2010 18th Annual IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems

Expanding the Event Horizon in Parallelized Network Simulations

Georg Kunz, Olaf Landsiedel[1], Stefan Götz, Klaus Wehrle
Distributed Systems Group
RWTH Aachen University
{kunz,landsiedel,goetz,wehrle}@cs.rwth-aachen.de

James Gross, Farshad Naghibi
Mobile Network Performance Group
RWTH Aachen University
{gross,naghibi}@umic.rwth-aachen.de

***Abstract*—The simulation models of wireless networks rapidly increase in complexity to accurately model wireless channel characteristics and the properties of advanced transmission technologies. Such detailed models typically lead to a high computational load per simulation event that accumulates to extensive simulation runtimes. Reducing runtimes through parallelization is challenging since it depends on detecting causally independent events that can execute concurrently. Most existing approaches base this detection on lookaheads derived from channel propagation latency or protocol characteristics. In wireless networks, these lookaheads are typically short, causing the potential for parallelization and the achievable speedup to remain small.**

This paper presents *Horizon*, which unlocks a substantial portion of a simulation model's workload for parallelization by going beyond the traditional lookahead. We show how to augment discrete events with durations to identify a much larger horizon of independent simulation events and efficiently schedule them on multi-core systems. Our evaluation shows that this approach can significantly cut down the runtime of simulations, in particular for complex and accurate models of wireless networks.

I. Introduction

Discrete event-based wireless network simulation currently faces at least two significant changes: First, recent advances in wireless communication technology demand highly accurate simulation models, resulting in a steep increase in *model complexity* and runtime requirements. Second, multi-processor computers constitute the de-facto default hardware platform even for desktop systems, thus providing cheap yet powerful "private computing clusters". As a result, the *parallelization* of discrete event simulations significantly gained importance and is therefore (again) in the focus of active research.

Model Complexity: Simulation models of wireless networks typically require a considerably more detailed modeling of the lower network layers than models of wired networks. In particular, the wireless channel and the physical layer demand precise models to capture the subtle effects and interactions of advanced wireless communication technologies such as MIMO transmissions or successive interference cancelation. Additionally, those systems depend on accurately timed models that take even short delays such as the processing time of algorithms and hardware components into account. Moreover, the computational complexity of simulation models is further aggravated by the fact that the wireless channel is a broadcast domain. This causes a much higher number of simulated nodes to be involved in the communication process than in wired networks. Consequently, simulation runtimes increase drastically which in turn hampers the development process and in-depth evaluations. Researchers often work around these issues by trading accuracy for shorter runtimes [1], [2]. Such trade-offs, however, need to be applied carefully as they may lead to incorrect simulation results [3]. In addition, researchers often utilize parallel hardware to execute independent sequential simulations in parallel to cover a wide range of simulation parameters and/or to establish statistical confidence. However, our experience shows that particularly during early development phases of new protocols or distributed systems, it is important to timely obtain approximate results in order to speed up and guide the development process. Furthermore, studies of the long-term behavior of complex systems demand an efficient simulation execution. Consequently, we identify the need for truely parallelized simulations.

[1]While at the Distributed Systems Group. New affiliation is School of Electrical Engineering, Royal Institute of Technology (KTH), Sweden

Parallel Discrete Event Simulation: Having been an active field of research for more than two decades, parallel discrete event simulation is supported by a wide range of network simulation frameworks [4], [5], [6], [7], [8]. Despite this tool support, creating a parallel simulation model is challenging and running simulations on a distributed simulation cluster is complex [9]. At the same time, the increasing number and speed of processing cores in today's commodity hardware makes a higher degree of parallelization very attractive and cost-effective for speeding up network simulation. Nevertheless, a key challenge in parallel simulations, in particular of wireless networks, is that the lookaheads of parallelizable events are very small due to short propagation delays and the broadcast nature of the wireless channel.

In this paper we address these challenges by introducing Horizon, a simple parallelization scheme that enables an efficient simulation of wireless networks on multi-processor systems. The key concept of Horizon is to identify independent simulation events by means of extended timing information and to execute such events dynamically on different processing units in parallel. To this end, Horizon extends discrete events with durations in simulation time in order to naturally model delays and exploit parallelism in the simulation model. Horizon specifically focuses on multi-processor systems to take advantage of fast shared-memory synchronization primitives and global control over the simulation. Summarizing, our paper makes the following two key contributions:

1) We introduce a methodology for augmenting discrete events with *durations* to explicitly and naturally model delays in discrete event simulation.
2) We present a *horizontal parallelization* scheme that exploits the given event durations for efficient parallel simulation on multi-processor systems.

Our evaluation shows that Horizon achieves up to the theoretical maximum of a linear speedup with respect to the number of CPUs. Furthermore, we analyze the performance characteristics of Horizon on the basis of synthetic benchmarks and a case study of a complex cellular LTE (Long Term Evolution) network model. Additionally, we derive approximate metrics for estimating the applicability of Horizon to a specific simulation model. The remainder of this paper is structured as follows: In Section II, we introduce state-of-the-art approaches to parallel network simulation and put them in context with our approach. Based on this background, we present the timing-information based parallelization scheme of Horizon in Section III and discuss advantages and limitations. Section IV details on the implementation of Horizon, followed by an performance evaluation in Section V. Finally, we discuss related work in Section VI and conclude in Section VII.

II. Background

Before introducing the design of Horizon, this section briefly discusses the relevant basics of discrete event simulation and gives an overview of state-of-the-art approaches to parallel simulation. A discussion of closely related research efforts is presented in Section VI.

In a discrete event simulation, each event e is associated with a specific timestamp t_e denoting its time of occurrence. The execution of an event does not consider the delay of the simulated process, but instead events happen instantaneously and take zero simulation time. The rationale behind this paradigm is most notably that i) it simplifies simulation by allowing a sequential execution of events according to a total ordering. This prevents causal violations [10], i.e., the execution of two events in the wrong order. ii) the actual durations of simulated processes are often not accurately known and iii) in practice, this paradigm still enables the modeling of delays by adjusting timestamps accordingly. However, we argue that the explicit integration of durations in discrete event simulation fosters a more natural and thus accurate modeling of timing. Moreover, the additional timing information can be exploited to efficiently parallelize simulation models with small lookaheads.

Traditional parallel discrete event simulation relies on two classes of synchronization algorithms to avoid causal violations. Conservative synchronization algorithms [11], [12], [13] strive to strictly avoid causal violations by identifying independent events which are safe for parallel execution. To efficiently identify such events, these algorithms heavily rely on the lookahead which is a lower time bound for the interaction of entities in a parallel simulation model [14]. In network simulation, the lookahead is typically based on simulated link delays and/or protocol properties [15], [16], [17]. In contrast, optimistic algorithms [18] speculatively execute events in parallel and perform roll-backs once a causal violation is detected. These algorithms considerably reduce the synchronization effort for loosely coupled systems and are less dependent on the lookahead than conservative algorithms.

