

AUTOMATED DIAGNOSIS OF DISTRIBUTED SIMULATION PERFORMANCE

Alois Ferscha*

Abstract *Distributed simulation (DS) aims at the acceleration of discrete event simulation experiments by decomposing the simulation model into spatially separated event sets, which are then executed in a distributed environment via synchronized logical processes (LPs). Synchronization protocols like Time Warp and Chandy/Misra/Bryant are employed to preserve global causality constraints and temporal consistency among events that occur locally within LPs. The performance of such LP based DS is influenced by a variety of factors such as the event structure underlying in the simulation model, the partitioning into submodels, the performance characteristics of the execution platform, the implementation of the simulation engine and optimizations related to the protocols. The mutual performance effects of parameters exhibit a prohibitively complex degree of interweaving, so that analytical performance models are of relative importance only. Nevertheless is a DS performance diagnosis of utmost practical interest when it comes to decide on the suitability of a certain DS protocol for a specific simulation model before substantial efforts are invested in developing sophisticated DS codes.*

DS has been known as a particularly challenging performance analysis domain, and still is. While working with Günter Haring at the University of Vienna, we addressed the issue of performance prediction for DS. Since prediction based on analytical performance models appeared doubtful wrt. adequacy and accuracy, a methodology was introduced involving the simulated execution of skeletal implementations of DS codes. The prediction methods were implemented and integrated into N-MAP, our distributed software prototyping and performance diagnosis environment. N-MAP, although reliant on models in predictive performance analysis, avoids engaging explicit models. Early code skeletons, together with their resource requirement specifications serve as prototypes for distributed programs, and simultaneously as their implicit performance models. Implicit, because a simulated execution of the prototype (real program code) yields a prediction of performance indices, and no explicit (static) model representation is involved. As such, N-MAP is not liable to modelling lapses. Here, N-MAP is used to help with DS performance diagnosis problems like (i) given a simulation model and a distributed execution platform, then which DS strategy promises best performance, or (ii) given a simulation model and a DS strategy, which execution platform is most appropriate, or (iii) which class of simulation models can gain from a given execution platform using a certain DS protocol. This paper summarizes some of the findings on how to predict the performance of a DS system with intricate dynamics, and which is influenced by a prohibitive manifold of performance factors.

*Institut für Praktische Informatik, Johannes Kepler Universität Linz, Altenbergerstr. 69, A-4040 Linz, AUSTRIA, Email: ferscha@soft.uni-linz.ac.at

“As the mountain is not coming to the Prophet,
evidently the Prophet must go to the mountain.” [25]

1 Why is DS performance evaluation hard?

In order to accelerate the execution of self-contained simulation models, distributed and parallel discrete event simulation techniques [12] have proposed the spatial decomposition of that model and the concurrent simulation of the submodels by so called *logical processes* (LPs). A DS relies on the presence of *independent events*, i.e. events occurring at different simulated time instants that do not affect one another. The critical issue for a DS protocol is hence the chance of *causality errors*, i.e. the possibility of dependent events being executed in different LPs. Correctness of the execution can be assured if the (total) event ordering as produced by a sequential (or non-distributed) simulation is consistent with the (partial) event ordering as generated by the distributed execution in the various LPs. Violations to this correctness property, i.e. the occurrence of *causality errors* is not possible if and only if every LP adheres to processing events in nondecreasing timestamp order only. Such LPs are said to adhere to the *local causality constraint*, *lcc*. Although sufficient, it is not always necessary to obey *lcc*, because two events occurring within one and the same LP may be concurrent (independent of each other) and could thus be processed in any order. The two main categories of DS protocols adhere to the *lcc* in different ways:

- *conservative* methods strictly avoid *lcc* violations, even if there is some nonzero probability that an event ordering mismatch will *not* occur.
- *optimistic* methods hazardously use the chance of processing events even if there *is* nonzero probability for an event ordering mismatch.

By now, more than 20 years of research have been devoted to studying various issues related to this goal, establishing techniques along two lines: conservative (Chandy/Misra/Bryant, CMB) and optimistic (Time Warp, TW) distributed simulation protocols. The main focus of this research has been to establish distributed simulators that operate in a *causally correct* way, i.e. simulators that despite the asynchronous execution of their components would generate consistent simulation sample paths. Causal correctness is defined in terms of timestamps of discrete events, and a simulation sample path is considered consistent if each event t that *happens before* some other event s in the real system, is generated into the simulation's sample path before s [19].

Yet another distinction among DS protocols can be made according to the way how they progress the execution: *synchronously* or *asynchronously* [27]. Assuming that P be the number of LPs executing concurrently on independent processors and each time step in every LP_i takes $T_{step,i} \sim exp(\lambda)$ ($E[T_{step,i}] = \frac{1}{\lambda}$) time, then the potential performance improvement of an *asynchronous* over a *synchronous* simulation is at most $O(\log P)$, because the expected simulation time $E[T^{sync}]$ for a k time step synchronous simulation is:

$$E[T^{sync}] = k E[\max_{i=1..P} (T_{step,i})] = k \frac{1}{\lambda} \sum_{i=1}^P \frac{1}{i} \leq \frac{k}{\lambda} \log(P),$$

and the expected on time $E[T^{async}]$ for asynchronous simulation is:

$$E[T^{async}] = E[\max_{i=1..P}(k T_{step,i})] > \frac{k}{\lambda}$$

Therefore we have

$$\lim_{k \rightarrow \infty, P \rightarrow \infty} \frac{E[T^{sync}]}{E[T^{async}]} \approx \log(P)$$

i.e., with increasing simulation size k , an asynchronous simulation can complete (at most) $\log(P)$ times as fast as the synchronous simulation (max speedup: $\frac{P}{\log(P)}$). Assuming, however, the execution time be uniformly distributed over $[l, u]$ then $\lim_{k \rightarrow \infty, P \rightarrow \infty} \frac{E[T^{sync}]}{E[T^{async}]} \approx 2$, i.e. the speedup an asynchronous DS can gain over a synchronous DS is 2 at best:

$$T^{sync} \leq k u, \quad E[T^{async}] \geq k \frac{(l + u)}{2} \quad \Rightarrow \quad \frac{E[T^{sync}]}{E[T^{async}]} = \frac{2}{(k u)(k (l + u))} \leq 2$$

We immediately find, that an analysis of the above kind is more than doubtful for the purpose of performance prediction: weak assumptions are being made on the system behavior while nothing is said about factors like the simulation strategy, the simulation model, the execution platform, the implementation details like memory management, data structures, etc. The analysis is however a typical example of performance evaluation in the early days of DS, i.e. applying fast and routine techniques.

Up until today, presumably some 2000 research papers have appeared (since the pioneering works by Chandy and Misra [5] and Jefferson [18]) which have significantly contributed in the scientific sense, but nevertheless failed to bring the field also to an industrial and/or commercial success. Due to the focused nature and involved DS publications, the field has often been criticized as an academic playground, and the community as living in an ivory tower. This pessimism has also diffused into the community and has led to an existential discussion on the chances of survival as reflected in a collection of papers following Fujimoto's position statement in [14]. Despite successful attempts to escape from this image expressing the hope that simulation in industry (the "mountain") would at least meet "halfway" [25] on their way to integrating DS techniques into commercial/industrial simulators in a transparent way, this discussion has not stopped [21]. One of the reasons for this was that the co-influence of the DS strategy, the execution platform and the simulation model on performance was not understood well enough. At the level of DS implementations, the preference among TW and CMB, given a simulation model and platform attributes, is neither conclusive nor can protocol optimizations establish a general rule of superiority. Moreover, for both these simulation strategies, neither fast processors (on their own), nor can fast communication (on its own) guarantee performance gains.

With this work we aim to bring the performance interaction among factors [20] like the simulation strategy, the simulation model and the execution platform (see Figure 1) to a transparent understanding. Successful DS applications are reliant on that kind of understanding, and the development of distributed simulators must be driven by the respective performance engineering activities.

