Time-Parallel Simulation with Approximative State Matching

Tobias Kiesling, Siegfried Pohl
Institut für technische Informatik
Universität der Bundeswehr München
85577 Neubiberg, Germany
{kiesling,pohl}@informatik.unibw-muenchen.de

Abstract

Time-Parallel Simulation offers the potential of massive parallelization of a simulation application, due to the amount of achievable parallelism not being restricted by the decomposability of the state space of a simulation model. Unfortunately, the potential speedup of a time-parallel simulation highly depends on the ability to match final and initial states of adjacent partitions. However, depending on the properties of the underlying simulation model, it might be feasible to accept a simulation iteration, even if the states of adjacent partitions do not match exactly. This leads to the concept of approximative state matching in time-parallel simulation, which is introduced in this paper. Experiments with a prototypical implementation of a simple simulation model show encouraging results in terms of simulation speedup and introduced error.

1 Introduction

In traditional parallel discrete-event simulation (cf. [2]) the set of state variables of a simulation model is decomposed into subsets. Each of these is assigned to a logical process that manages the corresponding substate. The drawbacks of this approach are the introduction of an overhead for the synchronization between processes and the limited amount of parallelism, which is achievable through this approach. This amount is restricted by the number of state variables and the decomposability of states in the model. Time-parallel simulation takes a different approach by partitioning the time axis of an intended simulation execution and performing the simulation of these partitions in parallel [1]. Afterwards, the results of all partitions are combined to create the overall simulation result. This has the potential for massive parallelism, as the number of processes is determined by the number of possible partitions, which is only restricted by the granularity of the time representation in the simulation implementation.

However, without further mechanisms, the final and initial states of adjacent partitions do not neccessarily coincide at the partition boundary, leading to incorrect state changes. A straightforward approach to handle this problem is to rerun the simulations of those partitions that started from an initial state inconsistent with the final state of the previous partition. Of course, the effective speedup of the parallel simulation quickly decreases with every additional simulation run of the same partition.

Several different methods for reducing the cost of achieving state consistency between partitions have been proposed. [7] introduces the notion of regeneration points which are states that keep reoccuring throughout a simulation execution. If these points are known a priori, a number of simulations can be executed in parallel, starting from a regeneration point and continuing until the regeneration point is reached again. Afterwards the traces of the parallel simulations are concatenated to get a correct trace of the simulation over the whole time period. [6] uses fix-up computations to correct the simulation runs of those partitions of the time axis that are known to have started from incorrect states. For the specific application of simulating cache accesses presented in [6], it is possible to do these computations without completely rerunning the simulation of the incorrect partition. [5] reduces the problem of simulation of certain kinds of queueing networks to recurrence relations that can be solved on massively parallel computers.

The approaches mentioned above guarantee correct results in the sense that the states of adjacent partitions always match at the partition boundary (resp., are corrected, if they do not match). The alternative approach proposed in this paper is to allow a deviation of states at the partition boundary according to predefined rules. An obvious drawback of this approach is the error introduced by the approximate state matching. It must be considered that this error might even invalidate simulation results.

[12] utilizes the concept of unmatched sets to weaken the state-match problem. In this paper, two system states partially match, if a subset of the state variables match. The *unmatched set* comprises the rest of the variables. The efficiency of a time-parallel simulation execution can be raised by artificially fixing the unmatched variables (of a simulation iteration), in order to remove them from the unmatched set. [12] further elaborates on two sophisticated algorithms for the simulation of G/G/1/K and G/D/1/K queues.

There are also approaches in traditional space-parallel simulation which accept an error introduced by the simulation algorithm in order to reduce the overhead for synchronization between processes ([8], [10], [11], [3]).

The challenge of the approach introduced here is to determine the rules for the allowed deviation of states for a given simulation model, such that the efficiency of the parallel simulation is maximized, while the introduced error stays below an acceptable level.

The rest of the paper is organized as follows. Section 2 presents the concept of approximative state matching in time-parallel simulation. A prototypical implementation is discussed in Section 3. Section 4 contains the results of experiments conducted for a simple simulation model. Finally, Section 5 concludes the work.