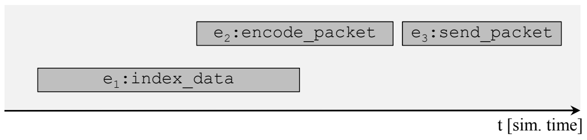


Fig. 1. Determining independent events: A simple example showing three expanded events that each span a certain period of simulation time. The events e_1 and e_2 cannot depend on each other due to their temporal overlapping and can thus be executed in parallel. e_3 must follow sequentially.

For both classes of algorithms, simulation models of wireless networks constitute a worst case scenario. Conservative algorithms suffer from extremely small lookahead values because of short propagation delays. In addition, the broadcast nature of the wireless channel results in a tight coupling of the simulated network nodes, thus causing frequent synchronization. In case of optimistic algorithms, synchronization usually implies roll-backs which drastically impact performance.

III. Design

A. General Idea

As pointed out previously, discrete events occur instantaneously at discrete points in simulation time. However, let's assume for now that we extend this modeling principle with the ability to handle events that span a period of simulation time. Given such functionality, we can now augment events with durations which naturally model the delay of simulated processes and algorithms.

Fig. 1 shows a simple example of three augmented events e_1, e_2, and e_3 representing a "packet encoding", a "packet sending, and a "data indexing" process. We observe that in the particular timing chosen for this example, e_1 and e_2 overlap in time while e_3 follows after the end of e_2. The overlapping implies that e_2 cannot depend on any results generated by e_1 because e_2 already begins while e_1 is still processing, i.e., its results are not yet available. Consequently, we conclude that both events are independent and can thus be processed in parallel. However, we cannot conclude whether or not e_3 is independent of the other two events since it begins after the earlier events finished. In this example, it is indeed dependent on e_2 which calculates the encoded packet that is sent by e_3.

Horizon bases its parallelization scheme on the observation of overlapping events. This parallelization scheme dynamically processes independent events from any layer and any node in the simulation model. Hence, we denote it *horizontal parallelization* to distinguish it from classic parallelization which partitions the simulation model vertically in terms of clusters of nodes. In the following we introduce the core concept in a formalized context, analyze its validity, and discuss limitations.

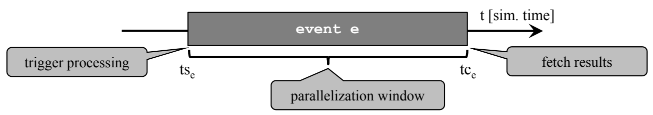


Fig. 2. Execution scheme of an expanded event e: The results of a simulated continuous task are not needed in the simulation before tc_e. Hence, the simulation scheduler offloads e to a worker CPU at ts_e and fetches the results at tc_e allowing other events to be processed in-between.

B. *Integrating Event Durations*

Horizon allows events to span a specific period of time. We denote this period the *event duration* td_e of an *expanded event* e. Consequently, an expanded event e is defined by a distinct starting time ts_e and a distinct completion time tc_e (see Fig. 2). We explicitly allow $ts_e = tc_e$ in order to handle traditional *discrete events*. On the one hand, this provides backwards compatibility to existing simulation models and enables their seamless transition to duration based modeling. On the other hand, discrete events may be used to perform maintenance tasks in a simulation such as propagating data via side channels. Such tasks can neither be mapped to a continuous process, nor do we want them to influence the simulation timing.

An expanded event begins when the simulation time reaches ts_e, but any results generated by this event are not needed within the simulation until the completion time tc_e. This follows directly from a correctly time-annotated simulation model: Consider the events e_2 and e_3 from the simple example given in Fig. 1. Obviously, e_3 depends on the output of e_2 since it can only send a packet after it was completely encoded. As a result, any overlapping timing in this case is incorrect and contradicts a valid simulation model. Conversely, we can conclude that an events e' that begins between ts_e and tc_e is independent to e. Thus, the time span between ts_e and tc_e naturally opens a time window (i.e., *horizon*) in which the parallel execution of independent events can be performed on different processing units. Specifically, at ts_e the simulation kernel offloads the execution of an event to an available processor and continues handling further independent events until the simulation time reaches tc_e. At tc_e, the simulation kernel waits for the calculation to finish if needed, fetches the results, and continues (see Fig. 3).

From a modeling perspective, the simulation time does not advance within an expanded event. Instead, each expanded event is executed in its own "time context" that always corresponds to its specific ts_e. The rationale behind this is two-fold: First, there is no general way of relating the actually executed operations within some expanded event to a continuously increasing time. After all, Horizon allows to model a long running continuous process by associating a comparatively long event duration to an expanded event, independent of the amount of code executed in this event. Second, the advancement of the simulation time should not be coupled to the global simulation time because this creates unwanted dependencies on the timing of other events: The

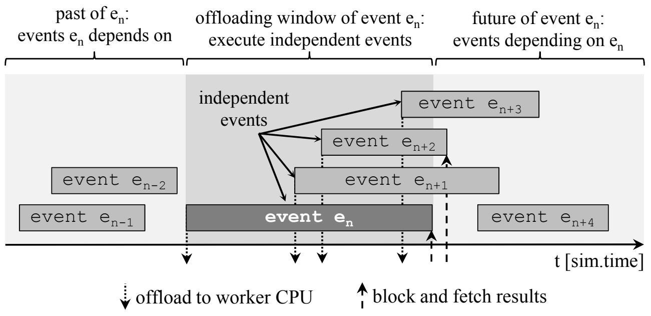


Fig. 3. Parallel event scheduling: The central scheduler advances the global simulation time by iteratively determining independent events, offloading them to worker CPUs and fetching the results of completed calculations.

global simulation time advances in discrete steps according to the ts and tc values of the scheduled events. Hence, depending on how many other events occur in parallel to an expanded event, the simulation time advances either in small time steps or in large chunks of time.

Typically, it is sufficiently accurate to predefine the actual length of an event duration at the time of modeling (e.g., for events of static runtime) or calculate it at runtime (e.g., the length of a received packet implies the time for decoding it). However, Horizon provides model developers with the means for explicitly controlling the length of an event duration at runtime if an extra level of accuracy is required. For instance, the execution time of an algorithm employed in a simulation event may depend on the number of loop iterations or conditional branches executed which need to be taken into account at runtime of the simulation. To handle such cases, Horizon allows to predefine a minimum event duration and to later adjust (i.e., extend) it at runtime. The event scheduler then directly incorporates any time adjustments by dynamically scheduling further events if the longer event duration results in new overlappings.