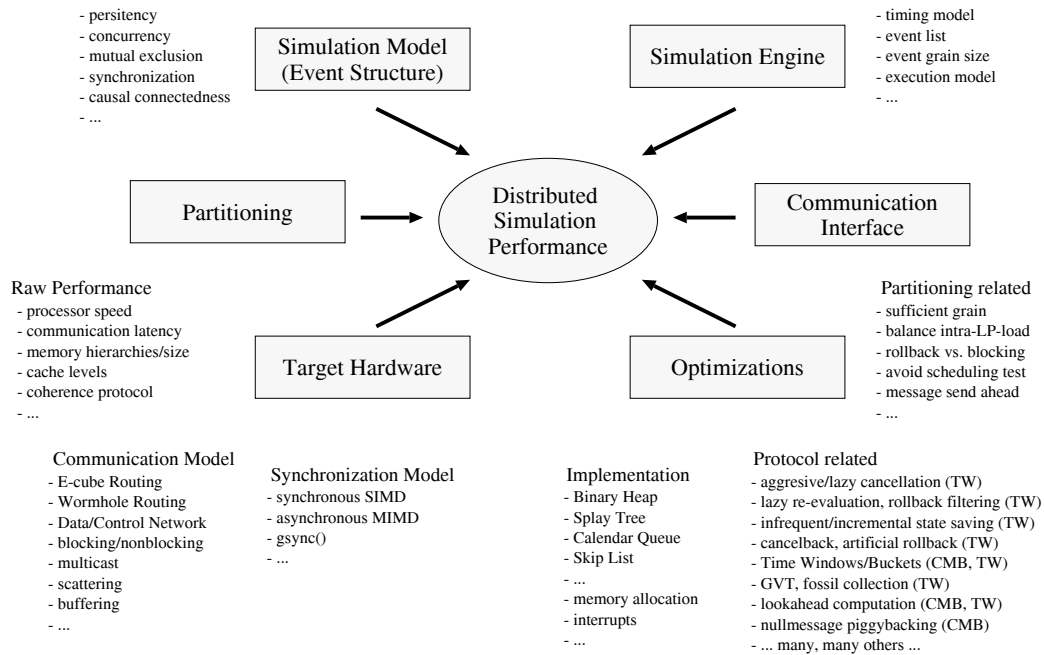


Figure 1: Factors that co-influence DS performance

2 CMB vs. TW: A never ending story

What certainly appears as the most crucial question among the above issues for practitioners is the choice of the simulation strategy, i.e. TW or CMB, for a particular simulation problem.

CMB protocols execute in each LP events that occur in the respective submodel in nondecreasing order of their occurrence timestamp, while strictly preventing the possibility of any event causality violation across LPs. The primary motivation for TW protocols is that events which occur in different LPs – irrespective of their occurrence time stamp – might not affect one another, thus giving rise to their distributed, out-of-timestamp-order execution. Both CMB and TW protocols have convincing advantages, but also suffer from shortcomings. While CMB protocols rely on the detection of when it is (causally) *safe* to process an event, TW is not reliant on any information coming from the simulation model (e.g. lookahead). Instead, while letting causality errors occur, TW employs a *rollback* mechanism to recover from causality violations immediately upon or after their detection. The rollback procedure in turn relies on the reconstructability of past states which can be guaranteed by a systematic state saving policy and corresponding state reconstruction procedures where space complexity is interchangeable with computational complexity (at least to some extent) [23]. The related state saving overheads are not present in CMB protocols, but the strict adherence to time-stamp-order event processing makes these protocols prone to deadlock due to cyclic waiting for safe-to-process concessions. A deadlock management mechanism is necessary to either avoid deadlocks, or detect and recover from deadlocks. In distributed system implementations of CMB protocols deadlock management is usually based on message exchanges, yielding severe communication overheads. Deadlock management is relieved from TW in a natural way, since deadlocks due to cyclic waiting conditions for messages able to make 'unsafe' events safe to process by exploiting information from their timestamps can never occur. Nevertheless, TW can suffer from communication overheads to a threatening extent: In situations where event occurrences are highly dispersed in space and time, rollback invo-

cations can recursively involve long cascades of LPs which will eventually terminate. An excessive amount of local and remote state restoration computations is the consequence of the annihilation of effects that have been diffused widely in space and too far ahead in simulated time, consuming considerable amounts of computational, memory and communication resources while not contributing to the simulation as such. This pathological behavior is basically due to the “unlimited” optimism assumption underlying TW, and has often been referred to as *rollback thrashing*. Overall, no general rule of superiority of the two strategies can be formulated [7], not even can performance improvement with protocol optimizations like lazy cancellation (TW), adaptive state saving (TW), optimism control mechanisms (TW), receiver initiated lookahead propagation (CMB), deadlock detection and recovery (CMB) etc. be assured.

A huge variety of TW protocols have been proposed in the literature, with a primary focus on correctness issues in their presentation. Performance aspects of the protocols, due to an overwhelming interweaving of influencing factors, have mostly been studied on the basis of abstractions (*models*) concerning the target execution platform, the TW implementation and the simulation task subject to a TW execution. In most of the literature, performance analysis is used to motivate the optimization of the TW protocol with respect to one or more of these concerns, or to assert the performance gain obtained with optimizations. Specifically the question on the relative qualities of TW protocols (in protocol optimization studies) has often been raised, but again, general rules of superiority cannot be formulated since performance cannot be sufficiently characterized by models.

In the first category of TW performance analysis, all of the performance factors are implicitly or explicitly abstracted into *models*, mostly stochastic models. As an example, in the analysis by [17], event processing time is assumed to be exponentially distributed, time stamps of messages are Poisson distributed in virtual time, the destination logical process (LP) for each message is randomly chosen and equally likely for every LP, etc. Performance investigations based on analytical models often fail to achieve a satisfactory accuracy due to (i) unrealistic and inadequate assumptions in the models (e.g. the memoryless property), (ii) the complexity of detailed models, (iii) simplifying assumptions that make the evaluation of those models tractable (e.g. symmetry, homogeneity, $M \rightarrow M$ property), and (iv) the possibility of modeling errors. Specifically the latter is the reason why only relative trust can be placed in the results obtained, although exceptions do exist [17, 1]). As an improvement, performance investigations have been conducted upon full TW implementations, but operating under synthetic workloads, thus defining a second category of TW performance analysis [16] [4].

In this second category, the analysis is based on a *real system* as far as the hardware and TW software is concerned, but the simulation task to be executed in this environment is still an abstraction (*model*) of the real simulation task – leaving just a single source of potential performance analysis bias due to modeling errors. The most prominent synthetic workload model for TW is PHOLD [13], which has been widely used in the community to demonstrate the *performance sensitivity* of TW implementations to the (event) structure of the potential simulation task. Another approach where a mixture of a real simulation task and a model of the execution environment is studied is trace driven simulation. Here, the behavior of a hypothetical environment is studied under the real load.

Even full TW implementations with real workload sets – as the third category – often prohibit performance comparisons if different implementation strategies were followed or different target platforms were selected for the execution. So in this third category, even after eliminating all possibilities of bias due to modeling errors, performance analysis potentially suffers from an “incomparability” dilemma. For the DS of Petri Nets a performance comparable implementation design [15] has been developed,

which isolates hardware from implementation and TW protocol related performance influences. A maximum of source code reuse gained from a conditional compilation implementation technique allows for a further reduction of incomparability to an acceptable level.

3 DS will survive: With performance prediction!

In an existential discussion within the parallel and DS community (“*Will Parallel Simulation Research Survive?*”) [14], the availability of *performance prediction methods* for DS protocols has been pointed out as being critical for the future success and general acceptance of DS methods in practice [22]:

... It is important to predict the performance of DS before a simulationist invests a substantial effort in developing parallel codes. For example, if the inherent parallelism for the simulation application is not sufficiently large, it is not worthwhile to take the DS approach.

