Large-Scale Network Simulation: Leveraging the Strengths of Modern SMP-based Compute Clusters

Mirko Stoffers*, Sascha Schmerling*, Georg Kunz*, James Gross‡, Klaus Wehrle*
*Communication and Distributed Systems, RWTH Aachen University
‡School of Electrical Engineering, KTH Royal Institute of Technology
stoffers@comsys.rwth-aachen.de

ABSTRACT

Parallelization is crucial for efficient execution of large-scale network simulation. Today's computing clusters commonly used for that purpose are built from a large amount of multi-processor machines. The traditional approach to utilize all CPU cores in such a system is to partition the network and distribute the partitions to the cores. This, however, does not incorporate the presence of shared memory into the design, such that messages between partitions on the same computing node have to be serialized and synchronization becomes more complex. In this paper, we present an approach that combines the shared-memory parallelization scheme Horizon [9] with the standard approach to distributed simulation to leverage the strengths of today's computing clusters. To further reduce the synchronization overhead, we introduce a novel synchronization algorithm that takes domain knowledge into account to reduce the number of synchronization points. In a case study with a UMTS LTE model, we show that both contributions combined enable much higher scalability achieving almost linear speedup when simulating 1,536 LTE cells on 1,536 CPU cores.

Categories and Subject Descriptors

I.6 [**Simulation and Modeling**]: Miscellaneous

General Terms

Design, Performance, Algorithms

Keywords

Parallel simulation, Shared-memory, Distributed simulation

1. INTRODUCTION

Network simulation models have grown continuously in size. Higher degrees of detail and the desire for simulation of larger networks have driven the requirement for high-performance execution of such simulation models. Unfortunately, CPU power dissipation prevents hardware manufacturers from increasing the clock frequency of single cores. Instead, multiple cores are included in a single machine, such that software developers have to adapt their software to utilize the full power available.

Simutools 2014, March 17-19, Lisbon, Portugal

DOI 10.4108/icst.simutools.2014.254622

Further, computing clusters and supercomputers have been available for decades. This hardware has driven the development of *Parallel Discrete Event Simulation (PDES)* [5] since the 1970s. The traditional approach to PDES is the (static) decomposition of a simulation model into multiple Logical Processes (LPs) in order to map one or more LP(s) to each processor. LPs then communicate by exchanging messages via the interconnect of the computing cluster. We refer to this approach as *distributed simulation* in the following.

On modern multi-core machines this approach can be used as well: Instead of assigning LPs to processors, LPs are assigned to processor cores. The cores then communicate via exchanging messages through the shared memory. However, approaches to *shared-memory PDES* [8, 9, 24] have proved that it can be more efficient to conceptually design parallel simulation explicitly for multi-core machines rather than adopting distributed simulation. Obviously, these approaches rely on the availability of shared memory, and therefore do not scale beyond the size of a single computer.

Modern computing clusters and supercomputers are typically built by interconnecting a set of shared-memory systems [15]. Traditionally, researchers decompose the simulation model into sufficiently many LPs to run an LP on each CPU core. This, however, means that the cores inside a computing node treat the shared memory as a buffer for serialized synchronization messages rather than accessing other LP's memory directly. We argue that this does not utilize the full power available. Instead, we propose an approach which leverages the strengths of shared-memory PDES while allowing the utilization of multiple cluster nodes through distributed simulation. By significantly reducing the synchronization and communication overhead inside the simulation, we increase the scalability dramatically.

In particular, we make the following contributions:

1. We propose a *multi-level parallelization approach*, **Distributed Horizon** that combines the shared-memory PDES concept *Horizon* [9] with traditional *distributed simulation* to leverage the strengths of both.
2. We propose a novel synchronization algorithm, **Dynamic Barrier Synchronization (DBS)** that reduces the number of synchronization points by applying domain knowledge.

We investigate the influence of certain model properties and show the benefit of our contributions by means of a case study (large-scale UMTS LTE network). Our measurements show promising results for a wide variety of model properties. DISTRIBUTED HORIZON outperforms distributed simulation for many parameters in a Closed Queuing Net-

work benchmark. With domain knowledge applied, in the LTE case study both contributions combined achieve almost linear speedup on more than 1,500 cores.

The remainder of this paper is structured as follows: We first discuss related efforts targeting similar goals as well as the approaches underlying this work. After describing the challenges in Section 3, we present and discuss the contributions in close detail in Section 4 and 5. After that, we explain the evaluation setup and the results in Section 6. Finally, we outline future work and conclude the paper.

2. RELATED WORK

In this section we discuss traditional techniques to PDES as well as newer approaches targeting similar goals to our work. In *distributed simulation* the simulation model is decomposed into LPs and the LPs are assigned to processing entities. A synchronization algorithm is necessary to avoid out-of-order execution of events at each LP.

The choice of the best fitting synchronization algorithm is crucial for simulation performance. In the following, we discuss traditional synchronization algorithms before we focus on more advanced approaches in conservative synchronization. We furthermore introduce approaches to *shared-memory PDES* as well as *multi-layer parallelization*.

2.1 Basic Synchronization Algorithms

There are two classes of conservative synchronization algorithms: A synchronous algorithm synchronizes all LPs at the same point in time while in an asynchronous algorithm LPs can synchronize locally. The most common synchronization algorithms of these classes are *CMB* [3] (asynchronous) and *LBTS* [4] (synchronous). CMB, where LPs send null-messages to their neighbors based on the local simulation state, heavily suffers from the *time creeping problem* [6], especially in wireless network simulation. Therefore, LBTS is better suited for these situations. Here, each LP adds the minimum link delay to the next local event and the next global barrier is defined as the minimum of all these timestamps. After that, each LP can execute all its local events until that timestamp. This guarantees progress since each synchronization allows at least the execution of one event. Nevertheless, small link delays result in small parallelization windows and therefore poor parallelization speedup.

2.2 Advanced Synchronization Algorithms

To mitigate the drawbacks of CMB and LBTS, approaches have been proposed to increase the parallelization window.

In this area, the Bounded Lag algorithm by Lubachevsky [14] introduces a *bounded lag* to provide lookahead, but increases the parallelization window by the concept of so called *opaque periods*. Hence, each node in the simulation has a means to provide a guarantee that it will not generate any outgoing message during this opaque period. The author demonstrates the usefulness of this approach for an *Ising Model* where a node will never generate an outgoing message between two events of a certain type, and the time between the two events is determined by drawing a random number. However, this approach does not allow to discard the guarantee upon arrival of an incoming message. This works for the Ising Model since incoming messages do not generate new events during the opaque period. Nevertheless, it does, for example, not work for a network node that processes incoming packets independently such that the opaque period generated by one packet can not provide a guarantee on the opaque period of another packet arriving at a later point in time. Hence, we focus on the design of a more generic approach working in more diverse situations.

