Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Jan Raddatz

Evaluation based design of parallel simulation strategies for in vehicle networks

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Jan Raddatz

Evaluation based design of parallel simulation strategies for in vehicle networks

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Master of Science
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Korf
Zweitgutachter: Prof. Dr. Fohl

Eingereicht am: 2. Februar 2016

Jan Raddatz

Thema der Arbeit
Evaluation based design of parallel simulation strategies for in vehicle networks

Stichworte
Conditional Null Message, Conservative Synchronization, Discrete event simulation DES, Distributed Simulation, Multicore system, Null Message, Partitioning, Parallel discrete event simulation, PDES, Simulation

Kurzzusammenfassung
Diskrete Event basierte Simulationen haben sich zu einem weit verbreiteten Werkzeug zur Auslegung und Entwicklung von Fahrzeugnetzwerken entwickelt. Netzwerke im allgemeinen und Fahrzeugnetzwerke im Besonderen sehen sich mit ständig wachsender Komplexität konfrontiert. Dies führt unausweichlich zu immer weiter steigenden Simulationslaufzeiten. Diese Arbeit präsentiert das Design und die Entwicklung von parallelen Scheduling Strategien um die Simulationslaufzeiten zu verkürzen.
Jan Raddatz

Title of the paper
Evaluation based design of parallel simulation strategies for in vehicle networks

Keywords
Conditional Null Message, Conservative Synchronization, Discrete event simulation DES, Distributed Simulation, Multicore system, Null Message, Partitioning, Parallel discrete event simulation, PDES, Simulation

Abstract
Discrete event based simulations become a wide spread tool in constructing and developing in vehicle networks. Networks in general and in vehicle networks in particular face a steadily increasing complexity. This results inevitably in increasing simulation runtimes. The main contribution of this work is the design and development of parallel scheduling strategies in order to speed up simulation runtimes of in vehicle networks.

Contents

1 Introduction **1**

2 Theoretical foundations and related work **3**

2.1 Simulation . . . 3
2.2 Classifications . . . 4
2.3 Time in simulation . . . 5
2.4 Discrete event based simulation . . . 5
2.5 Parallel discrete event based simulation . . . 7
2.5.1 Basic idea behind PDES . . . 7
2.5.2 Challenges in PDES . . . 8
2.6 Synchronization . . . 9
2.7 Optimistic Algorithms . . . 9
2.7.1 Time Warp Algorithm . . . 10
2.7.2 Time warp implementation . . . 11
2.7.3 Global virtual time . . . 11
2.7.4 Optimization of optimistic algorithms . . . 12
2.7.4.1 Reducing memory footprint . . . 12
2.7.4.2 Preventing overly optimistic execution . . . 12
2.7.4.3 Improved cancellation . . . 13
2.7.5 Performance estimation of optimistic algorithms . . . 13
2.8 Conservative Algorithms . . . 14
2.8.1 Null Message Algorithm . . . 14
2.8.2 Optimization of conservative algorithms . . . 18
2.8.2.1 Conditional null message . . . 18
2.8.2.2 Carrier null message . . . 19
2.8.2.3 Conservative time windows . . . 20
2.8.2.4 Null message cancellation . . . 21
2.8.2.5 Demand driven . . . 21
2.8.2.6 Path lookahead . . . 21
2.8.3 Performance estimation of conservative algorithms . . . 22
2.9 Time-Triggered Ethernet . . . 22
2.9.1 Communication protocols . . . 23
2.9.1.1 Deterministic Time-Triggered (TT) traffic . . . 23
2.9.1.2 Event-Driven or rate-constrained (RC) traffic . . . 24
2.9.1.3 Best-effort (BE) traffic . . . 24

2.9.2 Common time base . . . 24

3 Problem Analysis 26
3.1 Simulation model . . . 26
3.2 Dataflow analysis . . . 29
3.2.1 Cyclic message traffic . . . 29
3.2.2 Rate constrained message traffic . . . 31
3.2.3 Time triggered message traffic . . . 31
3.2.4 Best effort message traffic . . . 31
3.3 Parallel potential analysis . . . 32
3.3.1 Analytical analysis . . . 32
3.3.1.1 The efficiency criterion . . . 32
3.3.1.2 Applying the efficiency criterion . . . 37
3.3.2 Empirical analysis . . . 39
3.3.2.1 Choosing a partitioning strategy . . . 39
3.3.2.2 Parallel potential measurement . . . 41
3.4 Analysis conclusion . . . 42

4 Optimization concept 43
4.1 Generic optimization . . . 43
4.1.1 Scheduling algorithm selection . . . 43
4.1.2 Conditional null message algorithm basic approach . . . 44
4.1.3 Increasing parallelism . . . 47
4.1.4 Final conditional null message algorithm approach . . . 49
4.1.4.1 Broadcast earliest internal event . . . 49
4.1.4.2 Calculate ECOT Matrix . . . 50
4.1.4.3 Calculate min ECOT Array . . . 50
4.1.4.4 Calculate phase 2 . . . 51
4.1.4.5 Final algorithm . . . 52
4.2 Domain specific optimization . . . 53
4.2.1 Weak point of the generic optimization . . . 53
4.2.2 Problem-solving approach . . . 53
4.2.3 Basic thoughts towards a generally applicable domain specific scheduling strategy . . . 54
4.2.3.1 Asymmetric event generation . . . 55
4.2.3.2 Processing delays . . . 56
4.2.3.2.1 Fixed processing delays . . . 56
4.2.3.2.2 Variable processing delays . . . 56
4.2.3.2.3 Keeping track of events . . . 58
4.2.4 Final domain specific optimization approach . . . 58
4.2.4.1 Scheduler architecture . . . 58
4.2.4.2 Obtaining a global view on all possible external events . . . 59
4.2.4.3 Event tracking . . . 60

5 Concept implementation **62**
5.1 The OMNET++ parallel simulation subsystem 62
5.2 The conditional null message algorithm scheduler 63
5.2.1 Basic approach implementation 63
5.2.2 Distributed lookahead calculation 64
5.2.3 Basic synchronisation 66
5.2.4 Increased parallelism implementation 66
5.3 Domain specific optimization realization 68
5.3.1 The TTE event pipeline 68
5.3.2 The CAN event pipeline 68
5.3.3 Exploiting the CAN event pipeline 69
5.3.4 Implementation of the domain specific approach in Partition 1 70

6 Test **73**
6.1 Verification of operation 73
6.2 Efficiency Comparison 73
6.3 Speedup Comparison 77
6.4 Why the domain specific approach lacks in performance 78

7 Final **80**
7.1 Summary 80
7.2 Conclusion 82
7.3 Outlook 83

Glossary **90**

Listings

5.1 Calculating and broadcasting local lookaheads 64
5.2 Calculating and broadcasting local lookaheads 65
5.3 Basic synchronisation algorithm. 66
5.4 Final earliest input time (EIT) algorithm implementation including phase 2 calculation according to section 4.1.4.5. 67
5.5 Intercepting CAN events arriving at the transform module. 70
5.6 Intercepting CAN events arriving at the transform module. 71
5.7 Reporting event ID changes to the scheduler. 72

1 Introduction

The Communication over Real-Time Ethernet Group (CoRE)[1] is conducting research on communication solutions for time-critical applications using the Ethernet technology. New communication concepts are developed and analysed by using discrete event based simulations in OMNET++[2].

Discrete event based simulations become a wide spread tool in constructing and developing in vehicle networks. Networks in general and in vehicle networks in particular, face a steadily increasing complexity [1, 2, 3]. This results inevitably in increasing simulation runtimes. As a consequence, simulation based development processes are slowed down [4, 5].

A possible solution to this problem lies in distributing the processing load by splitting up the simulations into several logical partitions and processing them in parallel. Computer scientists are working in this field of research since the early 1970s. Before the recent success of multicore systems, researchers concentrated on distributing the simulation partitions over several sequential systems, in order to decrease simulation runtimes. The recent advance of multicore processors has once again brought parallel simulation into the focus of current research [5, 6].

A major difficulty in using parallel simulation is that it does not automatically result in faster execution times [5, 6]. Even when done right, not every simulation can profit from parallel simulation [7]. The success of using parallel simulation heavily depends on the selection of parallelization technology as well as the underlying simulation's parallel capabilities.

The main goal of this work is to develop parallel simulation strategies that to speedup in vehicle network simulations. The approach should be generally applicable to a range of simulations similar to the ones investigated as part of this work. Therefore, two typical in vehicle network simulations, developed by the CoRE group, were chosen for investigation.

Since parallel simulation is a subject of scientific research since the 1970s, a lot of parallel simulation approaches have been developed in the past. However, since today there is no general approach towards parallel simulation [8]. Therefore this work starts with acquiring the theoretical foundations and a thorough evaluation of the current state of research in parallel

[1] http://core.informatik.haw-hamburg.de/
[2] https://omnetpp.org/

simulation. The first aim is to select an existing parallel simulation approach as basis for further optimizations. The next step is an analysis of the two simulations and their parallel potential. Based on the results of the analysis results and the parallel potential, scheduling strategies will be developed. The scheduling strategies will then be implemented for the OMNET++ simulation system.

This work finishes with a validation of correct operation and a performance benchmark against the sequential and null message algorithm.

2 Theoretical foundations and related work

This chapter gives an introduction to the theoretical foundations that this master thesis is based on. It starts with an introduction to simulation in 2.1 followed by a brief overview of existing simulation classifications in 2.2. Different time domains will be defined in section 2.3. Discrete event based simulation (DES) will be explained in section 2.4, followed by an introduction to parallel discrete event based simulation (PDES) in section 2.5. Section 2.6 will give an overview of existing synchronization algorithms needed for conservative parallelism. Section 2.7 presents the optimistic synchronization approach and section 2.8 presents the conservative approach. It will also give a detailed explanation on how the most commonly used null message algorithm (NMA) works. The NMA will serve as reference algorithm for performance optimizations done in this work. Detailed overviews on the topic of PDES are given in summaries in [8, 9, 10, 11, 12, 13, 14, 15, 16]. These summaries were created by different computer scientists over time with the objective to sum up the current state-of-the-art in research. This chapter closes with a brief overview of Time-Triggered Ethernet (TTE) in section 2.9.

2.1 Simulation

Simulation is the imitation of the operation of a real-world process or system over time [17]. Simulating a system requires the creation of an appropriate systems's model first. A model is a representation of a physical or theoretical system. The model may be mathematical, descriptive, logical, or some combination of these elements [9]. Models are often abstract and simplified representations of their real counterparts but their behaviour in all relevant aspects conforms to the real system. Since reality can not always be exactly reproduced in simulation, a simulation might have a deviation to the system [10]. This will result in inaccuracies of the simulation results which must be considered when using them. The purpose of simulation is to gain insight about a systems's behaviour in certain situations without having the real system at hand. It also makes it possible to do the experiments which could be too dangerous or might damage the system system [10]. Simulations are often used to verify or validate a system

during development process before crafting the real system in order to detect failures early during development process. Typical use cases for simulations according to [10] are:

- *Optimization* of system behaviour (for example by bottle neck detection)
- *Decision aid* during system conception
- *Prediction* of system behaviour
- *Verification* of system behaviour
- *Validation* of theories and system behaviour
- *Animation* of system behaviour

2.2 Classifications

Since the examination of a systems's temporal behaviour plays an important role for most dynamic systems, the modelling of time based processes plays an important role in simulation [10]. This lead to the introduction of different simulation classifications shown in Figure 2.1. The two major classifications are continuous and discrete simulations.

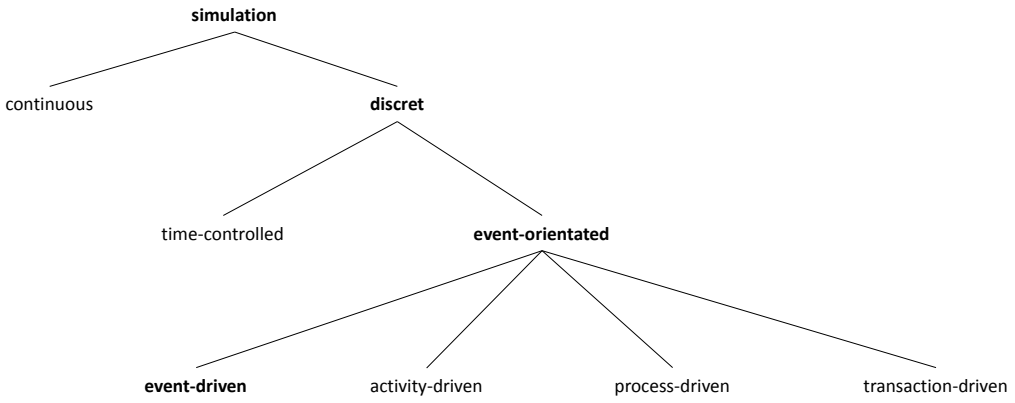


Figure 2.1: Simulation classifications [10].

In continuous simulation one assumes, that the model is changing its state continuously over time. Typical examples are the simulation of star systems or the simulation of electrical circuits and control loops. Common simulation system implementing continuous simulation are MathWorks MATLAB and Simulink.

Discrete simulation models change state at discrete points in time. The simulation system OMNET++ and therefore, all simulations developed by the CoRE research group are based on DESs. For this reason, this work will only attend to DESs. A detailed overview and further explanations on different simulation classifications are given in [10].

2.3 Time in simulation

Time is an essential part in simulating the temporal behaviour of dynamic systems. This section introduces two important time domains, that will be used throughout the remainder of this work. The simulation time denotes the time, that is passing inside the simulation, whereas

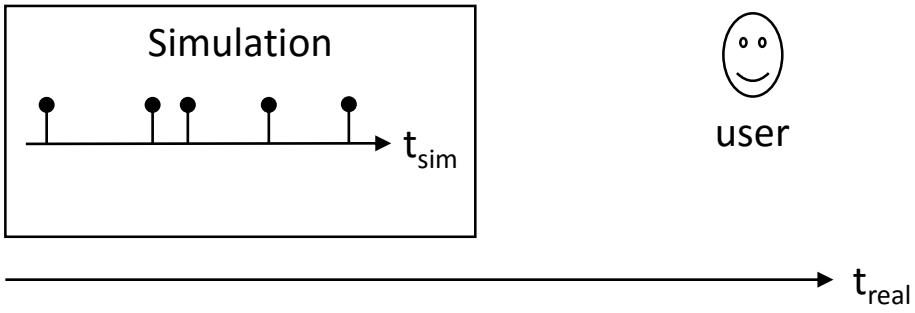


Figure 2.2: Time domains in simulation.

real time denotes the time, that is passing outside the simulation as shown in figure 2.2. Both time domains are defined from a user's point of view. For example simulating the behaviour of a star system for a timespan of five years, might take 5 minutes of real time, whereas for the simulation, five years of simulation time pass from a user's point of view.

2.4 Discrete event based simulation

In DES, the model is changing its state at discrete points in time. These discrete points in time are modelled as events, hence the name DES. Processing of an event happens immediately without time costs. Between two discrete points in time, there will be no change in state. For this reason, the simulation can, in contrast to continuous simulations, directly jump in time from one event to the next event. Figure 2.3 gives an example of discrete simulation time advancement. In this example, three events are shown with execution timestamps at $t = 1$, $t = 3$ and $t = 10$. Therefore, the simulation jumps to $t = 1$, processes the first event and then

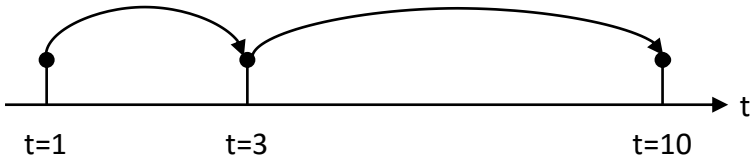


Figure 2.3: Advance in simulation time by jumping from one discrete event to the next.

advances to $t = 3$ in order to process the second event. At last it will advance to $t = 10$ and process the third event. When there are no more events left, the simulation terminates.

Events must be processed in time correct order. Processing events in time correct order is realized by an event scheduler. An event scheduler possesses an event queue, the future event set (FES) that stores events in time correct order. The event scheduler will always remove the event on top of the FES, that is the event with the smallest timestamp and process it. Processing an event might result in new events that must be added to the FES and carry a timestamp greater or at least equal to the currently processed event. In case an event timestamp is smaller

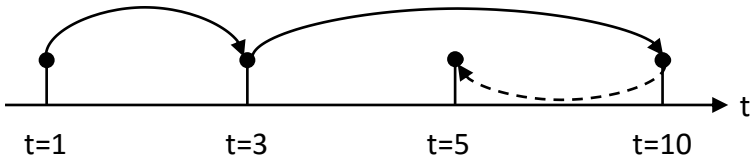


Figure 2.4: Causality violation caused by event at t=10 by scheduling a new event at t=5 which is in the past.

than the currently processed event, the newly created event is scheduled in the past as shown in Figure 2.4. This is called a causality violation because the newly created event at $t = 10$ might change the past, thus invalidating the results of the event at $t = 10$. In case of a causality violation, the simulation must abort with an error.

The growth in model complexity leads to ever increasing simulation times [11]. In the past years this was compensated by using new processors with increased clock frequencies. Lately the clock frequency of modern processors does not significantly increase due to physical limitations. For this reason, the development of modern processors has stepped into the era of multi-core [18]. Since the whole event processing is based on a purely sequential process, classical DESs can not make use out of modern multi-core processors and do not benefit from an increased amount of processing cores. In essence, this means, processing power stagnates

for classical DESs and simulation engineers will run into performance barriers, that used to be overcome in the past by replacing old processors with newer ones with increased clock frequencies.

2.5 Parallel discrete event based simulation

In order to increase simulation performance and to decrease simulation times, computer scientists began to look for new ways to overcome the hardware induced performance limitations (see 2.4). The basic idea is, to parallelize a simulation and thus distribute the simulation load over several processing units.

This section gives an introduction to the basic idea behind PDES in section 2.5.1. The challenges of parallel simulations are discussed in section 2.5.2.

2.5.1 Basic idea behind PDES

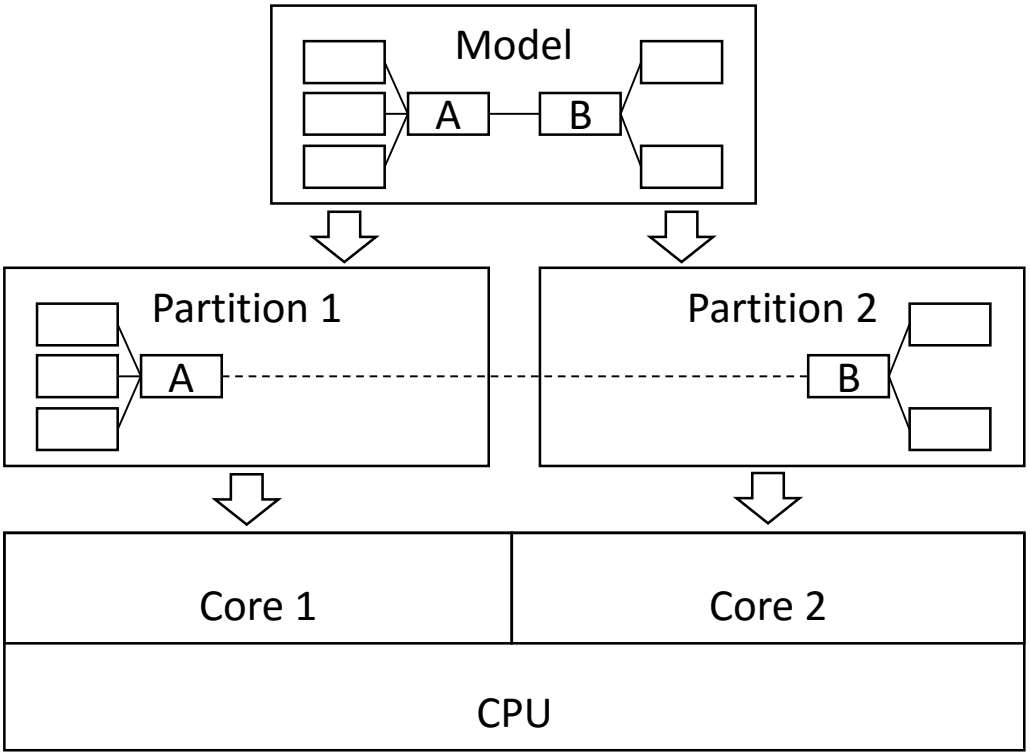


Figure 2.5: Model partitioning into two separate partitions and assignment to two separate processor cores.

The basic idea of parallel simulation is to split the model into independent parts that run in parallel as shown in Figure 2.5. These parts are called partitions. The process of splitting a model into independent parts is called partitioning. Each partition is running inside its own simulation system instance and is usually assigned to a dedicated processing unit. The different partitions can be realized as separate threads or most commonly as separate processes. In OMNET++ each partition is running inside its own process. Communication between partitions is realized through interprocess communication (IPC).

2.5.2 Challenges in PDES

As stated by [8], the implementation of PDES is difficult. The reason becomes evident, when taking a look at the example in Figure 2.5. Both partitions are running inside their own simulation system. Each simulation system contains its own event scheduler and its own FES. In order to enable communication between both partitions, a communication system is established between both simulation systems to make the event exchange possible (see 2.5.1). As described in 2.4 it is crucial, that events are processed in time correct order. With both partitions working independently, this paradigm might get violated. To demonstrate the

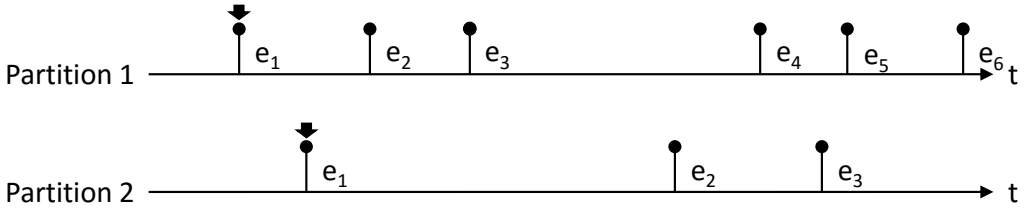


Figure 2.6: Event rising over time in partitions 1 and 2 - black arrows denote the currently processed event.

problem, the FES of Partition 1 contains two times as many events as the FES of Partition 2. This is shown in Figure 2.6. Furthermore, the following assumptions are made:

- Both partitions start at the same time.
- Both partitions process events at the same speed.
- Processing an event takes an equal amount of time for both partitions.

Since the event density inside FES 1 is two times higher than in FES 2, the effort needed to advance Partition 1 the same amount in time as Partition 2, is two times higher. For this reason, it will take Partition 1 two times longer in real time to arrive at the same point in simulation

time. In other words, Partition 2 will advance in simulation time much faster than Partition 1. As shown in Figure 2.7 processing of e_3 in Partition 1, causes a new event being sent to

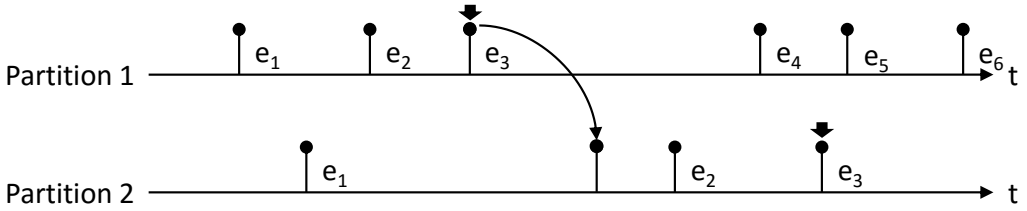


Figure 2.7: Causality violation caused by partition 1 sending an event to partition 2, which advanced in time way ahead of partition 1 - black arrows denote the currently processed event.

Partition 2. This new event is scheduled between e_1 and e_2 in the FES of Partition 2. Since both partitions process events at the same speed, Partition 2 already processed its third event. That means, the newly created event from Partition 1 lies in the past of Partition 2. Therefore, Partition 1 produced a causality violation.

To prevent causality violations and make parallel simulation practicable, some kind of time management is needed that coordinates and synchronizes the partitions.

2.6 Synchronization

Synchronization denotes the process that prevents causality violations from occurring or corrects them in PDES. Synchronization algorithms can be broadly categorized into two major categories: optimistic algorithms and conservative algorithms [19, 20]. The performance of PDES depends on a wide variety of factors. These include: the partitioning, the communication overheads of the parallel platform (both hardware and software overheads), and the overheads of the parallel synchronization algorithm [19]. Depending on the models architecture, a conservative algorithm might be more applicable or faster than an optimistic approach and vice versa. Therefore, the decision for one or another approach is not made easy.

2.7 Optimistic Algorithms

Optimistic algorithms do not actively prevent causality violations. Each partition will process events as fast as possible. Whenever a partition detects a causality violation, which is caused by receiving an event with a timestamp smaller than its current simulation time, the partition must perform a roll back of the simulation results and states. This roll back is also known as

time warp. The partition then restarts the simulation and reprocesses the events in timestamp order [12].

