Parallel Expanded Event Simulation of Tightly Coupled Systems

GEORG KUNZ, RWTH Aachen University
MIRKO STOFFERS, RWTH Aachen University
OLAF LANDSIEDEL, Chalmers University of Technology
KLAUS WEHRLE, RWTH Aachen University
JAMES GROSS, KTH Royal Institute of Technology

The technical evolution of wireless communication technology and the need for accurately modeling these increasingly complex systems causes a steady growth in the complexity of simulation models. At the same time, multi-core systems have become the de facto standard hardware platform. Unfortunately, wireless systems pose a particular challenge for parallel execution due to a *tight coupling* of network entities in space and time. Moreover, model developers are often domain experts with no in-depth understanding of parallel and distributed simulation. In combination, both aspects severely limit the performance and the efficiency of existing parallelization techniques.

We address these challenges by presenting *parallel expanded event simulation*, a novel modeling paradigm that extends discrete events with durations which span a period in simulated time. The resulting *expanded events* form the basis for a conservative synchronization scheme that considers overlapping expanded events eligible for parallel processing. We furthermore put these concepts into practice by implementing HORIZON, a parallel expanded event simulation framework specifically tailored to the characteristics of multi-core systems. Our evaluation shows that HORIZON achieves considerable speedups in synthetic as well as real-world simulation models and considerably outperforms the current state-of-the-art in distributed simulation.

CCS Concepts: •**Computing methodologies** → **Discrete-event simulation; Massively parallel and high-performance simulations;**

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Parallel discrete event simulation, Multi-core Systems, Wireless Systems, Simulation Modeling Paradigm, Conservative Synchronization

ACM Reference Format:
Georg Kunz, Mirko Stoffers, Olaf Landsiedel, Klaus Wehrle, and James Gross, 2013. Parallel Expanded Event Simulation of Tightly Coupled Systems. *ACM Trans. Model. Comput. Simul.* V, N, Article XX (2015), 25 pages.
DOI: http://dx.doi.org/10.1145/2832909

1. INTRODUCTION

Discrete event-based simulation of wireless networks currently faces two significant changes: First, recent advances in wireless communication technology demand highly

This research was funded by the DFG Cluster of Excellence on Ultra High-Speed Mobile Information and Communication (UMIC), German Research Foundation grant DFG EXC 89.
Author's addresses: G. Kunz, Chair for Communication and Distributed Systems, RWTH Aachen University; M. Stoffers, Chair for Communication and Distributed Systems, RWTH Aachen University; O. Landsiedel, Computer Science and Engineering, Chalmers University of Technology; K. Wehrle, Chair for Communication and Distributed Systems, RWTH Aachen University; J. Gross, School of Electrical Engineering, KTH Royal Institute of Technology.
 1049-3301/2015/-ARTXX $15.00
DOI: http://dx.doi.org/10.1145/2832909

accurate simulation models, as their interaction with the propagation channel as well as with other transceivers becomes increasingly hard to characterize mathematically. Example technologies to which this applies are multi-carrier transmission schemes [Koffman et al. 2002], multiple-input multiple-output systems [Gesbert et al. 2003] as well as techniques such as successive interference cancellation [Halperin et al. 2008] or interference alignment [Cadambe and Jafar 2008]. All these technologies are either already implemented in wireless networking standards or will become part of future standards soon. However, the simulation of these complex technologies leads to a steep increase in model complexity and runtime requirements. Unfortunately, this increasing model complexity cannot be met by increasing processing power any more, at least in the traditional sense. As excessive power dissipation has set an end to improving CPU performance through higher clock speeds, chip vendors invest increasing transistor counts in duplicate functionality while clock speeds remain relatively stable. Consequently, the performance of sequential programs, e. g., discrete event simulations, does not improve significantly with new generations of CPUs. Hence, simulations must employ parallel event execution to exploit multi-core systems.

However, parallel simulation especially of wireless systems is hard [Liu and Nicol 2002]. Traditionally, the primary application of parallel network simulation is large-scale models of wired networks [Fujimoto et al. 2003]. Yet, the proliferation of wireless technology has shifted the focus of interest in the research community from wired to wireless networks. Three main properties distinguish wireless from wired systems: i) the wireless channel is a broadcast domain; ii) wireless networks are generally of smaller size and iii) wireless systems have a much more complex interaction with the physical medium, i. e., the propagation channel. In terms of parallel simulation, these properties have significant implications. First of all, models of wireless networks exhibit substantially smaller lookaheads than models of wired networks [Liu and Nicol 2002]. The source for lookaheads in network simulation is typically the propagation delay along links between simulated entities. For this reason, parallel discrete event simulation divides large-scale wired networks, e. g., the Internet, along long-haul backbone links exhibiting usually delays in the range of tens of milliseconds [Markopoulou et al. 2006]. In wireless networks, however, links just range from tens of meters (e. g., Bluetooth) up to a few kilometers (e. g., Global System for Mobile Communications (GSM)). Therefore, propagation delays span merely nanoseconds to low microseconds, resulting in extremely small lookaheads. Furthermore, as the wireless channel is a broadcast domain, wireless networks comprise a highly connected topology. This in turn hinders partitioning of the model as the network cannot easily be divided in loosely connected clusters of network nodes. Instead, each Logical Process (LP) handling a partition is connected to many neighboring LPs, thereby increasing the synchronization overhead in conservative synchronization or the risk for receiving a straggler event in optimistic synchronization. Finally, the above two implications are aggravated by the fact that accurate physical layer models of wireless systems usually lead to computationally complex events. Hence, wireless networks are a good representative of *tightly coupled* systems, where – in this special case – the event complexities are unusually high (for typical network simulations).

Historically, the research community dedicated considerable efforts to investigate the feasibility and scalability of parallel simulation [Fujimoto 1990a; Liu 2009; Nicol 1996; Perumalla 2006] in general, thereby laying the foundation for parallel simulation frameworks [Chen and Szymanski 2005; Cowie et al. 2002; Riley 2003; Varga 2001; Bononi et al. 2006]. The primary focus of many of these works is on *distributed simulation* on computing clusters, which matches very well the exact opposite of tightly-coupled system models, i. e., large-scale models of wired networks with low connectivity, bigger lookaheads and rather low event complexities. However, as the tight cou-

pling significantly increases the synchronization overhead and limits the number of parallelizable events, these approaches do not scale very well anymore, partially due to the model characteristics but also due to the usage of compute clusters. To mitigate some of these problems, the research community devised lookahead extraction techniques that take the topology of the simulation model into account [Liu and Nicol 2002; Meyer and Bagrodia 1998]. However, these techniques require extensive manual effort and impose considerable runtime overhead. Therefore, the question for better simulation approaches of tightly-coupled systems like wireless networks still remains open.

In this paper, we contribute by presenting and evaluating such a new approach. We claim that today's approach of modeling time spans in discrete event simulation hides information about event dependencies. Processes, such as switching delays of the hardware, the computing time of algorithms, as well as the signal propagation generally span non-zero periods of wall-clock time. In contrast, the modeling paradigm in discrete event simulation defines that events occur at specific points in simulated time as defined by their timestamp, yet handling an event does not advance the simulated time beyond the timestamp. Consequently, a single event by itself cannot represent a period of simulated time. A typical approach to modeling time spans uses two separate events which indicate the beginning and the end of a physical process. However, this method of using two separate events to model the elapse in time hides dependency information that is valuable for parallel simulation. Based on these observations [Kunz et al. 2010; Kunz 2013], we make three contributions:

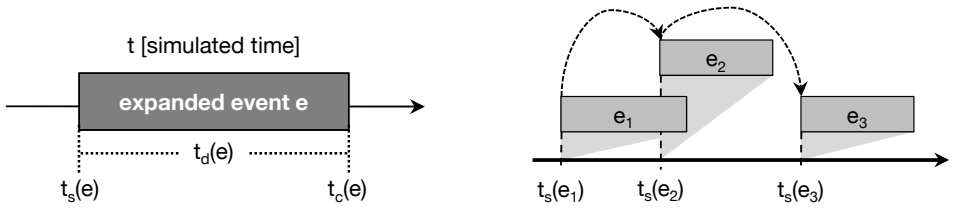
i) We develop *expanded event simulation*, a modeling paradigm that extends events with durations to eliminate the modeling mismatch between physical processes and discrete event simulation.

ii) We define a *parallelization scheme* that exploits the event dependency information provided by expanded event simulation to efficiently execute simulation models of *tightly coupled systems*.

iii) We tailor the parallelization scheme specifically to the *characteristic properties of multi-core systems* in order to fully utilize their processing power.

The last contribution is mainly motivated by the wide-spread availability of multi-core systems as well as the expected further increase in terms of available cores per CPU (as multi-core computers exhibit different hardware characteristics than distributed computing clusters, simply applying parallel simulation techniques designed for distributed simulation to multi-core computers does not achieve the best possible simulation performance). Apart from presenting our novel approach, we show by means of experimentation that our approach outperforms state-of-the-art approaches by two orders of magnitude in synthetic benchmarks, while it achieves a speedup of up to 6 for 11 worker threads for realistic simulation models of wireless networks. These performance improvements relate to tightly-coupled systems HORIZON is designed for.

In comparison to our previous publications [Kunz et al. 2009; Kunz et al. 2010], this paper contributes in several significant ways. First, this paper puts a stronger focus on the formal description of the concepts of parallel expanded event simulation and elaborates on our design decisions. Secondly, all presented performance results are based on an improved simulation engine. Thirdly, we stress in this paper much more the application domain of HORIZON with respect to tightly-coupled systems. In this respect, we initially compare HORIZON to the state-of-the-art simulation frameworks PRIME [Liu et al. 2009] and OMNeT++ and determine by means of a synthetic benchmark and a theoretical performance analysis under which conditions parallel expanded event simulation outperforms existing approaches. Additionally, while we have updated the performance results with respect to the 3GPP Long Term Evolution (LTE) case study, we



(a) An expanded event e spans a period of simulated time ranging from its starting time $t_{\mathrm{s}}(\mathsf{e})$ to its completion time $t_{\mathrm{c}}(\mathsf{e})$. The period between $t_{\mathrm{s}}(\mathsf{e})$ and $t_{\mathrm{c}}(\mathsf{e})$ is the event duration $t_{\mathrm{d}}(\mathsf{e})$.

(b) Our approach: Advancing the simulated time according to the starting times of the events.

Fig. 1. Basics of expanded event simulation.

present results of a second case study, which investigates the performance of HORIZON in the context of a wireless mesh network. This underlines the suitability of HORIZON for wireless system simulations either with regular or randomized timing behavior.