Path Lookahead by Meyer and Bagrodia [16, 17] allows to determine larger lookahead values. It computes the synchronization points by adding the delays along a potential path through the simulation entities and ignoring paths that could never be taken by the messages. Hence, it requires model annotations to distinguish possible from impossible paths. However, in this work we target a more generic approach where the next synchronization point can be determined not only by an additive function, but also, for example, by a ceiling function mapping the current simulation time to the next time slot where a message might be sent.

Furthermore, domain-specific approaches to more efficient synchronization have been taken. For example, Nicol [18] proposed a lookahead extraction scheme for FCFS queuing networks. In [7], Heidelberger and Nicol introduce a scheme specifically tailored to efficiently parallelize Continuous Time Markov Chains. However, we argue that it is necessary to design a synchronization algorithm feasible in a broader field of domains to accommodate more users.

2.3 Shared-Memory and Multi-Layer PDES

To utilize the full potential of today's compute clusters, approaches incorporating the presence of shared memory into the design have to be taken. Such an approach is *Horizon* [9], based on the concept of *expanded event simulation*. The idea of expanded event simulation is to provide a means for simulation modelers to naturally include processing durations into the simulation: Instead of simulating a time-consuming process by a start- and an end-event separately, Horizon offers expanded events spanning a duration of simulated time. Since the output of such an event can not be visible to the system before the calculation is completed, an expanded event naturally guarantees that no events can occur before the end time of the event. This way, the processing duration opens a parallelization window and all events that start during this window can be executed in parallel.

Horizon is based on a master/worker paradigm [10] where the master assigns parallelizable events directly to the worker threads. Therefore, the scheduler thread continuously picks the next event from the Future Event Set (FES) and determines the parallelizability. As soon as the event can be executed, the scheduler selects an idle worker thread and places a pointer to the event at a specific point in the shared memory. The spinning worker immediately starts processing the event. To use every core available in the system, we recently enabled the scheduler to change its role to a worker thread (if no worker is available for event execution) and the next finished worker takes over the scheduler role.

We have shown that Horizon is both correct and efficient on shared-memory systems [10]. However, at the moment, Horizon does not exploit cache locality such that superlinear speedup is unlikely.

Additionally, a multitude of hierarchical approaches has been proposed by the research community for efficient PDES and certain approaches also focus on shared-memory systems. One of the first approaches in this area was Local Time Warp by Rajaei et al. [20]. Local Time Warp joins multiple LPs into a *cluster* and performs optimistic synchronization inside each cluster, limited by a time window conservatively negotiated among the clusters. While this

approach was not specifically tailored to today's multi-core based compute clusters, it could be easily applied on this hardware by assigning each LP cluster to a computing node. However, the benefit would be limited since this approach would not use shared memory for inter-LP communication.

A different approach to multi-level PDES was performed by Xiao et al. in 1999 [24]. This approach (CCT) is an extension of CMB targeting single shared-memory machines to increase load balancing through decreasing LP sizes. Therefore, the authors introduce a hierarchical scheduling algorithm that schedules groups of LPs while each group schedules the LPs inside this group. However, only a single synchronization algorithm is applied here, such that this approach is a hierarchical scheduling algorithm, rather than a hierarchical synchronization algorithm.

Based on CCT, Nicol and Liu introduce *Composite Synchronization* [19] combining the asynchronous CCT with a synchronous approach to mitigate the disadvantages of both. Recently, this approach was extended to allow the execution of multiple Symmetric Multiprocessor (SMP) machines [13]. The approach requires thresholds to decide which synchronization algorithm to use for which channel, such that it is necessary to solve an optimization problem to determine these thresholds. A similar approach by Liu and Nicol, DaSSF [12] enables the distribution of SSF to run on shared-memory based computing clusters. SSF was initially based on CCT such that it only ran on a single SMP machine.

While these approaches target the same computing platform as we do, all of them rely on the partitioning of the model and maintain a separate FES for each LP. We argue that a more fine-grained parallelization on a per-event basis, as it is achieved by Horizon, allows for increased load balancing. Additionally, Horizon takes a different way of providing a parallelization window, such that simulations that can not benefit from traditional techniques can benefit from the parallelization window of Horizon. Furthermore, we showed that the centralized architecture of Horizon allows for more sophisticated synchronization techniques like probabilistic synchronization [11] since the scheduler maintains global knowledge of the simulation state. Hence, for our approach we identified Horizon as the most promising technique to distribute work among cores in an SMP system.

3. CHALLENGES

Today's compute clusters and supercomputers consist of a set of computing nodes linked with a high-speed interconnect like Infiniband. Each computing node is an SMP system, based on a set of multi-core processors sharing the same main memory with about 8 to 16 cores per node [15].

The standard approach to PDES on these systems is the decomposition of the simulation model into a large number of LPs and assigning each CPU core at least one LP. Hence, each multi-core processor executes several LPs in parallel. Synchronous synchronization algorithms like LBTS then require the synchronization of all LPs in the overall simulation resulting in a synchronization message for each pair of LPs at every synchronization point. This means that a computing node might receive and transmit several copies of the same synchronization message for the LPs on that node.

Furthermore, each processing core is statically assigned an LP. Hence, it is not possible to easily transfer load to other cores on the same compute node if the cores are not equally loaded. Dynamic load balancing techniques exist, but bear additional overhead for workload migrating.

To utilize the full processing power of today's computing clusters and supercomputers, scalability is the most important performance issue. Each additional CPU core should significantly contribute to the simulation performance such that the costs of purchasing and maintaining larger clusters pay off. While performance studies show the scalability of distributed simulations for certain synthetic benchmarks on up to 2 million cores [2], not every simulation model is that well-suited for the loose synchronization necessary.

The traditional approach to apply distributed simulation on today's clusters yields the following challenges which we describe in detail in the following:

- The synchronization overhead grows with the number of CPU cores.
- Proper load balancing over all LPs becomes harder with an increasing number of LPs.

Synchronization Overhead.

For asynchronous synchronization algorithms like CMB large lookaheads are vital to avoid time creeping. Hence, for simulation models with small link delays (like wireless networks), synchronous algorithms like LBTS are preferred.

In LBTS, a barrier needs to be negotiated between all LPs in the simulation. Thus, the number of MPI messages exchanged and therefore the synchronization overhead increases with the number of LPs in the simulation. This is particularly challenging for large-scale simulations since the performance gained by adding additional processing units is easily eliminated by the additional overhead. Note that every core added to the simulation needs to synchronize with all the other cores in the simulation.