2.7.1 Time Warp Algorithm

One of the first optimistic algorithms in parallel simulation was published by D. Jefferson in 1982 [21, 22]. Figure 2.8 shows Partition 1 causing a causality violation by sending an event to

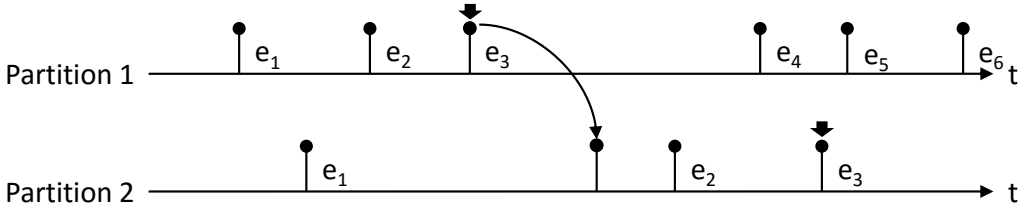


Figure 2.8: Causaltity violation caused by partition 1 (see also 2.4) - black arrows denote the currently processed event.

Partition 2. The timestamp of the event is way behind the current simulation time of Partition 2. When Partition 2 detects the causality violation, the time warp algorithm performs a roll back by resetting the partition state back to what it was at the time of the newly received event as shown in Figure 2.9. This roll back is called time warp, hence the name time warp algorithm.

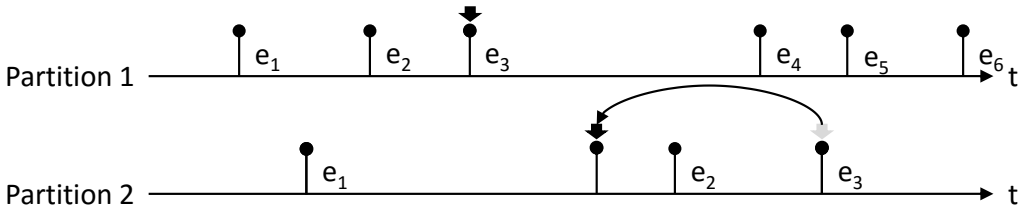


Figure 2.9: Time warp algorithm resets the model state to the point in time that is denoted by the newly received events timestamp - black arrows denote the currently processed event.

After roll back, the partition restarts simulation from that state on [13].

Although optimistic approaches are considered to be able to exploit a higher degree of parallelism [12, 23], they suffer a phenomena that poses a negative threat on simulation performance. Depending on the underlying model, time warps might initiate further time warps in neighbouring partitions which in turn might again cause new time warps [11]. This effect is called cascading time warps. Simulations experiencing this effect, will have a severely reduced simulation performance [24, 25]. The simulation performance might even drop below

that of its sequential counterpart. A performance analysis of time warp simulation with cascading rollbacks is presented in [25].

Besides the negative effect of time warps on performance, there is another important aspect to consider before using an optimistic algorithm. Optimistic algorithms are considered to be too complex or impractical to implement in practice [26]. The implementation of the time warp mechanism takes a considerable amount of effort. Simulations and simulation systems that were built without parallel simulation in mind, lack the mechanisms to realize roll backs. Using third party simulation libraries might make it impossible to create roll backs due to the possible lack of source code.

2.7.2 Time warp implementation

One approach to implement time warps is presented in [11, 22]. The authors describe the implementation of time warps by using negative events. For each event a negative counterpart exists. If both events happen to be in the same FES, they annihilate each other. In order to perform a time warp, the necessary negative events will be processed as shown in Figure 2.10.

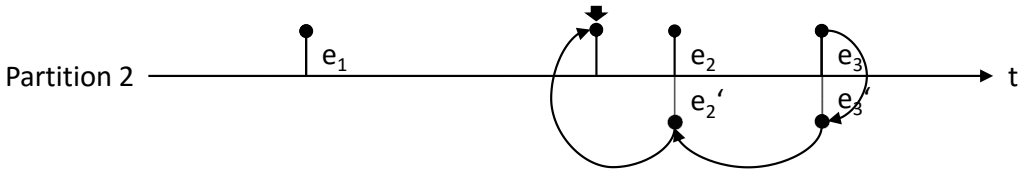


Figure 2.10: Performing a time warp by processing the negative events e_3' and e_2' to events e_3 and e_2. After reset, the simulation continues with the newly received event that caused the causality violation - black arrows denote the currently processed event.

2.7.3 Global virtual time

An important concept of the time warp is the introduction of global virtual time. Global virtual time is a lower bound on the timestamp of any future time warp that might occur in optimistic simulation. Time warps are triggered by receiving events with timestamps in the past. For this reason, the smallest simulation time over all partitions gives a value for the global virtual time. In [22] it is proved that rollbacks will not go further back than the global virtual time. Therefore, this discovery is exploited in many optimization approaches of optimistic simulations.

2.7.4 Optimization of optimistic algorithms

Optimistic algorithms suffer from two major problems. Cascading time warps have a negative impact on performance. The time warp algorithm requires extra memory to keep track of the simulation history to perform a time warp on detection of a causality violation. Therefore, two major branches in research try to reduce the occurrence of cascading time warps and to find efficient ways to reduce the memory footprint.

The remainder of this section presents current optimization approaches in optimistic algorithms grouped by optimization strategy.

2.7.4.1 Reducing memory footprint

Optimistic algorithms require a considerable amount of memory to safe model states and to keep a history of previously processed events. These informations are needed to perform a time warp in case a causality violation is detected. Without optimizations in memory management, the simulation will consume more and more of the limited available memory [12].

One strategy known as fossil collection deals with freeing no longer needed history information. It uses the global virtual time to free all the history information that is older than the global virtual time [12, 16]. As stated in 2.7.3 the simulation will never roll back to a simulation time earlier than global virtual time. Therefore, all history information with a timestamp smaller than the global virtual time can be considered fossil and is no longer needed. [27] presents a fossil collection algorithm that is using fixed time windows in which history is kept. It does not depend on the global virtual time and therefore makes the global calculation of the global virtual time unnecessary. In [28] the same authors map optimism to fossil collection in a time warp simulation. Further approaches are presented in [29, 30].

Another approach to overcome the problem is the cancelback protocol discussed in [31]. If the system runs out of memory, processed events that lay too far in the future are rolled back, thus freeing memory. Similar approaches are presented in [32, 33].

State saving in intervals rather than after each event is another approach to save memory and is described in [34, 35].

2.7.4.2 Preventing overly optimistic execution

Preventing overly optimistic execution reduces the probability for time warps and also reduces the memory footprint for saving the simulation history.

One possibility is to exploit the global virtual time to prevent overly optimistic execution. Basically these approaches ensure that none of the partitions simulate too far into the future

from global virtual time. An early technique uses a sliding window of simulated time which is defined as $[GVT, GVT + W]$ where W is a user defined parameter [12]. This approach was published in [36]. It prevents any partition advances further than $GVT + W$. Further similar approaches are listed in [12].

2.7.4.3 Improved cancellation

Another approach to improve optimistic simulation is to improve the cancellation strategy. Basically cancellation strategies are distinguished between lazy and aggressive algorithms. Aggressive cancellation means that upon detecting a causality violation the partition performs a roll back and immediately sends out cancellation messages for all messages that were processed prematurely. Lazy cancellation in contrast, performs a roll back and holds back cancellation messages until it is proved that previously sent events were incorrect. In [37] the authors present a dynamic switching strategy between lazy and aggressive cancellation in order to improve simulation performance.

In [38] the authors propose to implement inverse events to perform time warps. For each event an inverse event must be implemented that undoes the actions of its positive counterpart. The aim of this approach is to reduce the memory footprint by avoiding state saving. The authors also present an inversion compiler that automatically generates inverse events out of their positive counterparts.

2.7.5 Performance estimation of optimistic algorithms

The performance of time warp algorithms is often difficult to predict [25]. Little work on the performance issues of time warp algorithm schemes has been presented so far [39]. In [40] the authors present a performance analysis of "'Time Warp"' with limited memory. They state that prior work on time warp algorithms never considered that memory is only available in limited amounts. Therefore, they present an analytical model of time warp. This model is using the cancelback protocol and assumes a limited amount of memory for storing history data. They showed that the time warp algorithm performs considerably well even with a limited amount of memory. However, they assume a homogeneous distribution of events which might not be the case in realistic simulations and heavily depends on the partitioning strategy. In some approaches, the optimistic simulations create periodic checkpoints in order to roll back to an earlier state in time. In [41] the authors investigate the effects of varying the frequency of checkpointing on the time and space needed to execute a simulation. They prove the assumption that varying the checkpoint frequency has a considerable effect on the

amount of memory needed to save checkpoints and on execution times as well. The cancelback protocol is investigated in [39, 42]. The authors research the effect of limited memory and memory management on simulation performance similar to [40].

However, there are no approaches present so far to calculate the parallel potential of a sequential simulation when using an optimistic scheduling algorithm. Without knowing the parallel potential it is hard to decide whether an optimistic approach will bring the desired increase in performance. This might also be a reason why optimistic parallel simulation has not yet found a wide spread use in the field of DES.

2.8 Conservative Algorithms

In contrast to optimistic algorithms, conservative algorithms are built to prevent causality violations to happen. The first synchronization algorithms were based on conservative approaches [12]. For example, suppose a partition processed an event e_1 at $t = 10$ and therefore, is at simulation time 10. Furthermore, it is ready to process the next event e_2 at $t = 15$. How does the partition know, that it will not receive an event with a timestamp smaller than $t = 15$ from a neighbouring partition after processing event e_2 as shown in Figure 2.11? In summary

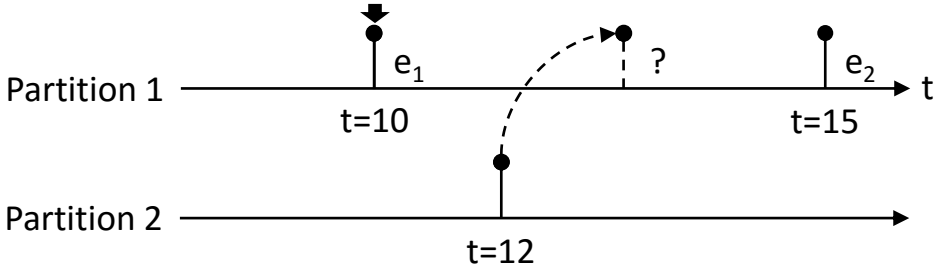


Figure 2.11: Partition 1 must know if it will receive any events with a timestamp smaller than $t = 15$ before processing e_2. Otherwise a causality violation might occur (see also [12]).

the main objective of conservative algorithms is to determine when it is "'safe"' to execute an event. That is, when it can guarantee that the partition will not receive any further events with a smaller timestamp at a later point in time. As a general rule, partitions must not process an event before it has been guaranteed to be safe [12].

2.8.1 Null Message Algorithm

One of the first conservative synchronization algorithms was the Null Message Algorithm [8, 12]. It was presented 1977 in [43] and 1979 in [44]. Further explanations of the null message

algorithm can be found in [8, 12, 19]. The basic principle of function will be explained in this section and is based on [12].

Explaining the NMA requires the definition of internal and external events. Internal events are events created and processed by the same partition. External events are events created by one partition, sent to another partition and processed by that partition.

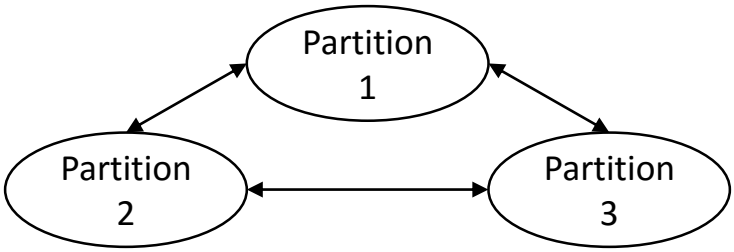


Figure 2.12: Example simulation consisting of three partition using a bidirectional links based on an IPC mechanism for event exchange.

The example simulation shown in Figure 2.12 consists of three partitions. The following assumptions are made:

1. According to 2.4 and 2.5 both partitions process events in increasing timestamp order.
2. Each partition will only send events to its neighbouring partition in increasing timestamp order.
3. Receiving an event with timestamp t from the neighbouring partition guarantees that there will not be another event received with a timestamp smaller t.

The following rules apply for each partition: external events received on one link must be stored in time correct order. As soon as the partition received at least one external event on each link, it can select the link containing the event with the smallest timestamp called t_{low}. This timestamp is called EIT [19]. It denotes the earliest time a partition can expect an external event to arrive from its neighbouring partitions. According to assumptions two and three, there will be no further events received from any neighbouring partition, with a timestamp smaller than the EIT. For this reason, the partition can process all internal events with a timestamp smaller than the EIT.

This approach successfully prevents causality violations to happen but in its current state it is prone to deadlocks. Suppose every partition starts with at least one internal event inside its FES. Processing this internal event will cause sending the first external events to neighbouring partitions. In order to process the first internal event the partitions must determine whether

it is safe. Before it can do this, it must have received at least one external event on each link. As stated before, the algorithm needs to receive at least one external event on each link to determine which events are safe to be processed. As a result, the partitions wait for each other to receive external events and the simulation is in deadlock state.

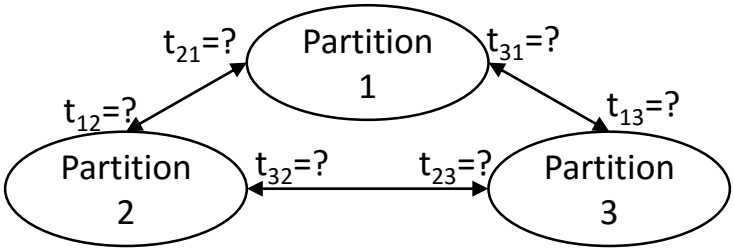


Figure 2.13: The simulation is in deadlock since the partitions do not receive external events on their links. t_{xy} denotes the time an event was last received on partition x, sent from partition y.

Null messages are used to solve deadlock situations. A null message with a timestamp t_{null} is a promise of the sending partition to the receiving partition that it will not send an event with a timestamp smaller than t_{null} at the same link in the future. This timestamp is called earliest output time (EOT) [19]. It denotes the earliest point in time, at which a partition will send an external event to its neighbouring partitions. Null messages are processed like ordinary events except, they do not cause any changes to the simulation or lead to any activity inside the simulation. The null message just contains a timestamp that is used by the receiving partition to determine which other events are safe to process. Suppose Partition 2 knows it will not send an event to Partition 1 with a timestamp earlier than $t_{21} = 5$ and that Partition 3 will not send an event to Partition 1 with a timestamp earlier than $t_{31} = 10$. Both partitions will then send a

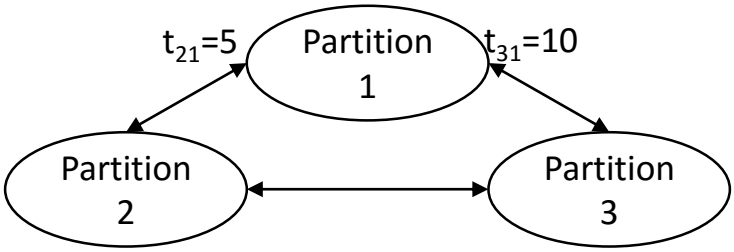


Figure 2.14: Partitions 2 and 3 send a null message to Partition 1 promising not to send any further events with a timestamp smaller their null messages timestamp.

null message to Partition 1 including the corresponding timestamp as shown in Figure 2.14. As

Partition 1 now received external events on both links, it can determine the events that can be processed safely, which are all events with a timestamp smaller than $t_{21} = 5$.

A question that arises is: How does a partition determine the timestamp of the null message it sends to its neighbouring partitions? If a partition is at simulation time t and it can guarantee that it will not send a message earlier than $t + L$, it can send a null message containing $t + L$ as timestamp. L is called lookahead. In other words: Lookahead is a lower bound on the duration after which the partition will send a message to another partition [19]. Reasons for a lookahead of length L could be internal processing time of the partition or the simulation time that is needed to exchange an event between two partitions. Lookahead is a key property by virtually all conservative synchronization algorithms [12]. It plays a vital role for the performance of parallel simulations and therefore, choosing an insufficient lookahead might lead to severe performance penalties.

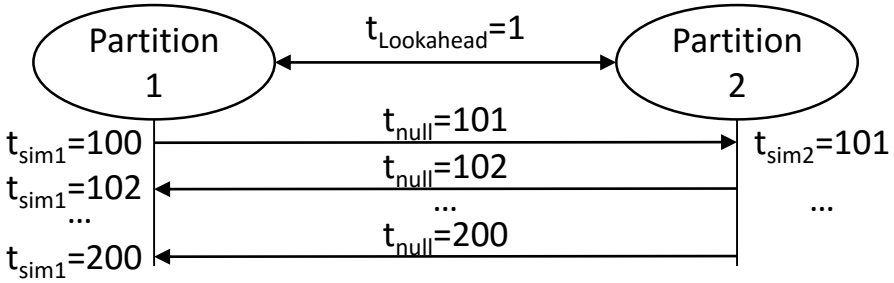


Figure 2.15: Typical example for the major drawback of the null message algorithm. Assuming a lookahead of 1 and a current simulation time of 100, it takes 100 null messages to advance the simulation time of Partition 1 to process the event at $t_e = 200$ safely.

A major drawback of the NMA is that it might create an excessive amount of null message [11, 45]. Consider a simulation consisting of two partitions. Let the exchange of an event between both partitions take the time $t_{transfer} = 1$ in simulation time and the lookahead therefore be $t_{lookahead} = 1$. Furthermore, both partitions are at simulation time $t = 100$ and the next event to process is at simulation time $t_e = 200$ in Partition 1 as shown in Figure 2.15. To advance the simulation time of Partition 1 from $t_{p1} = 100$ to $t_{p1} = 200$ the Partition 1 will start exchanging null messages with timestamps 101, 102, 103 and so on as shown in Figure 2.15. When the null message with timestamp 200 is received by Partition 1 it can finally process the event. In summary to process one single event it takes an exchange of 100 null message which has a severely negative effect on parallel simulation performance.

2.8.2 Optimization of conservative algorithms

One branch of research in conservative algorithms is working on improving the parallel simulation performance. Therefore, many scientific publications revolve around reducing the amount of null messages since they pose the major drawback of this algorithm [11, 46]. Until today there exists no generalized approach towards optimizing conservative simulation. Synchronization is a non trivial task and the different approaches presented in various publications are especially tailored to the nature of the underlying simulation. Therefore, speed-up varies considerably, depending on aspects of the simulation model [8].

The principal problem is that the algorithm uses only the current simulation time of each partition and lookahead [12]. It uses just these two informations to calculate a timestamp for its null message to tell other partitions about the minimum timestamp of messages it might send in the future. If both partitions of the previous example (see Figure 2.15) were able to recognize that the next event to process had timestamp 200, they could have advanced simulation time immediately to $t = 200$.

The remainder of this section presents current optimization approaches in conservative algorithms grouped by optimization strategy.

2.8.2.1 Conditional null message

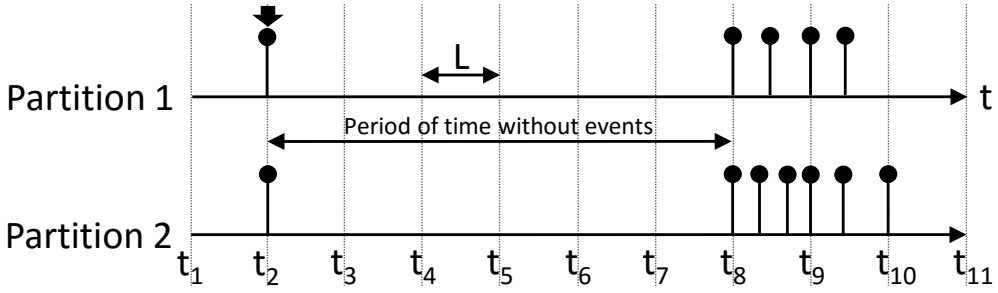


Figure 2.16: Simplified representation of message traffic with a larger period in simulation time without any simulation events. Vertical dotted lines represent the lookahead L between Partition 1 and 2 which is also the interval in which null messages are exchanged.

Figure 2.16 shows a typical event emergence that causes an excessive amount of null messages. In this example, events at t_2 will be processed in parallel. Then comes a period of time without any simulation events. In order to advance the simulation time to the next processable events at t_8, both partitions exchange null messages and advance the simulation time in steps, the size of the lookahead L. In this example it requires six null messages per partition to reach

t_8 without processing any meaningful simulation events. At t_8 the partitions can continue processing events in parallel again.

One approach that aimed at improving the NMA is the conditional null message algorithm (CNMA) first presented in [47]. The main idea of this approach is to let the partitions alternate between a common EIT computation phase and a common event processing phase [19]. During EIT computation phase, the partitions cooperatively compute a global EIT and then switch to the processing phase. In order to do this, each partition computes its earliest conditional output time (ECOT). ECOT is the earliest time, a partition will send an external event to its neighbouring partitions, assuming it will not receive any further external events with a lower timestamp. The global EIT is then calculated as the minimum ECOT of all partitions p.

$$EIT = min(ECOT_p) \tag{2.1}$$

During processing phase the partitions process all events with a lower timestamp than the computed EIT and within the current lookahead.

For example assume both partitions just finished their processing phase at t_2 as shown in Figure 2.16. They now enter the ECOT computation phase. According to Figure 2.16 Partition 1 and 2 both have their next events to process at t_8. Without any internal events with a timestamp smaller than t_8, both partitions will not send any external events earlier than t_8. Therefore, under the assumption that Partition 1 and 2 will not receive any further external events with a timestamp smaller than t_8, $ECOT_{P1} = ECOT_{P2} = t_8$. As a consequence, the ECOT computes to be t_8. Both partitions can safely advance their simulation time to t_8 without exchanging further null messages. From t_8 they can both process all remaining events with a timestamp smaller than $t_8 + L$ in parallel (same logic as for the NMA in section 2.8.1). After that, processing phase is complete and they must enter ECOT computation phase again.

2.8.2.2 Carrier null message

The carrier null message approach was introduced by Cai and Turner in [20]. Its main purpose is to reduce the amount of null messages and to increase lookahead by exploiting the underlying model network topology. Wood and Turner presented an advanced version in [48].

The basic idea is that null messages carry additional informations including route information and lookahead information [48]. This way partitions get a global view on the current simulations state. The route information keeps a record which partition the null message has passed since its creation. The lookahead information enables the receiving partition to calculate a lower bound on the time of the earliest possible message which can subsequently occur

within those partitions passed. Therefore, using this mechanism would reduce the amount of null messages by expanding the lookahead.

The carrier null message algorithm is an interesting approach since it could also be used to propagate an extended lookahead. For example if both partitions of the example in Figure 2.15 knew that the next event was scheduled at simulation time $t = 200$, both partitions could advance their simulation time immediately without sending further null messages.

An interesting addition to this approach is described in [49] in which they propose a technique called cooperative acceleration [46]. The mechanism allows partitions to cooperatively advance in simulation time by exchanging information about the timestamp of the next definite event to process. Once all partitions agree to the proposed timestamp, they collectively advance their simulation time.

These algorithms were developed with small lookahead simulations in mind.

2.8.2.3 Conservative time windows

The moving time window approach was first introduced in [50]. The idea is to define a time

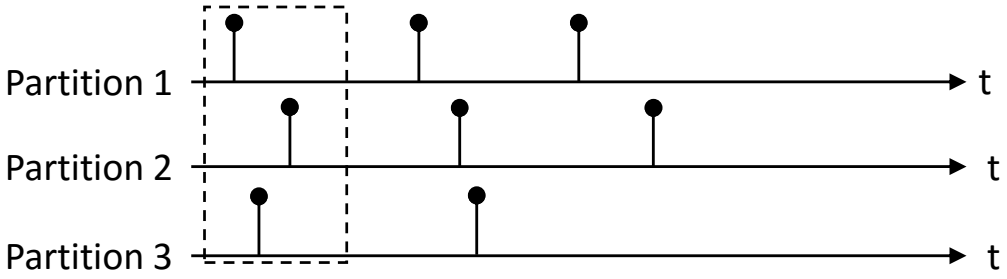


Figure 2.17: Moving time window shown in dotted lines whose inner events are safe to process.

window marking the events that can be processed in parallel as shown in Figure 2.17. The lower edge of the time window consists of the lower bound on timestamps of all events still to process. The upper edge or window size is given by the simulation programmer or by performing runtime analysis of the simulation. All events with a timestamp inside the moving time window are considered to be safe and can be executed in parallel.

An empirical performance study is performed in [51] and suggests the dynamic adjustment of the time window size at runtime in order to further increase performance.

2.8.2.4 Null message cancellation

The basic idea of null message cancellation is described and analyzed in [52]. The main idea is to cancel the processing of a null message if it is overtaken by another event with a higher timestamp. In case a partition receives an event, it will cancel or flush out all previously received and queued null messages with a smaller timestamp.

