Universität Stuttgart

INSTITUT FÜR KOMMUNIKATIONSNETZE UND RECHNERSYSTEME
Prof. Dr.-Ing. Andreas Kirstädter

Interner Bericht / Internal Report **Nº 57**

Titel / Title

Batch-Level Parallelism with the IKR SimLib

Verfasser / Author(s) M. Proebster, C. Blankenhorn, C. M. Mueller, J. Sommer, T. Werthmann

Datum / Date	15.04.2010
Umfang / Size	7 Seiten / Pages
Quelle / Source	
Schlüsselworte / Keywords	SimLib, Batch-Mean, Parallel Simulation

Beitrag der Arbeit / Achievement

Different possible levels for parallelization of simulations are introduced and explained shortly. An overview of the SimLib is given and the concept of batch-level parallelism is explained.

Kurzfassung / Abstract

A current trend in the performance evaluation of communication networks is the increasing complexity of simulation models. This particularly applies to wireless communication networks, where effects on different lower layers cannot be decoupled anymore. This results in higher computation cost and thus leads to longer execution times. Nowadays, single processor cores do not become significantly faster anymore. Only integrating more and more cores in a single processor can further increase the processing power. Parallelization can utilize the multi-core environment to reduce the simulation running times.
We discuss multiple levels of parallelism in event-driven simulations. We then focus on batch-level parallelism, which is a convenient way to distribute batches to several processor cores without the need for synchronization. We present the implementation of batch-level parallelism in our institute's simulation library.

Batch-Level Parallelism with the IKR SimLib

Magnus Proebster, Christian Blankenhorn, Christian M. Mueller,
Jörg Sommer, Thomas Werthmann
Universität Stuttgart
Institute of Communication Networks and Computer Engineering
Pfaffenwaldring 47, Stuttgart, Germany
magnus.proebster@ikr.uni-stuttgart.de

ABSTRACT

A current trend in the performance evaluation of communication networks is the increasing complexity of simulation models. This particularly applies to wireless communication networks, where effects on different lower layers cannot be decoupled anymore. This results in higher computation cost and thus leads to longer execution times. Nowadays, single processor cores do not become significantly faster anymore. Only integrating more and more cores in a single processor can further increase the processing power. Parallelization can utilize the multi-core environment to reduce the simulation running times. In this paper, we discuss multiple levels of parallelism in event-driven simulations. We then focus on batch-level parallelism, which is a convenient way to distribute batches to several processor cores without the need for synchronization. We present the implementation of batch-level parallelism in our institute's simulation library.

1. INTRODUCTION

Discrete, event-driven simulation often is the only way to evaluate the performance of new communication protocols, algorithms, or systems. In many cases, lab experiments with prototypical implementations are unfeasible, either because the technology under evaluation is not yet available, or because the scale of the experiment exceeds the capabilities of the lab environment. Analytical evaluation also quickly becomes intractable, e. g. if the variables and parameters are too numerous, the system is highly non-linear, or the distribution functions are unknown. The key to a sound simulation study that provides valuable insight is to create a credible simulation model at the right level of abstraction. The researcher has to decide which aspects of a complex real-world system to consider in the simulation model. Guidelines for modeling and validation of the models can be found in chapter 5 of [12] and the references therein.

Recent developments in wireless and wireline networks require simulation models that span over large parts of the protocol stack. In wireless networks, techniques such as channel-dependent scheduling, coordinated multi-point transmissions, interference coordination, and others couple effects on the physical layer with actions on the MAC and radio resource management layer [16]. Furthermore, performance gains of promising physical layer algorithms need to be investigated under different traffic patterns and in scenarios with multiple base stations and users. In wireline networks, the development of new transport protocols for the Internet often requires integrating the whole protocol stack of an operating system in the simulation model, e. g. the Network Simulation Cradle [1] follows this approach.

The necessity to take additional cross-layer effects into account leads to more complex simulation models. This translates to higher computational cost, which results in longer execution times when no counteractions are taken. While in the past, the continuously increasing computation power of general purpose CPUs helped to deal with this complexity, the situation today is different. The computation power of a single processor core does not increase at the same speed as it used to in the past. Instead, chip designers integrate additional processor cores on a single die. However, the simulation of communication networks cannot benefit from more cores as straightforward as simulations in other disciplines, such as fluid dynamics or climate models.