2 Time-Parallel Simulation with Approximative State Matching

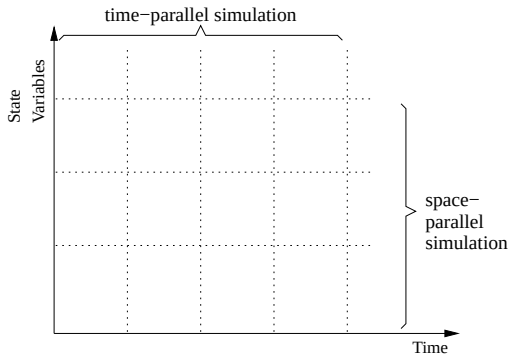
As already outlined in the last section, time-parallel simulation is a parallel simulation approach, which is not restricted by the decomposability of the simulation model. However, the state-match problem, i. e. the problem of inconsistent final and initial states of adjacent partitions, renders time-parallel simulation for many models almost useless. This is especially true for classes of models which exhibit complex states. Therefore, the concept of approximate state matching is proposed here as an efficient solution to the problem.

2.1 Time-Parallel Simulation

According to [4], a discrete event simulation essentially consists of the *state variables*, which establish the state space, the *event list*, which contains the events to happen in the future of the simulation, and the *simulation clock*.

To conduct a discrete event simulation in a parallel or distributed way, one possibility is to decompose the space-time diagram depicted in Figure 1 horizontally, in order to partition the set of state variables.

Figure 1 Space-Time Diagram of a Simulation



In time-parallel simulation, the space-time diagram in Figure 1 is partitioned vertically, i. e. every logical process manages all state variables, but only for a part of the simulated time.

Let $T := [0, t]$ be the intervall of simulation time. T is split into the subintervalls, or partitions

$$P_1 := [t_1, t_2], \ldots, P_m := [t_m, t_{m+1}],$$

with $t_1 = 0$, $t_2, \ldots, t_m \in (0, t)$, and $t_{m+1} = t$, such that every point in the simulated time is contained in some partition and the partitions overlap only at the partition boundaries, which are the times $t_2, \ldots, t_m$.

For performing the simulations of partitions in parallel, each partition has to start the simulation from some *initial state* $\hat{s}$. At the beginning of the simulation only the initial state of P_1 is known. For every other partition P_k, with $k \in \{2, \ldots m\}$, the initial state $\hat{s}_k$ has to be guessed. After the simulation of a partition P_k has been executed, the *final state* $\check{s}_k$ of P_k is known. As the simulations of different partitions are supposed to be executed in parallel, there is no guarantee that the final and initial states of adjacent partitions match, i. e. it is possible that $\check{s}_{k-1} \neq \hat{s}_k$ for partition P_k. This property of time-parallel simulation is known as the *state-match problem*.

Every time-parallel simulation algorithm must assure that the state-match problem is solved in order to perform a correct simulation of the whole simulation interval. A straightforward approach is to rerun the simulation of a partition, whose initial state deviates from the final state of the previous partition. However, without a successful strategy of guessing initial states, the repeated simulation of partitions leads to a degeneration of the efficiency.

2.2 Approximative State Matching

In many cases, where final and initial states of adjacent partitions deviate by a marginal amount, it might be feasible to accept the simulation of a partition even if this produces incorrect simulation results.

Allowed Deviation The *allowed deviation* $\mathcal{AD}(s)$ of the state s is a subset of the state space $\mathcal{S}$, such that changing the state of the simulation from s to $s' \in \mathcal{AD}(s)$ is accepted, even if it does not correspond to a valid state change defined in the model.

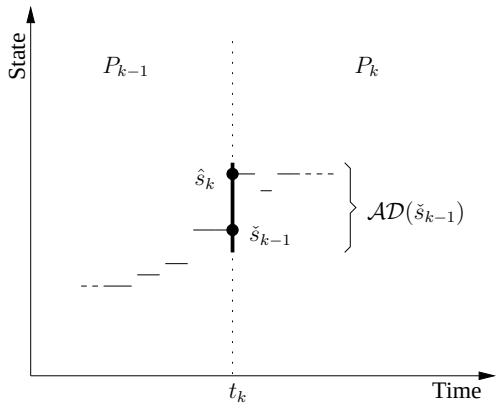
Reasonable implementations of $\mathcal{AD}(s)$ can only be found for a particular model, e. g. in a simulation of a G/G/1 queue, abrupt changes in the number of jobs waiting to be served can be accepted, as long as they do not influence the results of the simulation significantly. For a given model, $\mathcal{AD}(s)$ can be determined analytically or empirically, or it can already be defined by the modeler.