The remainder of this paper is structured as follows: Section 2 presents parallel expanded event simulation, a novel modeling paradigm addressing the previously identified challenges. Section 3 then introduces HORIZON [Kunz et al. 2010], our simulation framework that puts parallel expanded event simulation into practice. In Section 4, we present a high-level criterion for when our parallelization scheme outperforms classical approaches, followed by an in-depth performance evaluation of HORIZONin Section 5. Finally, we discuss the limitations of our contributions in Section 6, review related efforts in Section 7, and conclude the paper in Section 8.

2. EXPANDED EVENT SIMULATION

The core idea of our approach is to augment simulation models with additional domain specific information to support synchronization algorithms in identifying independent events. We first specify the novel modeling paradigm that explicitly augments events with time spans, referring to them as *expanded events* (see Figure 1(a)). Consequently, we denote a simulation implementing this paradigm an *expanded event simulation*.

Definition 2.1 (*Expanded Event*). An expanded event is defined by a distinct *starting time* and a distinct *completion time*. We refer to the difference in simulated time between start and completion time as *event duration*:

- $t_{\mathrm{s}} : \mathcal{E} \to T, \mathsf{e} \mapsto t$ maps an event e to its *starting time* in simulated time T,
- $t_{\mathrm{c}} : \mathcal{E} \to T, \mathsf{e} \mapsto t$ maps an event e to its *completion time* in simulated time T,
- $t_{\mathrm{d}} : \mathcal{E} \to T, \mathsf{e} \mapsto t$ maps an event e to its *duration* in simulated time T,

where $\mathcal{E}$ denotes the set of all events that occur in a simulation and T the simulated time. The event duration $t_{\mathrm{d}}(\mathsf{e})$ of an expanded event e is $t_{\mathrm{d}}(\mathsf{e}) = t_{\mathrm{c}}(\mathsf{e}) - t_{\mathrm{s}}(\mathsf{e}) \geq 0$. We explicitly allow $t_{\mathrm{s}}(\mathsf{e}) = t_{\mathrm{c}}(\mathsf{e})$ in order to represent traditional *discrete events* due to backwards compatibility and in order to enable the propagation of meta-data via side channels. Given an event duration, we furthermore define a restriction on the starting time of newly created events. To this end, we first define a successor relationship among events.

Definition 2.2 (*Successor Event*). Event e_2 is a successor of e_1 if and only if e_1 creates e_2, denoted by $e_1 \curvearrowright e_2$.

Based on the successor relationship, we restrict the starting time of successor events:

Definition 2.3 (*Starting Time of Successor Events*). For all expanded events e_1, e_2 with $e_1 \curvearrowright e_2$ holds $t_\mathrm{c}(e_1) \leq t_\mathrm{s}(e_2)$.

This means that new events may only start *after* the event (i. e., the physical process) that creates them, has finished. Hence, results of an expanded event may only become visible to the entire system after the event has been processed.

We next define a *sequential* execution model for expanded event simulation. Specifically, we address three questions: i) How to integrate event durations in the modeling process? ii) How to advance the global simulation time when executing an expanded event? iii) How to sort expanded events in the Future Event Set (FES)?

Modeling Event Durations. From a modeling perspective, the duration of an expanded event is not necessarily static. Much like the processing duration of physical processes, it can depend on dynamic inputs, e. g., the varying length of packets. Thus, we integrate dynamic event durations such that right before event execution the duration can dynamically be determined by the model. Note that in the remainder of this paper, the terms $t_\mathrm{c}(\mathsf{e})$ and $t_\mathrm{d}(\mathsf{e})$ for an expanded event $\mathsf{e} \in \mathcal{E}$ always refer to the *final* completion time and event duration, i. e., the final values after dynamic extension, unless stated otherwise.

Advancing the Global Simulated Time. In expanded event simulation, an event $\mathsf{e} \in \mathcal{E}$ carries two timestamps, $t_\mathrm{s}(\mathsf{e})$ and $t_\mathrm{c}(\mathsf{e})$, and spans a period of simulated time. Hence, we need to define how to advance the simulated time when executing expanded events:

Before executing an expanded event e, the event scheduler sets the global simulation time to $t_\mathrm{s}(\mathsf{e})$. Furthermore, the global time remains constant at $t_\mathrm{s}(\mathsf{e})$ throughout the entire wall-clock processing time of e. Thus, despite spanning a period of simulated time, the global virtual clock does not explicitly advance from $t_\mathrm{s}(\mathsf{e})$ to $t_\mathrm{c}(\mathsf{e})$ (see Figure 1(b)). From a model developer's perspective, when requesting the current simulated time from the simulation framework in an event handler, it always returns $t_\mathrm{s}(\mathsf{e})$. In addition, model developers can access and dynamically advance $t_\mathrm{d}(\mathsf{e})$ inside an event handler.

Event Ordering in the Future Event Set. Discrete event simulation sorts events in the FES in increasing order according to their timestamp. Since an expanded event $\mathsf{e} \in \mathcal{E}$ is defined by two timestamps, $t_\mathrm{s}(\mathsf{e})$ and $t_\mathrm{c}(\mathsf{e})$, sorting can utilize either timestamp, combinations of both, or even the event duration $t_\mathrm{d}(\mathsf{e})$. Taking the considerations of the previous section into account, our choice is to use the starting time $t_\mathrm{s}(\mathsf{e})$ as relevant sorting key. Moreover, similar to traditional discrete event simulation, the FES is sorted according to increasing timestamps, i. e., starting times. Note that in practice discrete/expanded event simulation frameworks apply additional sorting keys as tie breakers between events with equal (starting) timestamps. These tie breakers, e. g., the insertion order into the FES, user defined priorities, or event IDs, are needed to achieve a deterministic event ordering. We abstract from these tie breakers in the remainder of this paper.

Parallel Expanded Event Execution Model. As stated before, an expanded event e represents a physical process starting at $t_\mathrm{s}(\mathsf{e})$ and ending at $t_\mathrm{c}(\mathsf{e})$. We assume that the output of such a physical process is neither available nor visible to the entire system before the completion of the process. Therefore, we define that all *overlapping* expanded events, i. e., all events e' starting between $t_\mathrm{s}(\mathsf{e})$ and $t_\mathrm{c}(\mathsf{e})$ of an expanded event e, are independent of e. Hence, an overlapping event e' cannot depend on e because the input of e' cannot include the output of e which is not yet available at $t_\mathrm{s}(\mathsf{e}') < t_\mathrm{c}(\mathsf{e})$. As a result, the interval between $t_\mathrm{s}(\mathsf{e})$ and $t_\mathrm{c}(\mathsf{e})$ naturally opens a window for parallelization as shown in Figure 2. In terms of event processing, this means that when

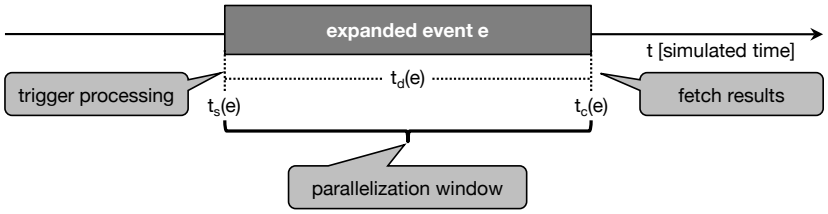


Fig. 2. Execution scheme of an expanded event e: The results of a simulated continuous task are not needed in the simulation before $t_c(e)$. Hence, the simulation scheduler offloads e to a worker CPU at $t_s(e)$ and fetches the results at $t_c(e)$, allowing other events to be processed in-between.

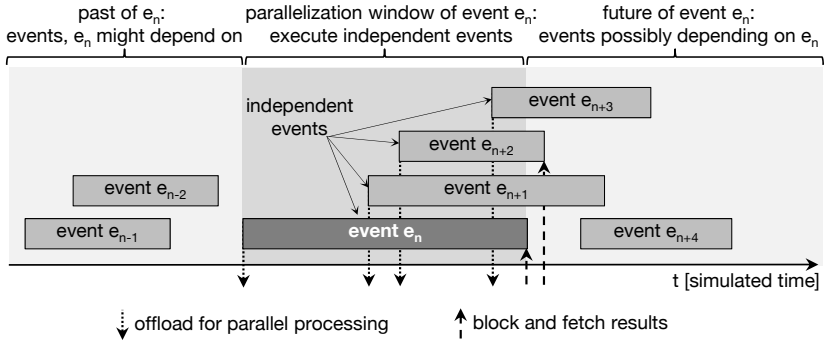


Fig. 3. Parallel event scheduling: The central scheduler advances the global simulation time by iteratively determining independent events, offloading them to CPUs and fetching the results of completed events.

the global simulated time reaches $t_s(e)$ for a given expanded event e, the simulation framework begins executing e. Specifically, the simulation kernel *offloads* e for parallel processing to an available processing unit and continues handling further events. When reaching $t_c(e)$ in simulated time, the results of e are available and needed in the model. Thus, the simulation blocks at $t_c(e)$ and waits for the processing unit to finish executing the offloaded event. Figure 3 illustrates the resulting parallel event execution model visually.

3. THE HORIZON SIMULATION FRAMEWORK

After presenting parallel expanded event simulation, we now introduce HORIZON, a parallel simulation framework that puts our novel modeling paradigm into practice. HORIZON targets multi-core systems in order to fully utilize their processing for simulation. The key challenge in this context is to provide a parallelization framework that is simple to use and tailored to the properties of typical simulation models of tightly coupled systems. We address this challenge by proposing a centralized parallelization architecture comprising a global FES and event scheduler.

Partitioning. The key to efficient partitioning a simulation model is to achieve an even workload distribution which avoids idle times and reduces cross-partition synchronization. Space-parallel partitioning schemes assign (groups of) components of a simulation model to the available processing units. However, the workload inflicted by the events executed on these components can be highly heterogeneous. As a result, partitioning a given simulation model is a difficult task and manually partitioning a given model is a considerable additional effort for model developers and users. In contrast, dynamic partitioning, i. e., load balancing, aims at maintaining an equally distributed workload by adapting the assignment of components to processing units

at runtime [Peschlow et al. 2007]. To this end, the simulation framework continuously measures the workload and idle times of the processing units and migrates simulated entities between partitions accordingly. However, these measurements as well as the migration process add to the complexity and the overhead of the simulation framework, and re-assigning entities can have a negative impact on the lookahead.

HORIZON addresses these issues by means of a multi-threaded master-worker architecture that avoids partitioning altogether. Instead of splitting the FES according to partitions, HORIZON retains a single global FES. A central event scheduler thread continuously dequeues events from the sole FES and distributes independent events to worker threads for parallel processing. As a result, the workload of a simulation model is evenly and automatically distributed across all available processing units, thereby eliminating the need for an explicit load balancing mechanism. This architecture is a direct consequence of the fact that HORIZON specifically targets multi-core systems. Such systems provide a global shared memory space across all processing units and threads, thus enabling any worker to handle any available independent event.