In this paper, we discuss different levels of parallelism for event-driven simulation with their advantages and disadvantages. We then focus on what we call *batch-level parallelism*, which allows distributing multiple batches of a single simulation run to a number of processor cores or different machines. Batches here refer to the widely applied batch-means-method, where a simulation is composed of a transient phase and the actual measurement phase, which is further subdivided into several intervals [12]. These intervals, denoted as batches, are treated as independent observations of the desired measurement value, from which a mean value, confidence intervals, and other statistical properties can be calculated.

While a simulation study usually consists of several simulation runs, each of which is composed of several batches, batch-level parallelism makes sense if the number of processor cores exceeds the number of simulation runs or if there are specific parameterizations that require significantly longer execution times. Another possible field of application is the design phase of algorithms, which usually is an iterative process. Here, quick performance estimates are desired without having to wait for hours or days until an entire simulation study completes. A simulation tool that supports batch-level parallelism is particularly useful if the simulation model, its implementation, and the output data analysis remain unchanged, irrespective of whether the simulation is run on a single or on many cores. Another important property is the flexibility in the number of cores, i. e. whether a simulation run with 20 batches can be distributed to one, two, or a dozen of cores.

The remainder of this document is organized as follows: In section 2, we discuss possible levels of parallelization for discrete, event-driven simulation of communication networks. In section 3, we give an overview of the *IKR SimLib*, which is our institute's simulation library. Our main contribution lies in section 4, where we focus on batch-level parallelism and describe how it is integrated in our simulation library. Finally, section 5 concludes this work.

2. PARALLELIZATION CAPABILITIES

As discussed above, parallelization can be utilized to reduce the runtime of simulation studies in comparison to a sequential execution. It further makes efficient use of distributed processing resources, e. g. in multi-core environments. Parallelization can be performed on different layers. Without touching the simulation model itself, different parameterizations can be evaluated in parallel. If the method of independent replications [12] is used, the execution of these independent replications can also be distributed onto different processing cores. Further parallelization can be achieved by distributing the simulation model by components or even single algorithms of the simulator. The advantages and disadvantages of these approaches will be discussed in the following paragraphs. A good reference for parallel simulation can be found in [9].

2.1 Parameter Level

Most simulation studies evaluate performance criteria in dependence on parameters. Therefore, it is common to perform multiple experiments with different parameter sets. These experiments are independent and can therefore be executed in parallel on multiple processor cores or different computers without additional synchronization overhead. A parallelization limit on this level is the number of parameter sets.

2.2 Batch Level

To obtain statistically significant results from a simulation, a sufficiently large share of the space, which is span by the influencing random variables, has to be covered. In addition, multiple independent results are required to calculate confidence intervals of the estimated results. Two approaches to this problem exist. In the first approach, many independent replications are simulated, each replication starting with a different initialization of the pseudorandom number generators. The second approach, the formerly mentioned batch-means method, splits one single simulation into batches. The batch results can be assumed to be independent if the correlation between the batches is sufficiently low. It's also possible to mix both approaches, so that each independent run is split into batches. [5] discusses the advantages and disadvantages of independent replications and batch-mean methods.

Which approach is preferable depends on the system properties. If the time correlation of the observed values is high, the length of the batches has to be increased to get sufficiently independent batch results. In this case, independent replications are more efficient, as there is no correlation between the runs per definition. Usually, we are interested in the steady state of the investigated system. Therefore, the initial transient (ramp-up) phase, when starting with an empty system, needs to be neglected. For independent replications, each run has to start with an own transient phase. If the length of this phase is in the order of magnitude of the length of the whole run, this method leads to a high amount of overhead.

The parallelization of the independent runs does not introduce synchronization overhead. Without transient phases, nearly ideal speedup can be achieved independent of the simulation model. Long transient phases reduce the efficiency of this approach, but the simulation runtime may still be reduced. An additional degree of freedom is introduced by the possibility to mix independent replications with the batch means method. This is supported by our simulation library. We further detail this in section 4.