Approximative State Matching The function $\mathcal{AD}$ is used to implement the concept of approximative state matching. Instead of performing an exact match of final and initial states at the partition boundaries of a time-parallel simulation, a simulation run of a partition P_n is accepted if the initial state of partition P_k is in the allowed deviation of the final state of partition P_{k-1}, i. e. $\hat{s}_k \in \mathcal{AD}(\check{s}_{k-1})$.

In Figure 2 the situation is explained in detail: a simulation run of partition P_k started with an initial state $\hat{s}_k$. When the simulation run of the previous partition P_{k-1} completes, it is found that $\hat{s}_k$ is an incorrect state, since it does not exactly match the final state $\check{s}_{k-1}$ of the simulation run of P_{k-1}. Nevertheless, the simulation run of P_k is accepted, due to $\hat{s}_k \in \mathcal{AD}(\check{s}_{k-1})$.

A simple algorithm for a time parallel simulation with approximative state matching is given in Algorithm 1. As the algorithm describes a parallel simulation approach, a notation inspired by the Occam programming language ([9]) is used to denote parallel execution of statements. The designator

Figure 2 Approximative State Matching



for all <list> **do in parallel** <statement> **end for**

indicates that the enclosed statement is executed in parallel for every element in <list>.

Algorithm 1 Iterative Time-Parallel Simulation

Input: $P_1, \ldots, P_m, \hat{s}_1, \ldots \hat{s}_m$
Output: $\check{s}_m$
repeat
  for all $k \in \{1, \ldots, m\}$ **do in parallel**
    simulate partition P_k with initial state $\hat{s}_k$
  end for
  for all $k \in \{2, \ldots, m\}$, where $\hat{s}_k \notin \mathcal{AD}(\check{s}_{k-1})$ **do**
    $\hat{s}_k := \check{s}_{k-1}$
  end for
until $\forall k \in \{2, \ldots, m\} : \hat{s}_k \in \mathcal{AD}(\check{s}_{k-1})$

In the algorithm of Algorithm 1, the parallel simulation of the partitions is performed in iterations until the initial state of every partition is in the allowed deviation of the final state of the previous partition. The algorithm shown here is a very simple approach that reruns the simulations of all partitions even if there is only one partition with an inacceptable deviation of states. However, if there is one logical process for every partition and partitions are statically assigned to processes, the efficiency of the simulation is not decreased by this simplification[1]. Note that, besides the amount of repeated simulations of the same partition, the overhead of this parallel simulation algorithm is determined by the search for partitions where $\hat{s}_k \notin \mathcal{AD}(\check{s}_{k-1})$, which can implemented in an efficient way.

[1] At least in the case where the runtimes of the parallel simulations are roughly the same.

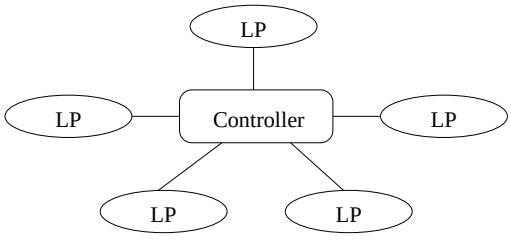
The algorithm directly shows the importance of the allowed deviation for the efficiency of the algorithm. The average number of repetitions of the simulations can be minimized by lowering the probability of $\hat{s}_k \notin \mathcal{AD}(\check{s}_{k-1})$. At worst, $\mathcal{AD}(\check{s}_{k-1}) = \check{s}_{k-1}$, in which case the algorithm is identical to the simple time-parallel simulation without approximate state matching. At best, with $\mathcal{S}$ representing the state space, $\mathcal{AD}(\check{s}_{k-1}) = \mathcal{S}$, is the trivial case where only one parallel run of the simulation is needed.