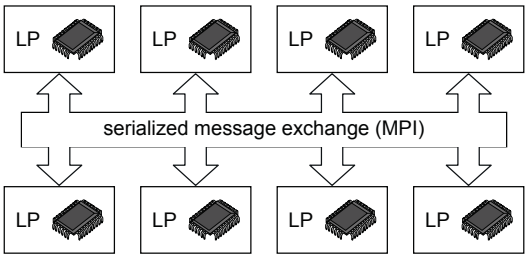
Load Balancing.

Proper load balancing is crucial for efficient large-scale simulation. The slowest LP always determines the simulation performance since all faster LPs need to wait for it to reach the specified barrier and potentially send out the necessary simulation events. Unbalanced load introduces idle times at less loaded LPs which waste computing resources, hence the overall simulation performance is reduced.

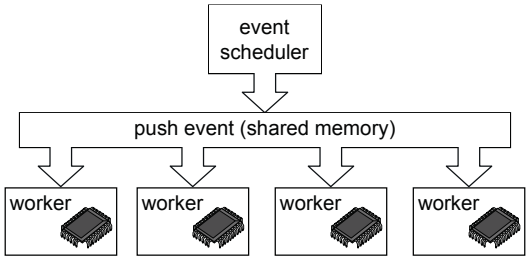
While load balancing is already a crucial issue in small-scale simulations, it gets more challenging at larger scales. Each LP added to the simulation needs to maintain the same pace as the other LPs. If the added LP is faster than the slowest LP, resources of the additional computing unit are wasted. If the LP is slower, it slows down the overall simulation wasting resources at all other computing units. Hence, each LP added might even slow down the overall simulation.

Furthermore, with varying load during a single simulation run there is no static partitioning that can ensure equal load during the whole run, hence waiting times occur on each LP. Dynamic partitioning could allow the migration of higher loaded simulation entities, but introduces additional overhead for the entity migration. Furthermore, predictions on the load in future simulation steps are required.

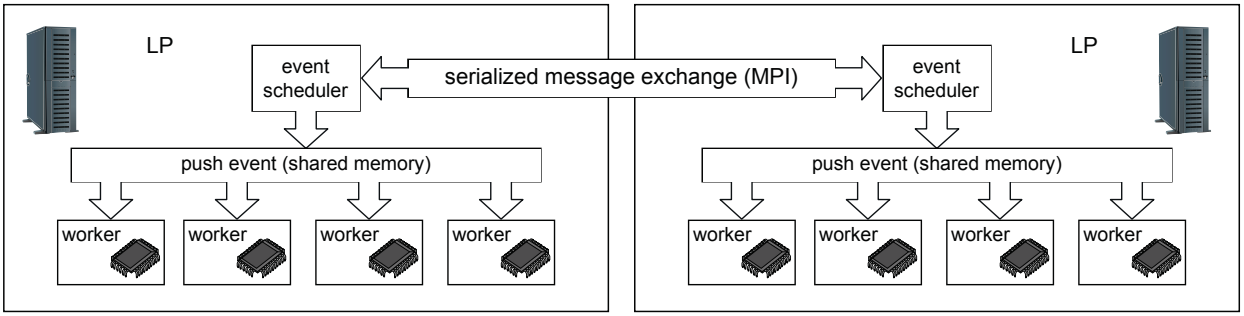
In summary, this work tackles two challenges by introducing a novel PDES approach: minimizing synchronization overhead and optimizing load balancing across computing units. In our approach we apply a multi-level parallelization scheme based on Horizon and distributed simulation to increase load balancing and decrease synchronization effort. By means of a novel synchronization algorithm we furthermore reduce the number of synchronization points necessary.



(a) Traditional Distributed Simulation. The model is decomposed into LPs and each processing unit executes an LP.



(b) Shared-memory Parallelization Horizon. The central scheduler thread dynamically offloads parallelizable events to idle worker threads.



(c) Distributed Horizon. We combine both schemes to leverage the strengths of SMP systems while enabling scalability beyond single computing nodes. Each LP now executes a Horizon process with multiple worker threads.

Figure 1: Distributed Horizon: The combination of (a) traditional distributed simulation with (b) the SMP approach Horizon leverages the strengths of today's SMP-based computing clusters and supercomputers.

4. EFFICIENT MULTI-LEVEL PDES

In this section we first describe the goals of our approach to efficient PDES simulations on SMP-based clusters before we introduce both contributions Distributed Horizon and the Dynamic Barrier Synchronization algorithm DBS.

4.1 Design Goals

In general, the goal of our multi-level PDES approach is to achieve high scalability on SMP-based clusters and maximize the performance of the simulation while the approach should be as easy to use by modelers as possible. More specifically, we state the following design goals:

Improve Load Balancing. Proper load balancing is a crucial property influencing the performance of parallel simulation. Idle CPU cores can not contribute to the progress of the simulation. The goal of our approach is to achieve better load balancing than distributing LPs over CPU cores in a straightforward way. We aim at parallelizing smaller work units inside each computing node to minimize the difference in work unit sizes.

Reduce Synchronization Overhead. Besides that, synchronization overhead is the most important factor influencing PDES performance. We summarize both the effort to agree on synchronization barriers and the effort to exchange event data between processing units under this notion. Synchronization barriers can be more efficiently agreed upon if they are negotiated locally between the CPU cores of a single computing node. Nevertheless, global barriers are required. We aim at minimizing both the number of synchronization points where global barriers need to be determined and the effort to determine each barrier. Furthermore, we target an efficient way of exchanging event data between the processing units of a single computing node avoiding the need for serialization and deserialization.

Increase Scalability. Our approach targets maximum scalability of a broad range of simulation models given that the model itself is scalable with the appropriate technique. This means that increasing the network size should not affect the simulation performance if at the same time the same amount of computing resources is added to the simulation. Our evaluation shows that the standard approach of distributed PDES with LBTS to simulate an LTE network does not maintain this property. Already a small-size network of only 48 LTE cells can not benefit from the total processing power available in four 12-core machines of a compute cluster. With our approach we target higher parallelization speedups for larger scale networks.

Ensure Ease-of-Use. Finally, PDES is only accepted by model developers if it is easy enough to apply. The goal of our approach is to ensure the ease-of-use such that the speedup achieved by the simulation is worth the effort required from the model developer. We aim at minimizing the effort for decomposing the network into LPs as well as specifying lookahead. However, we argue that certain manual effort can be tolerated if the performance boost can justify the overhead.

We divide our PDES approach into two independent concepts: Distributed Horizon leverages the strengths of SMP-based computing clusters by applying a multi-level parallelization scheme with distributed and shared-memory tailored parallelization concepts. The *Dynamic Barrier Synchronization* algorithm increases parallelization windows by enabling the model developer to easily include domain knowledge into the model. Combining both approaches then allows to utilize the full processing power of today's computing clusters achieving almost linear speedup in our case study.