2.3 Model Component Level

To further parallelize the simulation, the model itself has to be split and distributed. This introduces synchronization overhead and an additional source of errors. Thus, the splitting has to be planned carefully. In a typical simulation of a communication network, the behaviour of multiple nodes is modeled, while each node comprises multiple layers of a protocol stack. Two approaches to split the model arise from this architecture: Either distribute the nodes, including all their layers (vertical cut), or cut between the layers and distribute the processing of the layers (horizontal cut).

Horizontal splitting of the model often corresponds to a modularization of the simulation code. The interfaces of the modules are well defined, which makes synchronization less error-prone in this case. If there is only forward communication between two layers, the output of the first part can be stored to disk. This allows multiple simulations of the second part without simulating the first part again. For example, this is applicable for the simulation of mobile communication networks. There, the communication network usually does not influence the movement of the users and the channel attenuations. Therefore, the movement can be calculated in advance and stored to a trace file, which is then read during the simulation of the network under study. To get a high degree of parallelization, the model has to be split into many components. This increases the implementation overhead. If the model components do not have the same complexity, horizontal splitting leads to an unequal distribution of load on the processing cores.

In contrast to that, vertical splitting leads to a more even distribution of load, because the simulation blocks are of the same type, which results in equal computational complexity. If the nodes are instances of the same program code, distributing them can achieve a higher degree of parallelization with less implementation effort in comparison to horizontal splitting. In a typical model of a communication network, communication between nodes should only happen by using the lowest layer of the modeled protocol stack, which simplifies the separation of the nodes. However, often idealizations are implemented that allow direct communication of entities of the same protocol in multiple nodes. The approach of vertical splitting is used e. g. by the simulator OMNeT++ [17].

Independent of how the simulation model is split into parts, these can be parallelized using multiple techniques.

Shared Memory
Most common is the use of multithreading in combination with shared memory communication. All common programming languages and standard computers with multiple processor cores support this type of multithreading. It provides fast synchronization and very high bandwidth. As it is usually restricted to a single computer, the degree of parallelization is limited.

Distributed Memory
To increase the degree of parallelization, the simulation can be distributed to multiple computers. As these do not share a common main memory, the communication between the processes of the simulation has to be performed over a communication network. E. g. , this can be implemented by using message passing or remote procedure calls. The performance of this approach is limited by the performance of the network. Although a high degree of parallelization may be achieved, this approach is not feasible if the simulation requires much synchronization.

Job-Offloading
While the preceding approaches are most suitable for symmetric computer architectures, an additional approach can make use of asymmetric architectures with specialized hardware. This can be a device designed for floating point calculations, as the OpenCL architecture [13] or the IBM Cell processor [10], or a hardware implementation of a special functionality. By splitting the model horizontally and moving computational costly parts of the simulation to such a device, the flexibility of a standard computer can still be utilized for the greater part of the simulation. However, this approach requires porting some part of the simulation to a different architecture. In addition, this approach is useful only if the simulation has a central bottleneck. It has been used, e. g. for the calculation of a radio channel model in [2].

In general, splitting and parallelizing the model can increase the degree of parallelization significantly. However, a high synchronization overhead is introduced if the model is not coupled loosely. The necessity of synchronization increases the complexity of the whole simulation program. This introduces new sources of error and makes debugging difficult. Because of these drawbacks, parallelization on the level of model components is only sparely used by the authors.

2.4 Algorithm Level
If the main computational effort is caused by calculations of single algorithms, it may make sense to parallelize them. The mechanisms explained above for parallelization on model component level are also applicable. Besides, vectorization is an additional approach that can be utilized without great expense.

Most modern general purpose processors support instruction set extensions for vector operations. This Single Instruction Multiple Data (SIMD) functionality allows executing the same instruction on multiple operands in parallel. The most popular implementations are SSE and its extensions for the x86 architecture [18] and AltiVec for the Power architecture [8]. They allow operating either on four single precision or on two double precision numbers in parallel. An extension to double the register width and thereby the number of operands is planned [4]. Special architectures, e. g. the NVidia CUDA platform, support much wider vectors.