However, the increase in efficiency comes at the cost of an error that is introduced with the approximative state matching. Depending on the properties of the simulations, an incorrect initial state of a partition P_k might propagate through the partitions P_{k+1} to P_m and influence the results of the whole simulation. If the simulated model has recurring states that keep occuring throughout a simulation execution, chances are high that an error does not propagate too far, influencing simulation results only marginally, or not at all. Therefore, the viability of the approach of approximate state matching depends on the specificities of the underlying model.

3 Implementation

The approach outlined in the last section was implemented in order to gain a first impression of its viability. Basically, the developed prototype is a straightforward implementation of time-parallel simulation with the difference in the acceptance of state deviations at partition boundaries. Here, an overview of the implementation is presented and some interesting implementation issues are discussed. The results of experiments with the developed prototype for a simple simulation model are presented in the next section.

Figure 3 System structure



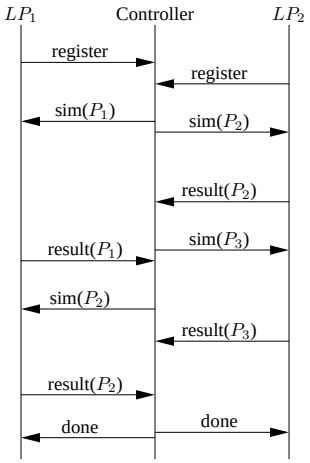
The basic structure of the simulation system is shown in Figure 3. The logical processes (LPs) are responsible for performing the simulations of particular partitions of the time axis. They are mainly sequential simulations that can be controlled over a network interface. The controller manages the whole simulation execution by assigning simulation tasks to the logical processes.

The communication between controller and logical processes is simple:

- the processes register with the controller,
- the controller assigns simulation tasks to the processes,
- the processes report simulation results to the controller, and
- the controller sends termination messages to the processes on completion of the simulation

For these tasks, a simple communication protocol has been defined and implemented on top of TCP/IP for usage in a distributed simulation.

Figure 4 Sample run of the system



Before starting a simulation, the controller initializes a given number of partitions and collects the logical processes that wish to participate in the simulation execution. A simulation run consists of the repeating assignment of partitions to simulation processes. The simulation ends if all partitions have been simulated with an acceptable deviation between their initial state and the final state of the previous partition.

Figure 4 depicts a sample run of the simulation system. The time axis of the simulation is separated into three partitions P_1, P_2, P_3, and two logical processes LP_1 and LP_2 are present. The sample shows the case where the first simulation run of P_2 is rejected after the results of P_1 are known. Therefore, P_2 is resimulated

by LP_1. The decision for rejection of a simulation run for a specific partition depends on the allowed deviation defined for a particular model, i. e. it is dependent on the specificities of the implemented model.

The example shows that the approach chosen here is fully dynamic, i. e. the partitions are assigned to the processes during the simulation execution, as opposed to a static assignment before the simulation start. This incurs an overhead for process management, but has the advantage that process idle time can be minimized as long as there are partitions to simulate. Therefore, it is important to be able to provide unsimulated partitions to idle processes most of the time. As also discussed in [13], the usage of a small number of partitions (e. g. equal to the number of processors) leads to an unbalanced processor load and consequently to a reduced efficiency of the overall simulation.

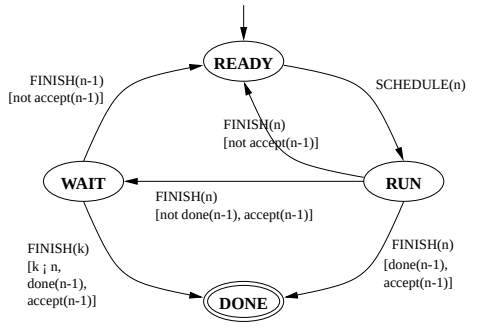
Figure 4 also depicts the situation where the result of a simulation run for P_2 is known, but it cannot yet be decided, if this simulation run can be accepted, as the result of the simulation of P_1 is not known yet. Partition P_2 has to be frozen, until the results of P_1 are known. If there are still partitions left to be simulated (P_3 in the example), the logical process that just finished execution does not have to remain idle, but can execute a remaining partition.