Simulation models in HORIZON exhibit a modular structure. For instance, individual parts of a simulation model represent separate components of the simulated system, e. g., hosts, network cards, protocols, the wireless channel. Besides common practice in software engineering, modularization is imperative for ensuring data consistency in our multi-threaded simulation framework: In shared-memory parallel simulation, event handlers are able to change the state of the entire simulation model. Thus, when executing events in parallel threads, the corresponding event handlers should not read from and write to the same state variables to avoid race conditions which result in an inconsistent state. Hence, we employ the modular structure of simulation models in HORIZON to encapsulate the state of components. An event handler is thus only allowed to modify the state locally to the module it belongs to. In order to avoid race conditions between multiple events occurring in the same local module, our event handling engine dispatches only one event per module at the same time.

If an event needs to change the state at a remote component, it has to create a new event that takes place at this particular module. Hence, modules in HORIZON are similar to logical processes in traditional parallel discrete event simulation. Structuring a simulation model in fairly independent components which facilitate parallel event execution is thus not unique to parallel discrete event simulation, but a challenge in nearly all parallel simulation approaches.

Synchronization. We now turn to the synchronization[1] algorithm of HORIZON. We initially decided to adopt a conservative synchronization algorithm due to its simplicity but also due to the importance of the lookahead for conservative synchronization algorithms. Thus, this allows an easier evaluation of the improvements through our expanded event approach over traditional parallel discrete event simulation. More precisely, we decided to utilize a barrier-based event synchronization. Recall that in parallel expanded event simulation overlapping events are independent. Hence, the event scheduler continuously dequeues the first event e from the FES F, checks whether or not it overlaps with currently offloaded events and if so, hands it to a worker for parallel processing (see Algorithm 1). Conversely, the scheduler does not immediately offload e if it does not overlap with *all* previously offloaded events. As a result, the minimum completion time among all offloaded events determines an upper bound, i. e., a *barrier*, for overlapping events. Thus, the synchronization algorithm coordinates par-

[1]Despite the centralized nature of HORIZON's event scheduling, we explicitly use the term synchronization algorithm for mainly two reasons: i) the scheduling algorithm is designed to achieve synchronization across the execution of non-independent events and ii) to use the same terminology as being used in related efforts.

Algorithm 1 Parallel scheduling of expanded events.

Procedure: ParallelEventScheduler()

```
1: shared variables: F, O, t_b
2: t_b := ∞, O := ∅
3: while F ∪ O ≠ ∅ do
4:   if F ≠ ∅ then
5:     e := arg min_{e∈F}(t_s(e))
6:     if t_s(e) ≤ t_b then
7:       O := O ∪ {e}, F := F\{e}
8:       determine t_d^min(e)
9:       t_b := min{t_b, t_c(e)}
10:      offload(e)
11:    else
12:      wait for t_s(e) ≤ t_b
13:    end if
14:  end if
15: end while
```

(a) Central Event Scheduler

Procedure: ParallelWorker()

```
1: shared variables: F, O, t_b
2: while true do
3:   e := getNextOffloadedEvent()
4:   execute e and t_d(e)
5:   O := O\{e}
6:   update t_b
7: end while
```

(b) Worker Thread

allel event execution by maintaining a barrier computed over all offloaded events. Deciding whether or not an event is offloadable hence boils down to checking if its starting time precedes the barrier. In order to formally state the event synchronization scheme, we first define the set O of all currently offloaded events.

Definition 3.1 (*Set of Offloaded Events*). The set $O \subseteq \mathcal{E}$ contains all currently offloaded expanded events, i. e., all overlapping expanded events being executed concurrently on all processing units. The sets F and O are mutually exclusive, i. e., $O \cap F = \emptyset$.

Based on O, we specify the synchronization barrier t_b as follows:

Definition 3.2 (*Synchronization Barrier*). The synchronization barrier $t_\mathrm{b} \in T \cup \{\infty\}$ is the minimum completion time of all events in O or infinity if O is empty:

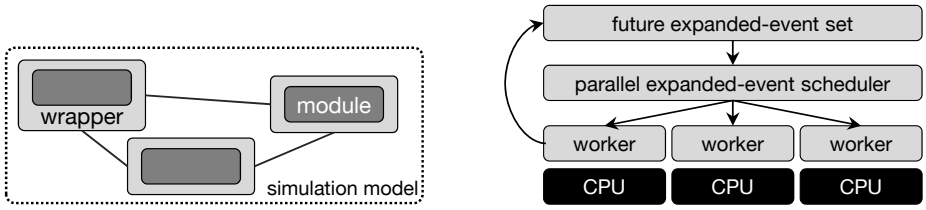
$$t_\mathrm{b} = \begin{cases} \min\{t_\mathrm{c}(\mathsf{e}) | \mathsf{e} \in O\} & , O \neq \emptyset \\ \infty & , \text{otherwise} \end{cases}$$

Algorithm 1 gives a formal definition (in pseudo-code) of the barrier-based synchronization scheme for the event scheduler and the workers separately.

Implementation of the HORIZON *Framework.* HORIZON is based on the widely used simulation framework OMNeT++ [Varga 2001; 2014] and is publicly available[2]. We decided to use OMNeT++ as implementation base for HORIZON, as the source code of OMNeT++ is publicly available and OMNeT++ has a large base of simulation models [Varga 2014]. The downside of building on top of an existing simulation framework is that integrating expanded event simulation creates non-trivial technical challenges. Despite supporting *distributed* parallel simulation, OMNeT++ is not designed for *multi-threaded* simulation. As a result, enabling thread-safe parallel event execution requires thorough analysis, modification, and verification of the simulation core.

To foster wide-spread use of parallel expanded event simulation, HORIZON is backwards compatible with OMNeT++, hence enabling a convenient transition to duration based modeling. However, to make use of expanded events, the model must be ported to HORIZON under consideration of two aspects. Every module of the simulation should provide one new method that determines the duration of a given event and

[2]https://code.comsys.rwth-aachen.de/redmine/projects/horizon-public.



(a) HORIZON wraps OMNeT++ modules. (b) Event handling in HORIZON.

Fig. 4. High-level overview of the architecture of HORIZON.

returns it to the event scheduler (cf. Section 2). If a module does not implement this method, the duration defaults to zero, hence rendering all events at this module discrete events. In addition, OMNeT++ utilizes global random number generators which are not thread safe. Locking these generators does not suffice since concurrent worker threads can access the generators in an arbitrary order, thereby breaking determinism across multiple simulation runs. Thus, every module in HORIZON employs only local random number generators. Since the event order locally at every module is deterministic, the mapping of random numbers to events is deterministic as well. Finally, the same modeling restriction applies as in distributed parallelization using OMNeT++: Event handlers must only modify data which is local to the module they execute on to preserve data consistency. Hence, exchanging data between modules relies on events and global variables are forbidden. Moreover, the event scheduler of HORIZON ensures that at most one thread is active per module at a time. In combination, these criteria prevent data races within and across modules.

Figure 4 shows the resulting architecture. HORIZON encapsulates the modules of classic OMNeT++ simulation models in wrappers which implement functionality for handling event durations, mutual exclusion of worker threads and additional housekeeping (see Figure 4(a)). Furthermore, the future event list and the event scheduler of OMNeT++ have been adapted to handle expanded events, and to forward them to worker threads for parallel processing (see Figure 4(b)).

4. THEORETICAL PERFORMANCE ANALYSIS

In this section, we present a theoretical model of the centralized parallelization scheme underlying HORIZON and a classical distributed parallelization approach. The goal of this section is to determine a high-level criterion for when parallel expanded event simulation is likely to outperform a classic parallelization scheme and vice versa.

Notations. We define the following additional terms:

- Π: set of partitions of a classical distributed simulation
- $\mathcal{E}_\pi \subseteq \mathcal{E}$: set of events in partition $\pi \in \Pi$
- Θ: simulation time, i. e., real time
- N: number of available CPU cores. For distributed simulation we assume $N = |\Pi|$.
- $p : \mathcal{E} \to \mathcal{N}$: assigns each event the number of parallel events
- $l \in T$: lookahead within the simulation
- X: random variable denoting the number of events which can be executed at a given point in simulated time
- Y: random variable denoting the number of events of a partition which can be executed at a given point in simulation time by an LP
- Z: random variable denoting the average degree of parallelization

We now determine the expected value of Z for HORIZON as well as for classical distributed simulation.

Parallel Expanded Event Simulation. The degree of parallelization $p(\mathsf{e})$ for an event $\mathsf{e} \in \mathcal{E}$ depends on the number of overlapping events:

$$p(\mathsf{e}) = |\{\mathsf{e}' \in \mathcal{E} | [t_\mathsf{s}(\mathsf{e}); t_\mathsf{c}(\mathsf{e})] \cap [t_\mathsf{s}(\mathsf{e}'); t_\mathsf{c}(\mathsf{e}')] \neq \emptyset\}| \tag{1}$$

To estimate the average parallelization degree, we make the following assumptions:

- The event starting times are independent and the arrival rate is exponentially distributed with parameter D.
- The durations are independent and exponentially distributed with parameter d.

The probability for n overlapping events at a given point in simulated time is then a Poisson distribution, i. e.:

$$P(X = n, \delta) = \frac{e^{-\delta} \cdot \delta^n}{n!} \tag{2}$$

for $\delta := d \cdot D$ and $n = 0..\infty$. We restrict this to the points in simulated time when actually events occur by means of the conditional probability $P(X = n, \delta | X \geq 1, \delta)$:

$$P(X = n, \delta | X \geq 1, \delta) = \frac{P(X = n, \delta \cap X \geq 1, \delta)}{P(X \geq 1, \delta)} = \frac{P(X = n, \delta)}{1 - e^{-\delta}}, \text{ for } n = 1..\infty \tag{3}$$

From this we derive the expected value of the parallelization degree as a function of the number of available CPU cores by calculating the weighted sum of the probabilities for each possible degree.

$$E[Z](N) = \sum_{i=1}^{N} i \cdot \frac{P(X = i, \delta)}{1 - e^{-\delta}} + N \cdot (1 - \sum_{i=1}^{N} \frac{P(X = i, \delta)}{1 - e^{-\delta}}) \tag{4}$$

$$= \sum_{i=1}^{N} \frac{i \cdot e^{-\delta} \cdot \delta^i}{i! \cdot (1 - e^{-\delta})} + N \cdot (1 - \sum_{i=1}^{N} \frac{e^{-\delta} \cdot \delta^i}{i! \cdot (1 - e^{-\delta})}) \tag{5}$$

We derive the parallelization degree for a sufficiently large number of CPUs by assuming $N \to \infty$: This function converges to the parallelization degree reachable by HORIZON using a sufficiently large number of CPUs:

$$\lim_{N \to \infty} E[Z](N) = \frac{\delta}{1 - e^{-\delta}}, \quad \delta \in]0, \infty[\tag{6}$$

Intuitively, for $\delta \to 0$, the expected parallelization degree converges to 1, corresponding to sequential execution. Conversely, for large values of δ, the expected parallelization degree is bounded by the number of overlapping (i. e., parallelizable) events.