None of the three above mentioned performance analysis categories appears adequate for these demands. Models, although able to give fast predictions, are too vague to be meaningfully applied. Analysis that considers full implementations, although accurate, just brings to light the design flaws committed earlier, which are irrevocable or very hard to repair at the time when major part of the development work is already done.

Conforming to these arguments, we think that for simulation practitioners at least a rough prediction of the performance potentials of a distributed simulator needs to be at hand in order to justify the respective coding efforts. For the purpose of prediction, “classical” (analytical) performance analysis techniques using formalisms like stochastic (e.g. Markovian) processes, queueing network models, Petri nets, etc. serve to abstract the simulation model, the DS protocol and target platform characteristics into *models* (examples are [17, 1]). Predictions based on analytical models, however, often fail.

Consequently, to overcome these shortcomings, and to potentially allow for an arbitrary forecast accuracy, performance prediction is based on simulation (rather than analysis) in our approach. We have chosen a *performance prototyping* method, where an implementation skeleton of an LP simulation protocol serves directly as the performance model. A performance prediction testbed, N-MAP [9], has been developed to support performance engineering endeavors from the early design phase of DS protocols in order to avoid late and costly re-engineering. Implementing DS protocols *incrementally* in N-MAP, i.e. starting from a code skeleton and providing more and more detailed program code towards the full implementation, allows for very early performance based design decisions, systematic investigations of performance sensitivities using an automated scenario manager, and a maximum of code reuse when trying different optimizations using an automated version manager.

4 Early performance prediction of DS protocols

The main goal of an early performance analysis of DS applications is to provide predicted, but sufficiently trustful data to solve decision problems before substantial manpower and financial investments are undertaken. The methodology we have is based on the identification of attributes characterizing the performance parameters on one hand, and on the identification of user oriented decision variables on the other hand. Conceptually, a simulation based performance prediction methodology is employed to allow for a systematic investigation of the effects of variations of strategy, platform and simulation model attributes upon predefined output (decision) variables. Methodologically, a set of relevant attributes for the respective factors is identified for which the analyst can provide a single or a whole range of possible parameter settings. N-MAP, the incremental code development and simulation tool component of the performance prediction testbed then in an automated way generates the full Cartesian product of the attribute combinations of interest, and conducts the related simulation experiments. The simulation output data is piped into a performance data mining component, where statistical reasoning (variance analysis, hypothesis testing, etc.) is conducted. A transformation system “translates” the statistical analysis results into “user oriented” decision values. Here (for space reasons) we shall present only the experiment design as far as the factor characterization of a 2^k factorial design is concerned.

To analyze DS performance we identify k factors x_1, x_2, \dots, x_k at 2 alternate levels each $x_i = [-1, +1]$ and then let N-MAP conduct 2^k simulations $E_{[x_1, x_2, \dots, x_k]}$ replicated r times with responses $y_{[x_1, x_2, \dots, x_k]_i}$ where $i = 1 \dots r$. The analysis yields 2^k effects, i.e. the k main effects q_1, q_2, \dots, q_k , the $\binom{k}{2}$ two-factor interactions $q_{1,2}, q_{1,3}, \dots, q_{1,k}, q_{2,3}, \dots, q_{k-1,k}$, the $\binom{k}{3}$ three-factor interactions $q_{1,2,3}, q_{1,2,4}, \dots, q_{1,k-1,k}, q_{2,3,4}, \dots, q_{k-2,k-1,k}$, and so on, such that the mean response for experiment $E_{[x_1, x_2, \dots, x_k]}$ is $\hat{y}_{[x_1, x_2, \dots, x_k]} = \sum_{i=1}^r y_{[x_1, x_2, \dots, x_k]_i} = q_1 \cdot x_1 + q_2 \cdot x_2 + \dots + q_k \cdot x_k + q_{1,2} \cdot x_1 \cdot x_2 + \dots$ and the allocation of variation yields confidence intervals for the effects q_1, q_2, \dots and predicted responses $\hat{y}_{[x_1, x_2, \dots, x_k]}$. We can then interpret effect impacts in terms of significance and magnitude.

4.1 Identification of DS performance factor attributes

The choice of attributes for the characterization of factors has a direct impact on the quality and value of the performance prediction analysis. A “rough” specification in terms of just a few attributes simplifies the experiments from a quantitative view, but yields performance reflections of a more general nature. A very detailed set of attributes, on the other hand, leads to excessive analysis efforts, but provides detailed performance insights. Depending on the purpose of the analysis, an “appropriate” set of attributes has to be found.

4.1.1 Protocol factors

The key to a successful performance diagnosis of LP simulators based on code skeletons is the choice of a tractably small, yet representative set of LP simulation protocols. Among the huge variety of protocols and respective incremental optimizations that appeared in the literature, only a few represent basic corner-stones in DS. We first identify both CMB and TW ($[TW, CMB]$) as the attributes (or

Factor Category	Issues		Factor	Level	
Protocol Factors	Time Warp	↔	CMB Protocol	PROTOCOL	[TW/CMB]
TW Factors	aggressive cancellation	↔	lazy cancellation	CANCELLATION	[-LC/+LC]
	full state saving	↔	incremental state saving	STATE SAVING	[-IS/+IS]
	no optimism control	↔	optimism control enabled	OPTIMISM	[-OC/+OC]
CMB Factors	receiver initiated nullmessages	↔	sender initiated nullmessages	NULLMESSAGES	[-SI/+SI]
	GVT reduction disabled	↔	GVT reduction enabled	GVT REDUCTION	[-GVT/+GVT]
Model Factors	low model parallelism	↔	high model parallelism	PARALLELISM	[-HP/+HP]
	uniform LVT progression	↔	non-uniform LVT progression	UNIFORMITY	[-NU/+NU]

Figure 2: Systematic Investigation of Factor Impacts: 2^k Factorial Design

levels) of the factor PROTOCOL, and then isolate representative variations of these main protocol categories together with their attributes.

4.1.2 CMB factors

Conservative protocols can be divided into three main approaches [12] (*i*) *deadlock avoidance* algorithms, (*ii*) *deadlock detection and recovery* algorithms, and (*iii*) *synchronous* algorithms. Deadlock avoidance algorithms such as CMB involve the use of null messages for synchronization purposes. An important variation of the deadlock avoidance approach is to send null messages on demand [3] rather than after each event. This strategy helps to reduce the number of null messages exchanged between LPs, but leads to a longer delay because a request message must now be transmitted for each null message. Deadlock detection and recovery algorithms make use of the fact that the deadlock can be broken by finding the event with the smallest timestamp amongst all LPs. This event is evidently safe to process. As with the GVT calculation of optimistic protocols, this involves a global reduction over all the timestamps of all unprocessed events. With synchronous algorithms (e.g. [2]), the simulation proceeds in cycles, where each cycle consists of a processing phase, followed by a global synchronization to determine the set of safe events to be processed in the next cycle. This is quite similar to the deadlock recovery algorithm described above, the main difference being the way that the global calculation is invoked.

It can therefore be seen that a fundamental factor in conservative algorithms is whether or not a global reduction is employed. A generalization of the CMB approach also allows it to be combined with null message sending at the initiation of the sender or the receiver, leading to four strategies which are representative of conservative protocols. Arguing for a factorial design as the statistical analysis method to be used, we therefore identify two *factors* (*i*) GVT reduction, symbolically referred to as GVT, and (*ii*) nullmessage sending initiation, symbolically referred to as SI. GVT and SI

are subject to an analysis at the *levels* +GVT (GVT reduction technique “on”) and –GVT (GVT reduction technique “off”), and +SI (sender initiated nullmessages) and –SI (receiver initiated nullmessages) respectively. As a shorthand notation, we write the four cases of interest as [–SI, –GVT], [–SI, +GVT], [+SI, –GVT] and [+SI, +GVT].