C. *Correctness of the Parallel Scheduling*

To guarantee the correctness of a parallel execution, Horizon needs to fulfill two requirements: i) the ordering of events must be deterministic and prevent causal violations and ii) the parallel execution of events must not cause data inconsistencies in the simulation model. In this section, we discuss how Horizon achieves these properties.

Event ordering: Horizon builds upon a centralized architecture consisting of a central scheduler and a single future event list (FEL) containing pending events. Within the FEL, events are ordered according to a total order "<" over their ts timestamps. Thus $e_1 < e_2$ holds iff $ts_{e_1} < ts_{e_2}$. The scheduler then drives the simulation by iteratively executing the first event from the FEL and/or fetching results from completed computations of previously offloaded events. Consequently, the global simulation time advances discretely between the corresponding ts and tc values of all scheduled events. Furthermore, an expanded event e represents an atomic operation whose effects become visible to other parts of the simulation only after its tc_e. Hence, the starting time $ts_{e'}$ of any new event e' created by the processing of e may only be equal or larger to tc_e. This guarantees that newly created events do not interfere with already offloaded events. As a result, Horizon preserves the correct ordering of events in increasing timestamp order (see Fig. 3).

Data consistency: State-of-the-art simulation models typically exhibit a component-based design (e.g., transmitters, physical channel, network protocols) according to best-practices in software engineering. Horizon exploits this fact by composing simulation models of extended components, called functional units. Similarly to the concept of logical processes in traditional parallelization, each functional unit maintains a private local state and interacts with neighboring functional units only via message passing [10]. To avoid data inconsistencies within the simulation model, the central scheduler of Horizon guarantees that only one event per functional unit is executed at any point in simulation time.

D. Discussion and Limitations

Horizon's architecture relies on a central scheduler and hence targets specifically shared-memory multi-processor systems. We believe that the increasing availability of multi-core computers renders such systems a cheap and valuable alternative to full-sized computing clusters for small to medium sized simulations. However, horizontal parallelization is orthogonal to existing distributed simulation schemes and can presumably be combined: In a hybrid approach, each partition of the distributed simulation model runs Horizon locally on a multi-processor cluster node, while the distributed simulation framework handles synchronization and communication across nodes. Thus, by utilizing existing parallel simulation mechanisms, Horizon transparently integrates with distributed computing clusters.

Moreover, an advantage of the centralized simulation architecture lies in its simple yet effective load balancing capabilities. In particular in wireless networks, node mobility can cause severe shifts in workload within the model when nodes move through the network or between different cells. By handling events from a central FEL and offloading them to a worker infrastructure, Horizon is not affected by these workload shifts and thus does not have to apply dedicated load balancing mechanisms.

The centralized event handling approach strongly benefits from events that exhibit a non-trivial processing complexity. By spending a considerable amount of runtime on their respective worker CPUs, such events easily amortize the overhead inflicted by the central event scheduling. As we will show in Section V-D, complex simulation models of wireless networks clearly exhibit this characteristic.

Extending discrete events with durations inevitably raises the question of how to obtain detailed timing information. We propose using existing and well understood techniques to calibrate simulation models with respect to timing. For instance, full system emulation [19], [20], [21], [22] employs detailed hardware models to exactly mimic the behavior of specific hardware platforms. It provides a highly accurate profiling tool for obtaining timing information that can be used to calibrate simulation models at the cost of increased development and implementation efforts. Additionally, recent research efforts [23], [24], [25] propose automatic calibration techniques for simulation models as a lightweight alternative to emulation. Such techniques automatically instrument simulation models with specific functionality to represent the (timing) behavior of a given hardware platform. Furthermore, communication protocols generally specify a number of timing constraints such as minimum and maximum timeout boundaries, waiting periods, or back-off durations. These properties provide a simple means for obtaining general timing information when no specific hardware platform is of interest. They also form the basis of well known approaches for maximizing the lookahead in distributed simulation [15], [16], [17]. Finally, domain experts may manually calibrate simulation models on the basis of their knowledge. This approach obviously demands a great amount of experience and careful judgment, but it also provides significant flexibility.

IV. Implementation

As an extension to the discrete event simulation paradigm, Horizon is generally applicable to any discrete event simulation framework. Our implementation of Horizon is based on OMNeT++ [6] – a simulation framework widely used in networking research. The highly modular design of simulation models in OMNeT++ closely ties into the concept of functional units and hence facilitates the adoption of Horizon.

Horizon implements the parallel execution of events by means of worker threads organized in a thread pool. Since OMNeT++ was not designed for multi-threaded execution, the primary implementation challenge is to ensure thread safety with minimum overhead. Fig. 4 presents an overview of the simulation architecture and illustrates the parallel event handling within Horizon. Among the typical core components of a simulation architecture is the central event scheduler along with a future event list (FEL). Horizon carefully extends this architecture by a thread pool, a work queue (WQ), and a so called offloaded event list (OEL).

To maintain a correct event execution order and avoid extensive locking, the central scheduler is the sole entity accessing the FEL. Once the scheduler determines that an event is safe for parallel execution, it removes the event from the FEL and enqueues it in the WQ. The WQ is shared among the scheduler and the worker threads which consume incoming events as soon as they become available. As a result, this work queue mechanism enables a simple yet effective load balancing because pending workload is equally distributed across CPUs. Additionally, by separating FEL and WQ, the scheduling policy of the WQ is not bound to that of the scheduler, but may be chosen flexibly (e.g., FIFO, EDF) to accommodate different workload characteristics. The OEL contains a set of meta-data for each offloaded event. Among this data is tc_e and a notification mechanism that indicates to the scheduler that the event was processed by a worker thread.

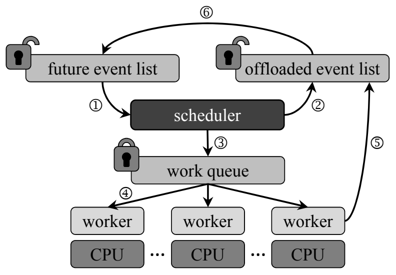


Fig. 4. Event data-flow in the simulation architecture: ① get next event, ② store information about offloaded event, ③ offload event to worker system, ④ execute events in parallel, ⑤ store results and new events, ⑥ update FEL with new events

Executing an event generally generates new events which need to be stored in the FEL accordingly. In Horizon however, events being generated during a threaded execution are not inserted in the FEL by the worker thread to avoid locking the FEL. Instead, newly generated events are temporarily stored in the meta-data of the corresponding offloaded event in the OEL. Once the scheduler gets notified about the completion of e at tc_e, it removes the new events from the entry in the OEL and inserts them in the FEL. Hence, the notification mechanism implicitly synchronizes the scheduler and the worker threads, thus eliminating the need for additional locking of the OEL.