Distributed Parallelization. The degree of parallelization $p(\mathsf{e})$ for an event $\mathsf{e} \in \mathcal{E}$ depends on the number of partitions which contain executable events within the lookahead:

$$p(\mathsf{e}) = |\{\pi \in \Pi | \exists \mathsf{e}' \in \mathcal{E}_\pi : t_\mathsf{s}(\mathsf{e}') \in [t_\mathsf{s}(\mathsf{e}); t_\mathsf{s}(\mathsf{e}) + l]\}| \tag{7}$$

We make the following assumptions:

- The event starting times are independent and the arrival rate is exponentially distributed with parameter D.
- Each partition has the same total number of events, i. e., $\forall \pi \in \Pi : |\mathcal{E}_\pi| = |\mathcal{E}|/|\Pi|$

The probability for n parallelizable events within a lookahead l is given by

$$P(Y = n, \delta) = \frac{e^{-\delta/N} \cdot (\delta/N)^n}{n!} \tag{8}$$

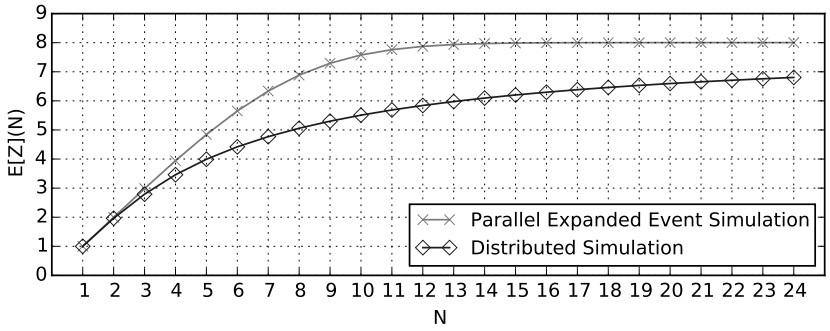


Fig. 5. Expected parallelization degree of parallel expanded event simulation and distributed simulation.

for $\delta := l \cdot D$, $N := |\Pi|$, and $n = 0..\infty$. In the following, we only have to consider

$$P(Y = 0, \delta) = e^{-\delta/N} \text{ and } P(Y \geq 1, \delta) = 1 - e^{-\delta/N} \tag{9}$$

We determine the probability for n parallel events by adding up all possible combinations of $N - n$ partitions without events and n partitions with at least one event:

$$P(X = n, \delta) = \binom{N}{n} \cdot P(Y = 0, \delta)^{N-n} \cdot P(Y \geq 1, \delta)^n \tag{10}$$

With this probability we can apply the same methodology we used for HORIZON to derive the expected value. We calculate the conditional probability:

$$P(X = n, \delta | X \geq 1, \delta) = \frac{P(X = n, \delta)}{1 - P(X = 0, \delta)} = \frac{P(X = n, \delta)}{1 - e^{-\delta}}, \text{ for } n = 1..\infty \tag{11}$$

We calculate $E[Z](N)$ as the weighted sum of the probabilities:

$$E[Z](N) = \sum_{i=1}^{N} i \cdot \frac{P(X = i, \delta)}{1 - e^{-\delta}} = \sum_{i=1}^{N} i \cdot \frac{\binom{N}{i} \cdot \left(e^{-\delta/N}\right)^{N-i} \cdot \left(1 - e^{-\delta/N}\right)^i}{1 - e^{-\delta}} \tag{12}$$

$$= \frac{e^{-\delta}}{1 - e^{-\delta}} \sum_{i=1}^{N} i \cdot \binom{N}{i} \cdot e^{i\delta/N} \cdot \left(1 - e^{-\delta/N}\right)^i \tag{13}$$

For a sufficiently large number of partitions, we yield

$$\lim_{N \to \infty} E[Z](N) = \frac{\delta}{1 - e^{-\delta}}, \quad \delta \in]0, \infty[\tag{14}$$

following the same intuition and ranges as equation (6). Note that this is the same value as for HORIZON yet with a different δ.

Conclusion. Figure 5 depicts the expected parallelization degree of HORIZON and distributed simulation from equations (5) and (13) over a varying number of CPU cores and an exemplary chosen $\delta = 8$. According to equations (6) and (14) both approaches should converge at ≈ 8.003, which the plot confirms. However, we also observe that HORIZON converges faster. We observed the same behavior for different values of δ. Hence, we conclude:

i) If sufficiently many cores are available, HORIZON outperforms classical distributed simulation when HORIZON's δ ($d \cdot D$) is greater than distributed simulation's δ ($l \cdot D$), i. e., $d > l$, and vice versa.

ii) For similar values of δ, however, HORIZON requires less CPU cores to achieve the same speedup. Hence, if the number of available CPU cores is relatively low compared to the parallelization degree of the model, HORIZON performs better than distributed simulation.

This formalism assumes that every LP processes the same total number of events. If this is not the case, less speedup is achieved by distributed simulation, while the HORIZON scheduler always distributes the load to a currently available CPU core.

5. EVALUATION

We evaluate HORIZON in three steps. First, we utilize synthetic benchmarks to characterize the performance properties of HORIZON with regard to the number of CPUs. Secondly, we compare HORIZON to the parallelization capabilities of OMNeT++ and PRIME [Liu et al. 2009]. Finally, we show the applicability of HORIZON by considering practically relevant simulation models (mesh networks, LTE networks).

Throughout this evaluation, we use the *speedup* [Bagrodia and Takai 2002] as performance metric which is defined as the ratio between the sequential execution time $t_{seq}(S)$ and the parallel execution time $t_{par}(S)$ of a simulation model S. Our evaluation is based on a implementation of HORIZON which builds upon OMNeT++ 4.1. All performance results show average values collected over 30 independent runs and the corresponding 99 % confidence intervals, which are however barely visible. We utilized an AMD Opteron compute server providing 32 GB of RAM and a total of 12 processing cores, organized in two six-core CPUs running a 64-bit Ubuntu 12.04.1 LTS server OS.

5.1. Performance Characteristics of Horizon

We investigate the scalability of the centralized architecture of HORIZON with regard to the number of worker threads and the workload. The synthetic benchmark model allows for adjusting two parameters: i) the degree of parallelism as well as ii) the computational complexity of the events. The latter defines the wall-clock time required to process a given event. It directly influences the parallelization speedup by changing the ratio of the event handling and synchronization overhead caused by the simulation framework and the actual workload of the simulation model. Consequently, we generally expect a better speedup for larger event complexities than for smaller ones. Moreover, the model consists of a configurable number of independent, i. e., not interconnected, benchmark modules. Each module continuously creates expanded events of specific computational complexity and schedules them for local execution. By maintaining a perfectly synchronous and overlapping timing among the events, we enable parallel execution. Since every module executes one expanded event at a time, we control the degree of parallelism by means of the number of benchmark modules.

To assess the scalability of HORIZON, we analyze the runtime performance of the benchmark model while varying the number of worker threads and the computational complexity of the events. Based on runtime performance profiles of publicly available simulation models and own measurements [Naghibi and Gross 2010] (see Figure 8), the event complexity ranges from 1 µs to 1 ms in this benchmark. We furthermore vary the number of workers between 1 and 11. In contrast, the degree of parallelism in this benchmark is fixed to 110 by using a total of 110 benchmark modules. This guarantees sufficient parallel workload for keeping the workers busy.

Figure 6 shows the resulting speedup. For event complexities of 1 ms and 0.1 ms, HORIZON achieves a speedup that grows linearly with the number of workers.[3] When

[3]Note that the slight drop in performance for 9 workers results from mapping 110 independent events to 9 workers, resulting in a sub-optimal resource utilization. We observe the same effect also for the other measurements points, yet it is less pronounced.

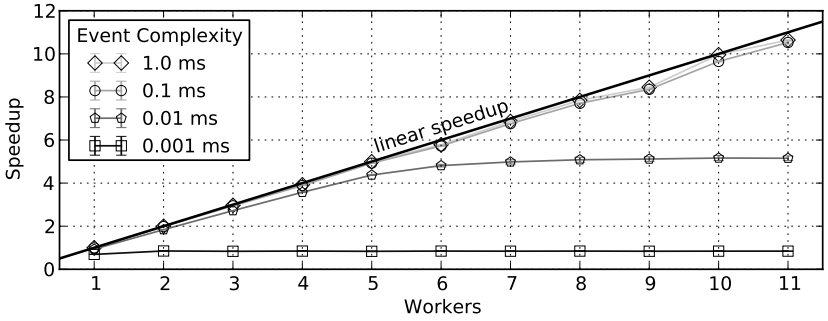


Fig. 6. Speedup of HORIZON in terms of the number of workers and the event complexity using a continuously parallelizable workload of 110 independent events.

reducing the event complexity to 10 µs, we identify a linear speedup for up to 5 workers. Beyond 5 workers, however, the speedup converges to 5. This is due to the fact that the centralized event scheduler handles events sequentially. In order to offload x events for concurrent processing, the event complexity has to be at least x times the offloading delay. Thus, given an event complexity of 10 µs, the scheduler is able to offload only 6 events, i. e., to keep at most 6 workers busy. For the same reason the speedup finally degrades to one when limiting the event complexity to merely 1 µs. [4]

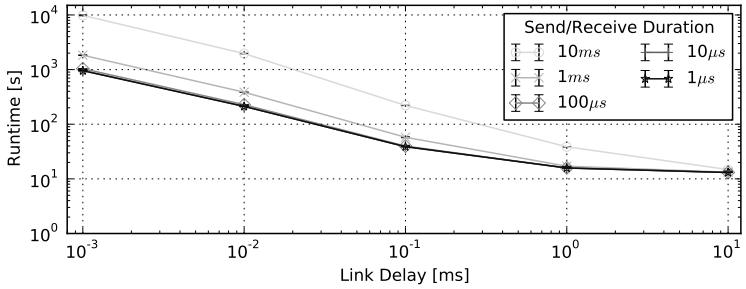
5.2. Comparison with Traditional Parallel Discrete Event Simulation

The goal of this benchmark is to analyze the performance improvement of parallel expanded event simulation over traditional parallel discrete event simulation. To this end, we compare HORIZON with OMNeT++ and PRIME utilizing their parallel discrete event simulation capabilities [Liu et al. 2009; Varga 2014; Sekercioglu et al. 2003]. OMNeT++ uses the Null Message Algorithm (NMA) [Chandy and Misra 1979] and relies on MPI (Message Passing Interface) for inter-LP communication, which in turn exploits shared memory on multi-core computers. PRIME employs a composite synchronization algorithm [Nicol and Liu 2002] and natively supports multi-threading.