To utilize vectorization, it is possible to hand-tune the algorithms by implementing central parts in assembler code. As this results in a high implementation effort and is error-prone, compiler support is often used. Vectorization can either be configured with special intrinsics, or be used automatically whenever the compiler detects suitable blocks [7]. On the one hand, manual configuration requires the programmer to have special knowledge of the architecture. On the other hand, it does achieve a higher efficiency, because common compilers have only limited support for automatic vectorization. A convenient way to use all capabilities of the architecture is to use specially tuned libraries like ACML [3]. However, these do not always support all the required functionality.

To use vectorization to speed up a certain algorithm, the algorithm has to be inherently parallelizable. If this is the case, a maximum speedup of 4 (for single precision) or 2 (for double precision) can be achieved on current main stream processors. Whether the vectorization can be performed by the compiler or special libraries can be used depends on the respective algorithm. Even if it has to be implemented manually, this parallelization approach has an outstanding advantage: While other approaches require additional processing cores to reduce the total simulation runtime, the vectorization blocks usually are idle and can be utilized without additional hardware expenses.

3. THE IKR SIMULATION LIBRARY
The Simulation Library (SimLib) of the Institute of Communication Networks and Computer Engineering (IKR) at the Universität Stuttgart is a tool for event-driven simulation of complex systems in the area of communications engineering.

3.1 Overview
The first version of the IKR SimLib was released in the 1980s. Since that time, we have improved the library continuously. The continuous improvements lead to a wide field of application, also outside of the institute. The IKR SimLib was used and is still used for several publicly and privately funded research projects, as well as student projects. Up to now, more than one hundred of these student projects have been finished. Furthermore, IKR's industrial partners use this library for complex simulations.

In 2008, we ported the IKR SimLib to Java while keeping all concepts and mechanisms of the predecessor C++ class library. Today, two editions of the IKR SimLib are available: The C++ edition and the Java edition. Each edition comes as a separate class library. We developed both editions in consideration of modern object-oriented design principles and clean software architecture. Both editions are publicly available under the GNU Lesser General Public License (LGPL). This allows changes within the libraries itself as well as proprietary programs to use it.

The design objectives of the IKR SimLib were manifold. The IKR SimLib is problem-oriented in a sense that it supports an effective implementation of an abstract communication system model. Each simulation model component can consist of submodels and other components. This leads to a hierarchical modeling approach. The components are encapsulated and communicate with each other by exchanging messages using ports. This offers a high reuse and an evolutionary redefining of new components by modifying existing ones.

Since the launch, the IKR SimLib has proved its applicability for performance evaluation in a multitude of communication areas, e. g. for IP, photonic, mobile, signaling, in-vehicle, and P2P networks. For getting the latest version of the IKR SimLib, a more detailed list of examples, and getting selected publications, please visit our website at *http://www.ikr.uni-stuttgart.de/IKRSimLib*.

Writing a simulation program based on the IKR SimLib requires a basic understanding of the library. The simulation libraries come along with extensive documentation, comprehensible tutorials, and examples. These help to get a quick insight on the library. The philosophy of the IKR SimLib is that the challenge lies in the appropriate modeling. The model has to reflect the object of investigation in an abstract, but specialized manner. Therefore, the library does not include ready-to-use implementations such as a HTTP/TCP/IP protocol stack or a WDM network. Instead, the library offers basic components such as queues, statistics, and generators that enable an easy and fast implementation of a pre-designed model.

3.2 Architecture and Conceptual Structure

The IKR SimLib is structured into three main parts as shown in Figure 1. The *basic concepts* include simulation support mechanisms as well as I/O concepts. Besides, the *modeling concepts* support a hierarchical modeling approach to create individual components that communicate with each other by exchanging messages. The *standard components* are composed entities like a traffic generator, which provide a simple model implementation. In the following sections, we describe each part in more detail.

3.2.1 Basic Concepts

The *basic concepts* support mechanisms and components that are necessary to control and execute an event-driven simulation. One of these mechanisms is the *simulation control* that handles the initialization, e. g. when to stop the transient phase and begin with the actual performance evaluation phase and finally when to stop the simulation batches. The control also signals the according changes to all objects needing this information. Furthermore, the basic concepts offer inherent support for *event handling*, e. g. by providing a calendar. While processing an event, it is possible to post new events, which are entered into the calendar. After processing of an event is finished, the next event in the calendar is processed.