The integration of Horizon in OMNeT++ is mostly transparent to existing models under the premise that functional units interact only via message passing, which is the default modeling paradigm in OMNeT++. For each functional unit, developers only need to provide the functionality for determining the duration of a given event when executed in the context of this particular unit. This allows for an effortless porting of existing models to Horizon.

V. Evaluation

As with every other parallelization scheme, the performance of Horizon heavily depends on the simulation model under investigation. Hence, the evaluation of Horizon follows three steps: We first derive a set of metrics that allow model developers to estimate the potential performance gain of Horizon for a given simulation model in order to decide whether or not the model is suited for horizontal parallelization. Then, we conduct synthetic benchmarks to investigate the performance of Horizon with regard to the number of CPUs, the workload, and the lookahead. Finally, we show that Horizon achieves significant speedup in realistic simulation models by means of a detailed simulation model of a cellular LTE network.

A. Performance Estimation

Horizon achieves parallelism by exploiting the overlapping of concurrent events in simulation time. Since the amount of overlapping events is a property of the simulation model, not every model is equally well suited for horizontal parallelization. Hence, it is important for model developers to estimate the applicability of Horizon to a specific model already before or early during development. In this section, we derive approximative metrics for doing so.

Let E be the set of all events that occur in a simulation model. Then, $\mathcal{I}(e)$ denotes the set of independent expanded events e' whose *starting times* $ts_{e'}$ overlap with a particular event e in simulation time

$$\mathcal{I}(e) = \{e' | ts_e \leq ts_{e'} \wedge ts_{e'} \leq tc_e\} \quad e, e' \in E. \tag{1}$$

Further, the number of concurrently to e offloadable events is

$$C(e) = |\mathcal{I}(e)| \quad e \in E, \tag{2}$$

and denotes that a maximum of $C(e)$ events can be executed in parallel (including e itself). Considering all events in a simulation model S, the average number of concurrent events is defined as

$$C(E) = \frac{\sum_{e \in E} |C(e)|}{|E|}. \tag{3}$$

$C(E)$ is a loose upper bound since the actual achievable parallelism of S is subject to the distribution of overlapping events throughout the simulation run and limited by the number of available CPUs and, as discussed in the next section, influenced by the processing complexity of the events. Nonetheless, Equation (3) allows for a rough estimation of how well a given simulation is suited for parallelization with Horizon. Additionally, $C_{max} = \max\{C(e) | e \in E\}$ is an upper bound for the number of CPUs that can be used to achieve a speedup. Having more than C_{max} CPUs does not further increase runtime performance as there are never enough concurrent events available to utilize all CPUs even during phases of maximum parallelism.

Obtaining exact values for $C(e)$ can be difficult because it requires tracing of the complete simulation run and determining $C(e)$ for each event in the simulation. Complex models and/or long simulation runtimes however generate large amounts of events which need to be recorded and analyzed. A faster online approach for finding an estimate of $C(E)$ is based on the notion of the event density [26]. The event density D denotes the number of events that occur in one simulated second during a simulation run. In combination with the average event duration $\overline{td}$, which can be determined by the model developer based on the calibrated model, it is possible to deduce the average number of overlapping events of a simulation model

$$\overline{\mathcal{I}}(E) = \overline{D} \cdot \overline{td}, \tag{4}$$

with $\overline{D}$ denoting the average event density. Obviously, high values of $\overline{\mathcal{I}}(E)$ indicate a good chance for achieving a speedup with Horizon. Concluding, despite being approximate, the presented metrics support model developers in assessing the applicability of Horizon to a given simulation model.

B. Benchmarking Methodology

Our practical evaluation of Horizon comprises two different classes of benchmarks, each targeting a unique evaluation goal. The first set of benchmarks (see Section V-C) aims at identifying the general performance characteristics and limitations of Horizon in terms of scheduling overhead, workload distribution capabilities, and lookahead size. The underlying benchmark model bases on a purely synthetic scenario designed for generating specific workload patterns. In contrast, the second set of benchmarks (see Section V-D) utilizes a concrete model of a cellular LTE network and acts as a case study to confirm the synthetic performance results.

Throughout this evaluation, we use two metrics to characterize the performance of Horizon: i) the *speedup*, which denotes the performance increase of a parallel simulation run over a sequential one, and ii) the *relative speedup*, which is a measure for the relative gain in performance when increasing the workload across parallel executions. While the speedup shows how a parallel architecture scales with a growing number of processing units, the relative speedup illustrates how well a parallel execution scheme can utilize a given workload. Given the runtimes for a sequential execution $t_{seq}(S)$ and a parallel execution $t_{par}(S)$ of a simulation model S, the speedup is defined as

$$\text{Speedup} = \frac{t_{seq}(S)}{t_{par}(S)}. \tag{5}$$

Accordingly, given the runtimes of two parallel executions of the models S and S' (of different workloads), the relative speedup is given by

$$\text{Relative Speedup} = \frac{t_{par}(S)}{t_{par}(S')}. \tag{6}$$

Our evaluation bases on a prototype implementation of Horizon which builds upon OMNeT++ 3.3. All performance results show average values collected over ten independent runs and the corresponding 95% confidence intervals. We utilized an AMD Opteron compute server providing 32GB of RAM and a total of 12 processing cores, organized in two six-core CPUs running a 64-bit Ubuntu 9.10 server OS.

C. Synthetic Benchmarks

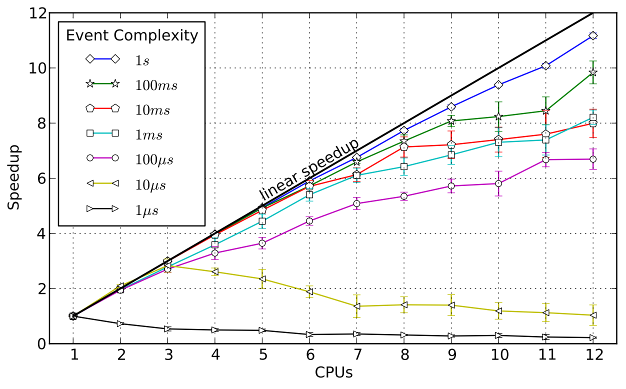
The synthetic benchmarks aim at characterizing the scalability of Horizon and identifying its limitations. Specifically, the benchmarks focus on the scalability in terms of the number of CPUs, the workload, and the size of the lookahead. By investigating a wide range of values for those parameters, the synthetic benchmarks span a parameter space that represents a large number of simulation models. Mapping concrete models into this space allows for deducing their potential for efficient parallelization with Horizon. The synthetic simulation model consists of a configurable number of workload-generating functional units. Each unit continuously creates events of a specific computational complexity and schedules them for execution. Furthermore, the synthetic model allows to define different workload patterns in terms of the number of events and their degree of parallelism $C(e)$. As a result, the event durations are chosen accordingly by the synthetic benchmark to yield the desired $C(e)$. Finally, all functional units are connected to each other via links with specific propagation delays, i.e., lookahead, thus forming a fully meshed network. A full mesh constitutes the worst case scenario in terms of synchronization, but represents real wireless networks in which every node is able to hear every other node.