2.8.2.5 Demand driven

An early adaption of the NMA is the Shared Ressource Algorithm for Distributed Simulation described in [53]. The basic idea is that the null message generation is demand driven. Whenever a partition runs out of events on one of its links, it will actively request a null message. Although this algorithm aims at reducing the amount of null messages it has performance limitations. This is because the transfer of a null messages requires twice the time due to the request and transmission mechanism. Similar demand driven approaches are presented in [54, 55].

2.8.2.6 Path lookahead

Path lookahead denotes a data oriented approach and is described in [46, 56, 57]. The basic idea is not to look at the physical links that connect the simulation partitions but at the data flow. Consider the simulation shown in Figure 2.18 consisting of three partitions. Suppose

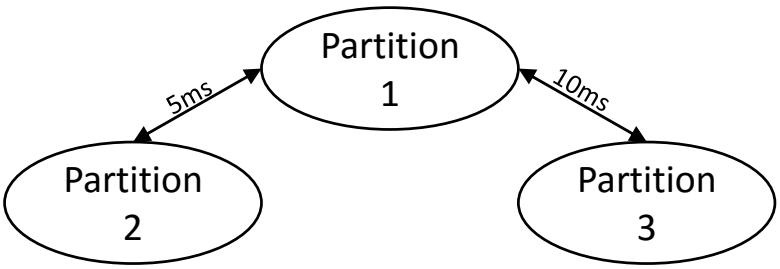


Figure 2.18: Example simulation for path lookahead.

Partition 1 is sending events at Partition 2 in intervals of $5ms$ and at Partition 3 in intervals of $10ms$. Transferring an event takes $1ms$ on each link. Therefore, using just the transfer delay of $1ms$ gives a small lookahead. Instead of using link delays for lookahead calculation, path lookahead considers the data flow. Figure 2.19 shows the data flow view of the example simulation. It shows that the simulation consists of two independent data flows. As a result the lookahed from Partition 1 to Partition 2 can be set to 5ms, whereas for the link between

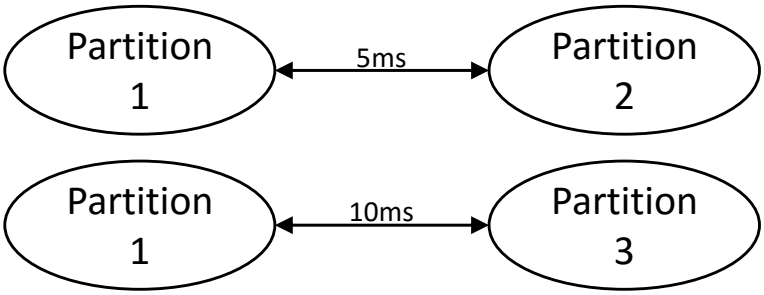


Figure 2.19: Dataflow view at the simulation.

Partition 1 and Partition 3 it can be set to 10ms. This way the lookahead is increased by factors 5 and 10, thus successfully decreasing the amount of null messages.

2.8.3 Performance estimation of conservative algorithms

Another branch of scientific research revolves around the performance estimation of conservative algorithms. Computer scientists examine the different optimization strategies for efficiency and develop further improvements based on the results. A good example is [51] in which Sokol, Weissman and Mutchler performed a performance study on the moving time window approach and improved it by allowing dynamic changes at runtime. In [26] Varga et al. present a practical efficiency criterion for the NMA that allows to estimate the parallel potential of a sequential simulation model, before investing effort into turning it into a parallel simulation. Dong et al. study the effects of partitioning in [58]. They investigate the effects of partitioning on huge systems and compare different partitioning strategies. A performance evaluation of conservative algorithms is presented in [19] by Bagrodia et al. Such performance studies should be considered with care since the performance of the different algorithms always depends on the underlying model that was simulated. Therefore, the results should not be used for simulation models with different architecture. [59] presents an analytical approach to determine the optimal lookahead inside a parallel simulation. This is done on a mathematical base and aims at reducing the average amount of null messages between two real events.

2.9 Time-Triggered Ethernet

Ethernet has seen a tremendous success and is a wide spread solution in networking computers and devices. Its advantages are that it is easy to use, reliable and relatively cheap to purchase. However, it was not developed with realtime applications in mind. Therefore, the deterministic

capabilities of the original Ethernet are limited. Deterministic communication requires full control of jitter, constant message latency and a repeatable message order. TTE is a standard specified by TTTech and the university of Vienna. It adds the afore mentioned capabilities to the original Ethernet technology, thus making Ethernet available for time critical and realtime applications that require deterministic behaviour. First results on TTE were published in 2005 in [**?**]. TTE also known as AS6802, enables shared communication among real-time and safety-critical applications, event-driven applications and standard Ethernet communications over the same Ethernet backbone network [**?**].

2.9.1 Communication protocols

The TTE standard basically defines three different traffic types for messages inside the network:

- Deterministic Time-Triggered (TT) traffic - synchronous communication for realtime applications with high demands on determinism.
- Event-Driven or rate-constrained (RC) traffic - asynchronous communication for applications with high demands on bandwidth reliability and availablility.
- Best-effort (BE) standard Ethernet traffic - asynchronous communication that is realising standard 802.3 traffic.

In order to assign one of these traffic types to data connections between a sender and receiver, TTE is using the virtual link (VL) concept. Each VL represents a logical link inside the Ethernet connection. To each VL a sender, a receiver and a traffic type must be assigned. The VL is then acting as communication channel with the specified traffic type and the assigned senders and receivers. The physical Ethernet link is basically split into smaller logical links with predefined traffic types as shown in Figure 2.20. The quality of service can be further enhanced by assigning priorities to RC traffic for example. The TTE logic is implemented in special switches.

2.9.1.1 Deterministic Time-Triggered (TT) traffic

TT messages are sent at defined points in time. Messages are guaranteed to be sent at these times and are used by realtime applications which are highly dependent on deterministic behaviour. The time of sending underlies a globally synchronised system time that is generated by a synchronization algorithm described in section 2.9.2. TT messages always take precedence over all other messages at all times. Typical applications might be for example steer-by-wire or break-by-wire systems.

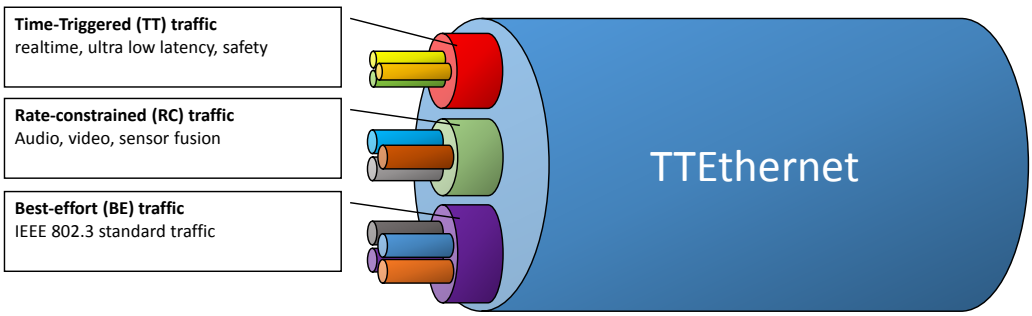


Figure 2.20: Time triggered Ethernet with many virtual links with individually defined traffic types

2.9.1.2 Event-Driven or rate-constrained (RC) traffic

RC traffic is used for applications with less stringent determinism and real-time requirements than strictly TT applications [?]. In contrast to TT messages, RC traffic does not underlie a globally synchronised system time. For RC traffic the required and predefined bandwidth is guaranteed to be available and delays and temporal deviations are within known limits. Therefore, delays might occur within these specified time limits. RC traffic has less priority than TT traffic but takes precedence of BE traffic. Typical applications might be sensor fusion or audio, video bridging.

2.9.1.3 Best-effort (BE) traffic

BE is the third traffic type in TTE. It denotes the classical Ethernet method for message transmission [?]. It does not offer any guarantees on message delivery and delivery times. Delays or loss of messages might occur depending on the current network load. BE traffic has less priority than TT and RC traffic. Therefore, it is using the free bandwidth next to the TT and RC traffic.

2.9.2 Common time base

Communication inside the TTE network follows a communication schedule, that is pre-defined at time of construction. Following this common time schedule requires that all nodes share a common time base that is necessary for timed-triggered communication (see also [60] for a comparison of time-triggered and event-triggered communication). For this reason TTE contains a system-wide clock synchronization mechanism. This mechanism consists of a startup

protocol that is responsible for establishing synchronization at startup, a clock-synchronization protocol that periodically synchronizes the network nodes and a clique-detection and resolution service to recover from network-wide transient upsets [**?**].

3 Problem Analysis

Using parallel simulation does not automatically yield performance gains. A linear increase in performance, compared to sequential execution, can also not be expected. This is due to the additional effort to send simulation messages across process borders and the additional synchronization overhead (see section 2.6) that is required [26]. Performance gains depend on a variety of factors [19]. Partitioning for example, has a significant effect on performance [58]. Using the wrong partitioning scheme will result in performance decreases. Using an optimistic scheduling approach for example, might result in bad performance, when the nature of the underlying simulation model causes a relatively huge amount of cascading rollbacks. Therefore, to benefit from parallel simulation, it is necessary to perform a careful analysis of the underlying simulation. Two simulation models, Simulation 1 and 2, are representative for the in vehicle network simulations at CoRE. This chapter presents the results from analysing these simulations. The results will be used as a decision base and as the foundation for a parallelism concept. Section 3.1 presents the two simulation models under investigation and explains the physical layout. Section 3.2 performs a dataflow analysis and gives an overview of the different message traffic types inside the simulated in vehicle networks. The parallel potential of both simulations introduced in section 3.1 will be analysed quantitatively and empirically in section 3.3. The results are summarized in the analysis conclusion in section 3.4.

3.1 Simulation model

Figure 3.1 shows Simulation 1. The model represents an in vehicle network with a TTE based backbone. It contains nine CAN buses, which are highlighted in green. These CAN buses, which are shown in red, are connected to the TTE by dedicated gateways. Each CAN bus connects at least one electronic control unit (ECU). An ECU is an embedded system inside a vehicle that is controlling one or more functions of the vehicle's systems. The gateways act as translation units between the CAN buses and the TTE. Therefore, each gateway is responsible for translating messages between its associated CAN bus and the TTE.

The TTE part of the simulation consists out of three TTE switches, seven network nodes and the afore mentioned CAN gateways. At this moment of development the network topology is

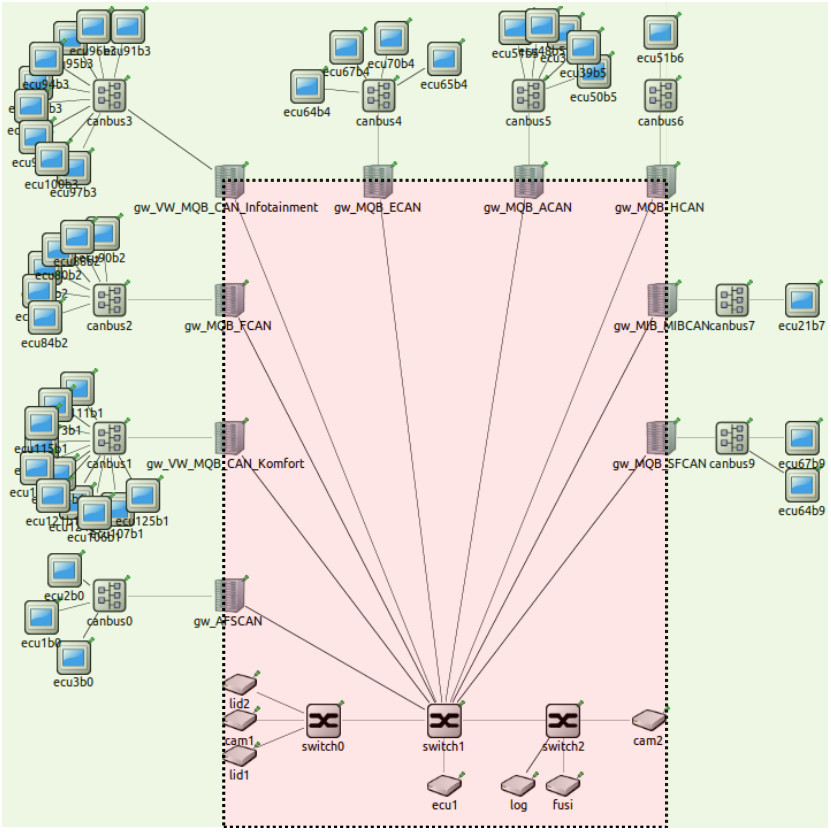


Figure 3.1: Simulation 1 - In vehicle network simulation model with TTE backbone (red) and nine controller area network (CAN) buses (green) connected via gateways to the TTE

stellar with switch 1 as central network node. Two laser distance sensor units (lid1, lid2) and one camera unit (cam1) are connected to switch 0. A logging unit (log), sensor fusion (fusi) and camera unit (cam2) are connected to switch 2. Switch 0, switch 2 all gateway units (gw_*) and a central control unit (ecu1) are connected to switch 1.

Simulation 2 is shown in Figure 3.1. It is a slightly modified version of Simulation 1. The main aim of Simulation 2 is to place the ECUs closer to their place of operation. Therefore, the CAN buses got split into two parts A and B and assigned to their own TTE switch (3 and 4).

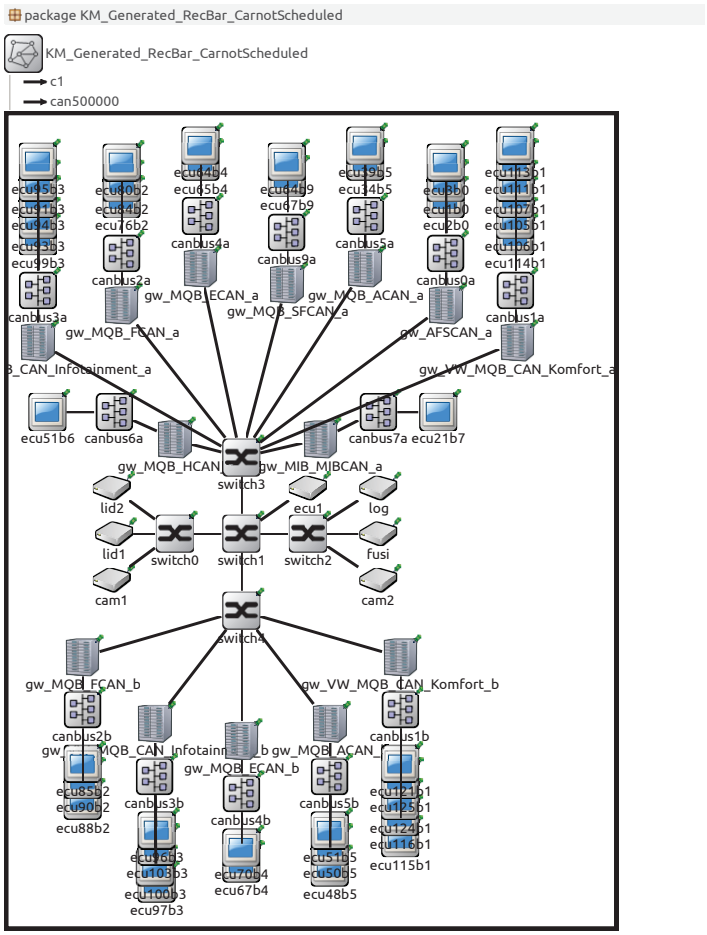


Figure 3.2: Simulation 2 - Modified version of Simulation 1 with TTE backbone and distributed CAN buses connected to two additional TTE switches to ease physical placement of ECUs in reality.

3.2 Dataflow analysis

This section presents a dataflow view of the underlying simulation. The dataflow view gives a first idea on how messages are sent inside the network and what kind of messages are sent. Understanding the dataflow is the foundation of lookahead based optimization strategies that were introduced in section 2.8.2.6.

Four different traffic types can be observed inside both simulations. Cyclic message traffic occurs inside the CAN buses. The TTE experiences all three traffic types mentioned in section 2.9. Namely:

- Rate constrained message traffic
- Time-triggered message traffic
- Best effort message traffic

3.2.1 Cyclic message traffic

Cyclic message traffic is generated by CAN nodes. The messages are sent in defined time intervals that range from 10 ms up to 2000 ms. Receivers are:

- CAN nodes on the same bus
- CAN nodes on another bus
- TTE nodes

CAN messages that are sent to neighbouring CAN nodes on the same bus, are directly transferred which is shown in Figure 3.3. CAN messages that are destined for CAN nodes on another CAN bus or one of the TTE nodes, must be forwarded using the TTE network. These messages are called heterogeneous messages. Heterogeneous message transfer is shown in Figure 3.4. The conversion of CAN messages to TTE messages is the task of CAN-TTE-Gateways (CTGs). These gateways are connected to both the CAN bus as well as to the TTE network. They collect CAN messages that must be forwarded and wrap them into TTE messages. These TTE messages are then routed forward to their destination TTE nodes and CTGs. Depending on the requirements, CTGs use RC or TT messages to forward the wrapped CAN messages. Although CAN messages are being sent in regular intervals, they are non-deterministic. Due to arbitration on the CAN bus, messages get delayed when another CAN node is already occupying the bus for sending a message.

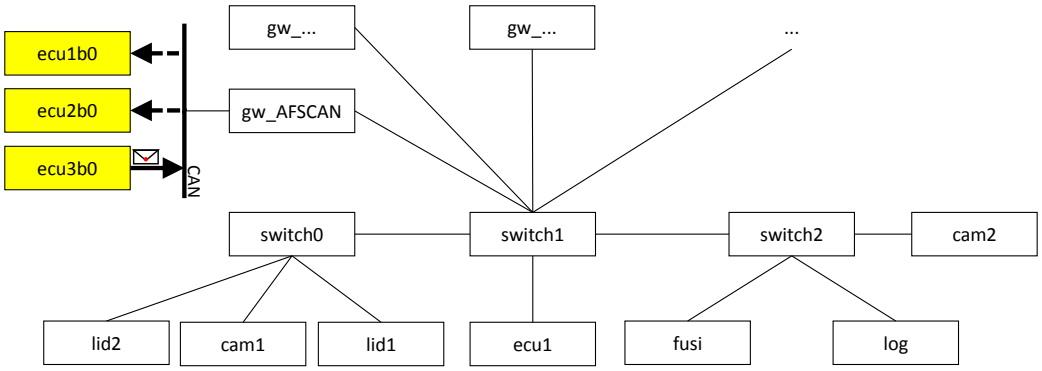


Figure 3.3: CAN node ecu3b0 sending a message to its neighbouring CAN nodes ecu1b0 and ecu2b0

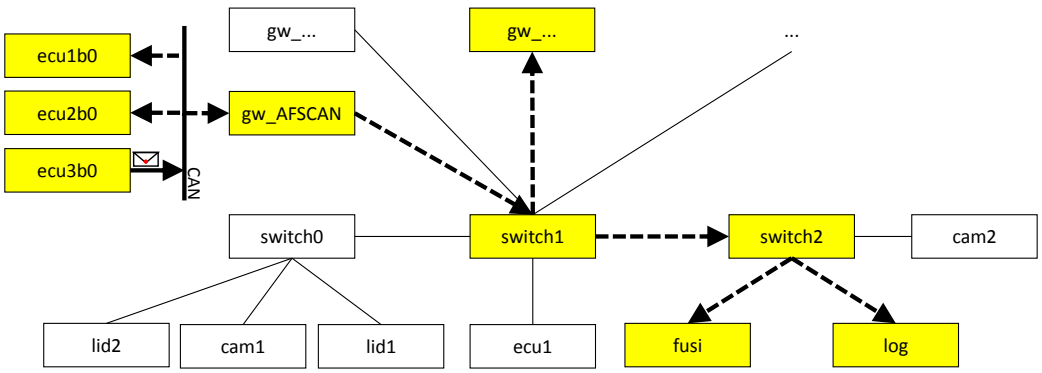


Figure 3.4: Heterogeneous message transfer from CAN node ecu3b0 to ecu1b0, ecu2b0, fusi, log and further CAN nodes

3.2.2 Rate constrained message traffic

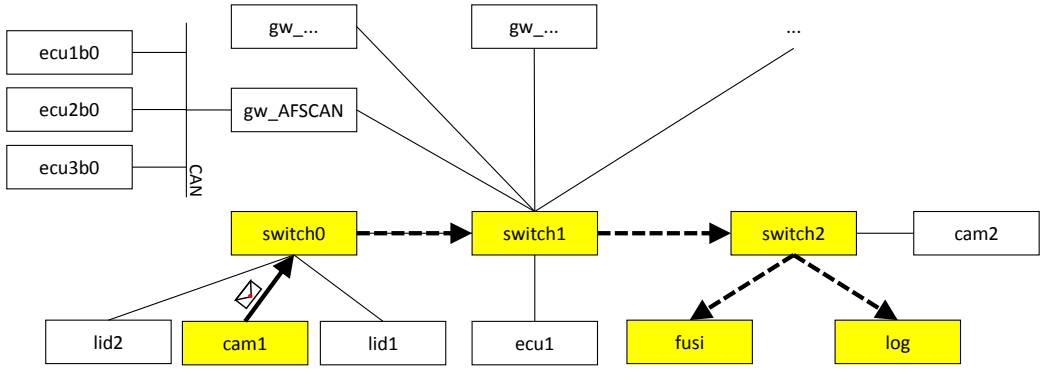


Figure 3.5: Example RC or TT message from cam1 to fusi and log

RC messages are used by TTE nodes for communication as shown in Figure 3.5. It is used by those nodes that have lower expectations on deterministic transportation but also need to transfer relatively huge amounts of data. An example node is the camera unit (cam1) shown in Figure 3.1. It is streaming its data using RC traffic. Using RC traffic has the advantage of guaranteed bandwidth (see section 2.9). Since it is non deterministic, there might be a jitter observable between RC messages that can be ignored. TTE nodes do not send any messages to CAN nodes using RC traffic.

3.2.3 Time triggered message traffic

TT messages are used by TTE nodes for deterministic communication which is shown in Figure 3.5. Therefore, TT traffic is chosen for those nodes with realtime requirements. Using TT traffic has the advantage of guaranteed transmission times (see section 2.9). Since it has deterministic behaviour, the message jitter will be kept to a minimum. TTE nodes do not send any TT messages to CAN nodes.

3.2.4 Best effort message traffic

In Simulation 1 and 2 BE traffic is used to stresstest the network and to validate that timing constraints are accomplished under heavy network loads. Therefore, BE traffic is optional and does not embody the main traffic type in both simulation's TTE networks.

3.3 Parallel potential analysis

This section presents the results of the parallel potential analysis. Before investing time into optimising a sequential simulation by parallelising it, one should have an idea of its parallel potential. This is because parallelization does not automatically yield performance benefits. In some cases it might even have a negative impact on simulation performance. Knowing the parallel potential, one can avoid the effort to optimise simulations that will just barely or not at all benefit from parallelization. Different techniques have been developed to estimate the parallel potential of a sequential simulation. The analytical analysis is using mathematical formulas to determine the parallel potential of a sequential simulation when using the NMA. The results of the analytical analysis are discussed in section 3.3.1. The empirical analysis uses specially tailored scheduling algorithms to minimize the simulation overhead as far as possible. The results of the empirical analysis are discussed in section 3.3.2.

3.3.1 Analytical analysis

In [26] the quantitative efficiency criterion is presented to calculate the parallel potential of a sequential simulation when using the NMA. This section explains the theory behind the efficiency criterion in section 3.3.1.1 and applies the efficiency criterion to Simulation 1 and 2 in section 3.3.1.2.

3.3.1.1 The efficiency criterion

The authors state, that their criterion is a hint towards a possible and not a guaranteed speed-up. Therefore, a guaranteed absolute performance gain can not be calculated with this method. The following abbreviations introduced in [26] will be used throughout the remainder of this section:

- *ev: Events (see event)*
- *sec: Real seconds (see real time)*
- *simsec: Simulated seconds (see simulation time)*

The practical efficiency criterion is based on a number of formulas which are using the following variables:

- *P performance* represents the count of events processed per second (ev/sec). The performance depends on the underlying hardware used and the average amount of computation needed to process an event.

- *E event density* is the number of events that must be processed per simulated second ($ev/simsec$). E solely depends on the simulation model.
- *R relative speed* measures the simulation time advancement per second ($simsec/sec$). Basically it tells how far the simulation advances in simulation time in 1 s of real time.
- *L lookahead* is measured in simulated seconds ($simsec$). For network simulations the lookahead is often assumed to be the link delay. At this point of time without any further optimization thoughts for lookahead, the link delay will be used for the calculation as well. The lookahead is calculated for the worst case possible. That is the smallest message possible on TTE. Therefore, a minimum message size of 64 Bytes is assumed. The transmission delay is 200 ns. The transmission speed is $100000000 bits/s$ in both simulations. The interframe delay has 96 Bits. The lookahead is then calculated as follows: $t_{lookahead} = (((64byte * 8\frac{bit}{byte}) + 96bit)/100000000\frac{bit}{s}) + 0.0000002s = 0.00000628s$
- τ *latency (sec)* is the real time it takes to exchange a message between two partitions. For this work messages are exchanged over process borders using interprocess communication. In this particular case the MPI interface[1] is used, which is considerably faster compared to named pipes or file I/O.