4.2 Distributed Horizon

The idea behind our multi-level parallelization scheme DISTRIBUTED HORIZON is straightforward: We observed that Horizon enables efficient utilization of today's multi-core shared-memory systems. Therefore, we apply *Horizon* as the first level of our approach. Since Horizon by design does not scale beyond the size of available shared-memory systems, we need to combine this approach with an approach for simulation on multiple machines. Hence, we apply traditional *distributed simulation* as the second level of our multi-level parallelization to enable scalability beyond a single computing node.

Figure 1 visualizes the multi-level parallelization. Like for distributed simulation (see Figure 1(a)), the simulation model needs to be decomposed into several LPs. However, it is not necessary to create as many LPs as CPU cores are available for execution, but we only need an LP for each computing node. Each node then maintains an FES for the events on that LP as well as input and output queues like in the standard design of distributed simulation. However, instead of running a single scheduler that executes the events on that LP sequentially, we execute the Horizon scheduler (see Figure 1(b)) on each LP. As in pure Horizon, this scheduler thread assigns the events to workers on the same computing node by a particularly lightweight pointer passing scheme via shared memory (see Section 2.3).

Figure 1(c) visualizes the execution of DISTRIBUTED HORIZON on a two-node cluster. The simulation is partitioned into two LPs, each maintaining enough parallel events to keep all workers busy. The two nodes exchange serialized messages when events are transferred across LPs or the LPs need to be synchronized.

After initialization, each event scheduler now performs the following steps in a loop:

1. Offload all parallelizable events to worker threads.
2. Receive incoming MPI messages.
3. Determine next barrier.

For the first step it is necessary to obey two different kinds of synchronization barriers: the first is derived from Horizon (intra-LP barrier), the second is derived from distributed simulation (inter-LP barrier). Hence, the two synchronization algorithms need to be combined in a way that ensures causal correctness while maximizing the parallelization window and minimizing the synchronization overhead. Furthermore, the determination of the next barrier (Step 3) needs to be performed in a modified way. We discuss the combined synchronization algorithm in more detail in the following.

Synchronization in Distributed Horizon.

We need to design the multi-level synchronization algorithm in a way that it ensures the causal correctness of the simulation results. We therefore combine the causality constraint for distributed simulation as defined in [5] with the causality constraint for Horizon as defined in [10]:

Definition 1. A DES obeys the causality constraint if and only if each LP processes each pair of non-overlapping events in non-decreasing starting time order.

This means that the local event scheduler of each LP has to ensure that i) it only offloads an event if it overlaps with all events currently being executed on the same LP, and ii) it only offloads an event e if no incoming message might cause another event prior to e.

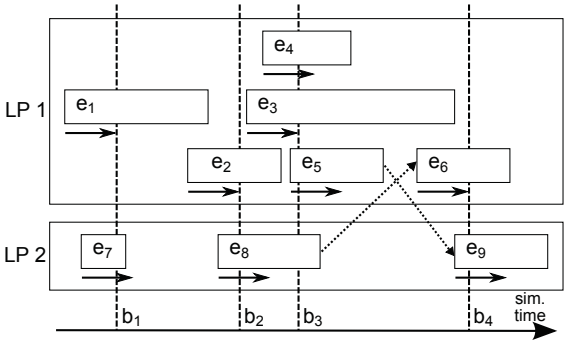


Figure 2: LBTS. The algorithm subsequently determines the (inter-LP) barriers b_1 to b_4 by adding the lookahead to the first event in the simulation. Solid arrows denote lookahead.

Hence, the scheduler has to maintain two barriers: an intra-LP barrier and an inter-LP barrier. The intra-LP barrier is determined by the basic Horizon rule: from all events currently being executed the earliest end time determines the intra-LP barrier. The inter-LP barrier needs to be determined by inter-process communication like in distributed simulation. For this purpose we implemented two different synchronization algorithms: LBTS [4] and our novel scheme Dynamic Barrier Synchronization (see Section 4.3). LBTS in DISTRIBUTED HORIZON works similar to the traditional LBTS algorithm (cf. Figure 2). Each LP determines the next event in the FES and adds the lookahead. Then, the minimum of these values determines the next inter-LP barrier.

The local event scheduler of each LP checks both barriers before offloading an event for parallel execution. If the intra-LP barrier prevents the event from being executed, the scheduler waits for the corresponding event to be completed, updates the barrier, and checks again. If the inter-LP barrier prevents the event from being executed, the scheduler initiates a new LBTS synchronization and waits for the other LPs to transmit their barrier timestamp. Only if both barriers allow the execution of the event, the event can be safely offloaded to a worker thread.

4.3 Dynamic Barrier Synchronization

Conservative synchronization algorithms essentially depend on sufficient lookahead between LPs, typically derived from link delays. Asynchronous synchronization protocols like CMB suffer from the time creeping problem, synchronous protocols like LBTS would, in the worst case, require a synchronization for each single event. On the other hand, optimistic synchronization requires expensive rollbacks, especially when simulation models maintain large states.

We argue that simple annotations created by a domain expert (e. g., the model developer) can significantly increase parallelization windows while reducing synchronization effort, for example in the following 3 cases: In an LTE model it usually suffices to exchange messages between LPs each 1 ms, i. e., at the beginning of each Transmission Time Interval (TTI), or even less often. A node in a computer network can predict a lower bound on the processing time of a packet traversing the network stack up and down and can this way create a guarantee for the earliest output time. In a queuing network, the sum of the delays between queues can as well provide a means to derive higher parallelization windows.

We therefore introduce our approach *Dynamic Barrier*

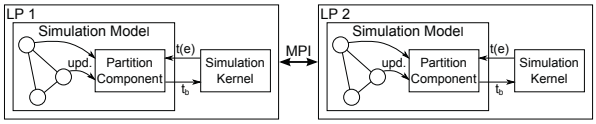


Figure 3: In DBS each LP maintains a Partition Component that can be implemented by the model developer and updated during the simulation by the simulation entities. The synchronization algorithm then determines the next barrier by requesting the Partition Component.

Synchronization allowing a modeler to *dynamically* specify the next synchronization barrier, by exploiting domain knowledge.

The general concept is depicted in Figure 3. Each LP maintains the simulation entities (e. g., modules) assigned to that LP. Additionally, we create an instance of a so called *Partition Component* that allows the modeler to store and provide information which do not belong particularly to a simulation module, but to the overall partition. For DBS to work properly, we only require a single function to be implemented in this component. This function is provided the timestamp $t(e)$ of the next event e in the FES and needs to return another timestamp t_b. The simulation framework considers t_b as a guarantee that this LP will not generate any messages prior to this timestamp. However, this guarantee is discarded if a new message is received from a neighboring LP, i. e., this message is still allowed to generate outgoing messages prior to t_b. More precisely, the minimum of the timestamps at all LPs yields a guarantee that no LP will send out a message to be executed before this timestamp.