Scalability w.r.t. CPUs: To assess the scalability of Horizon with respect to the number of CPUs, the synthetic benchmark model generates a static workload with $C(e) = 144$ and utilizes a varying number of CPUs. Choosing a large value for $C(e)$ guarantees that a sufficient amount of independent events exist in the model at any time to keep all CPUs busy. Moreover, the benchmark considers events of selected processing complexities, i.e., execution time on the worker CPU. The selected range of execution times reflects the result of profiling a wide range of publicly available simulation model frameworks [27], [28] and of our own models. All in all, the synthetic benchmark is designed to determine the runtime overhead imposed by the Horizon architecture and its impact on simulation performance under different levels of model complexity.

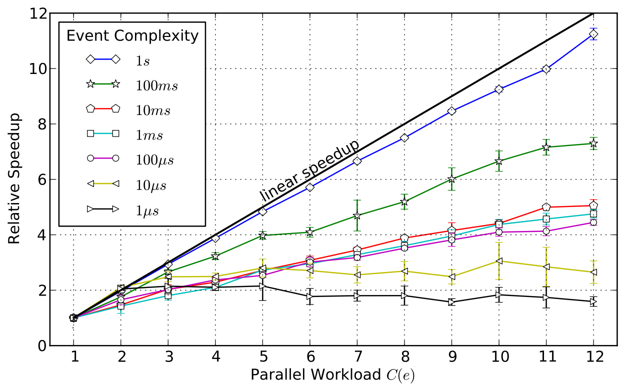
Fig. 5(a) shows the resulting performance gain for 1 to 12 CPUs and event processing times $p(e)$ from 1 microsecond to 1 second. The graph clearly illustrates a strong dependency of the speedup on the processing complexities of the events. For processing times of $100\mu s$ to $1s$, Horizon achieves a linearly increasing speedup with the number of CPUs. Furthermore, Horizon obtains a maximum speedup of 7 for events with $p(e) = 100\mu s$, and up to a speedup of 11 for events with $p(e) = 1s$. For events with $p(e) = 10\mu s$, peak performance is reached at three CPUs, while further increasing the number of CPUs causes a declining simulation performance. We accredit this to the overhead of the worker threads which increases with a growing number of CPUs due to contention of shared data structures. Finally, in case of events with $p(e) = 1\mu s$, the results indicate that the scheduling overhead outweighs the performance gain of parallelization, resulting in a performance degradation.

Scalability w.r.t. Workload: The complementary benchmark uses a fixed number of 12 CPUs while varying the workload between $C(e) = 1$ and 12. The purpose of this benchmark is to show how well Horizon scales with increasing model complexity and how well it realizes an even load distribution across CPUs given a static scheduling and contention overhead, i.e., a constant number of CPUs. Similarly to the previous benchmark, the event processing times range between $1\mu s$ and $1s$. The results presented in Fig. 5(b) are analogous to those previously discussed showing that Horizon obtains a significant speedup for events of longer processing duration ($100\mu s$ to $1s$). Furthermore, due to the static overhead, this benchmark still achieves a noticeable speedup of approx. 3 for events with $p(e) = 10\mu s$ and a speedup of 1.8 for events with $p(e) = 1\mu s$.

Scalability w.r.t. Lookahead Size: The last of the synthetic benchmarks aims at evaluating the impact of small lookaheads on parallel simulation performance. To this end, we compare Horizon with the traditional Null-Message Algorithm (NMA) [11] under different lookahead conditions. The benchmark generates events with uniformly distributed interarrival times



(a) Scalability in terms of CPUs over a static workload with $C(e) = 144$.



(b) Scalability in terms of workload over a static number of 12 CPUs.

Fig. 5. The results of the synthetic benchmarks show that Horizon scales well in terms of the number of CPUs (left) and the workload (right) for events of non-trivial complexity, i.e. $p(e) \geq 100\mu s$.

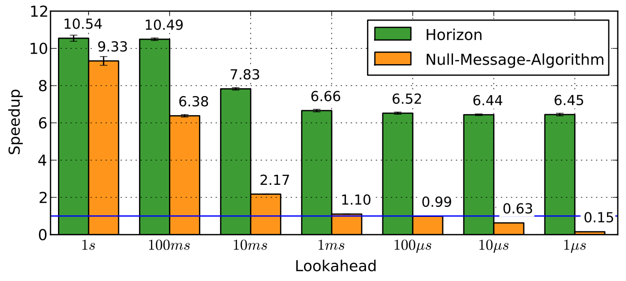


Fig. 6. Comparison of Horizon with the Null-Message Algorithm: Horizon delivers considerable speedups even for very small lookaheads while the performance of NMA rapidly declines with short lookaheads.

in the range of 0 to 1 seconds. In case of Horizon, this corresponds to uniformly distributed event durations and a back-to-back scheduling of events. Furthermore, we investigate lookaheads of $1s$ to $1\mu s$ in a fully meshed network of 100 benchmarking units. A full mesh typically constitutes a worst case scenario for parallel simulation schemes. Additionally, the benchmark considers static event complexities of $p(e) = 0.1s$ and employs 12 threads/NMA-partitions. Lastly, the benchmark runs the NMA implementation of OMNeT++ over MPI (mpich2 v1.2.1).

Fig. 6 shows that for a lookahead of $1s$, Horizon and the NMA deliver roughly similar speedups of approx. 10. This is due to the fact that this lookahead is larger than the maximum event interarrival time, thus providing good knowledge about future events for both parallelization schemes. However, for smaller lookaheads of $100ms$ to $1ms$, the NMA needs to perform increasingly more synchronization operations and is less able to identify independent events for parallel execution. Consequently, its performance degrades rapidly and at a lookahead of $1ms$ – which is still large for wireless networks – almost no performance gain is obtained. Finally, for lookaheads ranging from $100\mu s$ down to $1\mu s$, the NMA effectively requires longer runtimes than the sequential execution. In contrast, Horizon still achieves a significant speedup even for small lookaheads.

We conclude from the synthetic benchmarks that Horizon i) benefits from non-trivial processing complexities, ii) is able to equally distribute workload across worker CPUs, and iii) achieves considerably better performance than the NMA in networks with small lookaheads.

D. LTE Network Model Benchmarks

In order to underline Horizon's applicability to real simulation models, this section discusses a case study of a complex simulation model of an LTE network.