Actual values for these variables must be obtained from running the simulation in sequential mode. In OMNET++ these values are conveniently displayed by the IDE and can be directly read from screen while running the simulation.

In case the lookahead between partitions is smaller than the time between events, it takes more than one null message to advance in simulation time. The smaller the lookahead the more null messages must be sent. This behaviour is well known and described in section 2.8.1. Therefore, to optimize performance, the lookahead needs to be significantly larger than the time between events [26]. This relation is expressed in the following equation:

$$L >> \frac{1}{E} \tag{3.1}$$

The formula becomes reasonable when taking a look at the units.

$$simsec >> \frac{1}{\frac{ev}{simsec}} >> \frac{simsec}{ev} \tag{3.2}$$

[1] http://www.open-mpi.de

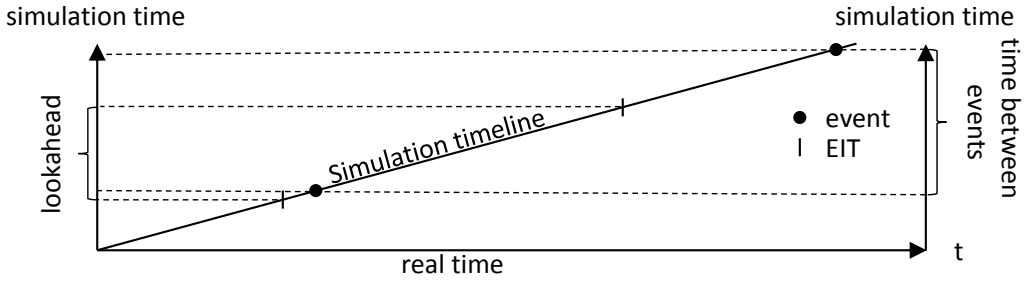


Figure 3.6: Equation 3.1: The lookahead L must be bigger than the average amount of simulated seconds that pass per event $\frac{simsec}{ev}$.

Equation 3.1 is more conveniently expressed as:

$$LE >> 1 \tag{3.3}$$

To benefit from parallelization one must ensure that partitions do not need to block at the EIT until the next null message arrives. That means partitions need to have enough events to process in their FES until the next null message arrives. To estimate whether partitions have enough events to process, the authors of [26] use the lookahead L and the relative speed R. If the advance in simulated seconds is less or equal than the time it takes until the next null message arrives, partitions do not have to block execution waiting for the next null message. Therefore, a null message must arrive before the partition reaches its EIT. Putting this into a mathematical expression, the authors assume ideal conditions for the simulation. That means, the link between partitions is idle, lazy null message sending is implemented and the simulation times of partitions advance in sync. Lazy null message sending is a technique to reduce the amount of null messages. It means that null messages are only sent on deadlock instead of sending a null message after each processed event [16, 19, 61]. A deadlock is reached, whenever the partition processed all events inside its FES with a timestamp smaller than its EIT. Under these conditions, partitions will periodically exchange null messages, which takes τ time to arrive at the receiving partition. The null message must arrive at the receiving partition, before the partition reaches its EIT that was received with the previous null message. Therefore, the latency of message transfer must be smaller than the amount of time it takes to advance by L in simulation time which is expressed by $\frac{L}{R}$. This relation is expressed in the following formula:

$$\tau < \frac{L}{R} \tag{3.4}$$

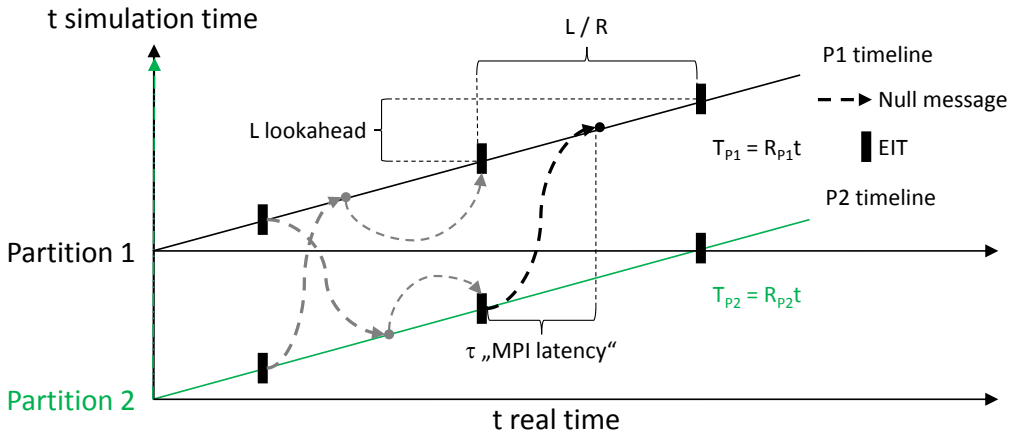


Figure 3.7: Example display of periodic null message exchange and correlation between simulation/real time, lookahead (L), relative speed (R) and earliest input time (EIT) [26]

Considering that $R = \frac{P}{E}$ this is more conveniently expressed as:

$$\tau P < LE \tag{3.5}$$

In reality, the assumption that simulation time passes evenly, does not withstand. The reason for this is because P and E tend to fluctuate. For this reason, a simulation with $\tau P = LE$ will experience frequent blocking of the simulation whereas $\tau P < LE$ allows some fluctuation of P and E without blocking the simulation. This insight let the authors define the coupling factor λ which is the ratio between LE and τP.

$$\lambda = \frac{LE}{\tau P} \tag{3.6}$$

If $\lambda < 1$, frequent blocking will occur since partitions will have to wait for the receipt of null messages. In this case one can not expect a performance benefit from parallelization. Basically one can expect that blocking is reduced, the bigger λ is. Experimental results conducted by the authors showed that λ values below 10 are "'small"' whereas values greater than 100 are considered "'large"'. The arising question is: When is a lookahead considered to be "'large"' enough to yield a satisfying performance benefit? This question is answered by the authors as follows: Lookahead is considered to be "'large"' enough when λ is greater than a λ_0 threshold that is chosen in the range of 10 to 100 so the following inequality is fulfilled:

$$L > \frac{\lambda_0 \tau P}{E} \tag{3.7}$$

In general, parallel simulations do consist out of n partitions. Therefore, Equation 3.6 needs to be extended to calculate λ for each link between two partitions separately (see also 3.8).

$$\lambda_{i,j} = \frac{L_{i,j} E_i}{\tau_{i,j} P_i} \tag{3.8}$$

To achieve good performance, all coupling factors must be sufficiently high enough. Therefore,

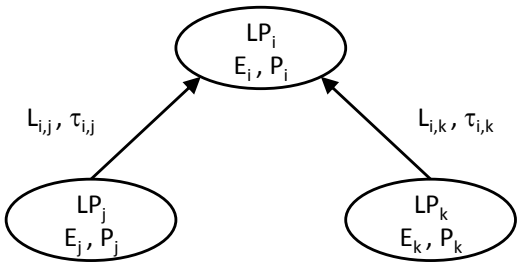


Figure 3.8: Calculating λ for different links between partitions [26]

the smallest coupling factor is the dominant constant and is defined as follows:

$$\lambda = \min_{i,j} \lambda_{i,j} \tag{3.9}$$

Another question investigated is: How does the NMA scale with the number of processors used? According to the authors, P_i and $\tau_{i,j}$ stay relatively constant with a growing number of partitions. However, the total amount of events will not change. Therefore, E_i will decrease since the total amount of events is distributed over a greater amount of partitions. Still assuming that the events are propagated evenly, the following equation is valid for the event density:

$$\min_i E_{n,i} \leq \frac{E_{seq}}{n} \tag{3.10}$$

With a decreasing event density, a partition will have less events to process between null messages. This leads to an increased chance that partitions must block and wait for the next null message because they run out of simulation events. The coupling factor will also decrease with an increase in the amount of partitions.

$$\lambda_n \leq \frac{\lambda_0}{n} \tag{3.11}$$

3.3.1.2 Applying the efficiency criterion

Applying the efficiency criterion requires the measurement of the required input variables. P and E_{seq} can be obtained from running the sequential simulation. Both values are already calculated and displayed by OMNET++.

The test system, which the sequential simulation was run on, is a Dell Latitude E6430 Notebook. It contains an Intel Core i5 quad core CPU with eight virtual cores (hyperthreading). Therefore, the maximum number of partitions would be 8. The Notebook is running Ubuntu Linux 14.04.

For a first parallel potential analysis, the Ethernet link delay between partitions is used as lookahead. The link delay is calculated as follows: Assuming the worst case scenario, the minimum Ethernet frame size is considered with 64 byte$<=>$ 512 bit. The propagation delay amounts to $0.000\,000\,2$ s. The TX speed is $100\,000\,000$ bit/s. The inter frame delay is specified with $96bit$. Therefore, the link delay is: $(\frac{512bit+96bit}{100000000\frac{bit}{s}}) + 0.0000002s = 0.00000628s$. Using this lookahead, it will take $\frac{1\,\mathrm{s}}{0.000\,006\,28\,\mathrm{s}} = 159235$ null messages to advance the simulation by 1 ms between two events (see also section 2.8.1). Therefore, using this relatively small lookahead will have a negative impact on the parallel simulation performance. In order to improve the parallel simulation performance, the lookahead should be extended as far as possible.

Table 3.1: Measurement results of the Message Passing Interface (MPI) latency using Intel(R) MPI Benchmark 4.0 Update 1

#bytes	#repetitions	$t_{min}[usec]$	$t_{max}[usec]$	$t_{avg}[usec]$	Mbytes/sec
64	1000	0,28	0,28	0,28	437,61

In this work, the MPI[2] is used to exchange events between partitions. The maximum event size that is exchanged was estimated with 64 bytes the worst case. Measuring of the latency τ was performed using Intel(R) MPI Benchmark 4.0 Update 1. As Table 3.1 shows, the latency was measured with an average of $0.000\,000\,28$ s. Equation (3.10) is used to calculate E for a different number of partitions. However, this calculation assumes an equal distribution of events over all partitions. This assumption might not hold true for Simulation 1 and 2 depending on the partitioning strategy. In case the events are not equally distributed over all partitions, the performance benefit of the parallel simulation might be less than predicted by the efficiency criterion. Criterion 1 is calculated according to equation (3.1). λ for criterion 2 is calculated according to equation (3.6). In order to achieve a performance benefit from parallel

[2] http://www.open-mpi.de

Table 3.2: Measured performance values for simulations 1 and 2

	Simulation 1	Simulation 2	Unit
P performance	28387	28302	$\frac{ev}{sec}$
E_{seq} event density	550302	583238	$\frac{ev}{simsec}$
R relative speed	0.05	0.05	$\frac{simsec}{sec}$
L lookahead	0.00000628	0.00000628	$simsec$
τ latency	0.00000028	0.00000028	sec

simulation, criterion 1 should be well over 1.0 and criterion 2 should be well over at least 10 [62]. Table 3.3 and 3.4 show the results for Simulation 1 and 2 respectively. According to the measured and calculated values, both simulations fulfil criterion one and two when using 2 and 3 partitions. At 4 and more partitions, the performance will decrease below that of the sequential simulation. However, as mentioned before, the calculations assume that the events

Table 3.3: Efficiency criterion for Simulation 1

n Partitions	E	$LE >> 1$	$\lambda > 10$
1	550302	3,46	434,79
2	275151	1,73	217,40
3	183434	1,15	144,93
4	137576	0,86	108,70
5	110060	0,69	86,96
6	91717	0,58	72,47
7	78615	0,49	62,11
8	68788	0,43	54,35

are distributed evenly over all partitions. This might not be the case for the simulations in question. For this reason, the parallel potential might be lower than expected depending on the partitioning strategy.

Table 3.4: Efficiency criterion for Simulation 2

n Partitions	E	$LE >> 1$	$\lambda > 10$
1	583238	3,66	462,20
2	291619	1,83	231,10
3	194413	1,22	154,07
4	145810	0,92	115,55
5	116648	0,73	92,44
6	97206	0,61	77,03
7	83320	0,52	66,03
8	72905	0,46	57,78

3.3.2 Empirical analysis

Until recently, researchers have not been able to directly measure the maximum achievable speed-up, and hence, PDES studies have been published without this comparison [26].

A solution to this problem was introduced in [19], called the ideal simulation protocol. It is a parallel scheduler that is reducing the synchronization overhead to a minimum. Instead of synchronizing partitions at runtime, by exchanging messages, it is using a pre-recorded eventlog file that contains informations about all events being sent. A similar eventlog based approach was presented in [7]. Both approaches were meant to find out a simulations parallel potential. They require at least one run to record the external events. Furthermore, the eventlog must be recreated, whenever the simulation is changed. Therefore, they are less suitable for use in a productive environment. Using one of these approaches consists out of several steps:

1. Select an efficient partitioning strategy.
2. Run the parallel simulation using the NMA and record all external events for each partition.
3. Run the eventlog based scheduler which will be using the recorded information about external events to determine which events are safe to be processed.

In this work, the eventlog based scheduler from [7] is used, to determine the maximum parallel potential of Simulation 1 and 2.

3.3.2.1 Choosing a partitioning strategy

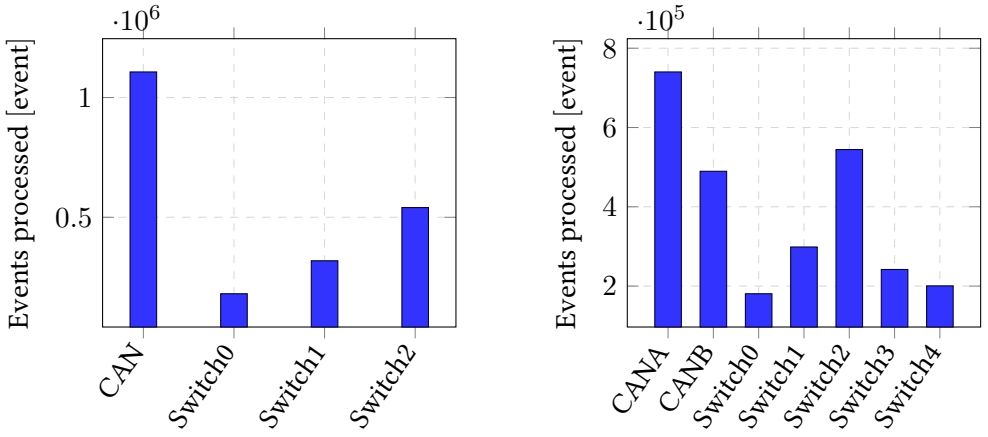
The eventlog based scheduler requires a partitioned simulation in order to evaluate its maximum parallel potential. An efficient partitioning strategy is one that distributes events almost equally

over all existing partitions (see section 3.3.1.1). In order to find such a partitioning strategy, both simulations are partitioned into functional groups that is CAN bus plus CAN nodes, TTE switch plus TTE nodes. The amount of processed events per partition of simulations 1 and 2 is then examined. Table 3.5 shows the final partitioning for Simulation 1 and 2. Both simulations

Table 3.5: Partitioning of simulation 1 and 2 to measure the amount of events per partition.

	Simulation 1	**Simulation 2**
Partition 0	CAN	CAN A
Partition 1	Switch 0	CAN B
Partition 2	Switch 1	Switch 0
Partition 3	Switch 2	Switch 1
Partition 4	-	Switch 2
Partition 5	-	Switch 3
Partition 6	-	Switch 4

are run for a total simulation time of 5 s and the amount of processed events per partition is measured. The results for Simulation 1 and 2 are shown in Figure 3.9. Both simulations show



(a) Simulation 1

(b) Simulation 2

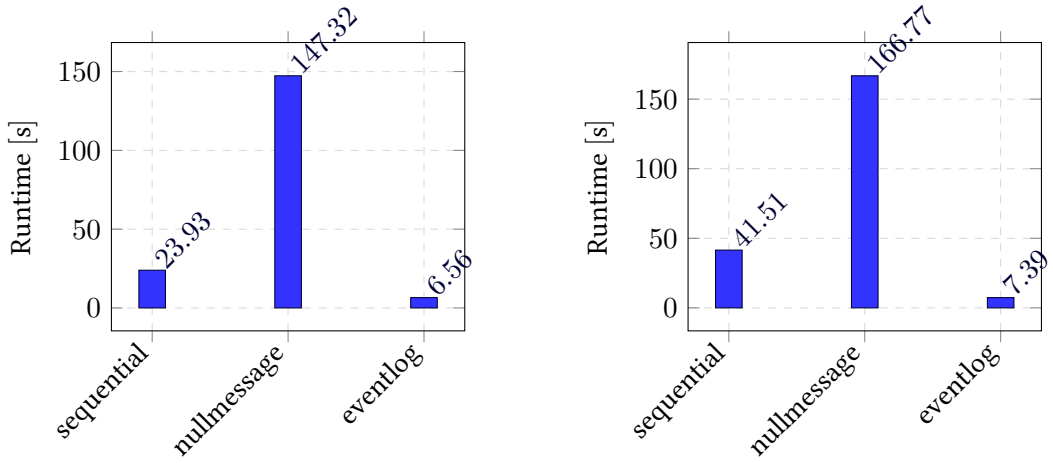
Figure 3.9: Event density measurement of Simulation 1 and 2. The bars show the total amount of events processed per partition, in order to simulate 5 s of simulation time.

that the amount of messages processed in the CAN partitions is almost similar to the total amount of messages in the TTE partitions. Simulation 1 has a CAN to TTE message ratio of 1.06 and Simulation 2 of 0.83. For this reason, splitting both simulations into a CAN partition

and a TTE partition seems to be the most efficient. This conclusion also corresponds to the results of the efficiency criterion in section 3.3.1.2.

3.3.2.2 Parallel potential measurement

To evaluate the maximum parallel potential, both simulations are run with the eventlog based scheduler. Partitioning is done according to section 3.3.2.1. The measurement results are shown in Figure 3.10. The runtime of the eventlog based scheduler proves that both simulations have



(a) Simulation 1

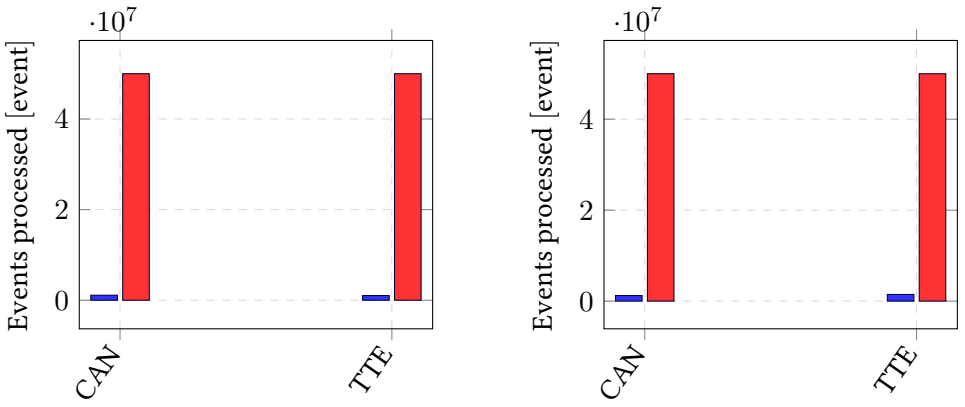
(b) Simulation 2

Figure 3.10: Parallel potential measurement for Simulation 1 and 2 using the sequential scheduler, NMA and eventlog based scheduler. Both simulations simulate 5 s of simulation time.

parallel potential when using the selected partitioning strategy. Simulation 1 takes 6.56 s in parallel, compared to 23.93 s in sequential execution mode. This corresponds to an optimization factor of: $\frac{23.93\,\mathrm{s}}{6.56\,\mathrm{s}} = 3.65$. The optimization factor of Simulation 2 is even better and corresponds to: $\frac{41.51\,\mathrm{s}}{7.39\,\mathrm{s}} = 5.62$. Another fact is that the sequential runtime of Simulation 2 is around 43% higher as that of Simulation 1. However, the parallel runtime of Simulation 2 is just 11% higher as that of Simulation 1. This non linear correlation shows, that the sequential simulation is not taking full advantage of the available hardware potential. The NMA is a lot slower than the sequential scheduler. For Simulation 1 it takes 147.32 s and for Simulation 2 it takes 166.77 s to finish.

One reason for the bad performance of the NMA is the excessive amount of null messages that is exchanged between partitions. Figure 3.11 shows the total amount of simulation events

processed versus the total amount of null messages that were exchanged in both partitions of Simulation 1 and 2. The amount of null messages exceeds the amount of simulation events by



(a) Simulation 1

(b) Simulation 2

Figure 3.11: Comparison of total number of simulation events (blue) to null messages (red)

a factor of 40.

3.4 Analysis conclusion

This chapter analysed the two simulations presented in section 3.1. Both simulated networks are mainly using cyclic and rate constrained message traffic according to section 3.2. This means that messages most likely pulse through the network. Which also means that there will be time windows with a lot of events to process and time windows without any events to process. As shown in section 3.3.2.2 the NMA produces a lot of null messages in order to jump ahead in time. Therefore, it is not an appropriate scheduler for this kind of simulations. This thesis proved true in section 3.3.2 in which both simulations were empirically evaluated for their parallel potential. The amount of null messages exceeds the amount of simulation events by a factor of 40. However, the analytical examination in section 3.3.1 as well as the empirical examination in section 3.3.2 promise parallel potential for both simulations. Therefore, a new scheduling strategy must be developed. It must decrease the amount of null messages and will have to exploit the behaviour of both simulations in order to optimally raise the parallel potential.

4 Optimization concept

This chapter presents the development of the parallel scheduling strategies. The main aim is to develop parallel scheduling strategies, that are able to raise the parallel potential of Simulation 1 and 2 that was predicted in section 3.3.1.2 and 3.3.2.2. The null message algorithm is not capable of doing this. Evidence is given in section 3.3.2.2. In a first step, a generic optimization approach will be developed in order to reduce the excessive amount of null messages that was measured in section 4.1. The main advantage of the generic optimization approach is that it does not require any adjustments of Simulation 1 and 2. It will be usable in all kinds of arbitrary simulations. The first step in developing the generic approach is to select an alternative scheduling algorithm. This is done in section 4.1.1. In the following steps, the selected scheduling algorithm is optimized in order to increase parallelism.

To further optimize the new scheduling strategies, the second step complements them by exploiting domain specific knowledge about the simulation. This is described in section 4.2.

4.1 Generic optimization

According to section 3.2, the message traffic of the simulated in vehicle networks is based on a combination of cyclic CAN messages and TTE traffic. For Simulation 1 and 2, the lookahead between partitions was calculated with $6.28\,\mu s$, whereas the intervals of the cyclic CAN messages for example, range between $10\,ms$ and $2000\,ms$ (see section 3.2.1). In relativity to the lookahead, this leads to larger periods of time without any simulation events. As a consequence, the message traffic is kind of pulsing through the network in intervals as shown in Figure 4.1. This behaviour causes corresponding simulation events and explains the excessive amount of null messages that is observed in section 3.3.2.2.

4.1.1 Scheduling algorithm selection

The message characteristic of both simulations is similar to the one that the conditional NMA in section 2.8.2.1 addresses. For Simulation 1 and 2 it seems to be the most promising scheduling approach from those presented in section 2.8.2. The carrier null message approach from section

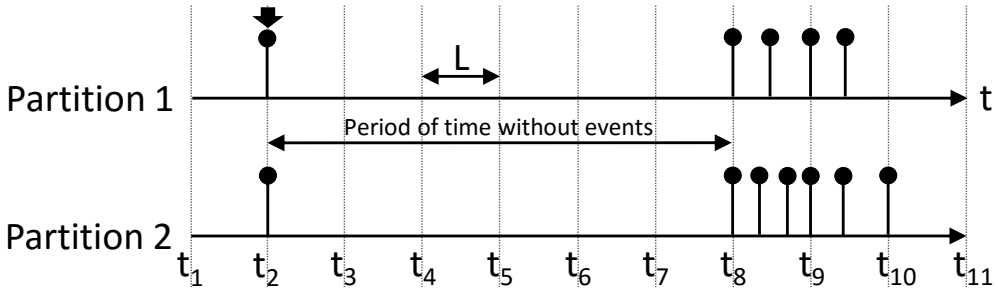


Figure 4.1: Simplified representation of message traffic as it occurs in Simulation 1 and 2 with two partitions according to section 3.3.2.1. Vertical dotted lines represent the lookahead L between Partition 1 and 2 which is also the interval in which null messages are exchanged.