4.1.3 TW factors

Important factors of TW are the cancellation policy and the state saving mechanism, as demonstrated by the number of papers published describing implementations of these mechanisms (see [7] for a survey). While it is impossible to include all variations, a representative set of strategies should include both aggressive and lazy cancellation and full and incremental state saving. A primary concern in optimistic protocols such as TW is their stability, in terms of avoiding a proliferation of cascaded rollbacks or echoing ([24]). A factor that must therefore be considered in any analysis of TW protocols is the use of an optimism control mechanism. Although a number of mechanisms have been published, they differ mainly in how the constraints on optimism are expressed. Examples include windowing mechanisms (e.g. [26]) and memory management techniques ([6]).

A consideration of the important factors of an optimistic protocol thus leads to eight representative strategies. In terms of the factorial design therefore we identify the three factors: (*i*) cancellation policy LC (at the levels +LC (lazy cancellation) and –LC (aggressive cancellation)), (*ii*) the state saving policy IS (at the levels +IS (incremental state saving) and –IS (full state saving)), and (*iii*) the optimism control mechanism OC (at the levels +OC (optimism control enabled) and –OC (optimism control disabled)). The cases of interest are accordingly notated as [–LC, –IS, –OC], [–LC, –IS, +OC], [–LC, +IS, –OC], [–LC, +IS, +OC], [+LC, –IS, –OC], [+LC, –IS, +OC], [+LC, +IS, –OC] and [+LC, +IS, +OC].

4.1.4 Model factors

To be able to study the behavior of the various DS protocols, an abstraction of the workload (simulation model) to be executed by the LPs is demanded. In order to allow different partitioning strategies to be explored, it is assumed that the simulation model is expressed in terms of simulation objects or nodes (e.g. queues in a queueing network model simulation, traffic lights in a road traffic simulation etc.). Simulation objects exhibit certain interactions with other objects, i.e. the occurrence of particular events in one node induces the occurrence of other events in remote objects. A very useful abstraction of the simulation model therefore is the consideration of a directed graph of simulation objects, with typed, weighted arcs, expressing the event causalities and occurrence frequencies among nodes. For studying different partitionings of the simulation model, arbitrary subsets of nodes of this graph are assigned to LPs, each of which executes on a dedicated processor of the target platform.

For a case study we assume the simulation model in Figure 3, which consists of 32 *nodes* connected in a 4×8 torus topology. Events are exchanged among nodes along closed paths in west-east and south-north directions. In order to allow a node to be able to forward an event along the correct path, each event is type tagged (type 1 or 2). When an event is received and subsequently simulated, a new internal event is generated and inserted into the EVL. The simulation of an internal event, in turn, creates an external event which is forwarded to the corresponding destination node. The model is

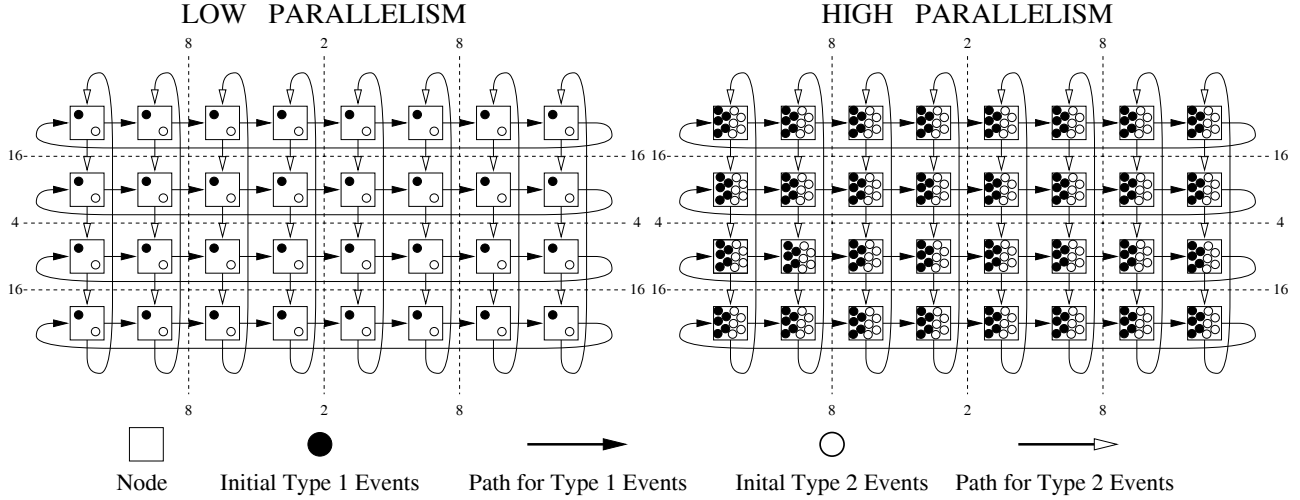


Figure 3: Structure of Simulation Model under Investigation

non-preemptive and constant with respect to the event population.

In order to partition the model among the LPs, the model is cut along the dotted lines for the corresponding number of LPs and contiguous nodes are then assigned to a single LP. For simulation on 32 LPs, each node is assigned to a single LP. During simulation, if the destination node of an “external” event lies within the same partition, it is treated as an “internal” event and immediately inserted into the EVL.

The degree of parallelism in the model is varied by the number of initial events assigned to each node. In the model with low parallelism ([-HP]), one event of each type is assigned to each node whereas with high parallelism ([+HP]), five events of each type are assigned to each node. With respect to the uniformity of the timestamp increments, the uniform model ([-NU]) assumes timestamps to be normally distributed with $\sim N(100.0, 0.01)$ as opposed to the non-uniform model ([+NU]) which assumes $\sim N(100.0, 25.0)$. We shall restrict here to the intuitive question of the impact of the factors NU at the levels +NU (nonuniform LVT progression) and -NU (uniform LVT progression), and HP at the levels +HP (high degree of inherent model parallelism) and -HP (low degree of parallelism).

4.2 Abstracted DS protocols and skeletal DS codes

For the implementation of the different representatives of DS protocols it is again important to find the best tradeoff between implementation detail and tractability of analysis. Another aspect of the implementation is to preserve the comparability of predicted performance for the various protocols, and most importantly, to preserve the preliminary coding efforts for the final implementation. The N-MAP [8, 10] toolset meets these requirements: first it supports an incremental coding process, i.e. a repetitive refinement of a preliminary code skeleton eventually yields the full implementation which allows for performance prediction at any level of completeness of the skeleton. Furthermore, N-MAP allows the use of mutables in pre-processor directives such as `#define` and `#ifdef` and therefore permits a conditional compilation of the source code based on the mutable settings defined in the scenario manager. Thus, using one and the same source program text, various code segments may be conditionally included or excluded in the resulting executable code in order to produce a DS