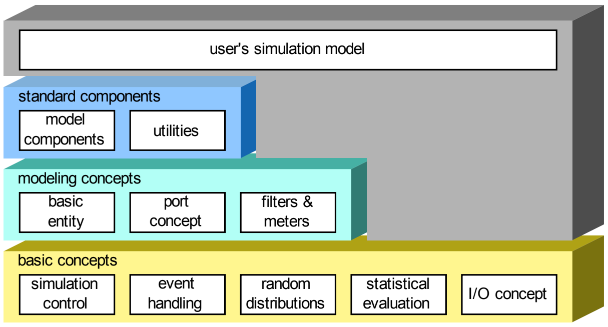

Figure 1: Basic structure of the IKR SimLib

The IKR SimLib supports stochastic processes and *on-the-fly* statistical evaluation. An important aspect is the distribution-oriented random number generation. The IKR SimLib implements many continuous and discrete *distributions*. *Statistical evaluation* is supported by many different statistics, too. Examples are the sample counter, conditional mean and correlation statistic. During a simulation run, the IKR SimLib computes statistical data, e. g. confidence interval. Therefore, a complex post-processing step is unnecessary. One distinguishing feature from many other simulation tools is the provisioning of metrics dealing with the statistical significance, which is in case of the IKR SimLib a student t-test based confidence interval. In addition, the library includes a flexible *I/O concept* which consists of a file parser for reading parameters and an XML-based output concept for printing results.

3.2.2 Modeling Concepts

The next main part of the IKR SimLib provides *modeling concepts*. In general, a model has a hierarchical structure and consists of several components and entities that communicate with each other. Entities are able to post and handle events. Each entity is derived from the base class *Entity* and has a unique local name which is chosen arbitrarily. This base class defines the common properties of all entities and methods for dealing with ports and events. The name helps to identify the entity and to locate it via a central component manager.

The hierarchical decomposition of an entity into a hierarchy of components or entities decreases the complexity. It enables a separate handling and treatment of each entity. This principle corresponds to the *divide-and-conquer* approach and leads to a tree structure of entities and components with the model itself as the root entity [11]. All entities are strictly encapsulated and communicate with each other by exchanging messages. This message exchange works by using so-called ports, which define a generic external interface of an entity. This *port concept* enables the interconnection of entities in a plug-n-play manner.

Furthermore, *filters and meters* are connected to ports. Filters inspect and may change messages based on certain rules, e. g. changing specific fields within the message. In contrast to this, meters primarily update statistics with values derived from the messages, e. g. the message length or time of arrival.

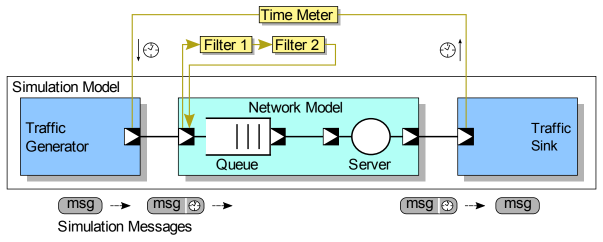

Figure 2: Example simulation model

3.2.3 Standard Components

The *standard components* are the third part. *Model components* like traffic generators, queues, servers, multiplexers, traffic sinks etc. are provided to ease model implementation. They also have a hierarchical structure. This offers a reuse of submodels and components that can be further redefined. Together with further utilities, they allow a simple model generation, especially for queuing networks.

3.3 Simple Simulation Model

For illustrating the concepts of the IKR SimLib, Figure 2 depicts a model of a simple single-server queuing network which comprises the network model, a traffic generator, and a traffic sink. The port concept and a message transfer protocol enable that the simulation messages are passed from component to component. After a component recognizes a new message at the output port, this port notifies the corresponding input port. For example, in Figure 2 each time when the traffic generator generates a new message, its output port informs the input port of the queue. Then, the receiving component decides if the message will be accepted.