2.8.2.2 improves lookahead. However, it does not help to skip longer periods of time at a reduced rate of null messages. Using conservative time windows from section 2.8.2.3, is a static and a less generic approach. It depends on domain specific knowledge to make it work. Therefore, it can not be used out of the box. Null message cancellation will not be needed when using the conditional NMA. The CNMAs uses null messages to exchange informations during computation phase. Therefore, partitions will not need to exchange more than one message with each other during EIT computation phase. The demand driven approach still has a problem with excessive null message generation at larger periods of simulation time without any simulation events. For this reason, the conditional NMA is chosen as foundation for the development of parallel scheduling strategies for in vehicle networks.

4.1.2 Conditional null message algorithm basic approach

The basic working principle of the CNMA is introduced in section 2.8.2.1. This section describes the CNMA and its optimizations, as it is realized as part of this work. The CNMA must alternate between a common EIT computation phase and a common event processing phase as described in section 2.8.2.1. The efficiency of the algorithm is dependent on the way this alternation is realized. Therefore, one major task is to specify when this alternation is supposed to happen, in order to keep the amount of alternations and therefore, the effort of synchronization as low as possible. Figure 4.2 illustrates the problem. Suppose both simulations are at simulation time t_2. The scheduler has to compute the EIT in order to process the next events. The scheduler does not have any insight into the nature of the events to process. Furthermore, it does not know, whether processing an event will create external events. For this reason, the scheduler must assume that processing any event will spawn further external events. Processing e_1 in

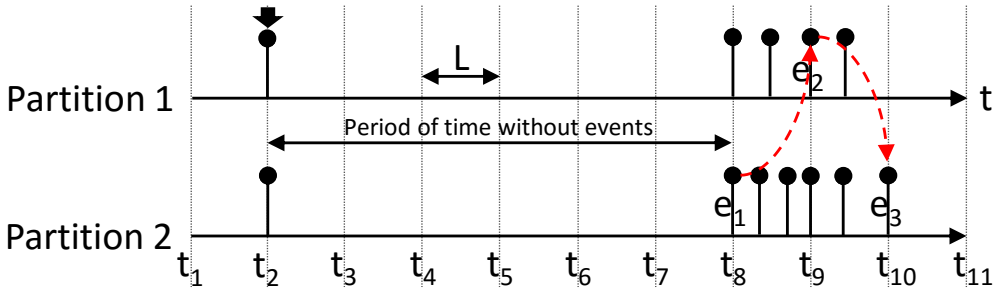


Figure 4.2: Generation of external events at unknown points in simulation time, makes it difficult to reduce the amount of necessary synchronization.

Partition 2 at t_8 for example, might produce a new event e_2 which is sent to Partition 1. This event will arrive Partition 1 at t_9, since it takes at least the lookahead L to transfer the event. Processing e_2 in Partition 1 at t_9 will in return produce another external event e_3 that will arrive Partition 2 at t_{10}.

The basic scheduling algorithm is supposed to follow a generic approach in order to optimize both simulations without putting further simulation specific development effort into it. For this reason, the CNMA approach realized in this work, determines the event with the smallest timestamp, among all partitions at computation phase. This event's timestamp will form the new EIT. Assume both partitions advanced their simulation time to t_2. This is shown in Figure

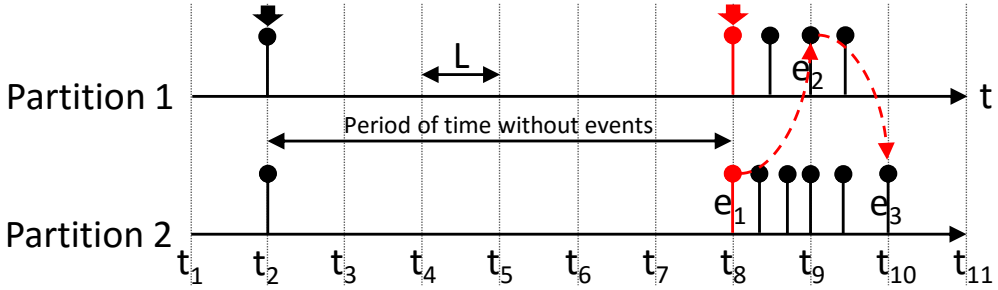


Figure 4.3: Calculating the next global EIT is done by determining the event with the smallest timestamp over all partitions. The timestamp of that event will for, the new global EIT.

4.3 by the black arrow facing down. The events to be processed next are at t_8 for both partitions, marked in red. Therefore, $EIT = min(t_8, t_8) = t_8$. Both partitions can safely advance their simulation time to t_8 and process the events at t_8. However, processing only the events at t_8, will not yield the desired parallel potential. Figure 4.4 shows an example situation, in which the simulation is basically processed sequentially. The events of Partition 1 and 2 are scheduled

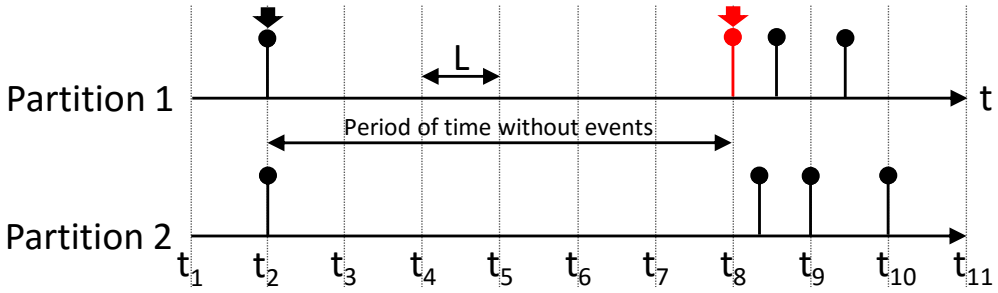


Figure 4.4: Basically sequentialized parallel simulation due to simulation events at uneven points in simulation time.

at uneven points in simulation time. In this particular example the partitions would alternate with event processing.

As a solution to this problem and in order to further increase the parallelism, the CNMA must exploit the lookahead between neighbouring partitions. This is shown in Figure 4.5.

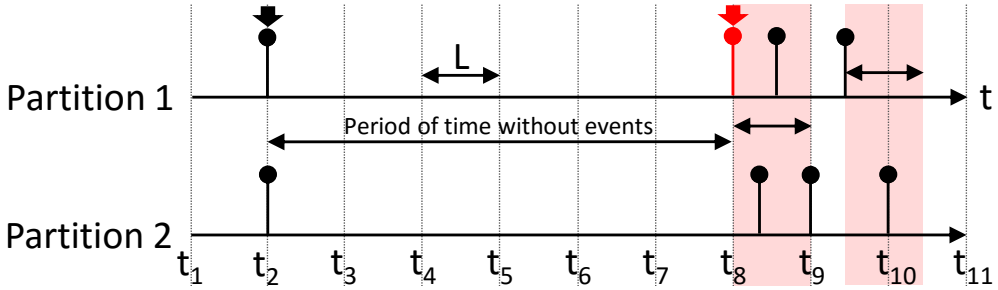


Figure 4.5: Exploiting the lookahead between partitions (red background), in order to further increase the parallelism.

In this example the current simulation time has reached t_2. After EIT computation phase, both partitions will advance the simulation time to t_8. Partition 1 will process the event at t_8. However, new events spawned as a result of event processing, will not arrive at Partition 2 earlier than $t_8 + L$. The same goes for events that are sent from Partition 2 to Partition 1. For this reason, both partitions can safely process every event with a timestamp smaller $t_8 + L$. That way, both partitions are capable of processing events in parallel, which leads to an increased level of parallelism. After processing all safe events, the partitions will enter another EIT computation phase. In this particular example the lookahead between both partitions is assumed to be equal. However, the algorithm will also work with varying lookahead between partitions.

4.1.3 Increasing parallelism

Section 4.1.2 explains how the CNMA exploits the lookahead between partitions at EIT computation phase in order to increase parallelism. This section aims at further improving parallelism by introducing a second computation phase that takes external events into account of the EIT calculation.

Consider the simulation shown in Figure 4.6. It consists out of three partitions. Figure 4.7

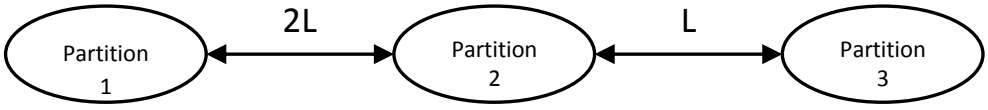


Figure 4.6: Example simulation consisting out of three partitions. Attend the differing lookahead between the partitions.

shows an exemplary event processing scenario. In this example all three partitions finish their processing phase at t_1 and enter the next EIT computation phase. For this reason, they

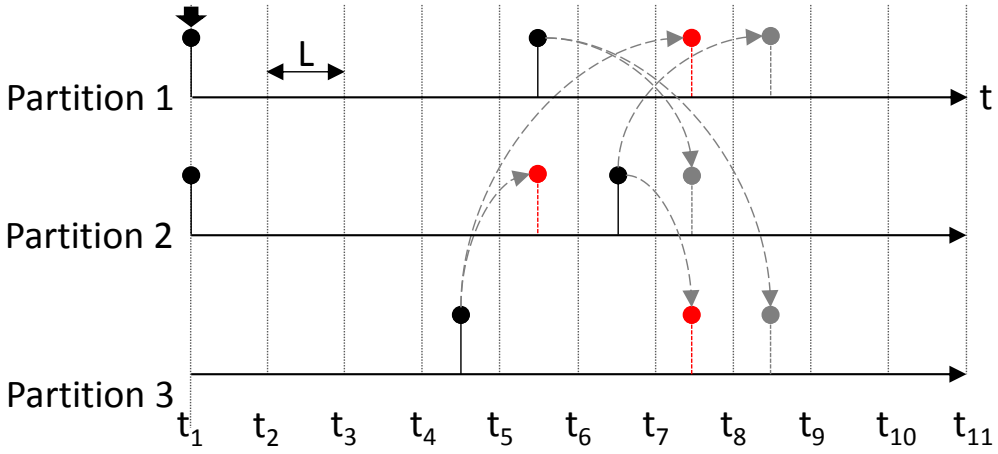


Figure 4.7: EIT computation phase step 1 with red events depicting the earliest external events that might be received.

exchange their ECOTs. This is shown by the grey arrows in Figure 4.7. Each grey arrow represents the ECOT of the sending partition to the receiving partition. As stated in section 2.8.1, the ECOT represents the earliest possible time that a partition might receive an event

from the sending partition. All under the assumption, that no further messages will be received by that partition, having a smaller timestamp. According to Figure 4.7, Partition 1 receives ECOTs $t_{7.5}$ from Partition 3 and $t_{8.5}$ from Partition 2. Partition 2 receives the ECOTs $t_{5.5}$ and $t_{7.5}$. Partition 3 receives ECOTs $t_{7.5}$ and $t_{8.5}$. The EIT is calculated as the minimum ECOT out of all ECOTs. Therefore, the EIT computation phase finishes with $t_{5.5}$ as new EIT. According to the current algorithm all three partitions advance their simulation time to $t_{5.5}$ and process all events with a timestamp smaller than $t_{5.5}$, as shown in Figure 4.8. However, taking a closer

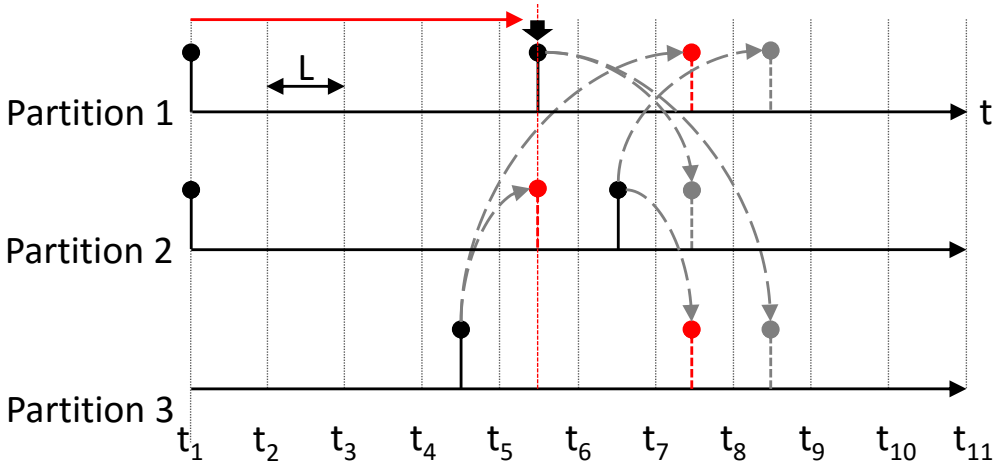


Figure 4.8: Partitions advancing in simulation time to $t_{5.5}$

look at Figure 4.8, one can see that all three partitions might advance their simulation time even further by considering that Partition 2 might receive an external event at $t_{5.5}$. Therefore, the CNMA will be extended by a second computation step. This step will be called phase 2 calculation through the remainder of this work. In Figure 4.9 Partition 2 might receive an external event at $t_{5.5}$. This external event has a smaller timestamp than the internal event that the previous ECOT calculation is based on. For this reason, the ECOT of Partition 2 is recalculated taking the timestamp of the external event into account. Processing the external event, might result in Partition 2 sending events to Partition 1 and 3. The external events will be received at $t_{6.5}$ in Partition 3 and at $t_{7.5}$ in Partition 1. Therefore, the ECOT of Partition 2 to Partition 1 is $t_{7.5}$ and from Partition 2 to Partition 3 it is $t_{6.5}$. This is shown in Figure 4.9. In conclusion by taking the external event into the ECOT calculation, the EIT of Partition 1 is increased by 2, whereas for Partition 3 it is increased by 1 (see Figure 4.9).

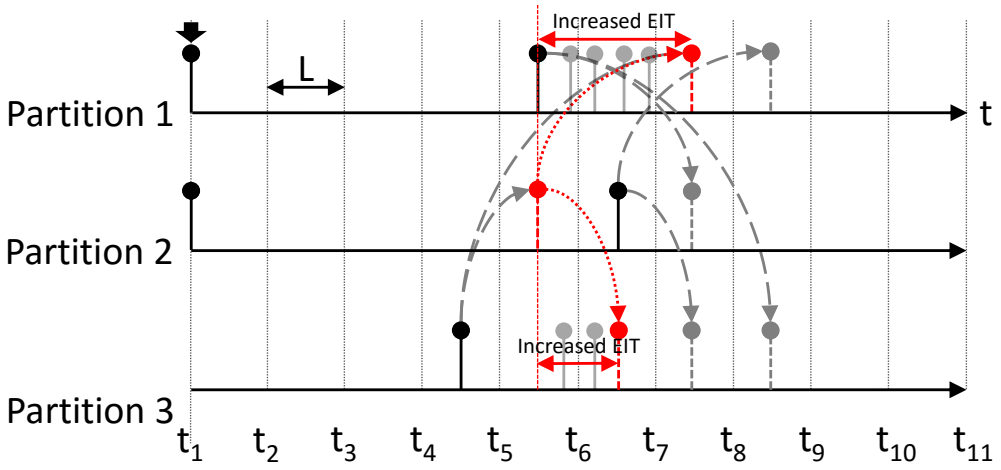


Figure 4.9: EIT computation phase step 2 also considers the EOTs to increase the lookahead and with it the amount of potential events that can be safely processed.

4.1.4 Final conditional null message algorithm approach

This section presents the final operating principle of the CNMA approach that is implemented in this work. Figure 4.10 shows the computation steps of the CNMA based synchronization approach.

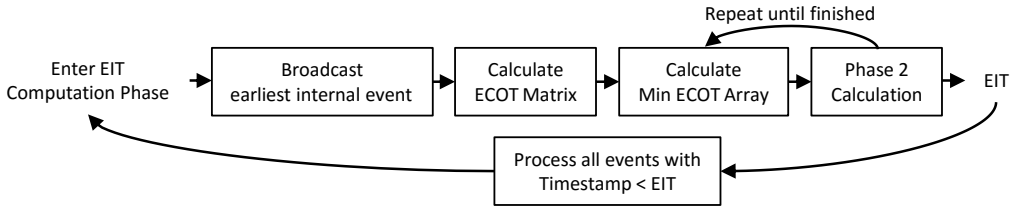


Figure 4.10: Computation steps of the CNMA based synchronization approach.

4.1.4.1 Broadcast earliest internal event

Whenever a partition processed all events inside its FES with a timestamp smaller than the EIT, it is blocked and must enter another EIT computation phase. The EIT computation phase is initiated by each partition broadcasting the timestamp of the next events inside its FES to all other partition. This event is called the earliest conditional event throughout the remainder of this work. It will be the next event that is processed under the condition that the partition will not receive any further events with a smaller timestamp.

4.1.4.2 Calculate ECOT Matrix

At receiving a broadcast from another $partition_n$, the earliest conditional event's timestamp is used to calculate the ECOT matrix at index $[n, n]$ by adding the lookahead between sending and receiving partition. The ECOT matrix is supposed to store the ECOTs between all partitions. Considering the example from Figure 4.8, the resulting ECOT matrix will look as shown in Table 4.1. The algorithm responsible for the creation of the ECOT matrix is shown in Listing 1.

Table 4.1: Resulting ECOT matrix derived from Figure 4.8, with earliest conditional event timestamps in bold.

From / To	Partition 1	Partition 2	Partition 3
Partition 1	**5.5**	7.5	8.5
Partition 2	8.5	**6.5**	7.5
Partition 3	7.5	5.5	**4.5**

Algorithm 1 CNMA ECOT broadcast

```
1: procedure CALCULATEECOTMATRIX
2:     for each partition sender ∈ Partitions do          ▷ Iterate sending partitions.
3:         for each partition receiver ∈ Partitions do    ▷ Iterate receiving partitions.
4:             ECOTMatrix[sender][receiver] =
5:             EarliestEvent(sender) + Lookahead(sender, receiver)
6:         end for
7:     end for
8: end procedure
```

4.1.4.3 Calculate min ECOT Array

The min ECOT array is derived from the ECOT matrix. It contains the minimum ECOT for each partition. Considering the ECOT matrix from Table 4.1, the minimum ECOT for Partition 2 is 5.5. The final ECOT array is shown in Table 4.2. The corresponding algorithm is shown in

Table 4.2: Resulting min ECOT array derived from ECOT matrix 4.1.

	ECOT
Partition 1	7.5
Partition 2	5.5
Partition 3	7.5

Listing 2.

Algorithm 2 Minimum ECOT array calculation.

```
1: procedure calculateMinECOTArray
2:     for each partition receiver ∈ Partitions do          ▷ Iterate receiving partitions.
3:         ECOTArr[receiver] = ∞
4:         for each partition sender ∈ Partitions do        ▷ Iterate sending partitions.
5:             if receiver ≠ sender then
6:                 if ECOTArr[receiver] > ECOTMatrix[sender][receiver] then
7:                     ECOTArr[receiver] = ECOTMatrix[sender][receiver]
8:                 end if
9:             end if
10:        end for
11:    end for
12: end procedure
```

4.1.4.4 Calculate phase 2

Phase 2 calculation tries to further extend the EIT for individual partitions. Therefore, it is checking each partition for an ECOT that is smaller, than the earliest scheduled internal event. An example for such an ECOT is shown in Figure 4.7, Partition 2 at $t_{5.5}$. In order to prevent causality violations, the algorithm must iterate the ECOTs, starting with the smallest one and advance in increasing timestamp order. In order to find the partition with the smallest ECOT, the ECOT array from section 4.1.4.3 is iterated as shown in Listing 3. The phase 2 algorithm is

Algorithm 3 Gets the next partition with an ECOT smaller than the earliest internal event in advancing order.

```
1: procedure getNextPhase2Partition
2:     nxtp = −1;
3:     for each partition p ∈ Partitions do                 ▷ Iterate over partitions.
4:         if p not yet processed then
5:             if Timestamp of earliest internal event of p > ECOTArray[p] then
6:                 if nxtp == −1 or ECOTArray[p] < ECOTArray[nxtp] then
7:                     nxtp = p
8:                 end if
9:             end if
10:        end if
11:    end for
12:    return nxtp
13: end procedure
```

shown in Listing 4. After finishing the phase 2 calculation, the min ECOT array contains the

Algorithm 4 Performs the phase 2 calculation.

```
1:  procedure PHASE2CALCULATION
2:      nxtp = getNextPhase2Partition();
3:      while nxtp ≠ (−1) do
4:          for each partition p ∈ Partitions do          ▷ Iterate over partitions.
5:              if p ≠ nxtp then
6:                  ECOTMatrix[nxtp][p] = ECOTArray[nxtp] + AdjMatrix[nxtp][p]
7:              end if
8:          end for
9:          calculateMinECOTArray()                      ▷ Recalculate the min. ECOT array.
10:     end while
11: end procedure
```

EITs for the different partitions.

4.1.4.5 Final algorithm

The final algorithm that determines the EIT for a specific partition is shown in Listing 5. It begins with a broadcast of the earliest conditional event (see section 4.1.4.1). It then waits for receiving the earliest conditional event timestamps from all other partitions of the parallel simulation. In the next step it calculates the ECOT matrix and the minimum ECOT array (see sections 4.1.4.2 and 4.1.4.3). The following Phase 2 calculation is based on the minimum ECOT array (see section 4.1.4.4). After the Phase 2 calculation, the minimum ECOT array contains the EIT for every partition of the simulation and is returned for the respective partition.

Algorithm 5 Final algorithm to calculate performs the EIT calculation phase.

```
1: procedure EITCALCULATIONPHASE
2:     Broadcast earliest conditional event timestamp.
3:     Receive all earliest conditional event timestamps from other partitions.
4:     calculateECOTMatrix()
5:     calculateMinECOTArray()
6:     phase2Calculation()
7:     return ECOTArr[PartitionNumber]                  ▷ Return new EIT.
8: end procedure
```

4.2 Domain specific optimization

Section 4.1 introduces the generic approach of the new scheduling concept developed in this work. The advantage of the generic scheduler is, it can be used out of the box for any kinds of parallel simulation. No adjustments on the simulation code will be necessary. However, its performance benefits are somewhat limited due to the generic approach that does not have any insight about the simulation's internals. Exploiting knowledge about the simulation's internal behaviour might help to further improve performance of the parallel scheduler. This section presents the second part of the scheduling concept that is offering a way, to integrate simulation specific knowledge into the scheduling concept.

4.2.1 Weak point of the generic optimization

Section 4.1.4 presents the final CNMA approach. The CNMA scheduler is basing its EIT calculation on the next earliest conditional event of the parallel simulation's partitions. A weakness of this method is that the majority of events inside a partition's FES will not produce any external events. Therefore, a lot of synchronization is based on events that will never produce any external events. Consider the situation shown in Figure 4.11. Only the events

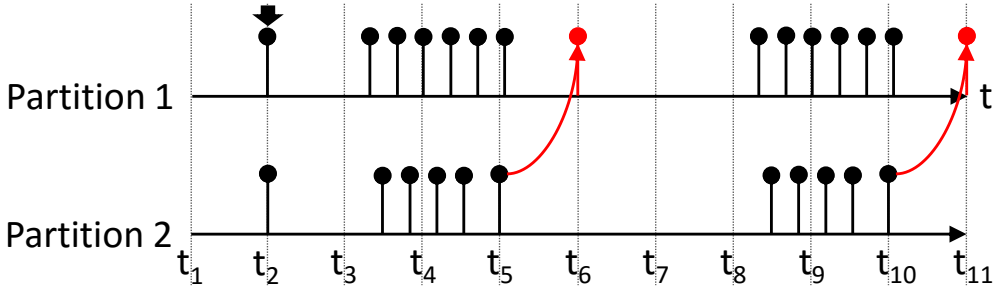


Figure 4.11: Example situation in which the generic approach fails to exploit the maximum parallel potential.

of Partition 2 at t_5 and t_{10} create external events which Partition 2 sends to Partition 1. At its current state, the generic CNMA approach needs five synchronization steps in order to advance simulation time past t_{10} as shown in Figure 4.12.