Model Description: The simulation used for this evaluation models a system compliant to 3GPP-LTE [29]. The system features c cells, each containing a base station (BS) serving m mobile stations (MS). The simulation time is slotted into downlink and uplink frames called transmission time intervals (TTI) with a duration of $1ms$ each. In each cell, arriving data packets destined for different terminals are queued separately at the base station. Then, prior to each downlink TTI, the base station schedules the available packets to be transmitted during the next downlink TTI with respect to a specific optimization goal, e.g., to minimize delay. Subsequently, a resource allocation unit tries to fulfill the scheduled transmission requests by assigning system resources to terminals accordingly. To achieve an optimal assignment in each downlink TTI, the allocation unit needs to solve an optimization problem which is known to have a considerable computational complexity. The formulation of this optimization problem along with more detailed descriptions can be found in [30].

On the physical layer, the system uses OFDMA as its transmission scheme with n resource blocks. Each resource block consists of 12 subcarriers, equivalent to a frequency width of 180 kHz. For each terminal/resource block pair, the channel gain varies randomly over time and frequency, i.e., it depends on a deterministic component (path loss) and a random, time- and frequency-variant fading component. We assume this gain to be exponentially distributed based on a multi-path propagation environment with no dominant path. A summary of the key system parameters is shown in Table I.

The model uses an inter-site distance of 500m, resulting in a maximum propagation delay between the mobile stations

TABLE I
CONFIGURATION PARAMETERS OF THE LTE SIMULATION MODEL.

Parameters	**Value**
Carrier frequency	2 GHz
Channel bandwidth	10 MHz
Mode	TDD
Number of resource blocks	55
Subcarriers per resource block	12
Subcarrier spacing	15 kHz
Scenario	SISO
Inter Site Distance	500 m
Fading model	Rayleigh

and the base station of merely 0.8μs. Thus, deriving the lookahead from the propagation delay yields a significantly small value and hence a tight coupling among the simulated nodes within the model. Instead, for use with Horizon, the model is time calibrated according to the protocol specification based technique discussed in Section III-D and pioneered by Liu and Nicol [15]. Specifically, building on the knowledge of the length of a TTI, this technique assigns durations to simulation events such that the durations of all events within a TTI add up to 1ms.

Results: We first analyze the workload characteristics of the LTE model to establish a foundation for understanding the performance results and to put them into perspective with respect to the findings of the previous section. Fig. 7 shows a histogram illustrating the distribution of events exhibiting a certain event processing complexity in the model. The majority of events comprise a processing complexity of $1\mu s$ to $100\mu s$ seconds (note the double logarithmic scale). These events account for simple functionality such as traffic generation, queue management, etc. Moreover, the figure shows that a large number of events are of considerable complexity, ranging from approx. $400\mu s$ to $4s$ (as indicated by the labeled peaks). The second graph in the figure, a CDF over the total runtime, illustrates that the events with processing complexities larger than $400\mu s$ contribute almost exclusively to the total runtime of the simulation model, hence making them a primary target for Horizon's offloading scheme (again, note the logarithmic scale). Summarizing, the LTE model exhibits the two key properties of modern wireless simulation models: i) a small propagation delay (i.e., lookahead) and ii) computationally complex modeling of physical layer effects, making it a well-suited use-case scenario for Horizon.

Fig. 8(a) shows the speedup for one to twelve CPUs given LTE networks of 12 and 30 cells, each serving 30, 50, or 70 mobile stations per cell respectively. In these scenarios, Horizon achieves a maximum speedup of 4.2 (12 cells, 30 MS/cell) up to 5.6 (12 cells, 70 MS/cell). Hence, despite the fact that the majority of events is of relatively small complexity, Horizon is able to exploit the computationally complex events to arrive at a noticeable speedup. Furthermore, we accredit the convergence of all scenarios at a speedup of approx. 5 to the overwhelming number of small events. Nevertheless, a close examination of the results shows that for

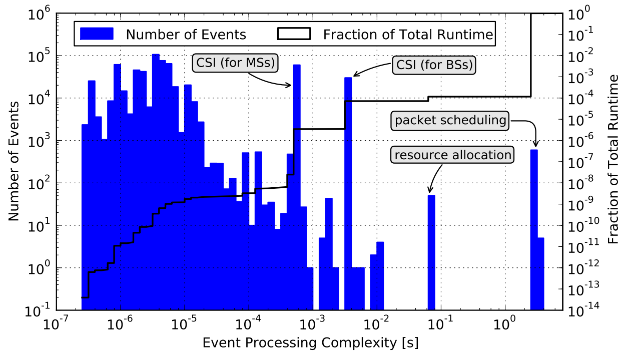


Fig. 7. Distribution of event processing complexities in the LTE model (12 cells, 50 MS/cell) and the fraction of the total simulation runtime according to those complexities (black CDF). The figure also indicates the four most complex types of events in the model. Note the logarithmic scales on all axes.

both basic network topologies (12 and 30 cells), the scenarios considering more mobile stations per cell still generate a higher speedup.

In Fig. 8(b), the relative speedup for networks of one to twelve cells and 30, 50, and 70 MS/cell is shown. Similarly to the previous results, the relative speedup ranges between 4.6 (30 MS/cell) and 6.5 (70 MS/cell). Additionally, the graphs show a steady performance increase, as the number of simulated cells grows. Both observations indicate that Horizon indeed scales with increasing workload and is able to distribute the additional load across the available processing units.

In conclusion, this case study shows the viability of horizontal parallelization and the successful application of Horizon to complex wireless network simulation models.

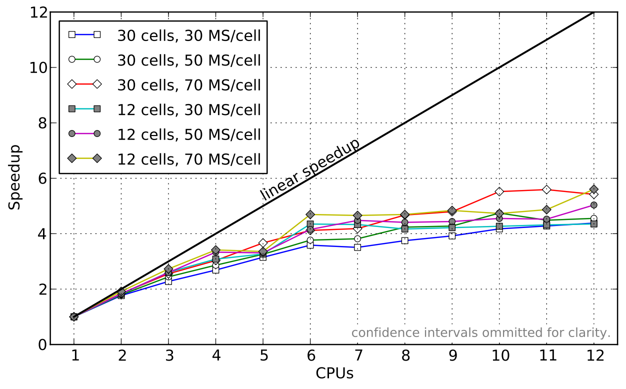
VI. RELATED WORK

This section discusses closely related approaches to parallel network simulation that are not covered by the general introduction in Section II.

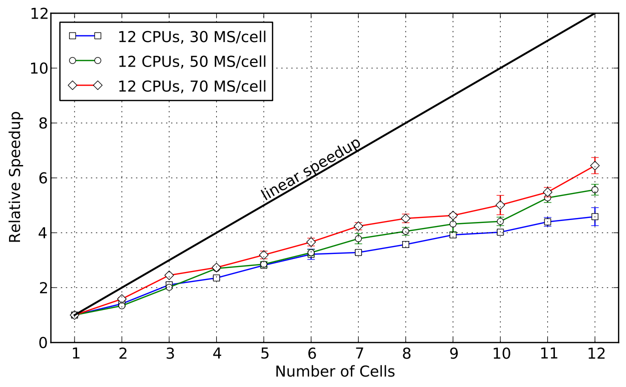
A. Enhanced Time Information

Previous efforts in network simulation research extend classic discrete event simulation with additional timing information to enhance simulation performance and scalability.