Because of the flexible port concept, the integration of filters and meters into the model is easy. They read and evaluate the flow of messages at various points within the model. In Figure 2 the integrated *Time Meter* measures the processing time in the network model including the waiting time in the queue and the holding time in the server. For this purpose, the time meter adds a time stamp to the message when it passes the output port of the traffic generator. When the message passes the input port of the traffic sink, the time meter reads and removes the time stamp. The two other filters in this figure observe the messages that are passing by the input port of the queue. For example, one of these filters might record a trace of messages of a defined traffic class.

3.4 Extensions

Today additional libraries exist that are built on top of the IKR SimLib. An example is the IKR Emulation Library (IKR EmuLib) [14, 15]. This library can emulate a system that is specified as a simulation model, i. e. we can use the same model in simulation and emulation in an efficient and lightweight manner. Consequently, the effort for enhancing an existing simulation tool with emulation capabilities is minimal. The IKR EmuLib is also available in two editions: The C++ edition and the Java edition.

A further example is the IKR TCP Library (IKR TCPLib) [6]. This library offers a basic implementation with all important

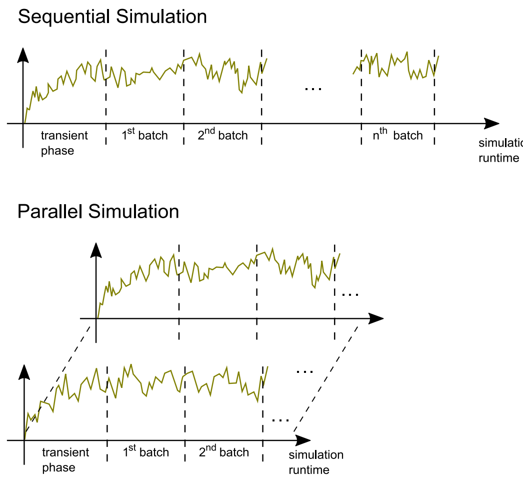

Figure 3: Principle of batch parallelism

TCP mechanisms (e. g. , flow and congestion control) and allows simulation of elastic applications and elastic traffic flows. The TCP components, which are included in this library, enable to model unidirectional TCP connections. The simulations results of the IKR TCPLib are comparable to other simulation environments, such as ns-2 (UC Berkeley, LBL, USC/ISI, and Xerox PARC). At the moment, the IKR TCPLib is only available in C++. We are working on porting the library to Java. The usage of the extensions, namely the IKR EmuLib and the IKR TCPLib, is optional.

4. BATCH-PARALLELISM IN THE IKR SIMLIB

In many network simulations, we want to know the steady-state mean of a certain value (e. g. queue length). This means that the initial transient phase, when starting with an empty system, needs to be neglected as this would impose a bias on the simulation results. For this type of simulations either the independent replications or the batch-mean methods are common. The design of the IKR SimLib supports the batch-mean method, but it can also be configured to run independent replications.

When trying to reduce the runtime of a simulation, the following questions arise:

- Which processing time is needed to achieve the same accuracy with either of the approaches?
- To what extent can the simulation be parallelized without the need for synchronization?

As shown before, parallelization is one way to reduce simulation runtime. If we use a fixed number of processor cores in the simulation environment, we can obtain results after the shortest time, when all the cores are busy during the whole time. This is easily possible when the number of simulation points is a multiple of the number of available cores. Then, without synchronization, each simulation point can be executed with the batch-mean method on a different processor. However, when this is not the case, another parallelization