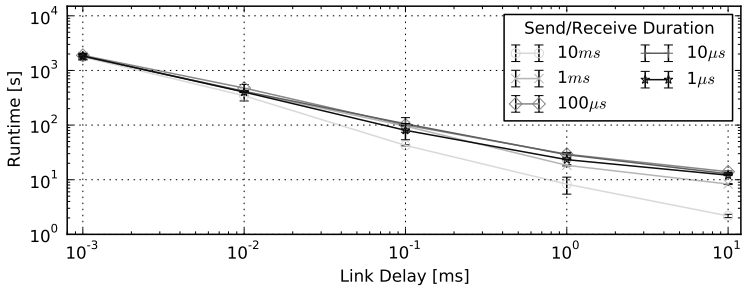
Comparing HORIZON to OMNeT++ allows to focus on investigating the efficiency of the parallelization schemes while excluding differences in the structure of the frameworks. Specifically, the simulation core and modeling API of HORIZON and OMNeT++ are, by design, nearly identical. This enables us to use one basic benchmark model for both frameworks. However, it is necessary to re-implement a separate model to accommodate the process-driven modeling paradigm underlying PRIME.

The key performance factors of expanded and discrete event simulation are the lookahead and event durations. We incorporate these properties in the benchmark models as follows: First, we interconnect all modules via links with a configurable delay since traditional parallelization in OMNeT++ and PRIME derives the lookahead from link delays. These links form a fully meshed topology, allowing benchmark modules to send abstract packets to randomly selected neighbors. Secondly, we add “send” and “receive”-processes spanning a period of simulated time to each module. In HORIZON, we model these processes by means of expanded events while the model for OMNeT++ resorts to using discrete start and end-events for both processes. The process-driven model for PRIME instead uses wait statements. We furthermore

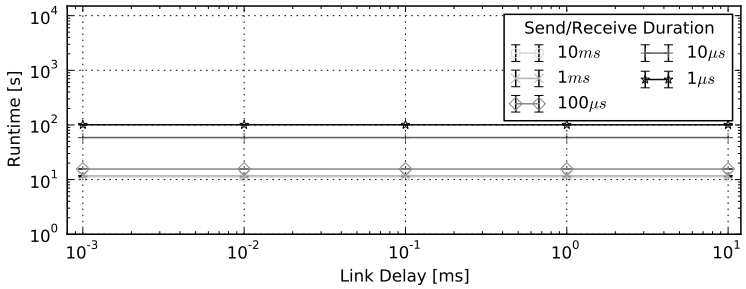
[4]Throughout these benchmarks all CPUs are fully utilized. This is in line with the push-based event handling scheme we presented in a previous publication [Kunz et al. 2011]. In terms of memory consumption, HORIZON imposes only a small overhead in comparison to OMNeT++ which is mainly due to meta-data maintained by the worker threads.



(a) OMNeT++.



(b) PRIME.



(c) Horizon

Fig. 7. Runtimes of OMNeT++, PRIME, and HORIZON.

vary the duration of the send and receive events analogously to the link delay since HORIZON uses this information to identify independent events. All benchmark models thus provide the same timing information, yet differently embedded. Third, each module generates (start-)send events with uniformly distributed interarrival times. These events trigger the sending process, however, only in 10 % of the cases, a packet is actually sent to one neighboring module, where it initiates the receive process. This behavior resembles the widely used and accepted PHOLD benchmark [Fujimoto 1990b] used to profile parallel simulations. Finally, the computational complexity of the events is 0.1 ms, which corresponds to the computational complexity found in simulation models of wireless systems (see Figure 8). Again, the model comprises 110 benchmark modules, distributed across 11 partitions/workers.

Figure 7 shows the runtimes of HORIZON, OMNeT++, and PRIME when executing the respective benchmark models. We observe that OMNeT++ and PRIME achieve a roughly similar performance which significantly depends on the link delay. For a link delay of 1 µs, the runtimes of both simulators exceed the runtimes for a link delay of

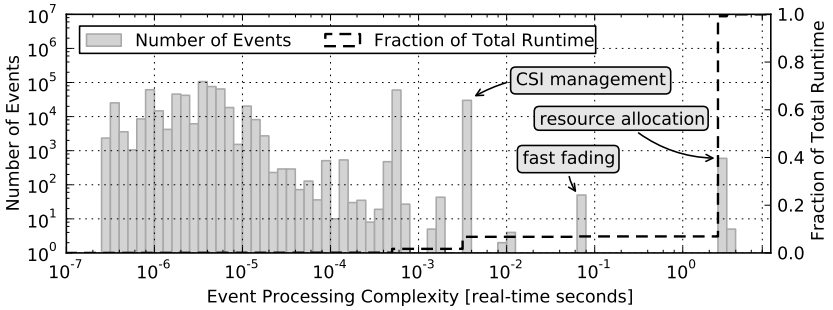


Fig. 8. Distribution of event processing complexities in the LTE model (12 cells, 50 MS/cell) and their corresponding fraction of the total simulation runtime (black dashed CDF). The figure further indicates the three most complex types of events. Note the double-logarithmic scales.

1 ms by a factor of up to 1000. Moreover, the runtimes of OMNeT++ increase drastically for short link delays as the event duration increases. This behavior is due to the time creeping problem of the NMA: If the lookahead does not cover a real simulation event, the NMA needs to send and process (multiple) null-messages to advance the simulated time, thereby increasing the synchronization overhead. For short link delays, the lookahead is small while long event durations increase the time between real simulation events, hence increasing the probability that the lookahead does not cover an event. PRIME shows a different behavior in this regard: With increasing link delays and event durations, the composite synchronization scheme is able to limit the synchronization overhead, resulting in much improved performance.

In contrast to OMNeT++ and PRIME, the performance of HORIZON does not depend on the link delay, but instead on the extent of the event durations: The longer the event durations, the more events overlap and can hence be processed in parallel. Moreover, HORIZON avoids synchronization messages altogether by focusing solely on shared-memory synchronization on multi-core systems. As a result, the central event scheduler has a global view of the FES and can thus immediately advance the simulated time to the next event. HORIZON hence explicitly trades off distributed simulation capabilities for a higher efficiency on our target platforms.

Overall, we observe that both approaches, traditional parallel event simulation and parallel expanded event simulation, outperform each other depending on the properties of the model. For tightly-coupled systems, characterized by short link delays, parallel expanded event simulation significantly outperforms the traditional schemes. Taking into account the typical link delays of wireless systems (in the range of a few μs at most), HORIZON promises a significant speed-up for these models.

5.3. Case Study: LTE Network Model

In order to underline HORIZON's applicability to real simulation models, we next present two case studies based on a complex simulation model of a 3GPP-LTE network as well as one for wireless mesh networks.

Simulation Model. LTE systems constitute the fourth generation of cellular networks and are currently deployed around the world. Their further evolution is of high interest and thus constantly pursued by academia and industry. Hence, this case study considers a currently very relevant system model.

Simulation models of LTE networks are known to be of high computational complexity. For instance, LTE systems perform involved resource allocation schemes at so called Transmission Time Intervals (TTIs) with a length of 1 ms. Specifically, at every downlink TTI each basestation (eNodeB) in the network dynamically assigns trans-

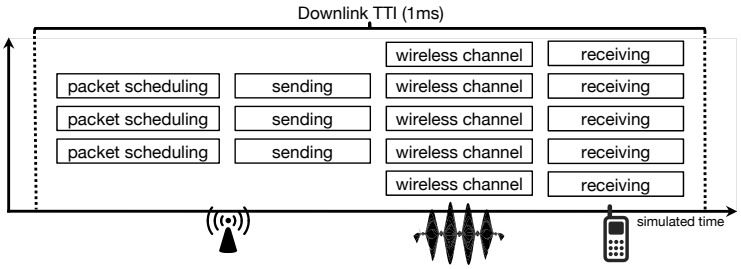


Fig. 9. Integrating parallel expanded event simulation into a time-slotted LTE model. Based on the specification that each TTI lasts 1 ms, we assign pseudo durations that jointly span the entire TTI. Moreover, the event durations of equal event types overlap to enable parallel execution.

mission resources (mainly power and bandwidth) to its associated mobile terminals (UEs). As these computations are based on the instantaneous channel state of each link between an eNodeB and a UE, these states need to be generated every TTI, i. e., every millisecond, as well. In this context, particularly the fast fading behavior needs to be calculated accurately for every transmission channel. Since a typical LTE network utilizes more than 50 channels and comprises hundreds of UEs per eNodeB, the complexity of the channel generation process is considerable. This is further aggravated by the fact that all eNodeBs operate on the same frequency band. Consequently, the channels of all interfering eNodeBs need to be considered as well, leading to an exponential growth in runtime as the number of eNodeBs and UEs increases.

Figure 8 shows a histogram over the distribution of event processing (wall-clock) times when executing the model with 12 eNodeBs and 50 UEs/eNodeB. In general, the complexity of events ranges from $2 \cdot 10^7$ s up to 4 s with the majority of events comprising a relatively small complexity. The figure indicates that the events modeling the resource allocation algorithm and the physical channel are of considerable complexity, ranging from approx. 400 µs to 4 s. In particular, these events contribute almost exclusively to the total runtime of the simulation model as visualized by the dashed CDF. The CDF shows the fraction of the total runtime taken up by the sum of runtimes of all events up to a certain complexity. For example, the figures shows that all events up to a complexity of 1 ms account for only 2 % of the total simulation runtime.

Time Calibration. The timing of the events in the model is highly regular: All events belonging to one downlink TTI take place at the beginning of the respective TTI at the *same* point in simulated time. A timer event indicates the beginning of a downlink TTI and triggers a recursive creation of all subsequent events, i. e., a send event creates a channel event which in turn creates a receive event and so on. Since all events of a TTI exhibit the same timestamp, we do not require explicit event durations to enable parallel execution. Instead, we consider discrete events taking place at the same point in time as expanded events with zero-time duration which overlap. Moreover, the recursive creation of events ensures a correct event ordering. For this reason, we leave the LTE model unchanged in terms of timing.

Nevertheless, we stress that parallel expanded event simulation is applicable to such an abstract system model. The general idea is to exploit the fact that TTIs last exactly 1 ms. Specifically, we assign pseudo durations to the events of a TTI such that the resulting expanded events i) jointly span an entire TTI, and ii) overlap according to physical processes that occur concurrently, as shown in Figure 9.