Lubachevsky [31] pioneered techniques for increasing the lookahead by introducing so called opaque periods and minimum propagation delays. During an opaque period, a simulated entity does not interact with its neighboring entities, hence allowing those to process further events. In [32], Fujimoto replaces the accurate timestamps of discrete events with time intervals representing a period of simulation time in which an event can occur. This approach is based on the observation that events do not necessarily occur at a fixed point in simulation time resulting in a certain event ordering, but the actual time of occurrence and hence the event ordering is subject to uncertainty. In terms of performance enhancements, these time intervals allow to increase the lookahead within distributed simulation models. Loper [33], [34] extends this concept by choosing discrete timestamps from these time



(a) Speedup for networks of 12 and 30 cells with 30, 50, and 70 MS/cell.



(b) Relative speedup for a network of 12 cells with 30, 50, and 70 MS/cell.

Fig. 8. Horizon achieves speedups of up to 6 for the LTE network model and thus underlines its applicability to complex real-world simulation models.

intervals according to a random distribution and reports a considerable speedup for the PHOLD benchmark [35] and queueing network simulations. However, utilizing time intervals introduces inaccuracies in the simulation results and influences the determinism and repeatability of simulations. In contrast, Horizon uses time intervals to actually model the simulated duration of an event at deterministic points in simulation time and deduces independent events from this.

Peschlow picks up the idea of uncertainty intervals [36] and investigates the influence of different event orderings resulting from overlapping intervals on simulation results. To avoid repeated simulation runs for each ordering, a single simulation run is branched for each different event ordering. For a simple airline simulation, the authors report a speedup of up to 10 on a four-processor computer for the branching approach in comparison to the time needed to conduct the corresponding number of single simulation runs. However, the branching approach suffers from a state explosion problem and is hence not well applicable to complex wireless network models. Previous work in this area [37] follows a similar branching approach, yet considers discrete events with concrete timestamps instead of uncertainty intervals. Even in this simpler scenario, branching delivers considerable speedup compared to repeated simulation runs, but the approach suffers from the same state explosion problems, making it less applicable to larger network simulations.

B. Multi-Threaded Architecture

The recent proliferation of multiprocessor systems sparked research efforts that explicitly trade the scalability of computing clusters for shared memory simulation architectures. In the context of the ns-3 network simulator project [38], a multi-threading extension of the core simulator architecture was developed [39]. It utilizes a conservative barrier based synchronization algorithm which relies on link delays across nodes for calculating barrier points in simulation time. However, in contrast to Horizon, this extension does not yet support wireless networks.

A similar approach [40] specifically focuses on wireless IEEE 802.11 networks. It also utilizes a barrier-based synchronization mechanism where each barrier in simulation time is calculated based on protocol and event lookahead. Both techniques intent to extract lookahead information from either the protocol specification or certain hardware characteristics such as the RxTxTurnaround time specified in IEEE 802.11. In conjunction with an additional event bundling mechanism, the authors report a linear speedup and for large network sizes of up to 2000 node even super-linear speedup.

Finally, Lynch [41] proposes a dedicated hardware unit that maps the global state information of conservative synchronization algorithms to specialized hardware registers on multi-processor platforms to provide fast concurrent access. In comparison to software/Horizon, hardware based approaches provide unchallenged speed, but often lack flexibility as the hardware is specifically designed for a particular combination of processors and synchronization algorithm.

VII. Conclusion and Future Work

In this paper, we present Horizon, a lightweight *horizontal parallelization* scheme for multi-processor hardware. By explicitly modeling the delay of simulated processes through event durations, Horizon is able to derive extended lookahead information from simulation models to identify independent events for parallel execution. Horizon particularly targets wireless simulation models that exhibit extremely small lookaheads and thus perform poorly with existing parallelization schemes. Our evaluation analyzes the performance characteristics of Horizon by means of synthetic benchmarks and shows that Horizon achieves a significant speedup over sequential executions. Moreover, a case study involving a complex simulation model of an LTE network illustrates Horizon's applicability to real-world scenarios.

Future research efforts focus primarily on the event scheduler. Being a centralized component results in an interesting trade-off between a limited scalability and the availability of global knowledge of the simulation model. We intent to investigate online and offline optimizations that exploit this global knowledge in order to fine-tune offloading decisions. Additionally, further research addresses the question of how to carefully combine Horizon with classic distributed simulation. In such hybrid simulation architecture, each logical process internally handles events according to event durations while utilizing classic synchronization algorithms across logical processes and partitions for enhanced scalability. Hence, by coupling the advantages of both worlds, this approach seems particularly promising for computing clusters with multi-processor nodes.

Acknowledgments

This research was funded in part by the DFG Cluster of Excellence on Ultra High-Speed Mobile Information and Communication (UMIC), German Research Foundation grant DFG EXC 89.

References

[1] J. Heidemann, N. Bulusu, J. Elson, C. Intanagonwiwat, K. Chan Lan, Y. Xu, W. Ye, D. Estrin, and R. Govindan, "Effects of Detail in Wireless Network Simulation," in *Proceedings of the SCS Multiconference on Distributed Simulation*, January 2001, pp. 3–11.

[2] Z. Ji, J. Zhou, M. Takai, and R. Bagrodia, "Improving Scalability of Wireless Network Simulation with Bounded Inaccuracies," *ACM Transactions on Modeling Computer Simulation*, vol. 16, no. 4, 2006.

[3] M. Takai, J. Martin, and R. Bagrodia, "Effects of Wireless Physical Layer Modeling in Mobile Ad-hoc Networks," in *Proc. of the 2nd ACM Intern. Symposium on Mobile ad-hoc Networking & Computing*, 2001.

[4] J. Cowie, A. Ogielski, and D. Nicol, "The SSFNet Network Simulator," Software on-line: http://www.ssfnet.org/homePage.html, 2002.

[5] G. Chen and B. K. Szymanski, "DSIM: Scaling Time Warp to 1,033 Processors," in *Proceedings of the 37th Winter Simulation Conference (WSC '05)*, 2005, pp. 346–355.

[6] A. Varga, "The OMNeT++ Discrete Event Simulation System," in *Proc. of the European Simulation Multiconference (ESM)*, June 2001.

[7] "PDNS - Parallel/Distributed NS," http://www.cc.gatech.edu/computing/compass/pdns/.

