Optimistic Parallel Simulations using Reverse Computation. *ACM Trans. Model. Comput. Simul.*, 9(3):224–253, July 1999.

[3] M.-K. Chung and C.-M. Kyung. Improving Lookahead in Parallel Multiprocessor Simulation Using Dynamic Execution Path Prediction. In *Proc. of the 20th Workshop on Principles of Advanced and Distributed Sim.*, 2006.

[4] B. Cota and R. Sargent. A Framework for Automatic Lookahead Computation in Conservative Distributed Simulations. In *Proc. of the SCS Multiconference on Distributed Simulation*, 1990.

[5] A. Ferscha. Probabilistic Adaptive Direct Optimism Control in Time Warp. In *Proc. of the 9th Workshop on Parallel and Distributed Simulation*, 1995.

[6] A. Ferscha and G. Chiola. Self-adaptive Logical Processes: The Probabilistic Distributed Simulation Protocol. In *Proc. of the 27th Annual Simulation Symposium*, 1994.

[7] R. M. Fujimoto. Time Management in the High Level Architecture. *Trans of The Society for Modeling and Simulation International*, 71(6):388–400, 1998.

[8] R. M. Fujimoto. Exploiting Temporal Uncertainty in Parallel and Distributed Simulations. In *Proc. of the 13th Workshop on Parallel and Distributed Simulation*, 1999.

[9] V. Jha and R. L. Bagrodia. A Unified Framework for Conservative and Optimistic Distributed Simulation. In *Proc. of the 8th Workshop on Parallel and Distributed Simulation*, 1994.

[10] G. Kunz, O. Landsiedel, J. Gross, S. Götz, F. Naghibi, and K. Wehrle. Expanding the Event Horizon in Parallelized Network Simulations. In *Proc. of the 18th Annual Meeting of the IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2010.

[11] G. Kunz, M. Stoffers, J. Gross, and K. Wehrle. Know Thy Simulation Model: Analyzing Event Interactions for Probabilistic Synchronization in Parallel Simulations. In *Proc. of the 5th International Conference on Simulation Tools and Techniques*, pages 119–128. ICST, 2012.

[12] J. Liu and D. M. Nicol. Lookahead Revisited in Wireless Network Simulations. In *Proc. of the 16th Workshop on Parallel and Distributed Simulation*, 2002.

[13] R. A. Meyer and R. L. Bagrodia. Improving Lookahead in Parallel Wireless Network Simulation. In *Proc. of the 6th Intern. Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 1998.

[14] R. A. Meyer and R. L. Bagrodia. Path Lookahead: A Data Flow View of PDES Models. In *Proc. of the 13th Workshop on Parallel and Distributed Simulation*, 1999.

[15] K. S. Perumalla. μsik – A Micro-Kernel for Parallel/Distributed Simulation Systems. In *Proc. of the 19th Workshop on Principles of Advanced and Distributed Simulation*, 2005.

[16] K. S. Perumalla. Parallel and Distributed Simulation: Traditional Techniques and Recent Advances. In *Proc. of the 38th Winter Simulation Conference*, 2006.

[17] H. Rajaei, R. Ayani, and L.-E. Thorelli. The Local Time Warp Approach to Parallel Simulation. In *Proc. of the 7th Workshop on Parallel and Distributed Simulation*, 1993.

[18] T. K. Som and R. G. Sargent. A Probabilistic Event Scheduling Policy for Optimistic Parallel Discrete Event Simulation. In *Proc. of the 12th Workshop on Parallel and Distributed Simulation*, 1998.

[19] R. Toccaceli and F. Quaglia. DyMeLoR: Dynamic Memory Logger and Restorer Library for Optimistic Simulation Objects with Generic Memory Layout. In *Proc. of the 22nd Workshop on Principles of Advanced and Distributed Simulation*, 2008.

[20] S. Turner and M. Xu. Performance Evaluation of the Bounded Time Warp Algorithm. In *Proc. of the 6th Workshop on Parallel and Distributed Simulation*, 1992.

[21] A. Varga. The OMNeT++ Discrete Event Simulation System. In *Proc. of the European Simulation Multiconference*, 2001.