Results. At first, we investigate the performance of HORIZON for a variable number of worker threads over three workload scenarios comprising 10 eNodeB and 100, 200, and 300 UE, respectively. We execute all three scenarios sequentially and in parallel

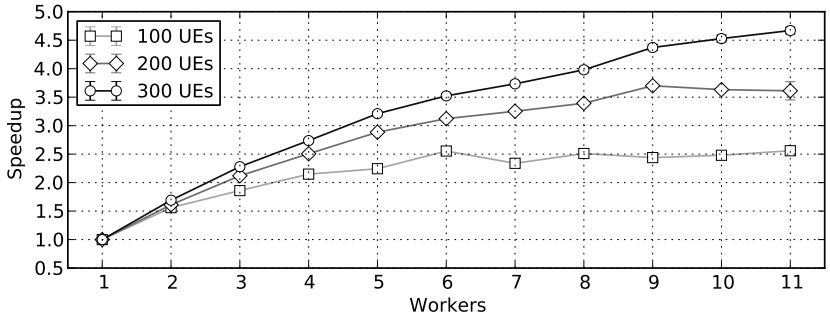


Fig. 10. Speedup over sequential execution for workers for a network of 10 cells with a total of 100, 200, and 300 UEs.

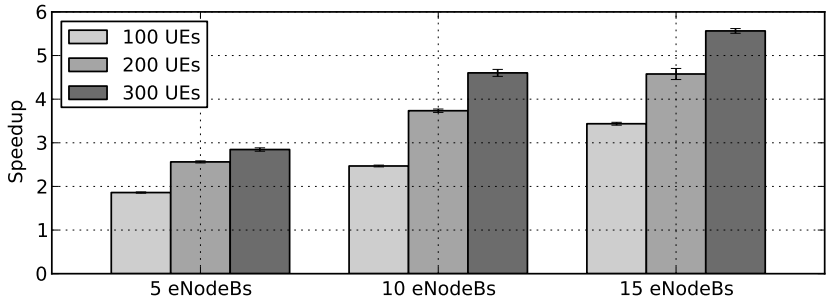


Fig. 11. Speedup of HORIZON using 11 workers over sequential execution for different workloads comprising networks of 5, 10, and 15 cells with a total of 100, 200, and 300 UEs.

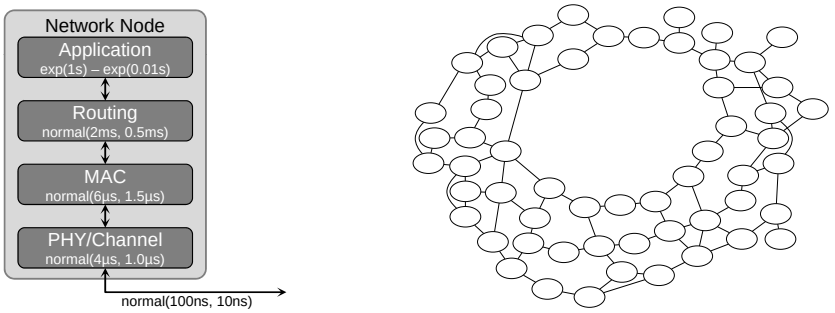
using 1 to 11 worker threads. Figure 10 shows the resulting speedups. We observe that the speedup increases in all scenarios with the number of worker threads and the number of UE, reaching a speedup of 2.5 for 100 UE, 3.5 for 200 UE, and 4.5 for 300 UE. We interpret these results as follows: A larger number of UE requires more channel state computations. These computations are independently modeled by individual events, hence increasing the number of parallelizable events in the model. Moreover, the complexity of the resource allocation algorithm performed on each eNodeB grows with an increasing number of UE. Since the algorithms execute in parallel events, the efficiency of the parallel simulation improves due to larger chunks of parallel work.

We verify this reasoning by varying the workload over a fixed number of 11 CPUs. Specifically, we vary the number of eNodeB between 5, 10, and 15 and distribute a total of 100, 200, and 300 UE among those eNodeB. Figure 11 illustrates the results which in fact confirm the previous reasoning.

5.4. Case Study: Wireless Mesh Network

We further evaluate the applicability of HORIZON by conducting a second case study using a wireless mesh network model. This case study complements the previous one by utilizing a network model which comprises a randomized timing behavior while, again, the simulated entities are tightly coupled. In addition, the event complexities are now lower when comparing to the LTE model.

System Model. The model is based on the "routing" sample shipped with OMNeT++ and has been extended with an accurate fading and error model. The simulated network consists of 60 nodes, each containing an application layer, routing layer, mac layer and physical channel layer which are represented as individual HORIZON modules, cf.



(a) Structure and processing-time distribution of a wireless mesh network node.

(b) Topology of the wireless mesh network.

Fig. 12. Overview of the wireless mesh network model using in the case study.

Figure 12(a). All nodes send packets which are routed through the network to one of three destination nodes via multiple hops. These packets are generated in the application layer and propagated through the stack down to the physical layer (PHY)/Channel layer from which they are broadcast to the PHY/Channel layer of neighboring nodes, cf. Figure 12(b). In the PHY/Channel module, accurate fading statistics are computed and depending on the outcome, packets may be dropped by an error model of the receiving mac layer. The fading and error models are based on work by Wang et. al [Wang et al. 2007] and Puñal et. al [Punal et al. 2011].

We control the density of the network by means of the PHY reception neighborhood (RN) of the nodes: A reception neighborhood of 1 means that all directly connected PHY/Channel modules receive a transmitted packet whereas in a reception neighborhood of 2, the packet is also received by the neighbors' neighbors, and so on. However, only the node selected by the routing layer forwards the packet further.

Besides the node structure, Figure 12(a) also shows the random distributions used to model the event durations, i. e., processing times, and link delays. For example, the events on the routing layer have durations which are normal distributed with a mean of 2 ms and a variance of 0.5 ms. As the event durations correspond to the processing durations in a real network stack, the durations reduce by orders of magnitude down the network stack. In addition to the event durations, we use an OMNeT++ channel delay to model the propagation delay on the link between two PHY/Channel modules. This delay is also randomly distributed to model the varying propagation delays to neighboring nodes at different distances but respects the tight coupling of the simulation model in general.

Results. We conduct the performance benchmarks of this case study with a reception neighborhood of 3 and 4, respectively. This parameterization causes a transmitted packet to be received by a large fraction of the wireless mesh network, yet not by all nodes, thereby retaining the characteristics of a multi-hop network. As a result, the model comprises regions of tightly coupled entities due to small propagation delays while at the same time the model exhibits an irregular timing behavior driven by the probability distributions on the higher network layers.

Figure 13 illustrates the speedup achieved by HORIZON for 1 to 11 worker threads and mean packet sending rates ranging from 1 packet/second to 100 packets/second. The figure shows that for a reception neighborhood of 4 HORIZON achieves a higher speedup than for a reception neighborhood of 3. Obviously, a larger reception area causes more network nodes to be involved in the reception of a packet. Due to the

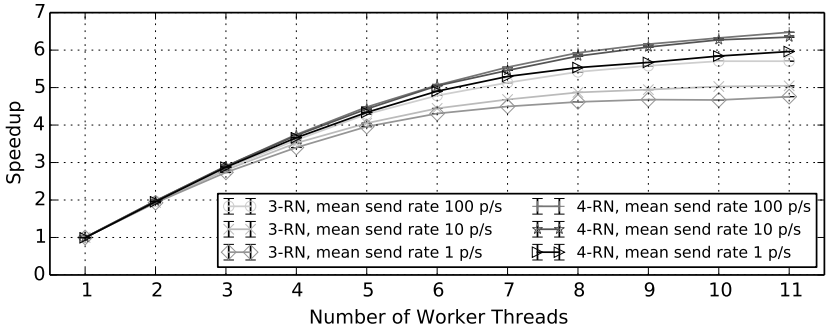


Fig. 13. Speedup of the wireless mesh network model over a varying numbers of worker threads. The speedup increases with the size of the reception neighborhood (RN) and the sending rate.

small propagation delay, however, this results in a tight coupling of a larger fraction of network nodes. Despite this coupling, HORIZON is able to extract a significant amount of parallelism from the simulation model due to the considerably longer durations of the receive events in comparison to the link propagation delay.

Similarly, the speedup increases with the mean packet sending rate in the investigated scenarios as there are more overlapping events in the simulation model. Specifically, for up to 4 worker threads, HORIZON achieves a nearly linear speedup which then diverges to a maximum speedup ranging from 4.8 for a reception neighborhood of 3 and a mean packet send rate of 1 packet/second up to a speedup of 6.4 for for a reception neighborhood of 4 and a mean packet send rate of 100 packets/second.

6. DISCUSSION AND LIMITATIONS

Parallel Expanded Event Simulation. Parallel expanded event simulation is backwards compatible to traditional discrete event simulation. Yet, porting a discrete event simulation to expanded events requires additional modeling effort. This effort can be quite low, e. g., in the case one expanded event replaces two discrete events. However, expanded event simulation fosters the inclusion of additional time information in simulation models which did not consider such information before. As a result, expanding discrete events changes the timing of the model, thus requiring careful adjustments and (re-)validation of the model. Nevertheless, we have demonstrated in [Stoffers et al. 2014a] that the redesign effort even for highly centralized models with global data structures like the OMNeT++ INET model is low to moderate.

Based on our experience, simulation modelers are typically experts in their particular domain of research, e. g., wireless systems, and not necessarily experts in parallel simulation. Hence, it is important to provide a simple-to-use simulation framework without the need to apply load balancing and model partitioning algorithms. Moreover, event durations are an inherent property of the simulated system and hence either known or easily definable by model designers. Hence, the major contribution of this work is to provide a simulation framework which is simple to use, due to a centralized architecture, and enables model developers to easily enrich parallel simulation models with domain specific properties by means of event durations.

Furthermore, modeling physical processes by means of expanded events supports the development of accurate energy models. Since the energy consumption is often directly linked to the duration of a process, e. g., the transmission time of a packet, expanded event simulation provides a solid foundation for detailed energy models. Moreover, despite being the primary field of application, parallel expanded event simulation is not limited to simulating wireless network models. Since parallel expanded event

simulation still retains the basic properties of discrete event simulation, any existing discrete event simulation model can be converted to make use of expanded events. Due to the fact that parallel expanded event simulation extracts its lookaheads from event durations instead of the traditional notion of link delays, it is particularly suitable for simulating tightly coupled systems, i. e., systems in which many entities interact on small timescales. Examples for such systems are parallel hardware platforms such as multi-core processors and highly integrated circuits.

The event scheduler does not offload expanded events which do not overlap, however, which are independent. Previous efforts [Liu and Nicol 2002; Meyer and Bagrodia 1998; 1999] showed that larger lookaheads can be derived when analyzing the interaction of simulated entities and combining their respective lookaheads. These combined lookaheads extend beyond event durations, thereby exposing additional parallelism in a simulation model. However, the smallest lookahead in a simulation model is typically the performance limiting factor (cf. time creeping problem). In this context, the event durations of parallel expanded event simulation aim at increasing the minimum available lookahead. Nevertheless, we explicitly extend parallel expanded event simulation with advanced lookahead extraction techniques by developing a probabilistic event synchronization scheme [Kunz et al. 2012].