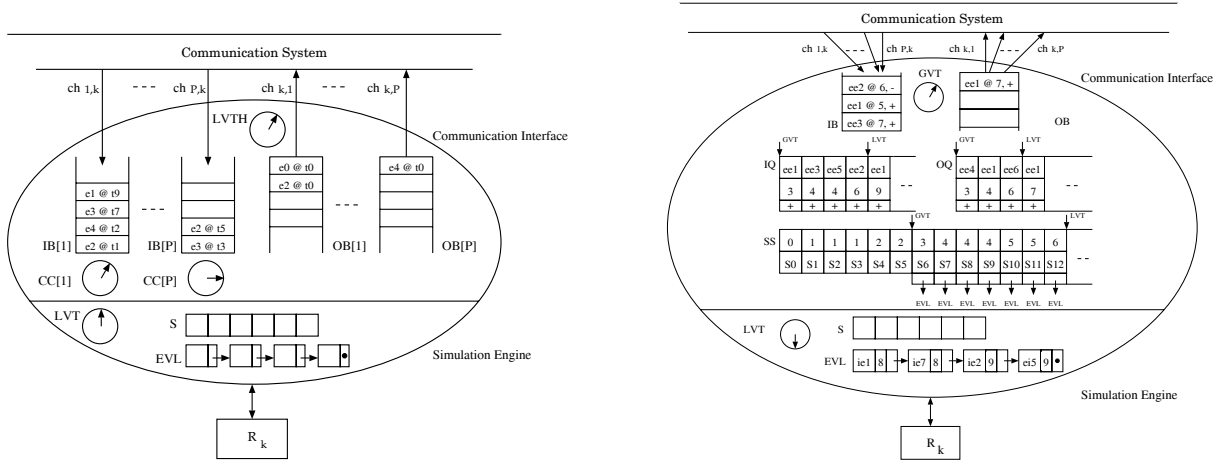


Figure 4: LPs for Chandy/Misra/Bryant (left) and Time Warp based (right) DS

protocol with the characteristics defined by the user in the scenario. In order to provide for the fairest comparison possible of the implementation strategies under investigation, the same source code is used, wherever possible, for identical functionalities in all strategies. Despite the obvious differences between the TW and CMB simulation protocols, a surprising amount of overlap can be identified.

Figure 5 shows the basic architecture of CMB (left) and TW (right) LPs, and Figure 5 illustrates the N-MAP source text used for generating all simulation protocols, explains the choice of the implementation detail of the skeleton used for performance prediction, and visualizes the amount of code overlap in the protocols. At the top, the various implementation options are listed for both TW and CMB. In order to determine, for example, which lines of code must be included in order to generate a TW simulator employing an *aggressive cancellation* policy and *full state saving*, find the column in the upper left-hand corner where both rows are marked with a solid square (first column) and follow this column down. Source code lines which have a solid bar in this column must be included in the final code and lines having no bar are excluded.

All strategies follow the same basic structural organization in that they all consist of the same series of consecutive functional segments (shown to the right of the source code in the figure). The actual implementation of a particular segment may, however, vary from strategy to strategy. When execution begins, the simulator and model are initialized in the `INIT` segment before the main simulation loop (`LOOP`) is entered. In the following `INPUT` segment, the receive buffer is repeatedly probed and incoming messages are read and processed until the buffer is empty. The `GUARD` segment ensures that it is safe to proceed with local simulation – if not, the simulator must loop back to the `INPUT` phase so long until conditions permit the passing of the `GUARD` segment. The next event to be simulated is then chosen in the `EVENT` segment. In the following `SIMULATE` segment, the actual occurrence of the event in the model is simulated and the state of the simulator is updated. External events generated during simulation are sent out in the `OUTPUT` segment and finally, the new state of the simulator is logged in the `LOG` segment.

The `INIT` code segment initializes both the protocol and model and inserts the initial internal events generated into the event list (EVL). In the case of CMB, an initial nullmessage with timestamp 0 is inserted into each of the input buffers. TW protocols using a *full state saving* (`-IS`) strategy must log this initial state onto the state stack whereas *incremental state saving* (`+IS`) strategies need not log the

		Aggressive Cancellation (-LC)	TW
		Lazy Cancellation (+LC)	
		Full State Saving (-IS)	
		Incremental State Saving (+IS)	
		Optimism Control (+OC)	
		Sender Init. Nullmessages (+SI)	
		Receiver Init. Nullmessages (-SI)	CMB
		GVT Reduction(+GT)	
		initProtocol(); initModel(); while (ie = nextIntEvent()) insert(ie,EVL); for (i=0; i<maxIB; i++) { e = createNullmsg(0); insert(e,IB[i]); }	INIT
		logFullState(); while(GVT < ENDTIME) { while(LVT < ENDTIME) { while(m = nextRecvBuffer()) { if (isMsg(m)) { dual=dualExists(m,IQ); if (ts(m) <= LVT && ((isPos(m) && !dual) (isPos(m) && dual))) { LVT = restoreCheckpoint(m); LVT = restoreBacktrail(m); generateAntimessages(LVT); while(ee = nextExtEvent()) { insertDup(ee,OB); insert(ee,OQ); } sendoutContents(OB); } if (dual) removeDual(m,IQ) else insert(m,IQ); } if (isMsg(m)) { insert(m,IB[m->source]); } if (isNullmsgReq(m)) { e = createNullmsg(LVT+lookahead()); insert(e,OB[m->source]); sendoutContents(OB[m->source]); } if (isGVTmsg(m)) { GVT = calculateGVT(m); fossilCollection(GVT); updateClocks(GVT); } } blocked=0; LVTH=MAXFLOAT; iMin=0; for (i=0; i<maxIB; i++) { if (empty(IB[i])) { blocked=1; e = createNullmsgRequest(i); insertEvent(e,OB[i]); sendoutContents(OB[i]); } else CC[i]=ts(first(IB[i])); if (CC[i] < LVTH) { LVTH=CC[i]; iMin=i; } } if (blocked) continue; if (memUsed > memLimit) continue; if (empty(EVL) && empty(IQ)) continue; e = removeLowestEVLorIQ(); if (ts(first(EVL)) <= LVTH) e = removeFirst(EVL); else e = removeFirst(IB[iMin]); LVT = ts(e); if (!isNullmsg(e)) { simulate(e); while(ie = nextIntEvent()) insert(ie,EVL); while(ie = nextPreempted()) remove(ie,EVL); while(ee = nextExtEvent()) { if (!dualUpdate(ee,OQ)) insert(ee,OQ); insertDup(ee,OB); insert(ee,OQ); insert(ee,OB[ee->dest]); } fillOB(LVT); sendoutContents(OB); } for (i=0; i<maxOB; i++) { if (empty(OB[i])) { e = createNullmsg(LVT + lookahead()); insert(e,OB[i]); } if (empty(OB[i])) sendoutContents(OB[i]); } logFullState(); logIncrState(); } }	Loop Straggler/Rollback INPUT NullMsgReq GVT Packet GUARD EVENT SIMULATE OUTPUT
		logFullState(); logIncrState(); }	LOG

Figure 5: Task Structure Specification of the Simulator

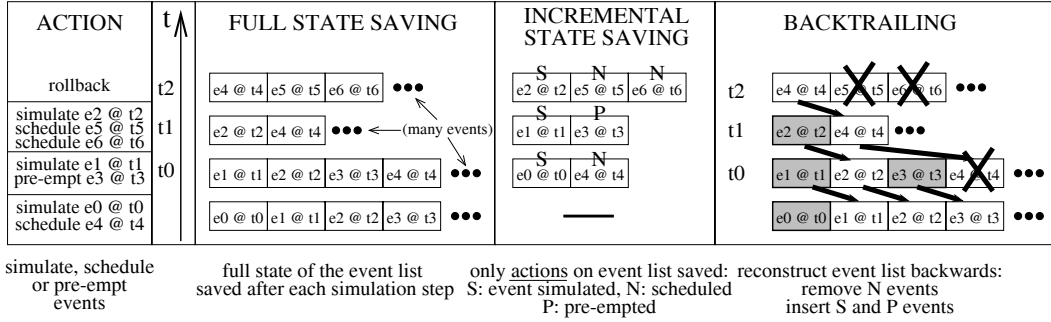


Figure 6: Full state saving (-IS) vs. incremental state saving (+IS) memory policies.

initial state (see Figure 6). The main simulation loop (LOOP) is then executed until GVT progresses to ENDTIME or, in the case of CMB protocols not using *GVT reduction* methods (-GVT), until LVT reaches ENDTIME.