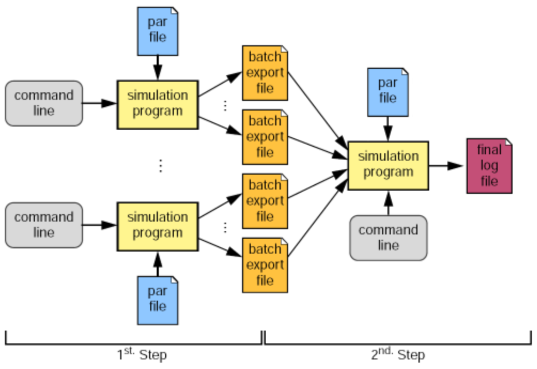

Figure 4: Workflow of batch parallelization with IKR SimLib

dimension comes in handy. It is obvious that the independent replications method can be run on several different processing cores. However, for the pure batch-mean method, this is not as easily possible. Therefore, the IKR SimLib offers the flexibility to combine both approaches, i. e. doing some independent runs with multiple batches. This means that a single simulation point is divided into multiple runs, each containing a fraction of the desired number of batches. Figure 3 shows the principle of the simulation flow and the difference to the conventional batch-means method. While in the case of sequential simulation the is just one process with a transient phase and each batch simulated after the other, with parallel simulation each process needs an individual transient phase to reach the stable state before the first batch starts. We can do this batch parallelization without changing anything to the post-processing steps, as it is already integrated within the statistical processing framework of the IKR SimLib. It is worth mentioning, that, when applying batch-parallelism, it is very important to have a reasonably defined transient phase. Otherwise, the initialization effects would influence the statistical properties of the simulation outcome.

4.1 Preparing the Simulation

After model implementation and testing, the execution of simulations is the next step. The defined simulation parameters span the parameter space. An extra tool called *IKR SimTree* generates a directory structure according to these parameters. Within this, it controls the whole workflow of the simulation, including the distribution of the simulation processes and the collection of the results. It also allows evaluating the results, which are written to an XML log file, in a user-friendly way.

The workflow of batch-parallelization with the IKR SimLib is depicted in Figure 4. Multiple processes of the simulation program are started from the command line or called by SimTree. Each process reads the topology of the system under study from a parameter file (.par-file) and creates the model. The results of each batch are written to a separate batch-export-file and collected and summarized in a final results file after all processes have finished.

Each component of the model may define one or several statistics about its values that shall be observed during the simulation. For example, a message queue reports statistics on waiting time and queue length by default. Also, meters between ports come with predefined statistics. These statistics are already specified in the source code of the simulation program.

The IKR SimLib provides a rich set of statistics for different purposes. For example, there are simple counting statistics just reporting the sum of observed events. More complex statistics like the so-called `SampleStatistic` give the mean, confidence interval, deviation and covariance with the respective confidence intervals, as well as the maximum and minimum of the observed samples. These measures also include applying statistical testing like the Student's T-test which is implemented in the library. Another example is the `DistributionStatistic` reporting the distribution function of a system property through predefined quantiles. All statistics are already prepared to be used with batch-parallelism.

4.2 Running the Simulation

After defining the model, we start the first simulation phase, which comprises running the actual simulation on one or several processing cores. If we simulate with the batch-mean method, this means we have one simulation process with a transient phase and serially executed batches (see Figure 3). For the case of independent replications or the combination of both methods, each process starts with a different seed for the random number generator and an own transient phase. So there is no correlation between the replications.

In the case of a distributed simulation, the results of each batch are stored in an intermediate log-file after the end of that batch. The purpose of these files is to export the state of a statistic concerning a certain observed value and making it possible to import this state again for merging the final results from all the batches. Hereby, it is important to export the exact floating point representation of the data in binary format. This is because the exact state of the statistics needs to be reproducible in order to avoid rounding errors. Otherwise, when simulating with many batches, the rounding errors could sum up and distort the overall simulation outcome. Which values to export depends on the respective statistical property. E. g. for a mean value, just this value is reported at the end of each batch. In contrast, when considering the quantile of an observed measurement value, it is not sufficient to export the batch-quantile because this cannot be combined with the results from the other batches anymore. Instead, all samples have to be exported in order to be able to determine the resulting distribution function.

4.3 Post-Processing

When the simulation of all desired batches is complete, a processing of the batch results needs to be done. The simulation program is started again with the command line option to collect the batch-results and to combine them into the final results. Note that in the case of the serial execution of batches on a single processing core, no intermediate files are written by default. Instead, the simulation framework holds the batch-results in memory and combines them automatically at the end of the simulation run. In both cases, IKR SimLib calculates the desired statistical properties of the observed values automatically and writes them to an XML-file.