The simulation procedure is then similar to LBTS. The scheduler processes local events and receives incoming messages until it reaches a barrier. However, instead of sending out the timestamp $t(e)$ of the next event e, it then requests the local partition component to determine a barrier timestamp t_b. After that, the LP broadcasts its local barrier t_b and receives the timestamps of the other LPs. We denote the set of all timestamps by T_b. Similarly to LBTS, the next global synchronization barrier is now derived as the minimum of all exchanged timestamps, i. e., $b := \min(T_\mathrm{b})$.

It is obvious that a modeler can simply mimic LBTS with this algorithm. The Partition Component is provided the timestamp of the next local event $t(e)$ to determine the next local guarantee t_b. For LBTS, this guarantee is the sum of $t(e)$ and the lookahead l. Therefore, it suffices to return $t_\mathrm{b} := t(e) + l$ to provide the same barrier as LBTS. However, by providing a timestamp $t_\mathrm{b} > t(e)+l$ through adding domain knowledge, modelers can increase the parallelization window and decrease the number of synchronization points.

We reconsider the example in Figure 2. LBTS subsequently creates 4 barriers and provides only small parallelization windows. However, there are only two cross-LP messages, such that – with global knowledge of the simulation model – it becomes clear that the parallelization windows are smaller than necessary. Provided that the timestamps of the next outgoing messages are known before the execution of e_1, the model developer can annotate this information into the Partition Component (see Figure 4). Then, the Partition Component of LP 1 yields $t_{\mathrm{b}1}$ and the component of LP 2 yields $t_{\mathrm{b}2}$. The minimum barrier ($t_{\mathrm{b}1}$) then opens a parallelization window that allows LP 1 to execute e_1 to e_5 while LP 2 executes e_7 and e_8.

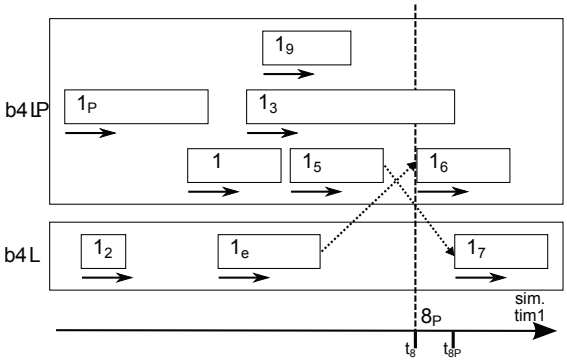


Figure 4: Through applying domain knowledge, model developers can significantly reduce the number of synchronization points with Dynamic Barrier Synchronization (compared to LBTS in Figure 2).

DBS in Distributed Horizon.

We designed DBS as a synchronization algorithm for both distributed simulation and Distributed Horizon. The integration into Distributed Horizon is straightforward: When the Horizon scheduler needs to determine a new global barrier, it does not execute LBTS but DBS as described above in order to negotiate a new (inter-LP) barrier. DBS then requests the local Partition Component for the next guarantee timestamp and the minimum of all these timestamps is the next inter-LP barrier for each LP. This increases the parallelization window like in distributed simulation, but additionally provides larger windows for Horizon to parallelize events.

5. DISCUSSION

Distributed Horizon can be easily used with any simulation model that is well suited for distributed simulation and Horizon. The user only needs to specify the partitions and can run the simulation in parallel.

However, with LBTS as a synchronization paradigm the parallelization window is limited. The Horizon scheduler can not execute two events in parallel if there is an inter-LP barrier in between, even if the events overlap and the Horizon paradigm therefore allows parallelization. Figure 2 depicts this problem. Here, e_1 and e_2 are not executed in parallel due to b_1. A conservative scheme without domain knowledge can not avoid this situation since e_7 on LP 2 might induce an event e' prior to e_2 not overlapping with e_2. In this case, e' needs to be executed before e_2. Hence, inter-LP barriers limit the parallelization window provided by Horizon and prevent parallel execution of actually independent events.

Our approach to Dynamic Barrier Synchronization provides a means to the user to annotate the next necessary synchronization point. We argue that this annotation can be achieved by the modeler through applying domain knowledge. For time-slotted systems like LTE where messages are only exchanged at certain points in time, this annotation can be trivially performed. The function only needs to determine the next time slot (e. g., next TTI) and return it.

However, this kind of annotation can become more complicated if the next message is not that easy to predict. For example, in a network router the timestamp of the next outgoing event can be determined by computing a lower bound for the time it takes for a network packet to traverse the

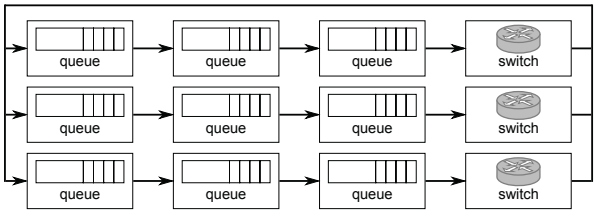


Figure 5: The closed queuing model consists of a set of tandem queues, each of which consists of a set of queues connected to a switch which randomly feeds back jobs to another tandem queue.

network stack up and down. If a packet has already been partially processed, the remaining computing time needs to be reduced accordingly. Note that in this example the barrier determined is the same that would be determined by the path lookahead concept of Meyer and Bagrodia [16].

To this end, a modeler has to carefully design and implement the Partition Component to avoid yielding too optimistic barriers. Such an error would result in a causal violation at the receiving LP. Since this might induce false simulation results, we decided to abort the simulation and yield an error describing the wrong guarantee timestamp. This allows to track down and fix the annotation bug.

It is also possible to provide too conservative guarantees resulting in poor performance. However, by just adding the lookahead to the current event timestamp it is always possible to provide at least the guarantee of LBTS and therefore achieve at least the performance of LBTS.

We recommend the following work flow for DBS: i) use LBTS to check the correctness and parallelizability of the model, ii) then use DBS with providing the guarantees as LBTS does, and iii) finally increase these guarantees step by step until the performance is satisfying. Furthermore, it is possible to develop tools that analyze previous simulation runs and provide the modeler information about the necessary synchronization points and potential barrier annotations.

6. EVALUATION

We evaluate the performance of our contributions by first investigating the influence of certain parameters on the performance of a rather synthetic queuing network simulation and then applying the approaches to a case study based on an abstract LTE model.

6.1 Methodology

We implemented Distributed Horizon and DBS for *OMNeT++* [21] since there is an implementation for both Horizon [9] and distributed simulation (called *Parsim* [23]) for OMNeT++. To ensure a fair comparison, we chose OMNeT++ as the base framework for all measurements. We extended Parsim with an implementation of LBTS as described by Chandy and Sherman in [4].

We measured the performance of four different configurations: i) Parsim (distributed simulation) with LBTS, ii) Parsim with DBS, iii) Distributed Horizon with LBTS, and iv) Distributed Horizon with DBS.