In the INPUT loop, the receive buffer is probed and messages are repeatedly read and processed until the buffer is empty. In the case of the TW protocol, it must first be ascertained whether the incoming message is a *straggler message* (i.e. a message having a timestamp less or equal to the current LVT) and, if so, whether a rollback must be initiated. Under the assumption that the communication channels are not guaranteed to be FIFO, a rollback must be initiated in two cases. First, if the straggler message is positive and no corresponding (dual) antimessage has yet been received, a rollback to the first local state having an LVT less than the timestamp of the straggler must be initiated. Second, if the straggler message is an antimessage and its dual positive message has already been received, a rollback to the state immediately preceding the processing of the corresponding positive message must be performed in order to annihilate the effects of the erroneously processed message. The TW protocol then generates antimessages for all messages sent as a result of simulation states which occurred after the restored state. In the case of *aggressive cancellation* (-LC), the antimessages generated are inserted into both the output buffer (OB) and output queue (OQ) and the contents of the OB are immediately sent out. If the protocol uses *lazy cancellation* (+LC), the antimessages are inserted into the OQ only and no immediate sendout takes place. Finally, regardless of if a rollback has occurred or not, the incoming message is then either inserted into the input queue (IQ) or, if a dual message already exists in the IQ, both the message and its dual are annihilated.

As opposed to TW, the handling of incoming messages in the CMB protocol is much simpler since all messages received, positive messages and nullmessages alike, are inserted into the corresponding input buffer (IB). A special case, however, arises for the *receiver initiated nullmessage protocol* (-SI), since the incoming message may also be a *nullmessage request*. In this case, the receiving LP responds immediately to the requesting LP by sending a nullmessage containing the LVT it guarantees on the corresponding channel. This time is calculated on the basis of the LP's current LVT and the *lookahead* for that channel.

The use of a *GVT reduction* method is mandatory for all TW protocols whereas it may be optionally included in CMB protocols (+GVT). The GVT reduction method implemented here exchanges *GVT calculation packets* along pre-defined circuits (rings) among the LPs. When a GVT calculation packet is received, the LP immediately calculates a new *local minimum LVT* (minLVT) based on its present LVT and the minimum timestamp of (unconfirmed) *in transit* messages. This minLVT is written to the GVT packet to inform the other LPs that no packets having a timestamp lower than minLVT

will be sent from this LP *provided no rollbacks occur on the LP in the meantime*. The minimum of all minLVTs can thus be taken as estimate for the actual GVT since no message will be sent before this time. TW protocols use this new GVT to perform an immediate fossil collection thereby returning freed memory (events) to the free pool. CMB protocols, on the other hand, use the new GVT to advance the channel clocks of the input buffers to this time by removing nullmessages from the buffers having a timestamp lower than GVT. If a buffer is empty after this process, a new nullmessage is inserted into the buffer with a timestamp of GVT.

After all incoming messages have been processed in the INPUT loop, a decision must be made whether local simulation may proceed or not. For CMB protocols, the GUARD segment ensures that it is *safe* to proceed whereas for TW protocols it determines if local simulation is *possible*. In the CMB protocol implemented, simulation must be halted when an input buffer becomes empty. With the *sender initiated nullmessage protocol (+SI)*, this will not be the case for any longer period of time since all LPs send out on all output channels during each simulation step. Conversely, with the *receiver initiated nullmessage protocol (-SI)*, the LP encountering this situation must explicitly request the sending of a nullmessage to unblock itself by sending a *nullmessage request* to the respective LP. In either case, the LP remains blocked and loops back to the INPUT segment so long until all input buffers contain at least one message. The LP then determines its *local virtual time horizon (LVTH)* by finding the lowest channel time of all input buffers.

The GUARD segment for TW protocols is somewhat simpler in that it must only determine that there are indeed events to process, i.e. the partition is not depleted of events. If there are no events, the LP loops back to the INPUT segment to await the arrival of new messages. If an *optimism control method (+OC)* is used and it is determined here that simulation may not continue due to constraints imposed by the throttling mechanism, the LP must also loop back to the INPUT phase and wait for conditions to arise that allow for further local simulation. The throttling mechanism implemented here controls optimism by adaptively limiting the number of available events. If the LP has exceeded the number of events it may use, it must loop back to the INPUT phase and wait until a sufficient amount of events be freed through the arrival of a straggler message (events freed by rollback) or a GVT calculation packet (events freed by fossil collection).

Now that it is safe to proceed, the event to be simulated in this simulation step must be chosen (EVENT segment). TW protocols choose the event with the lowest timestamp from either the EVL or the IQ. Similarly, CMB protocols choose the first event from the EVL if its timestamp is lower than the current LVTH and otherwise choose the event with the lowest timestamp from one of the input buffers.

Before the actual simulation of the chosen event, (SIMULATE segment), the LP's LVT is set to the timestamp of the chosen event. The event is then passed to the model for simulation and the model must, in turn, return three lists to the simulator. First, a list of new internal events resulting from the simulation of the current event. Second, a list of internal events which have now been pre-empted by the occurrence of the current event in the model. Third, a list of external events which must be sent out to other LPs as a result of simulation.

All protocols insert new internal events into the EVL and remove pre-empted events. In the CMB protocol, external events are inserted into the respective output buffers (OB). The TW protocol, on the other hand, requires that copies of external messages first be inserted into the output queue (OQ) in order to document the LP's sendout history. Before inserting into the OQ, the *lazy cancellation protocol (+LC)* first checks if a dual antimessage corresponding to the newly generated external message

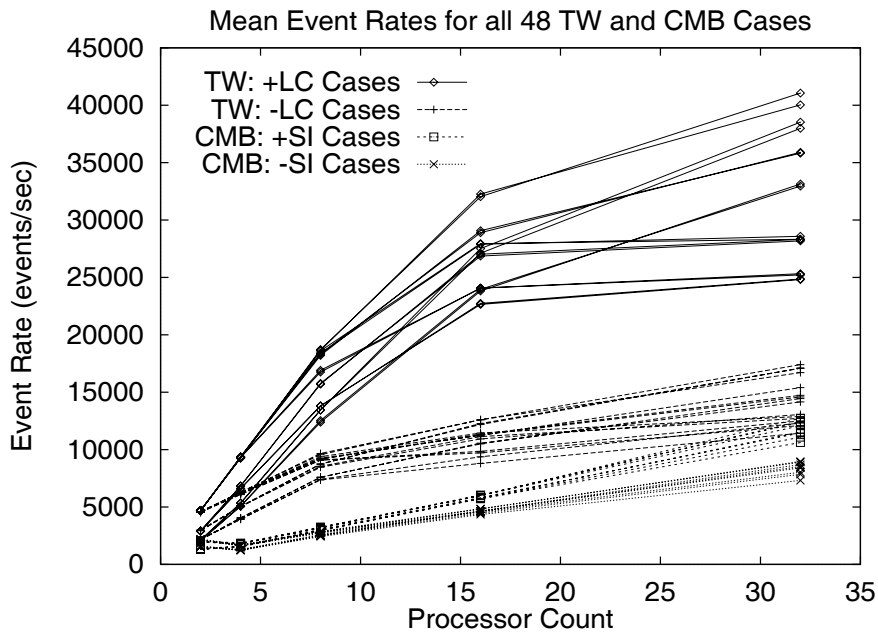
already exists in the output queue. If so, it is removed and both the event and the antimessages are annihilated (and freed) since re-simulation has produced the same event. After all external events have been inserted (or annihilated), the +LC protocol fills the OB with all messages in the output queue that have not yet been sent and that have a timestamp less or equal to the present LVT. In the case of *aggressive cancellation* (-LC), all external events are simply inserted into the OB.

In the OUTPUT phase, the contents of the OBs are sent out to the respective LPs. The CMB *sender initiated nullmessage protocol*, however, first inserts nullmessages into all empty output buffers so that all output buffers then contain either one or more positive messages or a single nullmessage. Finally, TW protocols log the new state of simulator onto the state stack (either in full or incrementally) and then loop back to the INPUT phase.