4.2.2 Problem-solving approach

To prevent the scheduler from including events that do not create external events into the synchronization process, it must get an ability to differentiate events into those that do create external events and those that do not. Knowing what events create external ones, the scheduler

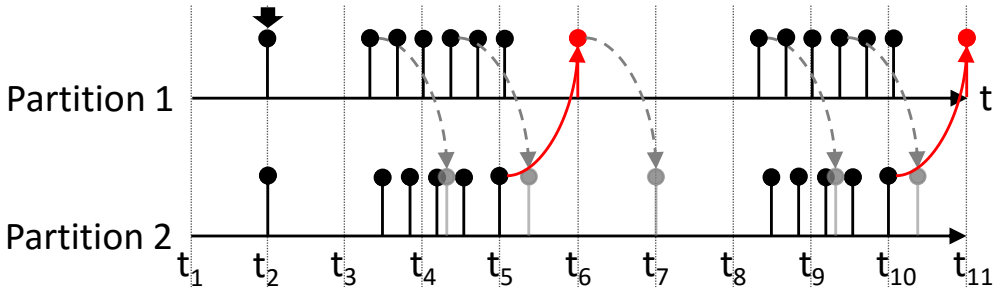


Figure 4.12: The generic CNMA approach from section 4.1 needs five synchronization steps to advance simulation time past t_{10}.

needs for the example situation from Figure 4.11 just two synchronization points as shown in Figure 4.13. The integration of this idea is straight forward. The broadcast of the earliest

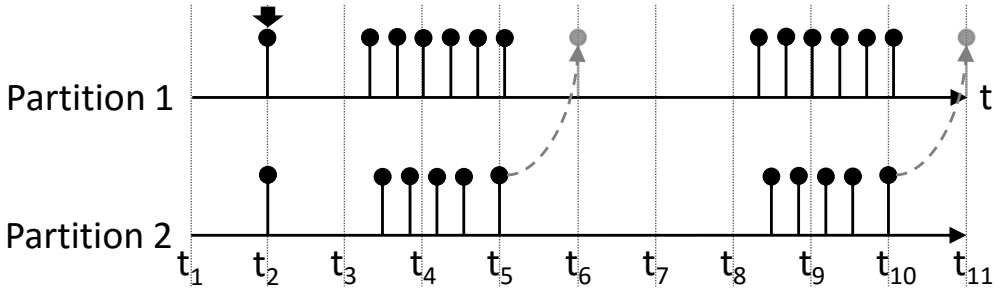


Figure 4.13: Knowing in advance which events create external events, reduces the amount of synchronization points to two.

internal event timestamp (see section 4.1.4.1) is slightly adjusted. Instead of broadcasting the timestamp of earliest internal event, the timestamp of the earliest internal event that creates external events is broadcast. To do this, the simulation itself must be adjusted in order to enable the scheduler to differentiate between events. The required changes to the simulation and the required implementation steps are examined in chapter 5.

4.2.3 Basic thoughts towards a generally applicable domain specific scheduling strategy

This section presents and discusses the basic thoughts towards a generally applicable domain specific scheduling strategy. Before developing a successful concept, a few important facts must be considered. Figure 4.14 shows an abstract depiction of a typical event pipeline, as it occurs in all DES. The underlying simulation consists out of two partitions, Partition 1 and Partition 2. Partition 1 contains an event source that generates events in fixed time intervals.

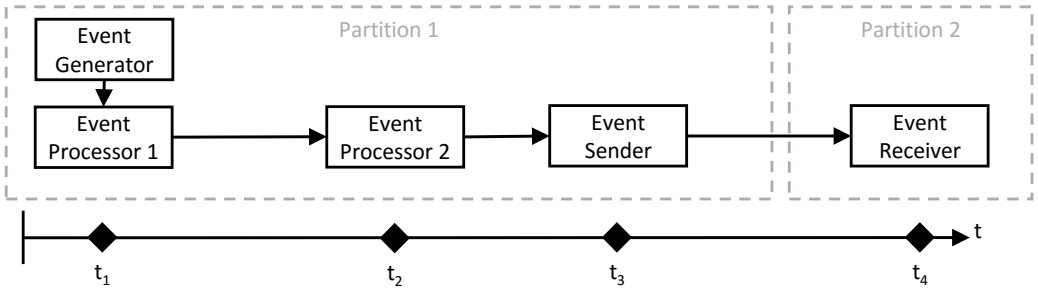


Figure 4.14: Typical event pipeline with several sequential event processing modules with and without processing delays.

Furthermore, it contains several event processing modules, with or without processing delays and an Event Sender that forwards events to Partition 2. Partition 2 contains an Event Sink that receives the events from the Event Sender and discards them.

In this particular example, the Event Source generates a new event at t_1. It is forwarded to Event Processor 1 with zero processing delay. Event Processor 1 however, has a processing delay Δ_{P1}. Therefore, Event Processor 2 receives the event at t_2. Event Processor 2 also has a processing delay Δ_{P2} so that the Event Sender receives the event at t_3. The Event Sender forwards it over partition borders to the Event Receiver in Partition 2. This takes a considerable amount of time, consisting out of the processing delay inside the Event Sender Δ_{P4}, plus the transmission delay Δ_L of the event between both partitions. Therefore, the event reaches the Event Receiver at t_4.

Assuming that Δ_{P1}, Δ_{P2} and Δ_{P4} were fix delays, this example would be an ideal candidate for domain specific optimization. At time t_1 the scheduler could predict that the next external event will be received by Partition 2 at $t_4 = t_1 + \Delta_{P1} + \Delta_{P2} + \Delta_{P4} + \Delta_L$. Therefore, both partitions can advance in simulation time to t_4.

However, reality differs from such ideal examples. Therefore, a number of different situations must be considered.

4.2.3.1 Asymmetric event generation

In general, simulations do not contain just one single event source. Instead, they contain a number of different event sources. These sources generate events at different points in simulation time. This leads to different points in simulation time that external events will be sent. These points in simulation time will be called time to broadcast (TTB) throughout the remainder of this work. Figure 4.15 shows an example of asymmetric generation of events that create external events when being processed. In conclusion, the scheduler must obtain a global

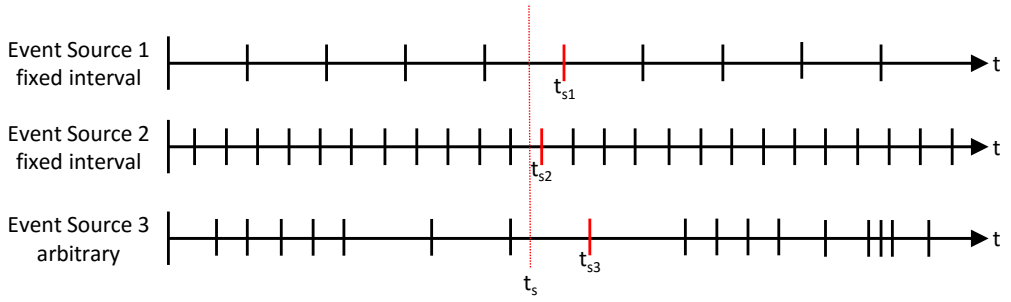


Figure 4.15: Example of asymmetric event generation by three different event sources in one simulation. The red dotted line denotes the time of the next synchronization.

view on all events that might cause external events in the future. In this particular example the scheduler must know at synchronization point t_s that there will be events generated at t_{s1}, t_{s2} and t_{s3}.

4.2.3.2 Processing delays

Processing delays are an ideal example of domain specific knowledge as a means of increasing lookahead between partitions. However, processing delays are not necessarily fixed time intervals. Therefore, they must be differentiated into fixed processing delays and variable processing delays.

4.2.3.2.1 Fixed processing delays Fixed processing delays can be directly used by the scheduler as a means of increasing lookahead between partitions. If the fixed processing delay between an event source in Partition 1 until the event is sent to an event receiver in Partition 2 was 5 s, the scheduler can add this delay to the lookahead between Partition 1 and Partition 2. Thus increasing the total lookahead between both partitions.

4.2.3.2.2 Variable processing delays Variable processing delays can not be directly used by the scheduler since they correspond to a time window in simulation time rather than a fixed point in time. They can occur whenever future event processing is based on decisions. Consider the example in Figure 4.16. Event Processor 2 forwards events directly to the Event Sender or Event Processor 3 in a randomly fashion. That implies, at event creation, the scheduler does not know at what time the event will reach the Event Sender and at what time it will be sent to Partition 2 as an external event. However, the scheduler can carefully assume that the event will always take the shortest path through the simulation. Doing so, it will always communicate the earliest possible time that an external event might be sent. In conclusion, the

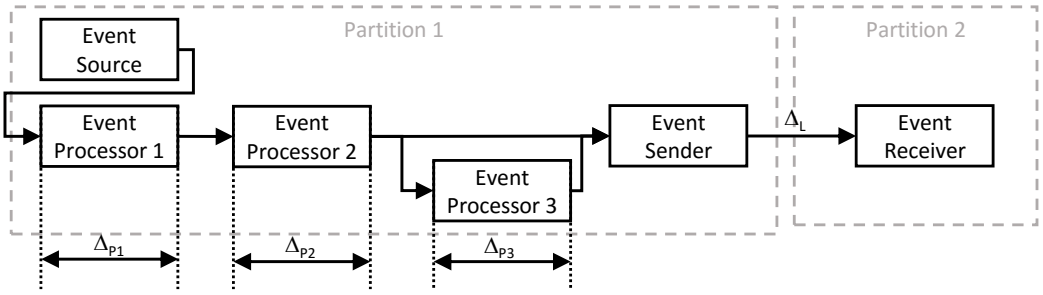


Figure 4.16: Example of variable processing delay. Event Processor 2 forwards events directly to the Event Sender or Event Processor 3 in a randomly fashion.

scheduler must always assume the smallest processing delay in case of variable processing delays.

However, always assuming the smallest processing delay is not sufficient for a reliable synchronisation concept. Figure 4.17 shows why. At t_1 the simulation generates an internal

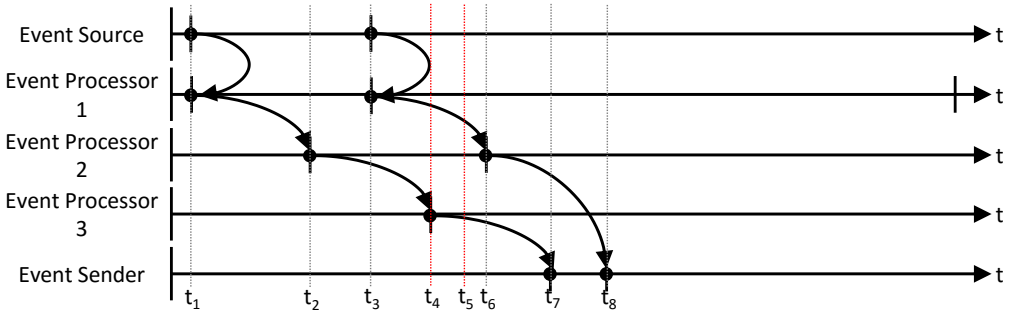


Figure 4.17: Example of variable processing delay and synchronisation problem.

event. This event will travel the event pipeline shown in Figure 4.16. Since the scheduler does not know which way the event will take at this point of time, it assumes the smallest processing delay. That means it expects an external event being generated at t_4. At t_3, yet another internal event is created and as a result the scheduler expects an external event being generated at t_8. At t_5, a new synchronization cycle is initiated. At this cycle the scheduler will expect that external event that was estimated for t_4, is already sent. Without further informations it will assume that the next external event will occur at t_8. However, since the first external event has not yet been sent because it did not take the smallest processing delay, it will be sent at t_7. Thus causing a temporal violation at t_7, which must not happen.

In conclusion, the scheduler must keep track of all predicted external events and their actual time of delivery.

4.2.3.2.3 Keeping track of events Keeping track of events inside the event pipeline as required by paragraph 4.2.3.2.2, requires each event carrying a unique identifier. In most simulation systems this is already the case. Often times though, the event will be transformed, deleted and recreated or in other ways modified. This causes the event loosing its original unique identifier and complicates event tracking. In conclusion, the new scheduler must get some kind of event tracking mechanism that allows to keep track of events, even when they change their unique identifier or get modified in any other way.

4.2.4 Final domain specific optimization approach

This section presents the core aspects of the final domain specific scheduling approach. The scheduling architecture is explained in section 4.2.4.1. It is followed by an explanation on how to obtain the global view on all possible external events in section 4.2.4.2 and the event tracking in section 4.2.4.3.

4.2.4.1 Scheduler architecture

The domain specific scheduler is based on the CNMA scheduler with phase two optimization, which was developed and introduced in section 4.1. Figure 4.18 displays the underlying class architecture. The interface IDspScheduler contains the required functions to realize

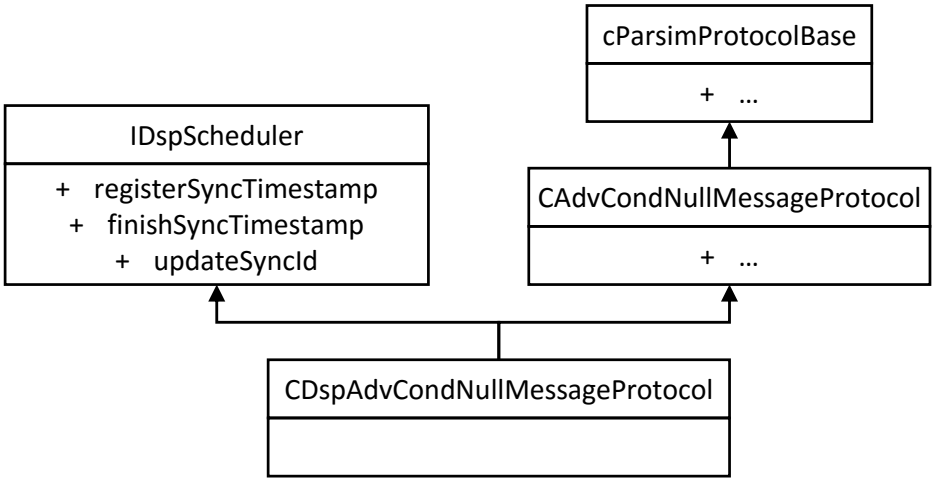


Figure 4.18: Class architecture for the domain specific scheduler approach.

the global view on possible external events and to realize event tracking. The interface is implemented by the concrete domain specific scheduler CDspAdvCondNullMessageProtocol. Since the new scheduler is derived from CAdvCondNullMessageProtocol, it acts like the

CAdvCondNullMessageProtocol scheduler in case there is no domain specific knowledge available. Whenever domain specific knowledge is available, it will use the domain specific information for synchronization.

4.2.4.2 Obtaining a global view on all possible external events

Whenever a simulation module creates an event that will create an external event in the future, it must inform the scheduler. The time of the external event must be calculated by the calling module. This is done according to section 4.2.3.2.2. The calculated timestamp and the event id are then passed to the scheduler by calling registerSyncTimestamp.

The scheduler keeps track of all announced timestamps, by keeping them inside a priority queue, which is sorted in time correct order. Basically the scheduler now contains two event sets. The original FES and the priority queue with timestamps of potential external events as shown in Figure 4.19. While the original FES contains all events of the simulation, the priority

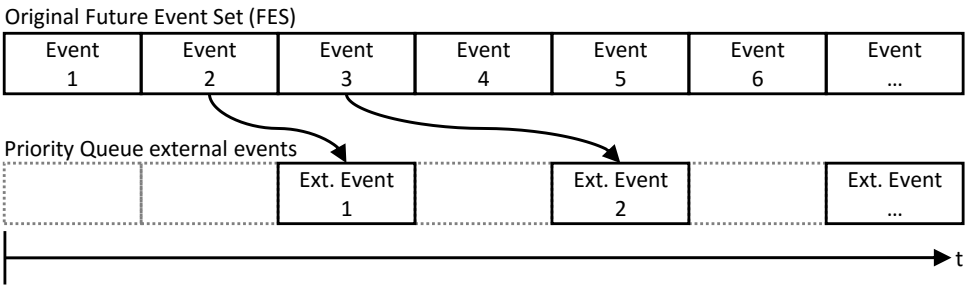


Figure 4.19: The domain specific scheduler contains two different event queues. The original FES as well as the priority queue with timestamps of possible external events.

queue just contains the possible external events. The scheduler is adjusted as follows. The

Algorithm 6 Calculating the ECOT now also includes the domain specific knowledge.

```
1: procedure getCondEOT
2:     tDspEOT = getNextDspTimestamp();
3:     if tDspEOT is valid timestamp then
4:         return tDspEOT;
5:     else
6:         return cAdvCondNullMessageProtocol::getCondEOT();
7:     end if
8: end procedure
```

ECOT is calculated by getCondEOT as shown in Listing 6. This function gets the next domain

specific timestamp to consider for synchronization in line 2. If a domain specific timestamp is available and valid, it will be returned in line 3. Otherwise, the generic synchronization timestamp of the generic synchronization approach is returned in line 6.

The next domain specific timestamp to consider for synchronization is determined by function getNextDspTimestamp as shown in Listing 7. The original CNMA synchronization

Algorithm 7 Determines the next domain specific timestamp to consider at synchronization.

```
1:  procedure getNextDspTimestamp
2:      while Priority queue NOT empty do
3:          tDspEOT = Take next DSP from priority queue;
4:          if External event at tDspEOT not yet sent then
5:              if tDspEOT > Current Time then
6:                  return tDspEOT
7:              else
8:                  return Invalid;
9:              end if
10:         else
11:             Remove tDspEOT from priority queue;
12:         end if
13:     end while
14:     return Invalid;
15: end procedure
```

algorithm as described in section 4.1.4.5, is unchanged. However, it is now fed with an ECOT derived from domain specific knowledge or a generic ECOT determined by the generic CNMA algorithm.

4.2.4.3 Event tracking

Event tracking after registration is realized by two additional functions, updateSyncId and finishSyncTimestamp. The function updateSyncId is used to tell the scheduler that one of the previously registered events changed its unique identifier. Input are the previous identifier and the new identifier.

Telling the scheduler that an external event has been sent to another partition, is done by using the function finishSyncTimestamp. It takes the external events unique identifier as input. Hence the effort to keep track of events when passing the event pipeline.

Under ideal circumstances, the external event is sent at the predicted time and the function is therefore called at the same time. However, in situations with variable processing delays, as

described in section 4.2.3.2.2, it might get called at some point later in simulation time. It must never be called earlier. In that case a causality violation would be created.

5 Concept implementation

This chapter presents the implementation of the final parallel simulation strategy, based on the optimization concept from section 4. Section 5.1 gives a brief overview of OMNET++'s parallel simulation system architecture and explains how to implement a new parallel scheduler. Section 5.2 presents the implementation of the CNMA scheduler. The domain specific optimization possibilities introduced in section 4.2, is further examined in section 5.3.

5.1 The OMNET++ parallel simulation subsystem

Omnet++ was initially not intended for parallel simulation. An extension to OMNET++ that enables parallel simulation is published in [63]. It presents the new parallel and distributed system architecture. It introduces two parallel simulation schedulers for OMNET++, the NMA and the Ideal Simulation Protocol. Figure 5.1 shows the basic system architecture. The

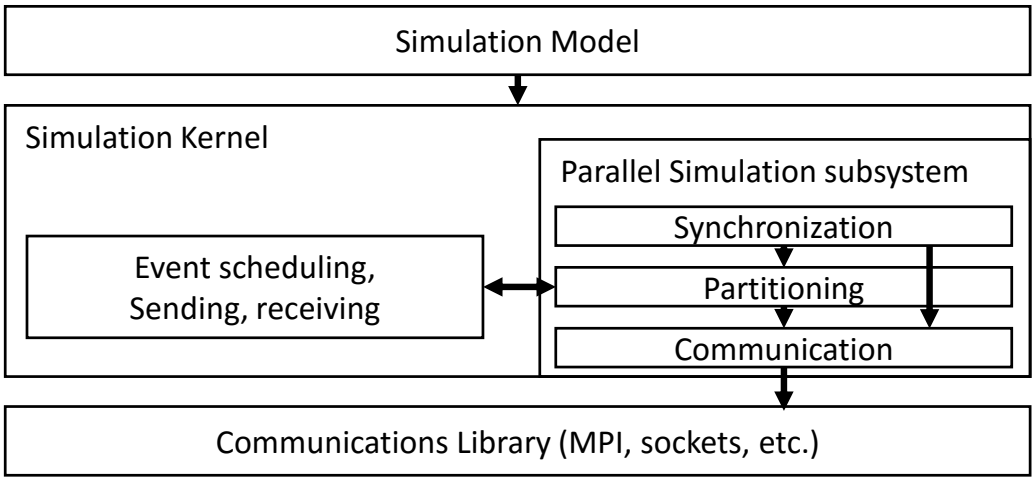


Figure 5.1: Architecture of OMNETT++ PDES implementation. [63]

parallel simulation subsystem consists out of three layers. The communication layer abstracts communication between partitions. Current implementations support sockets, named pipes and MPI. The partitioning layer is responsible for the instantiation of the parallel simulation

and for distributing the simulation modules on the different partitions. Every partition gets a complete model of the simulation. However, the partitioning system will replace modules that

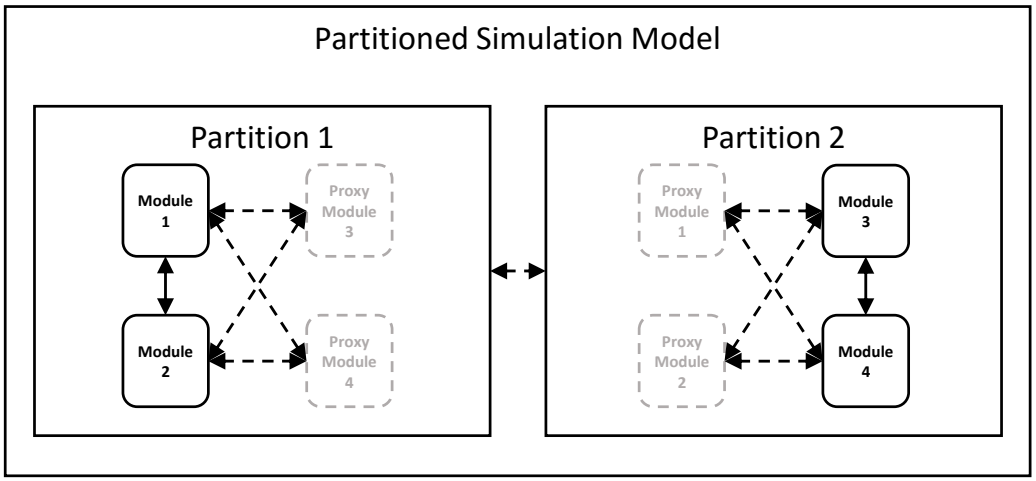


Figure 5.2: Example of a partitioned simulation model with two proxy modules per partition that forward any incoming events to the corresponding module at the other partition.

do belong to another partition by proxy modules as shown in Figure 5.2. When sending an event to a proxy module, the event will be intercepted by the partitioning system and forwarded via the communication layer to the partition containing the target simulation module. Finally the synchronization layer is representing the actual synchronisation algorithm which can follow either a conservative or an optimistic approach.

5.2 The conditional null message algorithm scheduler

The CNMA scheduler is implemented in two steps. In step one the basic approach from section 4.1.2 is implemented. The implementation highlights are presented in section 5.2.1. In step two, the parallelism is increased as described in section 4.1.3. The implementation details are presented in section 5.2.4.

5.2.1 Basic approach implementation

The basic CNMA scheduler is implemented in class cCondNullMessageProtocol which is derived from class cParsimProtocolBase which is part of the OMNET++ synchronisation layer (see section 5.1).

5.2.2 Distributed lookahead calculation

In this particular implementation, the lookahead calculation is just done once at simulation start for performance reasons. When initialising the simulation, the CNMA calculates the lookahead to all other partitions. Each partition is able to calculate the lookahead to its direct neighbours. However, for the CNMA each partition must also know the lookahead to partitions that are

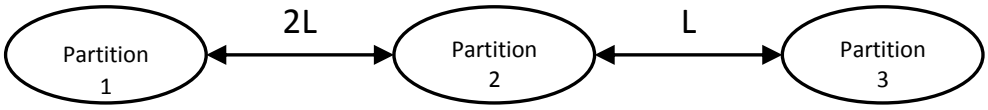


Figure 5.3: Example simulation with Partition 1 and Partition 3 beeing indirectly connected via Partition 2.

not directly connected as shown in Figure 5.3. For this reason, cCondNullMessageProtocol implements a distributed lookahead calculation.

It starts by calculating the lookahead to each neighbouring partition that is directly connected and broadcasting this information to all other partitions as shown in Listing 5.1.

```
void cCondNullMessageProtocol::broadcastSetupMessage()
{
  cCommBuffer *buffer = comm->createCommBuffer();
  for(int i = 0; i < numSeg; i++) {
    // Calculate lookahead to partition i.
    // m_MAX_TIME stands for no direct connection.
    segInfo[i].lookahead = lookaheadcalc->getCurrentLookahead(i);
    buffer->pack(segInfo[i].lookahead);
  }

  // Broadcast the calculated lookahead to all other partitions.
  for(int i = 0; i < numSeg; i++) {
    comm->send(buffer, TAG_SETUP_MESSAGE, i);
  }

  comm->recycleCommBuffer(buffer);
}
```

Listing 5.1: Calculating and broadcasting local lookaheads

Since the partition can not yet calculate the total lookahead to partitions that are not directly connected, getCurrentLookahead returns the maximum time as lookahead. This will be fixed on the receiving part of the lookahead calculation. On the receiving side, the lookahead broadcasts get sorted into an adjacency matrix. The adjacency matrix stores lookaheads between all partitions. Each row represents a sending partition, whereas each column represents a receiving partition. Table 5.1 shows the adjacency matrix for the simulation shown in Figure 5.3. After

Table 5.1: Example adjacency matrix for the simulation from Figure 4.6

	Partition 1	**Partition 2**	**Partition 3**
Partition 1	0	2L	inf
Partition 2	2L	0	L
Partition 3	inf	L	0

receiving the lookahead broadcasts from all other partitions, the adjacency matrix is used to find the lookaheads from the current partition to all other partitions of the parallel simulation. This is done by feeding the adjacency matrix the the Dijkstra algorithm for shortest path calculation.

```
// Instantiate Dijkstra algorithm and feed adjacency matrix.
CDijkstraOVsquare djks(m_AdjMatrix);
// Get process ID of the current partition.
int procId = comm->getProcId();
// Iterate partitions including current partition (L=0).
for (int i = 0; i < numSeg; i++) {
  // Store the calculate lookahead inside lookahead array.
  lookaheadArr[i].lookahead = djks.GetShortestPath(procId, i);
}
```

Listing 5.2: Calculating and broadcasting local lookaheads

The final lookahead array that is generated by listing 5.2 contains the lookaheads from the current partition, to all other partitions as shown in Table 5.2.