[8] G. F. Riley, "The Georgia Tech Network Simulator," in *Proceedings of the ACM SIGCOMM Workshop on Models, Methods and Tools for Reproducible Network Research (MoMeTools '03)*, 2003, pp. 5–12.

[9] R. M. Fujimoto, K. Perumalla, A. Park, H. Wu, M. H. Ammar, and G. F. Riley, "Large-Scale Network Simulation: How Big? How Fast?" in *Proc. of International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, October 2003, pp. 116–123.

[10] R. M. Fujimoto, "Parallel Discrete Event Simulation," *Communications of the ACM*, vol. 33, no. 10, pp. 30–53, 1990.

[11] K. M. Chandy and J. Misra, "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs," *IEEE Transactions on Software Engineering*, vol. SE-5, no. 5, pp. 440–452, September 1979.

[12] V. Jha and R. L. Bagrodia, "Transparent Implementation of Conservative Algorithms in Parallel Simulation Languages," in *Proceedings of the 25th Winter Simulation Conference (WSC '93)*, 1993, pp. 677–686.

[13] K. S. Perumalla, "Parallel and Distributed Simulation: Traditional Techniques and Recent Advances," in *Proceedings of the 38th Winter Simulation Conference (WSC '06)*, 2006, pp. 84–95.

[14] R. L. Bagrodia and M. Takai, "Performance Evaluation of Conservative Algorithms in Parallel Simulation Languages," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 4, pp. 395–411, 2002.

[15] J. Liu and D. M. Nicol, "Lookahead Revisited in Wireless Network Simulations," in *Proceedings of the 16th Workshop on Parallel and Distributed Simulation (PADS '02)*, 2002, pp. 79–88.

[16] 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*, July 1998, pp. 262–267.

[17] ——, "Path Lookahead: A Data Flow View of PDES Models," in *Proc. of the 13th Workshop on Parallel and Distributed Simulation*, 1999, pp. 12–19.

[18] D. R. Jefferson, "Virtual Time," *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 3, pp. 404–425, July 1985.

[19] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A Full System Simulation Platform," *Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[20] B. Titzer, D. Lee, and J. Palsberg, "Avrora: Scalable Sensor Network Simulation with Precise Timing," in *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, 2005, p. 67.

[21] F. Bellard, "QEMU, a fast and Portable Dynamic Translator," in *USENIX Annual Technical Conference*, 2005.

[22] K. Lawton, "Bochs: A Portable PC Emulator for Unix/X," *Linux Journal*, vol. 1996, no. 29, p. 7, 1996.

[23] T. Kempf, M. Doerper, R. Leupers, G. Ascheid, H. Meyr, T. Kogel, and B. Vanthournout, "A Modular Simulation Framework for Spatial and Temporal Task Mapping onto Multi-processor SoC Platforms," in *Proc. of the Conference on Design, Automation and Test in Europe*, 2005.

[24] V. Shnayder, M. Hempstead, B. rong Chen, G. W. Allen, and M. Welsh, "Simulating the Power Consumption of Large-scale Sensor Network Applications," in *Proc. of the 2nd Intern. Conference on Embedded Networked Sensor Systems*, 2004.

[25] O. Landsiedel, H. Alizai, and K. Wehrle, "When Timing Matters: Enabling Time Accurate and Scalable Simulation of Sensor Network Applications," in *Proc. of the 2008 International Conference on Information Processing in Sensor Networks (IPSN 2008)*, 2008.

[26] A. Varga, Y. Sekercioglu, and G. Egan, "A Practical Efficiency Criterion for the Null Message Algorithm," in *Proceedings of the European Simulation Symposium (ESS 2003)*, 2003, pp. 26–29.

[27] "Mobility Framework," http://mobility-fw.sourceforge.net/.

[28] A. Köpke, M. Swigulski, K. Wessel, D. Willkomm, P. Haneveld, T. Parker, O. Visser, H. Lichte, and S. Valentin, "Simulating Wireless and Mobile Networks in OMNeT++ – The MiXiM Vision," in *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, 2008, p. 71.

[29] 3rd Generation Partnership Project. Technical Specification Group Radio Access Network, *Physical layer aspects for evolved Universal Terrestrial Radio Access (UTRA)(Release 7), 3GPP TR 25.814 V7.1.0*, 2006.

[30] M. Bohge, J. Gross, M. Meyer, and A. Wolisz, "Dynamic Resource Allocation in OFDM Systems: An Overview of Cross-Layer Optimization Principles and Techniques," *IEEE Network Magazine, Special Issue: "Evolution toward 4G Wireless Networking"*, vol. 21, no. 1, 2007.

[31] B. D. Lubachevsky, "Efficient Distributed Event Driven Simulations of Multiple-loop Networks," in *Proc. of the 1988 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 1988.

[32] R. M. Fujimoto, "Exploiting Temporal Uncertainty in Parallel and Distributed Simulations," in *Proc. of the 13th Workshop on Parallel and Distributed Simulation (PADS '99)*, 1999, pp. 46–53.

[33] M. L. Loper and R. M. Fujimoto, "A Case Study in Exploiting Temporal Uncertainty in Parallel Simulations," in *Proceedings of the 2004 International Conference on Parallel Processing (ICPP '04)*, 2004, pp. 161–168.

[34] ——, "Pre-sampling as an Approach for Exploiting Temporal Uncertainty," in *Proceedings of the 14th Workshop on Parallel and Distributed Simulation (PADS '00)*, 2000, pp. 157–164.

[35] R. Fujimoto, "Performance of Time Warp under Synthetic Workloads," *Proceedings of the SCS Multiconference on Distributed Simulation*, vol. 22, no. 1, pp. 23–28, 1990.

[36] P. Peschlow, P. Martini, and J. Liu, "Interval Branching," in *Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation (PADS '08)*, 2008, pp. 99–108.

[37] P. Peschlow and P. Martini, "Efficient Analysis of Simultaneous Events in Distributed Simulation," in *Proceedings of the 11th IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT '07)*, 2007, pp. 244–251.

[38] T. R. Henderson, S. Roy, S. Floyd, and G. F. Riley, "ns-3 Project Goals," in *Proceeding of the 2006 Workshop on ns-2: The IP Network Simulator*, 2006, p. 13.

[39] G. Seguin, "Multi-core Parallelism for ns-3 Simulator," INRIA Sophia-Antipolis, Tech. Rep., 2009.

[40] P. Peschlow, A. Voss, and P. Martini, "Good News for Parallel Wireless Network Simulations," in *Proceedings of the 12th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM '09)*, 2009, pp. 134–142.

[41] E. W. Lynch and G. F. Riley, "Hardware Supported Time Synchronization in Multi-core Architectures," in *Proceedings of the 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation (PADS '09)*, 2009, pp. 88–94.