All results are based on 5 repetitions of the measurements on the "Bull MPI-S" cluster of the Computing Center of RWTH Aachen University [1]. In this cluster, each node is equipped with 12 physical CPU cores rated at 3 GHz. The

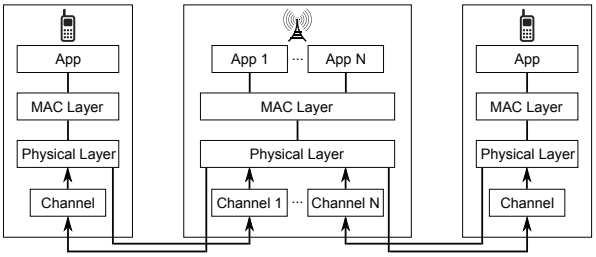


Figure 6: Layers and data flow in the abstract LTE model. The stations communicate by sending the packets along the arrows. Additionally, the MAC schedule and mobile station positions are broadcast every TTI to the two-hop neighbor cells.

plots show the average parallelization speedup as well as the 99 % confidence intervals.

We created two different simulation models: a CQN model to investigate the influence of certain parameters on the performance of the approaches and an abstract LTE model to determine the scalability of the techniques.

Closed Queuing Network.

We base our Closed Queuing Network (CQN) evaluation model on the queuing network example of OMNeT++. Here, a set of tandem queues (i. e., sequentially connected queues) is connected by switches that feed outgoing jobs from a tandem randomly into another tandem queue (see Figure 5). After a process reached the beginning of a queue, it is serviced for a certain amount of time. We model the service event by means of an expanded event spanning the service time. Replacing two distinct start- and stop-service events by a single expanded event is the primary idea of Horizon [9]. This does not change the semantics of the model, but provides a parallelization window to Horizon since the expanded event can not create new events before its end time. To mimic workload of a potentially sophisticated event handler, the service event includes a loop performing dummy operations for a certain amount of time. We further implemented DBS by estimating the time it takes until the next job leaves the LP and providing this time as the next barrier.

We created a base configuration that enables speedup with all investigated approaches and then modified event complexity, service time, and delays in a parameter study.

Abstract LTE Model.

We evaluate our contributions by means of a case study of an LTE simulation. For this purpose, we developed an abstract LTE simulation model for OMNeT++ which attaches importance to the structure of LTE rather than details of the implementation. We designed the model in a way that LTE engineers can include detailed algorithms for the parts they are interested in, e. g., the MAC scheduler which assigns transmission resource blocks to mobile stations. We argue that these algorithms typically only increase the complexity of the simulation but do not change the structure of the model, hence providing more parallelizable workload. Our model exchanges information at the beginning of every TTI with all cells in a two-hop neighborhood. Since LTE works in these time slots, we argue that relevant information can be exchanged at exactly these time points and no communication within a TTI is necessary. Furthermore, direct communication between far away cells should not be necessary

for interference calculation or cell synchronization. We conclude that typical LTE simulation applications scale similar or better with our approaches than this abstract model.

Our LTE model is constructed as follows (cf. Figure 6). On each mobile station there is an application module, generating traffic by means of a Poisson process, such that all stations cause an average load of 40 % in the cell. Similarly, the base stations maintain such an application for each connected mobile station. The base station MAC layer computes the schedule by means of a simple algorithm and distributes it to the mobile stations. At the beginning of the TTI, all packets that can be transmitted in that TTI are then forwarded to the physical layer which transmits the data to the corresponding channel at the destination. Here, path loss, fading, and interference is computed and the packet is either forwarded or marked corrupted and deleted. For load generation, we applied an accurate OFDM fading model as described in [22].

We evaluate our approaches by scaling the LTE model from 48 cells on 48 CPU cores up to 1,536 cells on 1,536 CPU cores distributed over 128 computing nodes.

6.2 Closed Queuing Network

For the CQN model, we created a base configuration and modified certain parameters one after the other.

CQN Base Configuration.

We perform a first comparison with the following parameters: The network consists of 4 tandem queues each maintaining 48 basic queues. With Distributed Horizon we partition this model into 4 LPs (a tandem queue per LP) to execute the simulation on 4 computing nodes. For traditional distributed simulation we further partition each tandem queue into 12 LPs of 4 basic queues each.

The service time is exponentially distributed with a mean of 10 s. The delay between two queues is fixed to 1 s, the delay between a switch and the subsequent tandem queue is set to 10 s. For the base scenario we set the average event complexity of the service event to 100 µs.

Figure 7 depicts the results of the base configuration. We observe that Distributed Horizon gains significantly more speedup than traditional distributed simulation. We attribute this to the fact that Horizon opens another parallelization window that allows for additional parallelization of events inside the tandem queues. With 10 s, the link delay between two LPs of Distributed Horizon is significant and the expanded events provide additional potential for parallelization. On the other hand, the link delays between two smaller LPs of traditional distributed simulation is shorter and since this approach does not use the service time for parallelization, speedup is rather limited.

We further experience that DBS provides an observable, but small gain. We attribute this to the fact that most of the time there is actually workload at the last queue in the tandem and it is therefore – even with knowledge of the model – not possible to provide significantly greater lookahead. We nevertheless observe that the approach is not worse than LBTS since the window is never smaller than the one determined by LBTS.

CQN Parameter Study.

In the following, we discuss the influence of certain parameters in the simulation on the parallelizability with the different approaches. In particular, we vary the event complexity, the mean service time, the queue-to-queue delay, and the switch-to-queue delay.

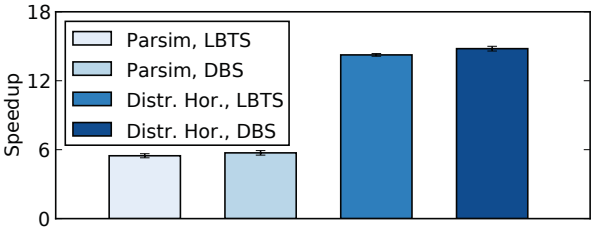


Figure 7: Speedup of CQN Base Setup. Event Complexity: 100 µs, Mean Service Time: 10 s, Switch-to-queue Delay: 10 s, Queue-to-queue Delay: 1 s.

With sufficient event complexity all approaches gain significant speedup. However, with decreasing event complexity we expect Distributed Horizon to loose performance since the Horizon schedulers have to offload each event separately for parallel execution and therefore introduce a small amount of overhead for each event. Figure 8(a) depicts the results of event complexities ranging from 1 µs to 100 µs. While we observe the expected drop in the performance of Distributed Horizon, we also observe a drop in the performance of OMNeT++ Parsim. This means that even with coarser grained parallel jobs, performance heavily depends on the complexity of each event. In the end, for lightweight events Distributed Horizon gains minimal speedup while the traditional approach becomes as slow as sequential execution for event complexities of only 1 µs.