5 Experimental results

The goal now for an experimental study is to explain DS performance in terms of the attainable event rate and the induced communication overhead (as our response variables) for CMB and TW. Within CMB, the factors SI at the levels +SI (sender initiated nullmessages) and -SI (receiver initiated nullmessages), GVT at the levels +GVT (GVT reduction technique “on”) and -GVT (GVT reduction technique “off”), shall be studied, whereas for TW, the coinfluence of the following factors is of particular interest: the message cancellation strategy LC at the levels +LC (lazy cancellation) and -LC (aggressive cancellation), the state saving policy IS at the levels +IS (incremental state saving) and -IS (full state saving), and the optimism control mechanism OC at the levels +OC (optimism control enabled) and -OC (optimism control disabled). We aim at explaining the respective DS protocol performance with respect to NU at the levels +NU (nonuniform LVT progression) and -NU (uniform LVT progression), and HP at the levels +HP (high degree of inherent model parallelism) and -HP (low degree of parallelism).

To this end, a collection of experiments have been conducted within our performance framework in a fully automated way. The output of this analysis, statistically aggregating 48 case executions at 5 different processor counts (i.e. 240 executions, each repeated 30 times), is condensed in the tables in Figure 7 and Figure 8. Figure 7 gives for each possible combination of factor levels the average event rate the respective protocol could achieve. A first observation here is, that TW with lazy cancellation achieved higher event rates than any other protocol, but CMB with sender initiated nullmessages was even able to outperform TW with aggressive cancellation. Figure 8 summarizes the respective probability of the F-value that the analysis of variance could identify, together with the sign and the value of coefficients of the fitted model. A bullet in the probability section of the table indicates significance at the confidence level $\alpha = 0.01$, a small circle indicates significance at $\alpha = 0.05$. We organize the interpretation of this analysis data into three sections. First, we collect arguments along and within the optimistic strategy, i.e. the factor TW, followed by some reasoning on the factor CMB. Finally, a cross-comparison of TW and CMB is given.



CMB Case	Event Rate (Events/Second)				
	2 LP	4 LP	8 LP	16 LP	32 LP
[-SI -GVT -NU -HP]	2159.9	1668.9	2816.3	4649.9	8699.2
[-SI -GVT -NU +HP]	2096.3	1610.5	2731.9	4569.6	8034.9
[-SI -GVT +NU -HP]	2207.4	1707.8	2881.9	4852.7	8909.7
[-SI -GVT +NU +HP]	2125.0	1635.6	2804.2	4827.8	8966.5
[-SI +GVT -NU -HP]	1554.1	1259.6	2511.3	4458.1	7850.3
[-SI +GVT -NU +HP]	1521.0	1226.3	2427.4	4350.3	7314.0
[-SI +GVT +NU -HP]	1587.2	1298.7	2592.5	4632.7	8411.5
[-SI +GVT +NU +HP]	1529.8	1239.7	2520.6	4635.6	8527.8
[+SI -GVT -NU -HP]	1998.1	1828.2	3235.6	5982.6	12733.3
[+SI -GVT -NU +HP]	1955.0	1784.5	3156.2	5996.3	12077.1
[+SI -GVT +NU -HP]	2040.6	1857.3	3248.7	6037.1	12373.2
[+SI -GVT +NU +HP]	1956.4	1782.1	3167.6	6019.6	12442.9
[+SI +GVT -NU -HP]	1342.9	1572.1	2994.4	5741.0	11079.5
[+SI +GVT -NU +HP]	1310.6	1530.1	2956.2	5716.9	10616.6
[+SI +GVT +NU -HP]	1371.1	1601.7	3035.7	5825.7	11446.5
[+SI +GVT +NU +HP]	1311.5	1538.0	2956.6	5771.4	11531.7

TW Case	Event Rate (Events/Second)				
	2 LP	4 LP	8 LP	16 LP	32 LP
[-LC -IS -OC -NU -HP]	2998.3	5045.2	8466.2	10933.3	13066.4
[-LC -IS -OC -NU +HP]	2929.2	5055.5	8733.1	12271.4	17102.1
[-LC -IS -OC +NU -HP]	2165.4	3933.8	7410.7	9374.2	12315.1
[-LC -IS -OC +NU +HP]	2148.7	4051.6	7601.7	10495.1	14448.8
[-LC -IS +OC -NU -HP]	2983.5	5042.1	8545.8	11136.7	12731.5
[-LC -IS +OC -NU +HP]	2950.3	5009.6	8747.6	12184.3	17089.4
[-LC -IS +OC +NU -HP]	2159.4	3947.8	7361.9	8802.4	11424.9
[-LC -IS +OC +NU +HP]	2141.8	4052.6	7584.8	10566.2	14157.2
[-LC +IS -OC -NU -HP]	4590.8	6259.2	9252.4	11241.7	15398.2
[-LC +IS -OC -NU +HP]	4592.8	6264.5	9661.1	12612.2	17404.0
[-LC +IS -OC +NU -HP]	4584.9	6112.8	9097.5	9850.0	12495.5
[-LC +IS -OC +NU +HP]	4547.1	6198.1	9181.9	11274.8	14588.1
[-LC +IS +OC -NU -HP]	4610.4	6222.0	9380.8	11455.0	12884.3
[-LC +IS +OC -NU +HP]	4574.3	6298.1	9555.1	12595.0	16714.2
[-LC +IS +OC +NU -HP]	4549.8	6083.9	9082.6	9717.7	12098.2
[-LC +IS +OC +NU +HP]	4529.4	6175.1	9190.9	11341.2	14731.8
[+LC -IS -OC -NU -HP]	2903.8	6819.5	15764.1	26862.8	28182.8
[+LC -IS -OC -NU +HP]	2905.7	6607.5	13821.4	22655.4	24819.1
[+LC -IS -OC +NU -HP]	2120.4	5357.5	13448.4	27511.7	38524.9
[+LC -IS -OC +NU +HP]	2065.1	5086.1	12527.8	23960.0	32958.1
[+LC -IS +OC -NU -HP]	2922.2	6836.9	15707.5	27026.6	28310.4
[+LC -IS +OC -NU +HP]	2894.7	6606.7	13782.9	22733.2	24856.4
[+LC -IS +OC +NU -HP]	2106.2	5319.7	13440.0	27108.3	37988.6
[+LC -IS +OC +NU +HP]	2052.1	5069.1	12385.0	23819.0	33143.0
[+LC +IS -OC -NU -HP]	4682.5	9368.4	18605.8	27940.1	28325.2
[+LC +IS -OC -NU +HP]	4697.4	9332.6	16904.6	24074.5	25326.8
[+LC +IS -OC +NU -HP]	4695.8	9287.0	18709.1	32031.1	41063.3
[+LC +IS -OC +NU +HP]	4678.8	9329.5	18204.3	29073.8	35824.3
[+LC +IS +OC -NU -HP]	4663.6	9370.9	18408.9	27891.1	28585.6
[+LC +IS +OC -NU +HP]	4698.6	9302.4	16753.5	24064.5	25216.7
[+LC +IS +OC +NU -HP]	4696.2	9251.7	18661.6	32261.8	40031.9
[+LC +IS +OC +NU +HP]	4663.9	9292.5	18331.4	28898.5	35894.8