Horizon. HORIZON's centralized master-worker architecture avoids the need for explicit load balancing mechanisms by exploiting the shared-memory space of multi-core systems. This approach, however, limits scalability in terms of i) the size of the simulation model and ii) the number of processing units: First, large scale simulation models, e. g., peer-to-peer networks, exhibit a considerable memory footprint, easily exceeding the total memory available in one multi-core system [Fujimoto et al. 2003]. Secondly, high performance multi-core computers will feature tens to hundreds of processing units. However, increasing the number of workers correspondingly results in extreme contention on the central FES and a bottleneck at the scheduler that cannot distribute events fast enough to keep all workers busy. Based on a measured event handling overhead of 1.5 µs [Kunz et al. 2011] and the observed event complexities in our simulation models ranging up to multiple (micro)seconds (see Figure 8), we believe that the limit in scalability is at 50 to 100 processing cores.

Parallel expanded event simulation is orthogonal to existing distributed simulation schemes, enabling a hybrid of both: Each partition of a distributed simulation model runs HORIZON locally on a multi-core cluster node, while a distributed simulation framework handles synchronization and communication across nodes. Previous efforts in the research community successfully investigated hybrid synchronization schemes [Liu and Nicol 2001], while we recently proposed such an extension of HORIZON in [Stoffers et al. 2014b].

Lastly, the event scheduler of HORIZON does not consider caching effects or the physical memory layout when assigning events to CPUs. Currently, the scheduler offloads a given event to the next available CPU core. However, this can cause events of the same event type taking place at the same module to constantly move between different CPU cores. Besides inefficient utilization of the CPU cache, processing events at changing CPU cores can result in prolonged memory access times on the prevalent Non-Uniform Memory Access (NUMA) architectures due to the need to copy data from a possibly remote memory location. Future work will focus on optimizing the event-to-CPU assignment algorithm in order to make more efficient use of NUMA architectures.

7. RELATED WORK

In this section, we review related efforts regarding extended modeling paradigms as well as multi-threaded parallel simulation frameworks.

Related Modeling Paradigms. Previous efforts in network simulation research also extend classic discrete event simulation with additional timing information to enhance simulation performance and scalability. Lubachevsky [Lubachevsky 1988] defines opaque periods which are closely related to the idea of expanded events. An opaque period defines a period in simulated time in which a simulated entity "promises" not to generate events. Thus, opaque periods are similar to expanded events in the sense that the starting times $t_s(e')$ of all events e' generated by an expanded event e must begin after $t_c(e)$ (cf. Definition 2.3) because the results of a physical process are only visible after its end. Due to the fact that opaque periods are not bound to the concept of physical processes, they constitute a more abstract concept. As a result, Lubachevsky needs to manually apply the concept of opaque periods to a given simulation model. In contrast, we deeply embed our approach of expanded events into an intuitive modeling paradigm which fosters the inclusion of expanded events by the model developers.

A different approach is taken by Fujimoto [Fujimoto 1999] who basically observes that real-world events do not occur at pre-determined points in time. Hence, the idea is to replace the accurate timestamps of discrete events with intervals representing a period of uncertainty in which an event may occur. Fujimoto furthermore defines an approximate-time partial-order among events along with corresponding synchronization algorithms. However, approximate-time partial-ordering introduces inaccuracies in the simulation results and limits determinism and the repeatability of simulations. Loper et al. [Loper and Fujimoto 2000; 2004] extends this concept by drawing discrete timestamps from uncertainty intervals according to a random distribution. Selecting discrete timestamps from uncertainty intervals increases the available lookahead while allowing to re-use existing discrete timestamp based synchronization algorithms. It hence solves the problem of limited repeatability and determinism inherent to approximate time partial ordering. In contrast, the respective starting and completion times of expanded events occur at deterministic points in simulated time, hence guaranteeing repeatability of the results.

Peschlow et al. [Peschlow et al. 2008] picks up the idea of uncertainty intervals and investigate the effects of different event orderings resulting from overlapping intervals. To avoid executing one individual simulation run for every possible interleaving of events, the authors propose *interval branching*. In this approach, a single simulation run branches for every possible ordering of events due to overlapping intervals, hence spanning an execution tree representing all possible interleavings. The key performance improvement over executing individual runs for each event interleaving is that equal event interleavings in the individual runs collapse to a common path in the tree which is executed only once. From an implementation perspective, the branching operation relies on logical processes and simulation cloning techniques [Hybinette and Fujimoto 1997; 2001] developed for distributed simulation. However, the branching approach suffers from a state explosion problem. Considering accurate simulations comprising millions of events, creating a branch for each possible event interleaving can easily exceed the available memory resources. Hence, interval branching is not generally applicable to complex simulation models.

Related Simulation Frameworks. In the context of the ns-3 project [Henderson et al. 2006], Seguin [Seguin 2009] implemented a multi-threaded extension of the ns-3 simulation engine. The framework employs both the Null Message Algorithm (NMA) and a synchronous barrier-based algorithm for conservative event synchronization. Due to the time-creeping problem inherent to the NMA, Seguin prefers the barrier-based algorithm for synchronization. Both synchronization schemes derive the lookahead from link delays, however, the framework only supports simple point-to-point links with static delays, corresponding to wired connections. In terms of partitioning, the frame-

work handles each network node as a separate partition, thereby eliminating the need for manual partitioning and simplifying load balancing. The project reports a 20 % performance increase on the DARPA NMS Campus Network model [Nicol 2003] using an eight-core computer. The authors blame the limited performance improvement on the overhead due to locking within the simulation framework. As a result, research on multi-threaded parallelization is discontinued in the ns-3 project, focusing instead on traditional distributed simulation over MPI [Barnes et al. 2012].

The multi-threaded simulation framework HiPWiNS [Peschlow et al. 2009] is based on JiST/SWANS [Barr et al. 2004] aiming for efficient parallel simulation of IEEE 802.11 networks by making two contributions: First, they propose event lookaheads which are conceptually similar to event durations since they exploit the fact that physical processes span a period of time in which they cannot influence the surrounding system. The authors propose using the delay of switching an IEEE 802.11 transceiver from sending to receiving mode (RxTxTurnaround), since during this switch, the transceiver can neither transmit nor receive. The second contribution is called event-bundling and aims at reducing the number of events exchanged between LPs by sending only few meta events which trigger the creation of potentially many local events. Despite the similarity of event lookahead and event durations, the concepts underlying HORIZON are more general. Specifically, event lookahead is applied only to the RxTxTurnaround duration which in turn is implemented by means of two events that are specially treated in the simulation framework. In contrast, expanded event simulation is a generalized modeling paradigm, equally able to utilize the RxTxTurnaround switching delay. Furthermore, event bundling is only effective when utilizing LPs, yet, the static assignment of network nodes to LPs as exercised in HiPWiNS severely restricts load balancing in contrast to the dynamic approach of HORIZON.

The latest incarnation of the Scalable Simulation Framework (SSF) [Cowie et al. 1999] is the PRIME simulation framework [Liu et al. 2009]. The primary focus of PRIME is on achieving real-time simulation as basis for co-simulation, i. e., the interaction of real networking systems with a simulated network. In order to harvest the required processing power needed for large scale networks, it combines multi-threaded simulation with distributed simulation. To this end, the framework utilizes conservative composite synchronization [Nicol and Liu 2002] in combination with hierarchical synchronization [Liu and Nicol 2001] to integrate multi-threaded with distributed simulation. In contrast to PRIME, HORIZON does not aim for large scale co-simulation, but instead focuses on speeding up small to medium scale simulations on desktop or workstation computers. Hence, architecture and event synchronization of HORIZON is considerably simpler than in PRIME.

In [Jagtap et al. 2012] the authors consider the architectural changes required to efficiently re-implement a previously distributed simulation engine based on MPI for multi-core environments. They find that depending on the simulation model and computational environment a speed-up of up to three can be achieved (in comparison to MPI-based implementations) based on the synthetic benchmark PHOLD. In [D'Angelo et al. 2012] the authors propose a novel implementation of the Time warp optimistic synchronization algorithm specifically tailored for multi-core architectures. This implementation is based on Go, a newer programming language that has efficient support for parallelization on multi-core machines. Based on this reimplementation, the authors show that for synthetic benchmarks the reimplementation can reach good speed-ups. However, the novel approach is not benchmarked with comparison schemes. Similarly, in [Chen et al. 2011] the authors investigate novel load-balancing techniques for optimistic synchronization algorithms based on multi-core architectures. In particular the authors add a global scheduler to the optimistic approach which mostly allows for a more efficient load balancing between the different logical partitions. Based on

this new approach, the authors show that the rollback rates can be significantly reduced while the load balancing becomes more efficient between the LPs. A further load-balancing approach, optimizing the simulation execution time of optimistic synchronization for multi-core architectures is presented in [Vitali et al. 2012].

In [Yoginath and Perumalla 2013] the authors address the problem of executing parallel discrete-event simulations over a set of virtual machines. The authors first show that due to the specific characteristics of PDES, the usage of virtual machines (and specifically their generic workload scheduler) does not scale well. Subsequently, the authors propose an optimized scheduler, which improves the performance significantly. Related to that, in [Li et al. 2014] the authors consider the performance of large-scale simulations performed on federates of data centers under optimistic synchronization. They first observe load imbalances leading to excessive synchronization overhead in roll-backs and timer updates. Next, they subsequently develop a hierarchical load-balancing approach.

8. CONCLUSIONS

This paper introduced parallel expanded event simulation as a novel modeling paradigm for efficient parallel simulation of tightly-coupled systems such as wireless networks. By modeling the duration of physical processes by means of one expanded event instead of two discrete events, expanded event simulation enables a parallel event scheduler to derive dependency information about expanded events, eventually allowing for conservative parallel execution of independent events. We furthermore presented HORIZON, a parallel simulation framework that puts expanded event simulation into practice by extending the well-known OMNeT++ simulation framework. In particular, HORIZON exploits the processing power of ubiquitous multi-core systems available to model developers and networking researchers. It thus employs a simple multi-threaded master-worker architecture, thereby avoiding explicit partitioning and load balancing mechanisms. The evaluation of HORIZON by means of synthetic and real-world models underlines the viability of parallel expanded event simulation especially in the context of tightly-coupled systems: For synthetic benchmarks HORIZON outperforms state-of-the-art by up to two orders of magnitude, while for real-world simulation models of wireless networks a speed-up of up to six can be reached. However, these performance advantages diminish for models with looser coupling, i. e., larger lookaheads between model partitions. As future work, we consider two further steps. On the one hand, the multi-core approach is still limited for very large simulation models. In this case, the combination of HORIZON with distributed simulation methods could be employed. Initial steps in this direction have been conducted in [Stoffers et al. 2014b] but further improvements of the synchronization methods between HORIZON partitions can be achieved. Furthermore, additional speed-up of HORIZON might be achieved by optimistic synchronization. Again, initial steps have been presented already [Kunz et al. 2012], but more work is required to compare the scheme to others.