Figure 8(b) depicts the influence of the mean service time of the jobs. In Distributed Horizon, we observe rather low sensitivity to the mean service time. On the one hand, shorter service times result in shorter parallelization windows for Horizon. On the other hand, the number of events in the parallelization window of the distribution level increases. In the end, this results in similar performance for different service time values. As opposed to Distributed Horizon, Parsim shows significantly more sensitivity to the service time. If the service time is as short as the queue-to-queue delay, Parsim nearly reaches the performance of Distributed Horizon.

The switch-to-queue delay has, as depicted in Figure 8(c), no influence on the performance of traditional parallel simulation. Since the limiting lookahead is determined by the switch-to-switch delay, the switch-to-queue delay does not influence the parallelization windows. On the other hand, the coarser grained LPs of Distributed Horizon determine their inter-LP lookahead exclusively from the switch-to-queue delay. Hence, we observe higher speedups for longer switch-to-queue delays in Distributed Horizon. Furthermore, DBS gains more speedup over LBTS when the lookahead is small. We attribute this to the fact that DBS additionally incorporates the state of the simulation as well as the current service time and can therefore hide the influence of the lookahead to a certain degree.

The queue-to-queue delay is the parameter that determines the lookahead for the simulations with Parsim. The coarser grained partitions of Distributed Horizon only determine the lookahead from the switch-to-queue delay. Therefore, this parameter has no influence on the performance of Distributed Horizon (see Figure 8(d)). On the other hand, larger lookahead values allow Parsim to out-

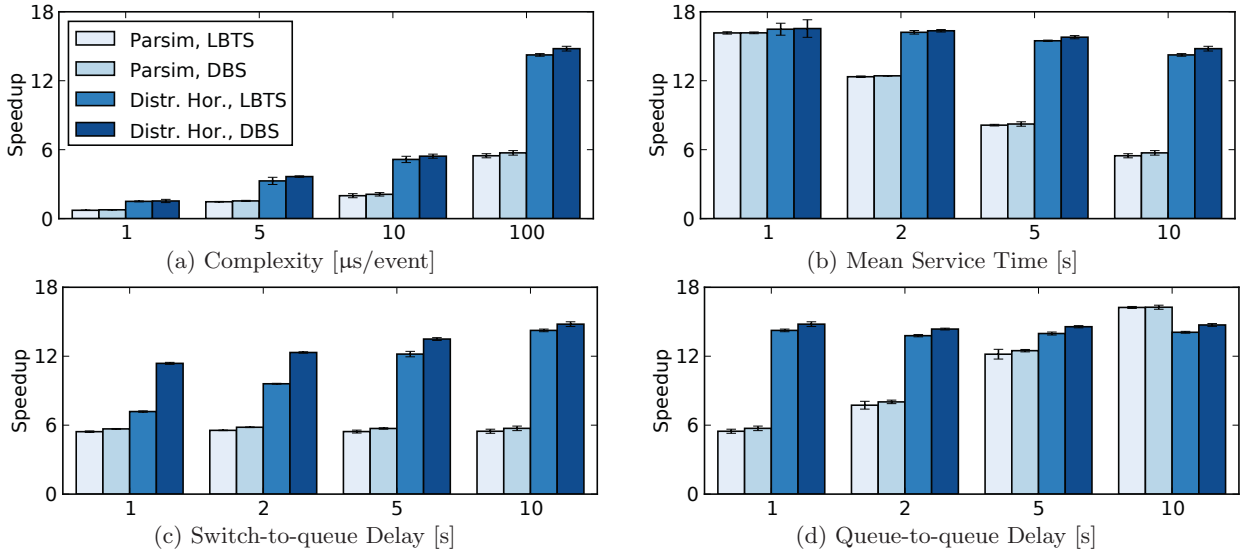


Figure 8: CQN Parameter Study: Speedup when varying certain parameters from the base case in Figure 7.

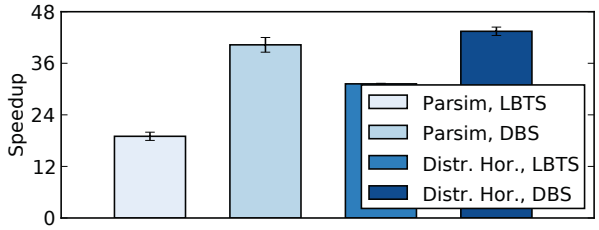


Figure 9: Parallelization speedup of LTE with 48 cells on 4 computing nodes with 12 CPU cores each.

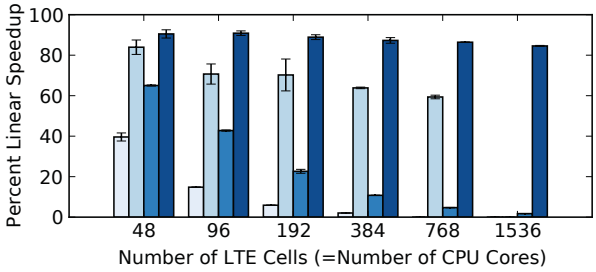


Figure 10: LTE scalability study: Performance compared to linear speedup, extrapolated from a sequential simulation run with 48 cells. For legend see Figure 9.

perform Distributed Horizon. This means that in situations where traditional synchronization algorithms can provide better parallelization windows than Horizon, traditional distributed simulation is better suited than Distributed Horizon.

In summary, Horizon depends on significant durations of the most complex events while Parsim requires significant link delays to extract lookahead. Hence, Distributed Horizon achieves the best speedup if both is available while traditional distributed simulation might be the better choice for simulations with high link delays and short event durations.

6.3 LTE Case Study

For the LTE case study, we first compare the performance achieved in a 48 cell setup before we increase the number of cells to investigate the scalability of the approaches.

Performance Comparison.

For the first evaluation setup, we created an LTE network comprising 48 cells and executed the simulation on 4 computing nodes equipped with 12 physical cores each. Hence, we have 4 LPs in Distributed Horizon and 48 LPs with Parsim. For a fair comparison, we ensured that in both setups we have the same assignment of cells to compute nodes. We performed this distribution in a way that neighboring cells are on the same computing node if possible.

Figure 9 depicts the results of this setup. We observe that OMNeT++ Parsim reaches with LBTS only a speedup of about 20 on this 48-core system. Applying Distributed Horizon reduces the complexity of the inter-LP synchronization and achieves therefore a speedup of about 30. On the other hand, applying DBS reduces the number of synchronization points significantly since the necessity of synchronizing each packet generation event at the application layer modules is omitted. This increases the speedup from 20 to 40. With both schemes combined, an almost linear speedup of 44 is achieved on the 48-core system.

Scalability Evaluation.