5. CONCLUSIONS

The performance evaluation of current communication networks requires the combined modeling of several network layers and results in long simulation running times. The IKR SimLib addresses this problem by supporting the evaluation of abstract system models and by supporting batch-level parallel simulation. The latest implementation of the IKR SimLib is freely available in both C++ and JAVA. Beside the well-known batch-means and independent-replications methods, it supports any combination of both methods, which allows for adapting the number of used processing cores to the available computing infrastructure. Since the IKR SimLib does the statistical evaluation of the batches, the user does not need to care about this complex post-processing stop. Our framework provides an easy way to perform batch-level parallelism for simulations by avoid pitfalls in the implementation which allows the user to focus on modeling.

6. ACKNOWLEDGMENTS

The authors would like to thank Joachim Scharf and Matthias Kaschub for many fruitful discussions.

7. REFERENCES

[1] Network Simulation Cradle. http://research.wand.net.nz/software/nsc.php.

[2] A. Abdelrazek, M. Kaschub, C. Blankenhorn, and M. Necker. A Novel Architecture using NVIDIA CUDA to speed up Simulation of Multi-Path Fast Fading Channels. In *Proceedings of the 69th IEEE Vehicular Technology Conference (VTC 2009)*, April 2009.

[3] Advanced Micro Devices, Inc. AMD Core Math Library (ACML). Technical report, 2008.

[4] Advanced Micro Devices, Inc. AMD64 Architecture Programmer's Manual Volume 6: 128-Bit and 256-Bit XOP, FMA4 and CVT16 Instructions. Technical report, May 2009.

[5] C. Alexopoulos and D. Goldsman. To batch or not to batch? *ACM Trans. Model. Comput. Simul.*, 14(1):76–114, 2004.

[6] S. Bodamer, M. Lorang, and M. Barisch. Ikr tcp library 1.2 user guide. Technical report, University of Stuttgart, IKR, June 2004.

[7] S. El-Shobaky, A. El-Mahdy, and A. El-Nahas. Automatic vectorization using dynamic compilation and tree pattern matching technique in jikes rvm. In *ICOOOLPS '09: Proceedings of the 4th workshop on the Implementation, Compilation, Optimization of Object-Oriented Languages and Programming Systems*, pages 63–69, New York, NY, USA, 2009. ACM.

[8] Freescale Semiconductor, Inc. AltiVec Technology Programming Environments Manual. Technical report, April 2006.

[9] R. M. Fujimoto. *Parallel and Distribution Simulation Systems*. John Wiley & Sons, Inc., New York, NY, USA, 1999.

[10] IBM. Cell broadband engine technology. Technical report, 2007.

[11] H. Kocher and M. Lang. An object-oriented library for simulation of complex hierarchical systems. In *Proceedings of the Object-Oriented Simulation Conference (OOS '94)*, pages 145–152, 1994.

[12] A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill Inc., December 1990.

[13] A. Munshi. The OpenCL Specification. Technical report, Khronos OpenCL Working Group, 2008.

[14] M. C. Necker, C. M. Gauger, S. Kiesel, and U. Reiser. Ikremulib: A library for seamless integration of simulation and emulation. In *Proceedings of the 13th GI/ITG Conference on Measurement, Modeling, and Evaluation of Computer and Communication Systems (MMB 2006)*, 2006.

[15] M. C. Necker and U. Reiser. Ikr emulation library 1.0 user guide. Technical report, University of Stuttgart, IKR, December 2006.

[16] A. Osseiran, E. Hardouin, A. Gouraud, M. Boldi, I. Cosovic, K. Gosse, J. Luo, S. Redana, W. Mohr, J. Monserrat, T. Svensson, A. Tolli, A. Mihovska, and M. Werner. The road to imt-advanced communication systems: state-of-the-art and innovation areas addressed by the winner + project. *Communications Magazine, IEEE*, 47(6):38–47, June 2009.

[17] Y. A. Sekercioglu, A. Varga, and G. K. Egan. Parallel simulation made easy with omnet++. In *European Simulation Symposium*, volume 15, 2003.

[18] S. Thakkar and T. Huff. The internet streaming simd extensions. Technical report, Microprocessor Products Group, Intel Corp., 1999.