REFERENCES

R. L. Bagrodia and M. Takai. 2002. Performance Evaluation of Conservative Algorithms in Parallel Simulation Languages. *IEEE Transactions on Parallel and Distributed Systems* 11, 4 (2002), 395–411.

P. D. Barnes, J. M. Brase, T. W. Canale, M. M. Damante, M. A. Horsley, D. R. Jefferson, and R. A. Soltz. 2012. A Benchmark Model for Parallel ns-3. In *Proc. of the 5th Inter. ICST Conf. on Simulation Tools and Techniques*.

R. Barr, H. Zygmunt, and R. van Renesse. 2004. JiST: Embedding Simulation Time Into a Virtual Machine. In *Proc. of EuroSim Congress on Modelling and Simulation*.

L. Bononi, M. Di Felice, M. Bertini, and E. Croci. 2006. Parallel and Distributed Simulation of Wireless Vehicular Ad hoc Networks. In *Proc. of the 9th Inter. Conf. on Modeling, Analysis and Simulation of Wireless and Mobile Systems*.

V. Cadambe and S. Jafar. 2008. Interference Alignment and Degrees of Freedom of the K-User Interference Channel. *IEEE Transactions on Information Theory* 54, 8 (2008), 3425.

K. M. Chandy and J. Misra. 1979. Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. *IEEE Transactions on Software Engineering* SE-5, 5 (September 1979), 440–452.

G. Chen and B. K. Szymanski. 2005. DSIM: Scaling Time Warp to 1,033 Processors. In *Proc. of the 37th Winter Simulation Conf.*

L. Chen, Y. Lu, Y. Yao, S. Peng, and L. Wu. 2011. A Well-Balanced Time Warp System on Multi-Core Environments. In *Proc. of IEEE Workshop on Principles of Advanced and Distributed Simulation (PADS)*.

J. Cowie, A. Ogielski, and D. M. Nicol. 2002. The SSFNet Network Simulator. Software on-line: http://www.ssfnet.org/homePage.html. (2002).

J. H. Cowie, D. M. Nicol, and A. T. Ogielski. 1999. Modeling the Global Internet. *Computing in Science & Engineering* 1, 1 (Jan. 1999), 42–50. DOI:http://dx.doi.org/10.1109/5992.743621

G. D'Angelo, S. Ferretti, and M. Marzolla. 2012. Time Warp on the Go. In *Proc. of the 5th Inter. ICST Conf. on Simulation Tools and Techniques*.

R. M. Fujimoto. 1990a. Parallel Discrete Event Simulation. *Commun. ACM* 33, 10 (1990), 30–53.

R. M. Fujimoto. 1990b. Performance of Time Warp under Synthetic Workloads. In *Proc. of the SCS Multiconference on Distributed Simulation*.

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

R. M. Fujimoto, K. S. Perumalla, A. Park, H. Wu, M. H. Ammar, and G. F. Riley. 2003. Large-Scale Network Simulation: How Big? How Fast?. In *Proc. of 11th Inter. IEEE Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*.

D. Gesbert, M. Shafi, D. Shiu, P. J. Smith, and A. Naguib. 2003. From Theory to Practice: An Overview of MIMO Space-time Coded Wireless Systems. *IEEE Journal on Selected Areas in Communications* 21, 3 (April 2003), 281–302.

D. Halperin, T. Anderson, and D. Wetherall. 2008. Taking the Sting out of Carrier Sense: Interference Cancellation for Wireless LANs. In *Proc. of the 14th ACM Inter. Conf. on Mobile Computing and Networking*.

T. R. Henderson, S. Roy, S. Floyd, and G. F. Riley. 2006. ns-3 Project Goals. In *Proc. of the 2006 Workshop on ns-2: The IP Network Simulator*.

M. Hybinette and R. M. Fujimoto. 1997. Cloning: A Novel Method for Interactive Parallel Simulation. In *Proc. of the 29th Winter Simulation Conf.*

M. Hybinette and R. M. Fujimoto. 2001. Cloning Parallel Simulations. *ACM Transactions on Modeling and Computer Simulation* 11, 4 (Oct. 2001), 378–407.

D. Jagtap, N. Abu-Ghazaleh, and D. Ponomarev. 2012. Optimization of Parallel Discrete Event Simulator for Multi-core Systems. In *Proc. of Ithe EEE 26th Inter. Parallel and Distributed Processing Symposium*.

I. Koffman, V. Roman, and R. Technol. 2002. Broadband Wireless Access Solutions based on OFDM Access in IEEE 802.16. *IEEE Communications Magazine* 40, 4 (2002), 96–103.

G. Kunz. 2013. *Exploiting Multi-core Systems for Parallel Network Simulation*. Shaker Verlag. PhD Dissertation.

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

G. Kunz, O. Landsiedel, and K. Wehrle. 2009. Poster Abstract: Horizon - Exploiting Timing Information for Parallel Network Simulation. In *Proc. of the 17th Inter. IEEE Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*.

G. Kunz, M. Stoffers, J. Gross, and K. Wehrle. 2011. Runtime Efficient Event Scheduling in Muti-threaded Network Simulation. In *Proc. of the 4th Inter. Workshop on OMNeT++*.

G. Kunz, M. Stoffers, J. Gross, and K. Wehrle. 2012. Know Thy Simulation Model: Analyzing Event Interactions for Probabilistic Synchronization in Parallel Simulations. In *Proc. of the 5th Inter. ICST Conf. on Simulation Tools and Techniques*.

Z. Li, X. Li, L. Wang, and W. Cai. 2014. Hierarchical Resource Management for Enhancing Performance of Large-scale Simulations on Data Centers. In *Proc. of the 2nd Conf. on Principles of Advanced Discrete Simulation*.

J. Liu. 2009. *Parallel Discrete-Event Simulation*. John Wiley & Sons.

J. Liu, Y. Li, and Y. He. 2009. A Large-scale Real-time Network Simulation Study Using PRIME. In *Proc. of the 2009 Winter Simulation Conf.*

J. Liu and D. M. Nicol. 2001. Learning Not to Share. In *Proc. 15th Workshop on Parallel and Distributed Simulation*.

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

M. Loper and R. M. Fujimoto. 2000. Pre-sampling as an Approach for Exploiting Temporal Uncertainty. In *Proc. of the 14th Workshop on Parallel and Distributed Simulation*.

M. Loper and R. M. Fujimoto. 2004. A Case Study in Exploiting Temporal Uncertainty in Parallel Simulations. In *Proc. of the 2004 Inter. Conf. on Parallel Processing*.

B. D. Lubachevsky. 1988. Efficient Distributed Event Driven Simulations of Multiple-loop Networks. In *Proc. of the ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*.

A. Markopoulou, F. Tobagi, and M. Karam. 2006. Loss and Delay Measurements of Internet Backbones. *Computer Communications* 29, 10 (June 2006), 1590–1604.

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

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

F. Naghibi and J. Gross. 2010. How Bad is Interference in IEEE 802.16e Systems?. In *Proc. of the 16th European Wireless Conf.*

D. M. Nicol. 1996. Principles of Conservative Parallel Simulation. In *Proc. of the 28th Winter Simulation Conf.*

D. M. Nicol. 2003. Darpa Network Modeling and Simulation (NMS) Baseline Network Topology. online, [last accessed November 28, 2014]. (2003). http://www.ssfnet.org/Exchange/gallery/baseline/index.html

D. M. Nicol and J. Liu. 2002. Composite Synchronization in Parallel Discrete-Event Simulation. *IEEE Transactions on Parallel Distributed Systems* 13, 5 (May 2002), 433–446.

K. S. Perumalla. 2006. Parallel and Distributed Simulation: Traditional Techniques and Recent Advances. In *Proc. of the 38th Winter Simulation Conf.*

P. Peschlow, T. Honecker, and P. Martini. 2007. A Flexible Dynamic Partitioning Algorithm for Optimistic Distributed Simulation. In *Proc. of the 21st Inter. Workshop on Principles of Advanced and Distributed Simulation*.

P. Peschlow, P. Martini, and J. Liu. 2008. Interval Branching. In *Proc. of the 22nd Workshop on Principles of Advanced and Distributed Simulation*.

P. Peschlow, A. Voss, and P. Martini. 2009. Good News for Parallel Wireless Network Simulations. In *Proc. of the 12th Inter. Conf. on Modeling, Analysis and Simulation of Wireless and Mobile Systems*.

O. Punal, H. Escudero, and J. Gross. 2011. Performance Comparison of Loading Algorithms for 80 MHz IEEE 802.11 WLANs. In *Proc. of the 73rd IEEE Vehicular Technology Conf.*

G. F. Riley. 2003. The Georgia Tech Network Simulator. In *Proc. of the ACM SIGCOMM Workshop on Models, Methods and Tools for Reproducible Network Research*.

G. Seguin. 2009. *Multi-core Parallelism for ns-3 Simulator*. Technical Report. INRIA Sophia-Antipolis.

A. Sekercioglu, A. Varga, and G. Egan. 2003. Parallel Simulation Made Easy with OMNeT++. In *Proc. of the European Simulation Symposium*.

M. Stoffers, R. Bettermann, J. Gross, and K. Wehrle. 2014a. Enabling Distributed Simulation of OMNeT++ INET Models. In *Proc. of the 1st OMNeT++ Community Summit*.

M. Stoffers, S. Schmerling, G. Kunz, J. Gross, and K. Wehrle. 2014b. Large-Scale Network Simulation: Leveraging the Strengths of Modern SMP-based Compute Clusters. In *Proc. of the 7th Inter. ICST Conf. on Simulation Tools and Techniques (SIMUTools'14)*.

A. Varga. 2001. The OMNeT++ Discrete Event Simulation System. In *Proc. of the 15th European Simulation Multiconference*.

A. Varga. 2014. *OMNeT++ Website*. Retrieved November 28, 2014 from http://www.omnetpp.org

R. Vitali, A. Pellegrini, and F. Quaglia. 2012. Towards Symmetric Multi-threaded Optimistic Simulation Kernels. In *Proc. of IEEE Workshop on Principles of Advanced and Distributed Simulation (PADS)*.

C.-X. Wang, M. Pätzold, and Q. Yao. 2007. Stochastic Modeling and Simulation of Frequency-correlated Wideband Fading Channels. *IEEE Transactions on Vehicular Technology* 56, 3 (2007).

S. B. Yoginath and K. S. Perumalla. 2013. Optimized Hypervisor Scheduler for Parallel Discrete Event Simulations on Virtual Machine Platforms. In *Proc. of the 6th Inter. ICST Conf. on Simulation Tools and Techniques*.