In order to support the dynamic approach presented in the example of Figure 4, the states of partitions have to be tracked during the simulation execution[2]. There are five possible states of a partition: `RUN` indicates that the simulation for the partition is currently run by a process, `READY` denotes a partition ready to be simulated, a partition that is `DONE` does not have to be simulated any more, and `WAIT` signifies a partition that has been simulated, but that is still waiting for some event in the previous partition. Every partition is in one of the four states. A state diagram defining the behaviour of the partitions is shown in Figure 5.

Upon initialization, every partition is in state `READY`, which is only left when the partition is scheduled for execution by a logical process, which changes its state to `RUN`. When a logical process finishes execution of a partition P_n, the event `FINISH(n)` is sent to the state machines of all partitions, possibly resulting in further state changes. A partition P_n that receives a `FINISH(n)` event in state `RUN`, checks whether its starting state is in the allowed deviation of the final state of the previous partition (this is indicated in the state diagram by the function call `accept(n-1)`). If the two states do not

[2]Note that the state of a partition does not correspond with the state of a simulation run of a partition.

Figure 5 State Diagram for Partition P_n



match approximately, the partition returns to the state `READY`. If the difference in simulation states is acceptable, the partition either enters the `DONE` state if the previous partition is also `DONE` (this is checked by the function call `done(n)`), or enters the `WAIT` state to wait for the previous partition. A waiting partition P_n reenters the `READY` state if the simulation of P_{n-1} is finished and the difference in simulation states is not acceptable. A partition's state changes from `WAIT` to `DONE` if any earlier partition is finished, the difference in simulation states is acceptable, and the previous partition (P_{n-1}) is `DONE`.

The state diagram in Figure 5 shows that partitions P_k with $k > n$ are dependent on earlier partitions but not vice versa. Therefore, the scheduling of partitions to processes, performed by the controller, uses the simple approach of always selecting that partition in state `READY` which has the lowest index. This gurantees that partitions in the `WAIT` state can be resumed as early as possible.

4 Experimental Results

The prototype described in the last section was adapted for the simulation of a M/M/1 queue. The state $s_t = (j_t, A_T, D_t)$ of the simulation at a given time t consists of the number of present jobs j_t and the time of the next arrival A_t, as well as the next departure D_t if a job is present. In order to perform approximate matching of states at partition boundaries, the allowed deviation is defined as:

$$\mathcal{AD}(j_t, A_T, D_t) := \{(j'_t, A'_t, D'_t) \in \mathcal{S} : |j_t - j'_t| \leq \delta\}.$$

Note, that $\mathcal{AD}(s)$ is a function of the number of jobs present in the system, i. e. any deviations of the next arrival and departure times are accepted. This is reasonable, because as long as the ratio of number of partitions