Table 5.2: Final lookahead array of Partition 1 after shortest path calculation using the Dijkstra algorithm.

	Partition 1	**Partition 2**	**Partition 3**
Partition 1	0	2L	3L

5.2.3 Basic synchronisation

The basic synchronisation algorithm as described in section 4.1.2 is realized inside cCond-NullMessageProtocol::takeNextEvent. It is shown in listing 5.3.

```
// Cycle until a safely processable event is detected.
cEvent *event;
while (true) {
  // Get next event to process from fes.
  event = sim->msgQueue.peekFirst();
  // Verify that timestamp is below EIT.
  if (event->getArrivalTime() <= m_EarliestInputTime) {
    // This even can be processed safely.
    break;
  }
  else {
    // EIT reached, initiate new synchronization cycle.
    runSynchronizationCycle();
    // Keep waiting until EIT is raised.
    if (!receiveBlocking())
        return NULL;
  }
}

// Remove event from FES and return it.
cEvent *tmp = sim->msgQueue.removeFirst();
```

Listing 5.3: Basic synchronisation algorithm.

It peeks at the next processable event inside the FES without removing it. The algorithm then checks, wether its oricessable by verifying its timestamp is below the calculated EIT. Is this the case, the event is processed. If that is not the case, a new synchronisation cycle is initiated in order to calculate a new EIT.

5.2.4 Increased parallelism implementation

This section presents the implementation of the CNMA including the phase 2 calculation developed in section 4.1.3. After initiating a new synchronization cycle, each partition waits until it receives the EOTs from all other partitions of the parallel simulation. Deciding whether a synchronization cycle is complete is realized inside a finite state machine. Whenever a new synchronization is running, it keeps track of all received EOT broadcasts. The FSM switches

into ready state under the condition that an EOT broadcast was received from every partition of the simulation.

The final EIT calculation is realized inside calculateEIT shown in Listing 5.4. It is called whenever a running synchronization cycle finishes.

```
simtime_t cAdvCondNullMessageProtocol::calculateEIT()
{
  ...
  // Calculate initial ECOT matrix (see section 4.1.4.2).
  calculateECOTMatrix();
  // Calculate initial ECOT array (see section 4.1.4.3).
  calculateMinECOTArray();

  int sender = -1;

  // Calculate phase 2 (see section 4.1.4.4).
  while((sender = getNextPhase2Partition()) != -1) {
    for(int receiver = 0; receiver < numSeg; receiver++) {
      if(receiver != sender) {
        if ( (m_MinECOTArray[sender] != m_MAX_TIME) &&
             (m_AdjMatrix[sender][receiver] != m_MAX_TIME) ) {
          m_ECOTMatrix[sender][receiver] =
                          m_MinECOTArray[sender] +
                          m_AdjMatrix[sender][receiver];
        }
        else {
          m_ECOTMatrix[sender][receiver] = m_MAX_TIME;
        }
      }
    }
    calculateMinECOTArray();
  }
  // Return the new EIT
  return m_MinECOTArray[comm->getProcId()];
}
```

Listing 5.4: Final EIT algorithm implementation including phase 2 calculation according to section 4.1.4.5.

5.3 Domain specific optimization realization

This section presents the implementation of the domain specific optimization approach. As already mentioned in section 4.2, domain specific knowledge about the internal workings of the simulation is necessary. Analysing a simulation in order to gain domain specific knowledge, proved to be a time consuming task. Therefore, only Simulation 1 is investigated in order to prove the concept is working.

Section 5.3.1 presents the event pipeline of the TTE partition and is followed by section 5.3.2 which presents the event pipeline of the CAN partition. Section 5.3.3 discusses possibilities on how to exploit the insights about the CAN event pipeline (For partitioning details see section 3.3.2.1). Section 5.3.4 presents the implementation of the domain specific approach in Partition 1.

5.3.1 The TTE event pipeline

Figure 5.4 shows the TTE event pipeline. The reason for the higher complexity is because the

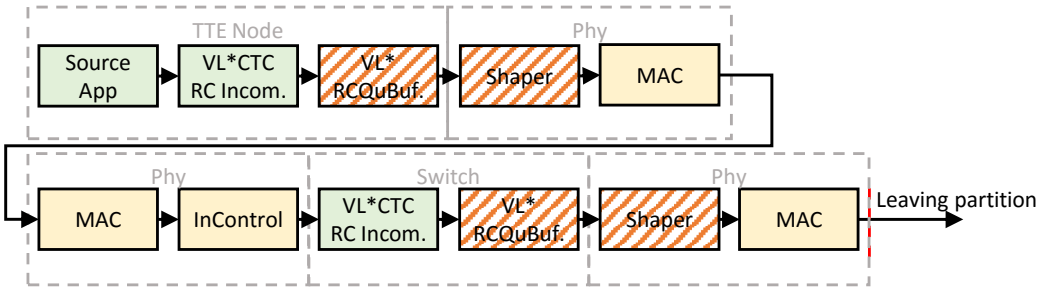


Figure 5.4: TTE event pipeline

TTE event pipeline does not have any static delays. Instead it consists out of a sequence of modules with variable delays, marked in orange. The buffers as well as the traffic shapers add variable delay to passing events depending on the current traffic volume and traffic priorities. In contrast to the CAN event pipeline the TTE event pipeline seems less complex at first. However, the effort to exploit domain specific knowledge is higher and requires a significant amount of changes in existing simulation code. This is why exploitation of domain specific knowledge is implemented for the CAN event pipeline in a first attempt.

5.3.2 The CAN event pipeline

The CAN event pipeline of Partition 1 is shown in Figure 5.5. Multiple messages are created at

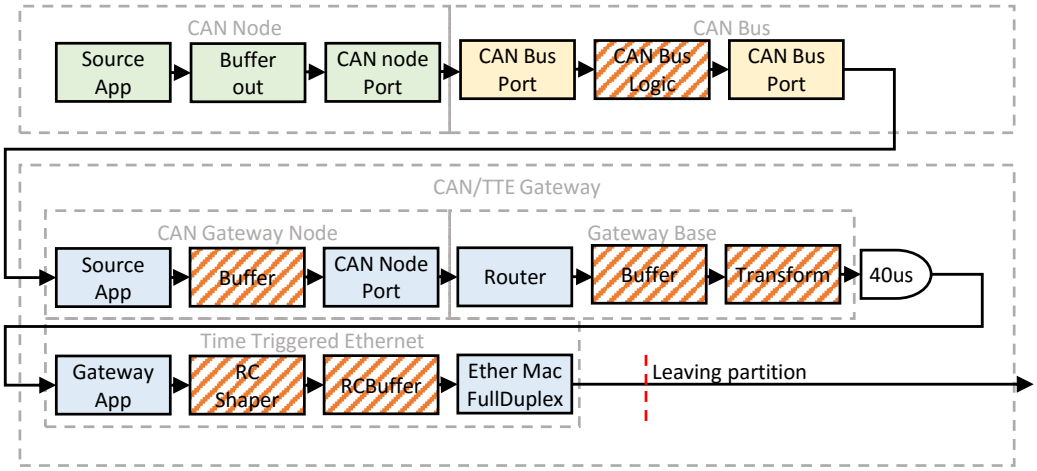


Figure 5.5: CAN message pipeline.

different intervals inside the Source App module. Each event passes an output buffer module and leaves the simulated CAN node through the CAN node port module. It then passes the CAN bus modules. The CAN bus logic module is responsible for event arbitration. This module is the first of the event pipeline that might delay the event. Next, the event enters the CAN gateway node. The CAN gateway node is responsible for transforming the event into a TTE event. The Router module decides whether CAN event are supposed to be forwarded into the TTE network or not. In case the router forwards a event towards the TTE network, the event enters the transform module next. The transform module transforms a CAN message into a TTE event. It possesses a fixed processing delay of 40 µs. After leaving the transform module, the event is passed on to the TTE traffic shaper. The traffic shaper is responsible to forward events according to their traffic class and priority. This module might add further delay until the actual event is sent. Finally the event leaves the CTG through the Ether MAC Full Duplex module. By leaving the CTG, the event is also becoming an external event since the receiving switch Switch 1 is assigned to Partition 2. Thus the event is crossing partition borders.

5.3.3 Exploiting the CAN event pipeline

Section 5.3.2 introduced the CAN event pipeline. It also pointed out the simulation modules that may add delays to the event processing. Figure 5.6 shows the delays of the CAN event pipeline in an idealized way. The pipeline contains two unspecific delays and one specific delay of 40us. The pipeline also contains a branch that only forwards specific events to the TTE network. Only these events will be sent to Partition 2 and thus become external events.

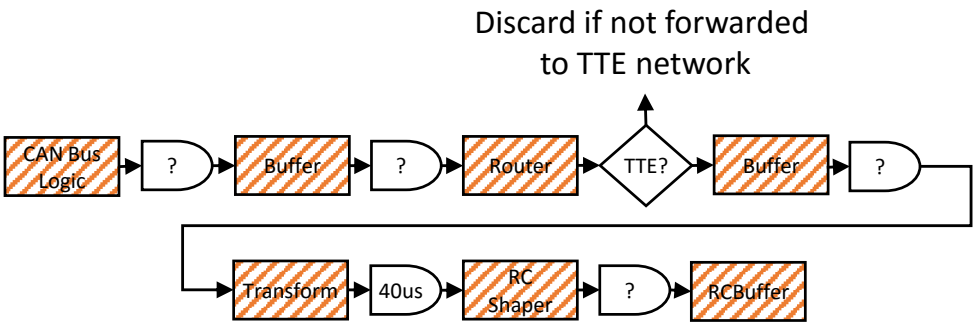


Figure 5.6: Idealized view on all delays inside the CAN event pipeline.

For this reason, the most convenient place to exploit domain specific knowledge is behind the router module. The transform module is located behind the transform module and is the only module with a static delay. In a first attempt on using domain specific knowledge, this static delay is used in order to increase the lookahead between both partitions. This means that from the time of event creation until leaving Partition 1 there will always be a delay of at least 40 µs. However, it might be even longer depending on the delays that will be added by the CAN bus logic and the traffic shaper module.

5.3.4 Implementation of the domain specific approach in Partition 1

The transform module is represented by class GatewayTransformation. The domain specific implementation starts with intercepting each event arriving at this module. This is done by modifying GatewayTransformation::handleMessage as shown in listing 5.5.

```
1  void GatewayTransformation::handleMessage(cMessage *msg)
2  {
3    ...
4    if (...){
5      if (m_pScheduler != nullptr) {
6        m_pScheduler->registerSyncTimestamp((*it)->getId(),
7                                    simTime() + m_TransformDelay);
8      }
9    }
10   ...
11 }
```

Listing 5.5: Intercepting CAN events arriving at the transform module.

For every CAN event arriving at GatewayTransformation, registerSyncTimestamp of the domain specific scheduler is called. As input parameter the event ID is given. The second parameter denotes, at what time the scheduler is supposed to expect an external event being generated from the currently received event. Therefore, the static transform delay of $40\,\mu s$ is added to the current time.

As a result, the domain specific scheduler now has a number of timestamps inside its priority queue as shown in Figure 5.7. In this particular example, the priority queue contains just

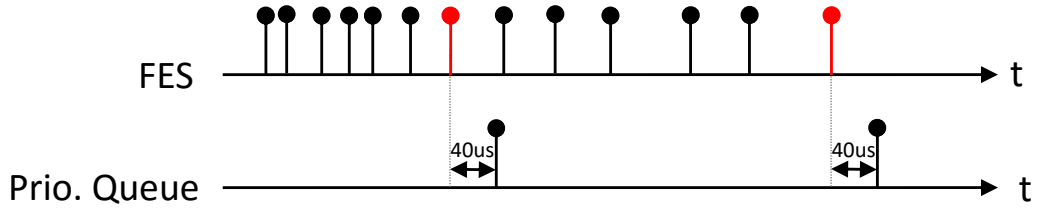


Figure 5.7: Example of the FES and priority queue in Partition 1 with calling registerSyncTimestamp inside GatewayTransformation::handleMessage. Actual CAN events inside the FES that reach the transform module are highlighted in red.

two timestamps, whereas the FES contains thirteen events. In conclusion the amount of synchronization cycles required by Partition 1 is reduced. This is the exact behaviour that is corresponding to the concept definition in section 4.2.4.2.

However, at the current state of the implementation, timing violations occur. This is, because there is a variable delay inside the event pipeline, which adds further delay to all events (see Figure 5.6). Therefore, it is required to tell the domain specific scheduler the time at which the external event finally leaves the partition. This is realized inside the Ether Mac FullDuplex module as shown in Listing 5.6.

```
1  void EtherMACFullDuplex::handleSelfMessage(cMessage *msg)
2  {
3    ...
4    if (msg->getKind() == ENDTRANSMISSION) {
5      cEndTxMessage* pEndTxMsg = (cEndTxMessage*) msg;
6      if (m_pScheduler) {
7        if (cSimulation::getActiveEnvir()->getParsimProcId() == 0) {
8          m_pScheduler->finishSyncTimestamp(pEndTxMsg->m_MsgId);
9        }
10     }
```

```
11   ...
12 }
```

Listing 5.6: Intercepting CAN events arriving at the transform module.

Whenever the transmission of an event is finished, EtherMACFullDuplex::handleSelfMessage is called and the transmitted event is passed as parameter. The scheduler is called with finishSyncTimestamp and the event ID is passed as parameter. Now the scheduler knows, that the event transmission is finished and it can remove the event from the priority queue.

One problem left is that while passing the CAN event pipeline, the event will change its event ID several times. This is because the event is either copied, destroyed and recreated or encapsulated in some other event. For this reason the scheduler can not correlate the event ID passed in Listing 5.6 with any event IDs inside its priority queue. In order to enable the scheduler to do so, it is required to inform the scheduler about every change of the event ID. One particular example where the event ID gets changed is the Buffer module that is in front of the transform module. This module is partly realized in class RCBuffer. Whenever this buffer sends an event, it sends a duplicate of the original event. The duplicate however, has a new event ID. Therefore, RCBuffer::handleMessage is slightly changed in order to let the scheduler know about the ID change. This is shown in Listing 5.7.

```
1  void RCBuffer::handleMessage(cMessage *msg)
2  {
3    ...
4    if (EtherFrame *outgoingMessage = getFrame()) {
5      ...
6      EtherFrame* pDup = outgoingMessage->dup();
7      if (m_pScheduler) {
8        m_pScheduler->updateSyncId(outgoingMessage->getId(),
9                                                  pDup->getId());
10     }
11     ...
12   }
13 ...
14 }
```

Listing 5.7: Reporting event ID changes to the scheduler.

6 Test

This chapter presents the performance and test results of the three scheduling strategies developed in this work. Section 6.1 presents the general verification of operation for all three algorithms. Section 6.2 presents the efficiency of all three scheduling strategies in regard to null message reduction. Section 6.3 presents the achieved speedup in regard to the sequential as well as the null message algorithm. Section 6.4 investigates why the most advance of all three scheduling strategies, the conditional null message algorithm that is using domain specific knowledge, does not perform as well as presumed.

6.1 Verification of operation

All three scheduling algorithms must be verified for correct operation before used in production environments. Conducting a formal verification is an expensive process. Therefore, it is decided to verify correct behaviour of all three scheduling strategies by comparing their output for equality. For this work the attestation of equal output should be sufficient to say that all three scheduling strategies perform correctly.

The simulation's output consists out of three files per partition. A vector file with *.vec file ending, and two data files with *.sca and *.vci file ending. The output of the null message algorithm that comes with OMNET++ and should be approved sufficiently is compared to the output of the three scheduling strategies developed in this work. All output files proved to be identical with minor differences consisting of different dates of creation and process numbers.

Furthermore the amount of simulation events that is exchanged between partitions is compared. It also proved to be identical as shown in Table 6.1 for Simulation 1 and in Table 6.2 for Simulation 2. In conclusion, all three simulation strategies produce identical output compared to the null message algorithm.

6.2 Efficiency Comparison

The next step in testing is to verify how efficient the simulation strategies perform in comparison to the null message algorithm. As stated in section 3.4, the NMA is producing an excessive

Table 6.1: This Table shows the amount of actual simulation events that are exchanged between both partitions of Simulation 1, for different simulation runtimes.

Simulation 1	5 sec	10 sec	15 sec	60 sec
NMA	37335	74728	112174	449062
CNMA	37335	74728	112174	449062
AdvCNMA	37335	74728	112174	449062
DspAdvCNMA	37335	74728	112174	449062

Table 6.2: This Table shows the amount of actual simulation events that are exchanged between both partitions of Simulation 2, for different simulation runtimes.

Simulation 2	5 sec	10 sec	15 sec	60 sec
NMA	62205	124500	186854	747936
CNMA	62205	124500	186854	747936
AdvCNMA	62205	124500	186854	747936

amount of null messages for synchronization purposes. Therefore, a major target of this work is to develop simulation strategies that reduce the amount of null messages. Table 3.11 shows

Table 6.3: Amount of total null messages sent to simulate 5 s of time.

	Simulation 1	Reduction	Simulation 2	Reduction
NMA	99990702		99989140	
CNMA	665692	150	756766	132
AdvCNMA	488454	204	477622	209
DspAdvCNMA	488190	204	N/A	N/A

the total amount of null messages generated in Simulation 1 and Simulation 2 in order to simulate 5 s of time. All newly developed simulation strategies manage to reduce the amount of null messages by a factor greater hundred. Therefore, the CNMA proves to be effective for this kind of simulations. The results also show a drawback with the domain specific approach. Although the advanced conditional null message algorithm with domain specific knowledge (see section 4.2.4) generates a slightly smaller amount of null messages compared to the advanced conditional null message algorithm. However, the reduction is neglect able and the average reduction factor computes to 204. This is an odd result, since the exploitation of domain specific knowledge is supposed to further reduce the amount of null messages generated. The reason for this lack in null message reduction is investigated in section 6.4.

In conclusion, all three scheduling strategies manage to keep the amount of required synchronization messages way below the amount of actual simulation messages. This is an indicator

for a scheduling algorithm's efficiency as described at the empirical analysis in section 3.3.2 and by the efficiency criterion in section 3.3.1.1. Table 6.4 shows the amount of null messages, simulation events and external events generated for each partition of Simulation 1 per scheduling strategy. The simulated time was 5 s. The measured values are now used to calculate the

Table 6.4: Simulation 1 amount of simulation events, external events and null messages for Partition 1 and Partition 2 measured for a period of time 5 s.

	Partition 1			**Partition 2**		
	Events	**Ext. Events**	**Null Msgs.**	**Events**	**Ext. Events**	**Null Msgs.**
NMA	1106758	12336	49995081	1039732	24999	99990702
CNMA	1106758	12336	332846	1039732	24999	332846
AdvCNMA	1106758	12336	244227	1039732	24999	244227
DspAdvCNMA	1106758	12336	244095	1039732	24999	244095

ratios between internal, external events and null messages. The internal event to null message ratio is expressed as in equation (6.1).

$$R_{int} = \frac{I}{N} \tag{6.1}$$

where

- R_{int} is the ratio between internal events and null messages.
- I is the total amount of internal events;
- N is the total amount of null messages.

The external event to null message ratio is expressed as in equation (6.2).

$$R_{ext} = \frac{E}{N} \tag{6.2}$$

where

- R_{ext} is the ratio between external events and null messages.
- E is the total amount of external events;

The calculated ratios for Simulation 1 are shown in Table 6.5. R_{int} shows, that all three scheduling strategies process an average of three to four internal events per null message, whereas the null message algorithm processes just one internal event every hundred null messages. From this point of view the new scheduling strategies prove superior over the

Table 6.5: This Table shows the ratio between internal events and null messages (R_{int}) and between external events and null messages (R_{ext}).

	R_{int}	R_{ext}
NMA	0.01431129	0.000248924
CNMA	3.224449145	0.056084496
AdvCNMA	4.394456796	0.076435038
DspAdvCNMA	4.3968332	0.076476372

null message algorithm. R_{ext} however, shows that all three strategies still have room for improvement. In an ideal situation, the amount of null messages almost equals the amount of external messages. This would result in $R_{ext} = 1.0$. This shows that all three strategies have a theoretical potential for improvement by a factor in the range of 14 to 20. However, with an R_{ext} between 0.05 and 0.07 they prove again far superior than the null message algorithm. The results for Simulation 2 in Tables 6.6 and 6.7 further support this statement.

Table 6.6: Simulation 2 amount of simulation events, external events and null messages for Partition 1 and Partition 2 measured for a period of time 5 s.

	Partition 1			**Partition 2**		
	Events	**Ext. Events**	**Null Msgs.**	**Events**	**Ext. Events**	**Null Msgs.**
NMA	1229767	16402	49998288	1466154	45803	49990852
CNMA	1229767	16402	378383	1466154	45803	378383
AdvCNMA	1229767	16402	238811	1466154	45803	238811

Table 6.7: This Table shows the ratio between internal events and null messages (R_{int}) and between external events and null messages (R_{ext}).

	R_{int}	R_{ext}
NMA	0.026962138	0.000622118
CNMA	3.562423523	0.08219846
AdvCNMA	5.644465707	0.130238976

6.3 Speedup Comparison

This section presents the achieved speed up that is reached by all three scheduling strategies in comparison to the sequential as well as the null message algorithm scheduler. Speedup in runtime is defined by formula (6.3).

$$S_{runtime} = \frac{T_{S1}}{T_{S2}} \tag{6.3}$$

where

- $S_{runtime}$ is the speedup in runtime of scheduler 2 with respect to scheduler 1.
- T_{S1} is the runtime of scheduler 1;
- S_{S2} is the runtime of scheduler 2.

The amount of null messages is reduced by a large amount by all three simulation strategies as shown in section 6.2. However, the speedup is not proportional to the amount of null messages. This is because the new scheduling strategies require some additional processing power. Another reason is that a null message requires much less processing power relative to an ordinary simulation event. Table 6.8 shows the total runtimes of Simulation 1. Table 6.9

Table 6.8: Total runtimes of Simulation 1 for different simulation runtimes and the different scheduling strategies. Green indicates runtimes smaller than the sequential counterpart, red indicates runtimes bigger than the sequential counterpart.

	5s	**10s**	**15s**	**60s**
Sequential	22,92	47,562	71,519	287,689
NMA	146,036	303,314	595,967	1960,704
CNMA	20,975	44,151	84,954	268,918
AdvCNMA	21,234	42,585	64,448	261,658
DspAdvCNMA	22,162	44,623	67,258	272,466

shows the average speedup against the sequential and the null message algorithm, calculated according to equation (6.3). For simulation the speedup is shown in Table 6.10. However, Simulation 2 lacks the results for the domain specific approach, since it is only specifically implemented for Simulation 1 [1]. The newly developed scheduling strategies manage to reduce the simulation runtime in both simulations. In Simulation 2 they manage to be faster than the null message algorithm by a factor of five. In Simulation 1 they are around seven times

[1] All the underlying measurement data can be found in Performance-Measurements.xlsx

Table 6.9: Speedup of Simulation 1 by the newly developed scheduling strategies against the sequential and the null message algorithm scheduler.

Average Speedup	**Sequential**	**NMA**
CNMA	1.07699253	7.506133389
AdvCNMA	1.10136857	7.685163415
DspAdvCNMA	1.05482246	7.360946287

Table 6.10: Speedup of Simulation 2 by the newly developed scheduling strategies against the sequential and the null message algorithm scheduler.

Average Speedup	**Sequential**	**NMA**
CNMA	1,10878881	5,277738271
AdvCNMA	1,09281726	5,208379756

faster than the null message algorithm. However, they do not perform that well against the sequential algorithm. All newly developed algorithms are slightly faster than the sequential algorithm. However, they just achieve an increase in performance by an average of five to ten percent.

6.4 Why the domain specific approach lacks in performance

The technically most advanced scheduling strategy which is using domain specific knowledge does not show the desired performance benefits. Actually, its performance arranges somewhere between the conditional null message algorithm from section 4.1.2 and the advanced conditional null message algorithm from section 4.1.4. This section presents the reason for this lack in performance.