We investigate the scalability of our approach by increasing the number of LTE cells. At the same time, we increase the number of computing nodes available to the simulation such that there is always a physical core per cell. We extrapolate the performance of a sequential simulation run for all scenarios from the run with 48 cells and compute how close the approaches get to linear speedup. The results are depicted in Figure 10. We observe that the LBTS synchronization algorithm is not scalable at all. Without Distributed Horizon, LBTS achieves a speedup of less than 8 simulating 384 cells on 384 CPU cores. We attribute this to the fact that all applications generating events have to be synchronized all over the simulation. However, Distributed

HORIZON performs significantly faster also with LBTS since it reduces the amount of LPs that need to be synchronized at each synchronization point.

With DBS we observe proper scalability in computing time of both Parsim and DISTRIBUTED HORIZON. Utilizing the shared memory with DISTRIBUTED HORIZON additionally reduces the memory requirements. For this reason, a simulation with 1,536 cells was still feasible with DISTRIBUTED HORIZON while Parsim ran out of memory. Furthermore, we observe less sensitivity to network size for DISTRIBUTED HORIZON allowing almost perfect scaling of the LTE simulation up to a network size of 1,536 cells. Here, we measure only a few percent longer simulation runtimes than for the 48 cell setup and compared to the extrapolated runtime of sequential simulation a speedup of 1,300, i. e., about 85 % of linear speedup.

7. CONCLUSION AND FUTURE WORK

In this work, we introduced two approaches to reduce the synchronization effort in parallel simulation and increase the load balancing to leverage the strengths of today's computing clusters. Our multi-level synchronization paradigm DISTRIBUTED HORIZON combines the shared-memory parallelization approach Horizon with the traditional approach of distributed simulation. This allows distributing workload across computing nodes while at the same time minimizing the synchronization effort inside a computing cluster node by applying a scheme specifically tailored to SMP nodes. With our novel synchronization algorithm DBS it is additionally possible to specify greater parallelization windows to further reduce the synchronization overhead.

Future efforts to DISTRIBUTED HORIZON target the emerging platform of many-core processors like Intel Phi. Here, a third synchronization layer between Horizon and distributed simulation could be introduced to reduce the contention to the central structures of Horizon if 100 or more threads run on an SMP node. Furthermore, different synchronization algorithms including optimistic approaches or advanced approaches like probabilistic synchronization [11] could be applied on the layers of DISTRIBUTED HORIZON to investigate the performance. Focusing on our novel synchronization algorithm DBS, future efforts could create an asynchronous synchronization algorithm from the guarantees of DBS.

In summary, in this work we introduced two contributions, DISTRIBUTED HORIZON and Dynamic Barrier Synchronization, which in combination showed to scale an LTE model almost linear on more than 1,500 cores.

Acknowledgments

This research was funded by the DFG Cluster of Excellence on Ultra High-Speed Mobile Information and Communication (UMIC).

8. REFERENCES

[1] D. an Mey et al. The RWTH HPC-Cluster User's Guide. Technical report, RWTH Aachen University, Aug. 2013.

[2] P. Barnes, C. Carothers, D. Jefferson, and J. LaPre. Warp Speed: Executing Time Warp on 1,966,080 Cores. In *Proc. of the 27th ACM SIGSIM Conf. on Principles of Advanced Discrete Simulation*, 327–336, 2013.

[3] K. Chandy and J. Misra. Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. *IEEE Trans. on Software Engineering*, 5(5):440–452, 1979.

[4] M. Chandy and R. Sherman. The Conditional-Event Approach to Distributed Simulation. Technical report, DTIC Document, 1989.

[5] R. Fujimoto. Parallel Discrete Event Simulation. *Communications of the ACM*, 33(10):30–53, 1990.

[6] R. Fujimoto. Parallel and Distributed Simulation. In *Proc. of the 31st Winter Simulation Conf.*, 122–131, 1999.

[7] P. Heidelberger and D. Nicol. Conservative Parallel Simulation of Continuous Time Markov Chains Using Uniformization. *IEEE Trans. on Parallel and Distributed Systems*, 4(8):906–921, 1993.

[8] D. Jagtap, N. Abu-Ghazaleh, and D. Ponomarev. Optimization of Parallel Discrete Event Simulator for Multi-core Systems. In *Proc. of the 26th Intl. Parallel Distributed Processing Symposium*, 520–531, 2012.

[9] 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 Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 172–181, 2010.

[10] G. Kunz, M. Stoffers, J. Gross, and K. Wehrle. Runtime Efficient Event Scheduling in Multi-threaded Network Simulation. In *Proc. of the 4th Conf. on Sim Tools and Techniques*, 359–366, 2011.

[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 Conf. on Sim Tools and Techniques*, 119–128, 2012.

[12] J. Liu and D. Nicol. Learning Not to Share. In *Proc. of the 15th Workshop on Parallel and Distributed Simulation*, 46–55, 2001.

[13] J. Liu and R. Rong. Hierarchical Composite Synchronization. In *Proc. of the 26th Workshop on Principles of Advanced and Distributed Sim.*, 3–12, 2012.

[14] B. Lubachevsky. Efficient Distributed Event-Driven Simulations of Multiple-Loop Networks. *Communications of the ACM*, 32(1):111–123, 1989.

[15] H. Meuer, E. Strohmaier, J. Dongarra, and H. Simon. Top500 List, June 2013. www.top500.org/list/2013/06/.

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

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

[18] D. Nicol. Parallel Discrete-Event Simulation of FCFS Stochastic Queueing Networks. In *Proc. of the ACM SIGPLAN Conf. on Parallel Programming: Experience with Applications, Languages and Systems*, 124–137, 1988.

[19] D. Nicol and J. Liu. Composite Synchronization in Parallel Discrete-Event Simulation. *IEEE Trans. on Parallel and Distributed Systems*, 13(5):433–446, 2002.

[20] 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*, 119–126, 1993.

[21] A. Varga. The OMNeT++ Discrete Event Simulation System. In *Proc. of the 15th European Simulation Multiconf.*, 2001.

[22] C. Wang, M. Pätzold, and Q. Yao. Stochastic Modeling and Simulation of Frequency-Correlated Wideband Fading Channels. *IEEE Trans. on Vehicular Technology*, 56(3):1050–1063, 2007.

[23] D. Wu, E. Wu, J. Lai, A. Varga, A. Şekercioğlu, and G. Egan. Implementing MPI Based Portable Parallel Discrete Event Simulation Support in the OMNeT++ Framework. In *Proc. of the 14th European Simulation Symposium*, 2002.

[24] Z. Xiao, B. Unger, R. Simmonds, and J. Cleary. Scheduling Critical Channels in Conservative Parallel Discrete Event Simulation. In *Proc. of the 13th Workshop on Parallel and Distributed Simulation*, 20–28, 1999.