Figure 6 4 processors and 100 partitions

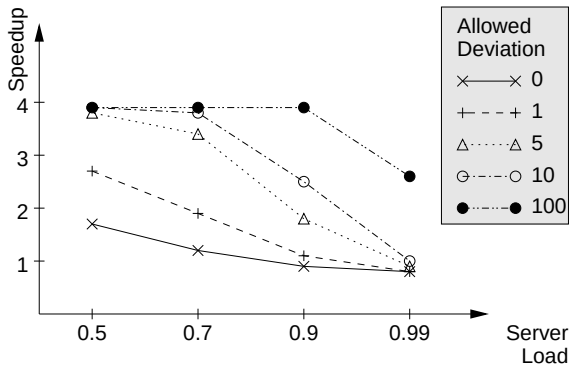


Figure 8 4 processors and 0.9 load

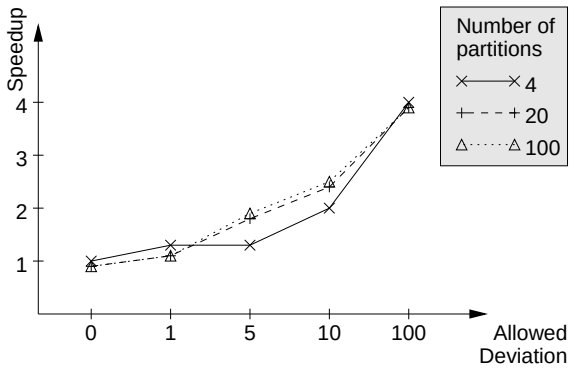


Figure 7 8 processors and 100 partitions

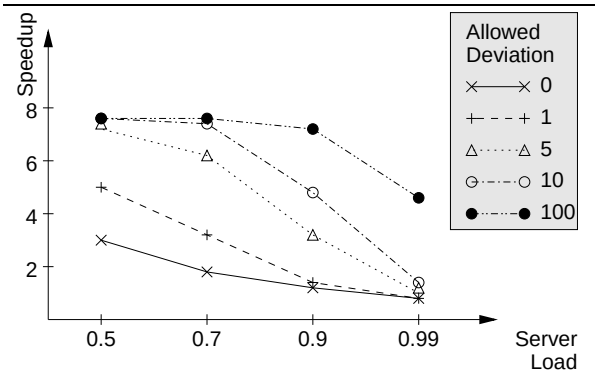
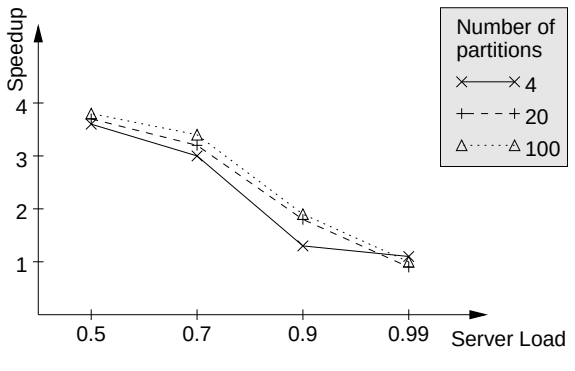


Figure 9 4 processors and allowed deviation 5



to mean number of arrivals in a partition is sufficiently low, the statistical error introduced by the allowed deviation is determined by the difference in job numbers. The allowed deviation still depends on the parameter δ, which can be varied for a higher or lower approximation in the state matching.

Experiments were conducted as a distributed simulation on a network of workstations with 4 and 8 logical processes, each running on a different processing node. The runtime of the simulation was measured and compared to the runtime of a sequential simulator in order to measure the speedup of the parallel simulation. The influence of the allowed deviation on the speedup of the simulation was measured by varying the parameter δ. Further, the server load is known to have a great influence on the achievable speedup. This is due to the fact that it is rather easy to guess the correct number of jobs present in the system at a given time if the server load is low[3]. As the server load increases, the variance of present jobs increases, leading to a lower probability that a guessed state is correct. Finally, also the influence of the number of partitions on the speedup was measured for a different number of partitions.

In order to be able to measure the introduced error, the maximum number of present jobs and the average waiting time were recorded during the simulation execution. However, no significant deviation of the parallel simulation results from the sequential simulation results could be observed, even for higher values of δ.

Figures 6-9 show the averages of the speedup that could be achieved for various parameter combinations. The results presented here are averages of the results of ten repetitions for every parameter combination. Figure 6 shows the speedup for 4 logical processes and 100 partitions. The speedup was measured for different server loads and allowed deviations. As suspected, the efficiency of the simulation decreases with a higher server load. However, even for high loads, an appropriate allowed deviation partly compensates the loss of speedup. For lower server loads, it is interesting to note the significant increase in speedup even with the intro-

[3]In particular, the number of jobs was always guessed to be 0, which is more probable to be correct if the server load is low.

duction of the allowed deviation with a marginal value of $\delta = 1$. Note also that with a reasonably high allowed deviation, it is possible to approach the maximal speedup of 4, which indicates the low overhead of the simulation algorithm.

Figure 7 shows the results for 8 logical processes and 100 partitions. These results are very similar to those for 4 processes. However, it can be noticed that the overall speedup slightly drops for 8 processors, which is due to higher communication overhead in the simulation system.

Figures 8 and 9 show the influence of the number of partitions on the speedup. In Figure 8, partition sizes are compared for various allowed deviations and a server load of 0.9. For small allowed deviations, almost all of the partitions have to be simulated more than once, wherefore the processes are busy regardless of the number of partitions. The same is true for large allowed deviations, where most of the partitions are simulated exactly once. However, for modest allowed deviations, with a small number of partitions, the idle time of processors reaches a significant level. Therefore, a higher number of partitions leads to a significant increase in speedup. Figure 9 shows similar results for different server loads and an allowed deviation of 5.