The main idea of using domain specific knowledge is to reduce the amount of synchronization. This is done by ignoring all those events inside the partition's FES that will not create any external events. However, exploiting domain specific knowledge is only implemented in Partition 1 of Simulation 1 as described in section 5.3. Therefore, it ignores all events inside the FES for synchronization and just uses the events inside the priority queue. However, Partition 2 does not possess any domain specific knowledge. Therefore, the scheduler falls back to using the event inside its FES for synchronization. This leads to the situation that is shown in Figure 6.1. While Partition 1 is aware of the fact which events will create external events and therefore, could advance way ahead in time, Partition 2 does not. Without any domain specific knowledge it must result on all events inside its FES. Therefore, it can only advance in simulation time in intervals the size of the lookahead between both partitions.

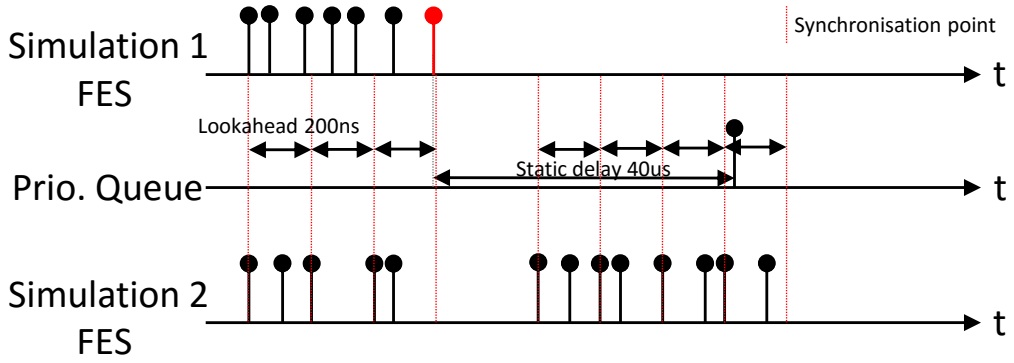


Figure 6.1: Example of the synchronization between Partition 1 with domain specific knowledge and Partition 2 without domain specific knowledge.

This insight leads to the conclusion that in order to make efficient use of domain specific knowledge, it must be implemented in every partition. Furthermore, in order to benefit from parallelism in the best way possible, all partitions should determine the same points in time for synchronization if possible. The more these synchronization points are displaced, the more synchronization takes place and less parallelism is achieved. An example is shown in Figure 6.2. The left half of the Figure shows displaced synchronization points. That means events that

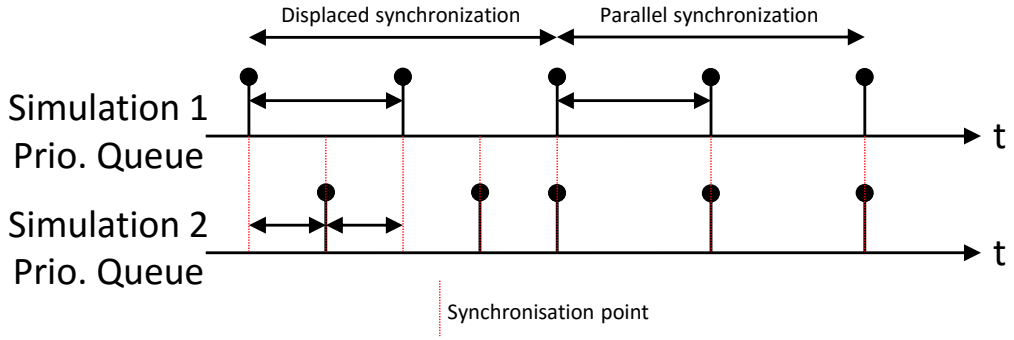


Figure 6.2: Example for displaced and optimal synchronization.

might create external events are scheduled at different points in simulation time for Partition 1 and Partition 2. This leads to an increased amount of synchronization cycles as can be seen by the red lines. The right half of the Figure shows parallel synchronization points. That means events that might create external events are scheduled at the same points in simulation time for Partition 1 and Partition 2. This leads to an decreased amount of synchronization cycles and an increase in parallelism.

7 Final

7.1 Summary

This master thesis describes the evaluation based design and development of parallel simulation strategies for in vehicle networks.

The first step consisted in gathering and presenting the necessary theoretical foundations of parallel simulation. These are described in chapter 2. A definition of simulation is given in section 2.1. The different classifications are described in section 2.2. The existing in vehicle network simulations were classified as classic discrete event based simulations. The basic concepts of parallel discrete event based simulations were developed next. They are presented in sections 2.4 and 2.4. The next step consisted in gathering the required knowledge about synchronization in section 2.6 and the basic differences about conservative and optimistic parallel simulations in sections 2.7 and 2.8. The most common and successful approaches to optimize these algorithms, were researched in a thorough literature research of more than fifty scientific publications about parallel simulation. The optimization approaches for optimistic simulation strategies are presented in section 2.7.4, those for conservative simulation strategies in section 2.8.2. Finally, a brief overview on time triggered ethernet is given in section 2.9.

Chapter 3 is devoted to a thorough problem analysis. The major goal was to get a basic understanding of the nature of in vehicle network simulations. Therefore, two typical simulations of the CoRE group were chosen for further investigation. Section 3.1 presents the simulation model of both simulations and explains the basic architecture. It is followed by a data flow analysis in section 3.2. The results of the data flow analysis were supposed to support the decision for a basic simulation strategy to continue with. Before starting with the concept development, the parallel potential of Simulation 1 and Simulation 2 was investigated in section 3.3. The parallel potential was analytically calculated in section 3.3.1. The results got empirically verified using the optimum eventlog based scheduler with zero synchronization overhead in section 3.3.2. Based on the analysis results and the theoretical evaluation of the available scheduling strategies, presented in section 2.8, the most promising approach is

identified in section 3.4. This approach was then selected as foundation for the evaluation based design of the parallel simulation strategies for in vehicle networks.

After coming to the conclusion that both simulations offer a sufficient enough parallel potential in section 3.4, the concept work was started with chapter 4. It was decided to split the concept development into a generic part in section 4.1 and a domain specific part in section 4.2. The generic scheduling strategy is supposed to work out of the box with all kinds of simulations, without further modifications. In contrast to the generic strategy, the domain specific scheduling strategy does not work out of the box. It is based on domain specific knowledge about the simulation's internals in order to optimize the amount of necessary synchronization. The advantage of this approach is that the generic scheduling strategies do not require modifications of existing simulations. If they prove successful, they could be used immediately. Therefore, the final concepts consist out of the original conditional null message algorithm which is described in section 4.1.2, an advanced conditional null message algorithm with additional optimization as described in section 4.1.4 and a domain specific scheduler that is based on the conditional null message algorithm in section 4.2.

After finishing the concept phase for all three scheduling algorithms, the implementation phase started in chapter 5. A brief introduction to the OMNET++ parallel simulation subsystem is given in section 5.1. It is followed by the implementation details of the generic scheduling strategy in section 5.2. The implementation of the basic conditional null message algorithm is explained in section 5.2.1. Section 5.2.2 gives an insight on lookahead calculation and how it is realized as part of the new scheduling strategy. The advanced conditional null message algorithm that was developed in section 4.1.3 and 4.1.4, is described in section 5.2.4. Section 5.3 describes the implementation of the domain specific approach. It starts with an analyzation of the TTE event pipeline in section 5.3.1, followed by an analyzation of the CAN event pipeline in section 5.3.2. Based on the decision it was decided to exploit domain specific knowledge in the CAN event pipeline only. This decision was based on the fact, that the implementation takes a considerable amount of effort and requires to change existing code.

After a successful implementation, the test phase started. The results are presented in chapter 6. Section 6.1 explains how the correct operation of all three scheduling strategies was verified. An efficiency comparison of all three scheduling strategies and the sequential and null message algorithm is presented in section 6.2. A presentation of the achieved speedup is given in section 6.3. All three scheduling strategies proved to be significantly faster than the already existing null message algorithm. However, the domain specific approach did not fulfil the expectations in regard to efficiency and speedup. The cause was investigated and the

reasons are presented in section 6.4. The theoretical maximum speedup that was empirically evaluated in section 3.3.2, was not reached by all three scheduling strategies.

7.2 Conclusion

The evaluation based design of parallel simulation strategies for in vehicle networks has been finished successfully.

The thorough analysis of the two underlying in vehicle network simulations was performed in order to get a thorough understanding of the simulation's behaviour. The efficiency criterion from [26] was applied successfully in order to estimate both simulation's parallel potential. The results promised parallel potential in both cases. A verification of these results was done by an empirical analysis with the eventlog based scheduler from [7].

Based on the analysis results, it was decided to use the conditional null message algorithm as a starting point to develop parallel simulation strategies. This decision proved successful by the following test results.

The first scheduling strategy, the conditional null message algorithm, proved to be way more efficient than the original null message algorithm. The amount of null messages was reduced by an average factor in between 130 to 150, in comparison to the null message algorithm. The speedup was around 700 percent in regard to the null message algorithm and around seven to ten percent in regard to the sequential algorithm. These results were surpassed by the second scheduling strategy. The advanced conditional null message algorithm managed to reduce the amount of necessary null messages by an average factor of 200, in regard to the null message algorithm. The achieved speedup was around 500 to 700 percent in regard to the null message algorithm and around nine to ten percent in regard to the sequential algorithm. The third scheduling strategy was using domain specific knowledge in order to further optimize runtimes. However, it did not prove as successful as desired. Its performance was similar to that of the other two scheduling strategies. At least from a theoretical point of view, it was supposed to be the fastest. An analysis showed that the third scheduling strategy suffered from the fact, that the domain specific part was just implemented for the CAN event partition.

All three scheduling strategies developed as part of this work proved far superior than the null message algorithm. They proved as well faster than their sequential counterpart. However, the speedup that was promised by the empirical analysis in section 3.3.2.2 was not reached. The intensive use of domain specific knowledge might improve these results. However, one must consider, that the analytical as well as the empirical analysis always assumed optimal conditions.

In conclusion, all three scheduling strategies form a solid foundation for further investigation on parallel simulation for in vehicle networks. Considering the empirical analysis, it seems safe to say that the maximum parallel potential is not yet reached. A big advantage is that two out of three scheduling strategies can be used out of the box. Therefore, it is not necessary to modify existing simulation code in order to take advantage of parallel simulation. A further advantage is that users do not need to obtain special knowledge on parallel simulation in order to use these schedulers. However, as soon as domain specific knowledge shall be used to further improve simulation times, a thorough understanding of parallel simulation is essential.

Finally, this work proved that the conditional null message algorithm and its variations are capable to speedup the two investigated vehicle simulations. This statement should be valid for all in vehicle simulations similar to those that were investigated in this work.

7.3 Outlook

This section gives a brief outlook on possible future work.

The scheduling strategy that is using domain specific knowledge did not perform as desired. Therefore, further investigations should be made in order to evaluate the full potential of using domain specific knowledge. In a first step one could implement the exploitation of domain specific knowledge for the TTE Partition. However, this step requires some effort and will result in modifications to a lot of existing code. A lot of those modules that need to be modified will as well be used in the CAN partition. Therefore, the modifications must be partition based. They must not be enabled for the CAN partition.

Dependent on the results, a second step would try to come up with a general concept on how to exploit and implement the usage of domain specific knowledge about a simulation's internals. This work again requires some effort and in depth knowledge about parallel simulation.

Another possibility is to take a look into optimistic scheduling strategies. However, this direction requires even more changes to existing code and should only be considered after maxing out the conservative scheduling strategies.

Bibliography

[1] T. Nolte, H. Hansson, and L.L. Bello. Automotive communications-past, current and future. *2005 IEEE Conference on Emerging Technologies and Factory Automation*, 1:985–992, 2005.

[2] N. Navet, Y. Song, F. Simonot-Lion, and C. Wilwert. Trends in Automotive Communication Systems. *Proceedings of the IEEE*, 93(6):1204–1223, jun 2005.

[3] Paolo Giusto, Alberto Ferrari, Luciano Lavagno, and Alberto Sangiovanni-vincentelli. Automotive Virtual Integration Platforms : Why ' s , What ' s , and How ' s Cadence Design Systems. 2002.

[4] Georg Kunz, Mirko Stoffers, James Gross, Klaus Wehrle, Olaf Landsiedel, Stefan Goetz, and Farshad Naghibi. Expanding the Event Horizon in Parallelized Network Simulations. In *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, pages 172–181, 2010.

[5] Georg Kunz, Mirko Stoffers, James Gross, and Klaus Wehrle. Runtime efficient event scheduling in multi-threaded network simulation. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, SIMUTools '11, pages 359–366, ICST, Brussels, Belgium, Belgium, 2011. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[6] SY Wang and CC Lin. Exploiting event-level parallelism for parallel network simulation on multicore systems. *... Distributed Systems, ...*, 23(4):659–667, 2012.

[7] Jan Raddatz, Franz Korf, and Till Steinbach. Determining the Optimization Potential of Conservatively Scheduled Parallel Simulations. *CoRE Group @ HAW Hamburg*, 2015.

[8] R. M. Fujimoto. Parallel discrete event simulation. *Commun. ACM*, 33(10):30–53, 1990.

[9] John S Carson and Jerry Banks. Introduction to Discrete-Event Simulation. *Simulation*, 1986.

[10] Friedemann Mattern and Horst Mehl. Diskrete Simulation – Prinzipien und Probleme der Effizienzsteigerung durch Parallelisierung. *Informatik-Spektrum*, 12(4):198–210, 1989.

[11] R. M. Fujimoto. Feature Article–Parallel Discrete Event Simulation: Will the Field Survive? *INFORMS Journal on Computing*, 5(3):213–230, 1993.

[12] R. M. Fujimoto. Parallel and Distributed Simulation Systems. *the 31st conference on Winter simulation: Simulation*, pages 147–157, 1999.

[13] Alois Ferscha and S K Tripathi. Parallel and distributed simulation of discrete event systems. *Event London*, pages 1–65, 1998.

[14] R M Fujimoto. Distributed simulation systems. In *Simulation Conference, 2003. Proceedings of the 2003 Winter*, volume 1, pages 124–134 Vol.1, 2003.

[15] K S Perumalla. Parallel and Distributed Simulation: Traditional Techniques and Recent Advances. In *Simulation Conference, 2006. WSC 06. Proceedings of the Winter*, pages 84–95, 2006.

[16] Shafagh Jafer, Qi Liu, and Gabriel Wainer. Synchronization methods in parallel and distributed discrete-event simulation. *Simulation Modelling Practice and Theory*, 30:54–73, 2013.

[17] Jerry. Banks, John S Carson, and Barry L Nelson. *Discrete-event system simulation*. Prentice Hall, 1999.

[18] Tang Wenjie and Yiping Yao. HSK: A Hierarchical Parallel Simulation Kernel for Multicore Platform. In *Parallel and Distributed Processing with Applications (ISPA), 2011 IEEE 9th International Symposium on*, pages 19–24, 2011.

[19] Rajive L. Bagrodia and Mineo Takai. Performance evaluation of conservative algorithms in parallel simulation languages. *IEEE Transactions on Parallel and Distributed Systems*, 11(4):395–411, 2000.

[20] Parallel Discrete and Event Simulation. An Algorithm For Reducing Null-Messages of CMB Approach in Parallel Discrete Event Simulation. Technical report, 1990.

[21] D Jefferson and H Sowizral. Fast Concurrent Simulation Using the Time Warp Mechanism. Part I. Local Control. Technical Report ADA129431, Rand Santa Monica, 1982.

[22] D Jefferson. Virtual time. *ACM Transactions on Programming Languages and ...*, 7(3):404–425, 1985.

[23] J S Steinman. SPEEDES - A multiple-synchronization environment for parallel discrete-event simulation. *International Journal in Computer Simulation; (United States)*, 2, jan 1992.

[24] Yaocheng Zhang and Ge Li. SafeBTW: A Scalable Optimistic Yet Non-risky Synchronization Algorithm. *2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation*, pages 75–77, jul 2012.

[25] Seng Chuan Tay Seng Chuan Tay, Yong Meng Teo Yong Meng Teo, and Rassul Ayani Rassul Ayani. Performance analysis of Time Warp simulation with cascading rollbacks. *Proceedings. Twelfth Workshop on Parallel and Distributed Simulation PADS '98 (Cat. No.98TB100233)*, 1998.

[26] A Varga, Sekercioglu, and G K Egan. A practical efficiency criterion for the Null Message Algorithm. In *Simulation in Industry: Proceedings of the 15th European Simulation Symposium (ESS 2003)*, 2003.

[27] C H Young and P A Wilsey. Optimistic fossil collection for time warp simulation. In *System Sciences, 1996., Proceedings of the Twenty-Ninth Hawaii International Conference on ,,* volume 1, pages 364–372 vol.1, jan 1996.

[28] Christoper H Young, Radharamanan Radhakrishnan, and Philip A Wilsey. Optimism: Not Just for Event Execution Anymore. In *Proceedings of the Thirteenth Workshop on Parallel and Distributed Simulation*, PADS '99, pages 136–143, Washington, DC, USA, 1999. IEEE Computer Society.

[29] M Chetlur and P A Wilsey. Causality Information and Fossil Collection in Time Warp Simulations. In *Simulation Conference, 2006. WSC 06. Proceedings of the Winter*, pages 987–994, 2006.

[30] Voon-Yee Vee and Wen-Jing Hsu. Pal:a new fossil ollector for time warp. In *Parallel and Distributed Simulation, 2002. Proceedings. 16th Workshop on*, pages 31–38, 2002.

[31] David Jefferson. Virtual Time II: Storage Management in Conservative and Optimistic Systems. In *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing*, PODC '90, pages 75–89, New York, NY, USA, 1990. ACM.

[32] Yi-Bing Lin and Bruno R Preiss. Optimal Memory Management for Time Warp Parallel Simulation. *ACM Trans. Model. Comput. Simul.*, 1(4):283–307, 1991.

[33] B R Preiss and W M Loucks. Memory management techniques for time warp on a distributed memory machine. In *Parallel and Distributed Simulation, 1995. (PADS'95), Proceedings., Ninth Workshop on (Cat. No.95TB8096)*, pages 30–39, jun 1995.

[34] Avinash C Palaniswamy and Philip A Wilsey. An Analytical Comparison of Periodic Checkpointing and Incremental State Saving. *SIGSIM Simul. Dig.*, 23(1):127–134, 1993.

[35] Yi-Bing Lin, Bruno R Preiss, Wayne M Loucks, and Edward D Lazowska. Selecting the Checkpoint Interval in Time Warp Simulation. In *Proceedings of the Seventh Workshop on Parallel and Distributed Simulation*, PADS '93, pages 3–10, New York, NY, USA, 1993. ACM.

[36] Lisa M Sokol and Brian K Stucky. MTW: Experimental Results for a Constrained Optimistic Scheduling Paradigm. In David Nicol, editor, *Distributed {S}imulation*, volume 22 of *Simulation*, pages 169–173. Society for Computer Simulation (SCS), San Diego, CA, 1990.

[37] R Rajan and P A Wilsey. Dynamically switching between lazy and aggressive cancellation in a Time Warp parallel simulator. In *Simulation Symposium, 1995., Proceedings of the 28th Annual*, pages 22–30, apr 1995.

[38] Christopher D Carothers, Kalyan S Perumalla, and Richard M Fujimoto. Efficient Optimistic Parallel Simulations Using Reverse Computation. *ACM Trans. Model. Comput. Simul.*, 9(3):224–253, 1999.

[39] Samir R Das and Richard M Fujimoto. A Performance Study of the Cancelback Protocol for Time Warp. *SIGSIM Simul. Dig.*, 23(1):135–142, 1993.

[40] Ian F Akyildiz, Liang Chen, Samir Ranjan Das, Richard Fujimoto, and Richard F Serfozo. Performance Analysis of "Time Warp" with Limited Memory. *Sigmetrics*, 20(1):213–224, 1992.

[41] Bruno R. Preiss, Ian D. MacIntyre, and Wayne M. Loucks. On the Trade-off between Time and Space in Optimistic Parallel Discrete-Event Simulation. *Computer Engineering*, (c), 1992.

[42] S.R. Das and R.M. Fujimoto. An empirical evaluation of performance-memory trade-offs in time warp. *IEEE Transactions on Parallel and Distributed Systems*, 8(2):210–224, 1997.

[43] R E Bryant. Simulation of Packet Communication Architecture Computer Systems, 1977.

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

[45] David Nicol and Richard Fujimoto. Parallel simulation today. *Annals of Operations Research*, 53(1):249–285, 1994.

[46] R.a. Meyer and R.L. Bagrodia. Path lookahead: a data flow view of PDES models. *Proceedings Thirteenth Workshop on Parallel and Distributed Simulation. PADS 99. (Cat. No.PR00155)*, 1999.

[47] K M Chandy and R. Sherman. The conditional-event approach to distributed simulation. *Contract*, (June), 1989.

[48] Kenneth R. Wood and Stephen J. Turner. A generalized carrier-null method for conservative parallel simulation. *ACM SIGSIM Simulation Digest*, 24(1):50–57, 1994.

[49] T.D. Blanchardt, T.W. Laket, and S.J. Turner. Cooperative distributed Acceleration : discrete robust event conservative simulation. *ACM SIGSIM Simulation Digest*, 24(1):58–64, 1994.

[50] B. D. Lubachevsky. Efficient distributed event-driven simulations of multiple-loop networks. *Communications of the ACM*, 32(1):111–123, 1989.

[51] L.M. Sokol, J.B. Weissman, and P.a. Mutchler. MTW: an empirical performance study. *1991 Winter Simulation Conference Proceedings.*, 1991.

[52] Bruno R Preiss, Wayne M Loucks, Ian D Macintyre, and James a Field. Null Message Cancellation in Conservative Distributed Simulation Department of Electrical and Computer Engineering University of Waterloo Waterloo , Ontario , Canada , N2L 3G1. *Time*, (c):1–6, 1991.

[53] Paul F. Reynolds. A shared resource algorithm for distributed simulation. *ACM SIGARCH Computer Architecture News*, 10(3):259–266, 1982.

[54] Jayadev Misra. Distributed discrete-event simulation. *ACM Computing Surveys*, 18(1):39–65, 1986.

[55] J Kent Peacock, Johnny W Wong, and Eric G Manning. Synchronization of Distributed Simulation Using Broadcast Algorithms. *Computer Networks*, 4:3–10, 1980.

[56] R.a. Meyer and R.L. Bagrodia. Improving lookahead in parallel wireless network simulation. *Proceedings. Sixth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (Cat. No.98TB100247)*, 1998.

[57] Moo Kyoung Chung and Chong Min Kyung. Improving lookahead in parallel multiprocessor simulation using dynamic execution path prediction. *Proceedings - Workshop on Principles of Advanced and Distributed Simulation, PADS*, 2006:11–18, 2006.

[58] Zhenjiang Dong, Jun Wang, George F. Riley, and Sudhakar Yalamanchili. A study of the effect of partitioning on parallel simulation of multicore systems. *Proceedings - IEEE Computer Society's Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, MASCOTS*, pages 375–379, 2013.

[59] Cheng-Hong Li, Alfred J. Park, and Eugen Schenfeld. Analytical Performance Modeling for Null Message-Based Parallel Discrete Event Simulation. *2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 349–358, 2011.

[60] Amos Albert. Comparison of event-triggered and time-triggered concepts with regard to distributed control systems. *Embedded World*, pages 235–252, 2004.

[61] W. Su. Variants of the Chandy-Misra-Bryant distributed discrete-event simulation algorithm. Technical report, 1988.

[62] A. Varga, Y. A. Sekercioglu, and G. K. Egan. Estimate the Parallel Potential of Your Model, 2007.

[63] Ahmet Y Sekercioglu, Andras Varga, and Gregory K Egan. Parallel Simulation Made Easy with OMNeT++. In *Proceedings of the European Simulation Symposium*, Delft, The Netherlands, 2003.

Glossary

Causality violation is the execution of an event out of time correct order.

DES discrete event based simulation.

Event is a discrete point in time of a simulation, associated with a change of state of its model.

Event scheduler is an algorithm responsible for processing events in time correct order.

External event is an event created by one partition, sent to another partition and processed by that partition.

FES is the future event set that stores events in time correct order.

Internal event is an event created by one partition and processed by the same partition.

IPC is interprocess communication and allows message exchange between processes.

Model represents the behaviour and functions of the simulated systems relevant key features.

NMA is the null message algorithm which is a conservative synchronization algorithm (see section 2.8.1).

Null message is a special event that is used to prevent deadlocks in NMA.

Partition is one part of a partitioned model.

Partitioning is the process of splitting a model into separate partitions.

PDES parallel discrete event based simulation.

Processing unit can be a central processing unit (cpu) or a single core of a multi-core processor.

Real time is the real time domain or in other words the real time passed for a user for running a simulation.

Simulation according to [17], is the imitation of the operation of a real-world process or system over time.

Simulation system is a software offering the necessary functionality to run a simulation on a model over time.

Simulation time is the simulations time domain.

System is a real world system or process that will be simulated. Common examples are networks, motors etc..

.

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 2. Februar 2016 Jan Raddatz