For the simple example of an M/M/1 queue, even with a high server load, the parameters that control the parallel simulation execution can be tuned to obtain a maximal speedup with a minimal amount of introduced error.

5 Conclusion

This work introduces the approach of approximate state matching in time-parallel simulation. An important characteristic of time-parallel simulation is the state match problem: if the simulation of a partition starts from an initial state that is inconsistent with the final state of the previous partition, the whole simulation is incorrect. Instead of an exact solution to the problem, approximate state matching is a heuristic approach that accepts deviations in final and initial states of adjacent partitions at the cost of a deviation of simulation results from the correct values.

Time-parallel simulation with approximate state matching is a general parallel simulation approach that can be applied to any simulation model. The challenge in applying approximate state matching is to find an allowed deviation of states that maximizes the speedup of the simulation, while keeping the introduced error below an acceptable level. More work is required to gain understanding of the allowed deviation of complex states in contrast to the rather simple state of a M/M/1 queue.

The experimental results in Section 4 show that approximate state matching can reasonably be applied to the simulation of a M/M/1 queue. It would be desirable to research the application of approximate state matching to other, more realistic models.

With the help of approximate state matching in time-parallel simulation, the massive parallelization of appropriate simulation models is viable. Together with the approach of dynamic scheduling used by the prototypical implementation presented in this paper, it should be possible to construct a simulation system that can be executed efficiently on unreliable and/or high-latency networks with a very high number of processing nodes.

References

[1] K. Chandy and R. Sherman. Space-Time and Simulation. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 53–57, 1989.

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

[3] R. M. Fujimoto. Exploiting temporal uncertainty in parallel and distributed simulations. In *Proceedings of the 13th Workshop on Parallel and Distributed Simulation*, pages 46–53, 1999.

[4] Richard M. Fujimoto. *Parallel and Distributed Simulation Systems*. John Wiley & Sons, Inc., 2000.

[5] A. G. Greenberg, B. D. Lubachevsky, and I. Mitrani. Algorithms for Unboundedly Parallel Simulations. *ACM Transactions on Computer Systems*, 9(3):201–221, 1991.

[6] P. Heidelberger and H. S. Stone. Parallel Trace-Driven Cache Simulation by Time Partitioning. In *Proceedings of the 1990 Winter Simulation Conference*, pages 734–737, 1990.

[7] Y. Lin and E. Lazowska. A Time-Division Algorithm for Parallel Simulation. *ACM Transactions on Modeling and Computer Simulation*, 1(1):73–83, 1991.

[8] P. Martini, M. R¨umekasten, and J. T¨olle. Tolerant Synchronization for Distributed Simulations of Interconnected Computer Networks. In *Proceedings*

of the 11th Workshop on Parallel and Distributed Simulation, pages 138–141, 1997.

[9] Dick Pountain and David May. *A Tutorial Introduction to Occam Programming*. BSP Professional Books, 1987.

[10] D. M. Rao, N. V. Thondugulam, R. Radhakrishnan, and P. A. Wilsey. Unsynchronized Parallel Discrete Event Simulation. In *Proceedings of the 1998 Winter Simulation Conference*, pages 1563–1570, 1998.

[11] N. V. Thondugulam, D. M. Rao, R. Radhakrishnan, and P. A. Wilsey. Relaxing Causal Constraints in PDES. In *Proceedings of the 13th International Parallel Processing Symposium (IPPS/SPDP '99)*, pages 696–700, 1999.

[12] J. J. Wang and M. Abrams. Approximate Time-Parallel Simulation of Queueing Systems with Losses. In *Proceedings of the 1992 Winter Simulation Conference*, pages 700–708, 1992.

[13] H. Wu, R. M. Fujimoto, and M. Ammar. Time-Parallel Trace-Driven Simulation of CSMA/CD. In *Proceedings of the 17th Workshop on Parallel and Distributed Simulation*, pages 105–114, 2003.