Figure 7: TW vs. CMB, Response Event Rate: A First Observation

CMB Case	Pr(F)					Coefficients				
	2 LP	4 LP	8 LP	16 LP	32 LP	2 LP	4 LP	8 LP	16 LP	32 LP
(Intercept)	-	-	-	-	-	1755.486	1571.399	2876.018	5260.535	10073.840
SI	● 0.000	● 0.000	● 0.000	● 0.000	● 0.000	-94.548	115.829	-217.301	633.636	1737.346
GVT	● 0.000	● 0.000	● 0.000	● 0.000	● 0.000	-314.079	-163.948	-126.610	-458.067	-1073.840
NU	● 0.000	● 0.000	● 0.000	● 0.000	● 0.000	10.872	11.399	25.426	63.445	274.541
HP	● 0.000	● 0.000	● 0.000	● 0.000	● 0.000	-27.601	-27.570	-40.084	-22.773	-117.414
SI:GVT	● 0.000	● 0.000	● 0.000	○ 0.024	● 0.000	-12.890	36.525	18.589	-13.534	-143.971
SI:NU	● 0.001	● 0.004	● 0.000	● 0.000	● 0.000	-2.406	-2.911	-20.235	-47.469	-129.025
SI:HP	0.703	0.296	0.915	0.144	0.999	0.275	-1.052	0.299	8.695	0.028
GVT:NU	● 0.005	○ 0.027	0.519	0.412	● 0.000	-2.038	2.235	1.797	4.873	116.692
GVT:HP	● 0.000	● 0.004	0.920	0.324	0.112	5.450	2.914	0.281	5.871	27.066
NU:HP	● 0.000	● 0.000	0.849	0.739	● 0.000	-7.389	-5.396	0.531	1.977	154.376
SI:GVT:NU	0.515	0.174	0.359	0.891	● 0.009	0.471	1.371	2.555	-0.812	45.069
SI:GVT:HP	0.550	0.066	0.532	0.533	0.931	-0.432	-1.858	1.741	-3.707	-1.482
SI:NU:HP	0.280	0.621	0.429	0.188	0.611	-0.781	-0.497	2.204	-7.835	8.647
GVT:NU:HP	0.200	0.937	0.304	0.997	○ 0.021	0.926	-0.079	-2.865	0.022	-39.567
SI:GVT:NU:HP	0.531	0.086	0.459	0.966	0.819	0.433	1.734	-2.063	-0.256	-3.890

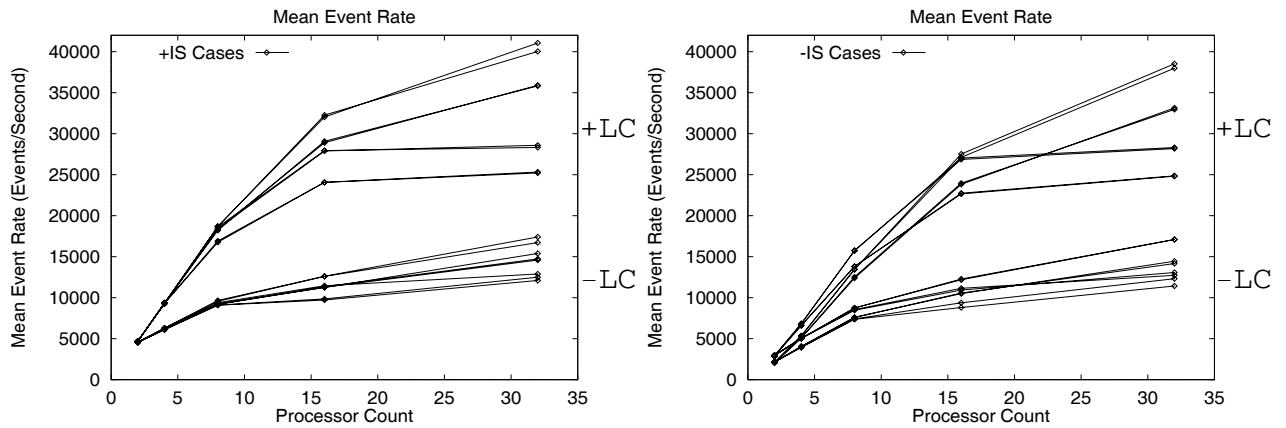
TW Case	Pr(F)					Coefficients				
	2 LP	4 LP	8 LP	16 LP	32 LP	2 LP	4 LP	8 LP	16 LP	32 LP
(Intercept)	-	-	-	-	-	3581.345	6498.233	12333.370	18912.950	23332.040
LC	● 0.000	● 0.000	● 0.000	● 0.000	● 0.000	13.360	1135.601	3635.617	7911.227	8604.754
IS	● 0.000	● 0.000	● 0.000	● 0.000	● 0.000	1052.739	1262.830	1355.210	919.802	540.820
OC	0.521	0.328	0.343	0.699	0.490	-1.632	-4.998	-14.217	-14.758	-47.311
NU	● 0.000	● 0.000	● 0.000	● 0.000	● 0.000	-210.260	-342.050	-307.352	292.470	1885.607
HP	● 0.000	● 0.001	● 0.000	● 0.000	● 0.000	-11.787	-17.719	-256.353	-598.515	-512.461
LC:IS	● 0.000	● 0.000	● 0.000	● 0.000	● 0.000	43.720	413.258	741.612	669.804	278.825
LC:OC	0.748	0.930	0.182	0.731	0.385	0.816	0.447	-19.998	13.095	59.546
LC:NU	0.834	● 0.000	● 0.000	● 0.000	● 0.000	0.533	-49.534	67.953	1121.648	3235.300
LC:HP	0.281	● 0.000	● 0.000	● 0.000	● 0.000	2.739	-49.859	-364.120	-1280.525	-1683.437
IS:OC	0.159	0.524	0.677	0.335	0.496	-3.576	-3.256	-6.235	36.830	46.620
IS:NU	● 0.000	● 0.000	● 0.000	● 0.000	● 0.000	197.626	295.854	431.636	533.124	246.539
IS:HP	○ 0.020	● 0.000	● 0.001	0.621	0.254	5.904	28.754	48.030	18.885	-78.181
OC:NU	0.675	0.445	0.790	○ 0.043	0.113	1.066	-3.897	-3.986	-77.367	-108.865
OC:HP	0.559	0.649	0.927	0.642	0.146	-1.482	-2.324	-1.375	-17.758	99.727
NU:HP	0.173	● 0.001	● 0.000	● 0.003	● 0.000	-3.459	17.363	120.047	112.961	-281.845
LC:IS:OC	0.232	0.598	0.911	0.184	0.348	3.039	2.690	-1.677	-50.737	-64.295
LC:IS:NU	0.180	● 0.000	● 0.000	● 0.000	● 0.000	3.408	69.249	238.439	412.486	315.341
LC:IS:HP	0.154	● 0.000	● 0.000	0.726	0.580	3.625	26.623	59.073	13.367	37.911
LC:OC:NU	0.747	0.322	0.188	0.627	0.652	-0.819	-5.055	19.740	-18.517	-30.895
LC:OC:HP	0.199	0.798	0.990	0.647	0.626	-3.265	1.308	0.188	-17.493	33.430
LC:NU:HP	0.407	0.296	● 0.000	0.058	● 0.001	-2.104	-5.341	159.434	72.497	-228.986
IS:OC:NU	0.059	0.535	0.617	0.200	0.698	4.795	-3.170	7.496	48.904	-26.632
IS:OC:HP	0.812	0.099	0.528	0.086	0.716	-0.604	8.445	9.454	-65.721	24.908
IS:NU:HP	0.306	0.195	0.380	0.835	0.357	-2.601	6.625	13.158	-7.954	63.152
OC:NU:HP	0.159	0.443	0.551	○ 0.034	○ 0.046	-3.583	3.921	8.945	81.112	137.035
LC:IS:OC:NU	0.591	0.164	0.957	0.981	0.733	1.365	7.114	-0.805	-0.888	-23.418
LC:IS:OC:HP	0.483	0.381	0.237	0.916	0.764	1.783	-4.478	17.718	4.009	20.537
LC:IS:NU:HP	● 0.006	● 0.006	● 0.005	0.575	0.181	6.950	14.239	42.607	-21.424	-91.801
LC:OC:NU:HP	0.314	0.391	0.326	0.997	0.564	2.557	4.386	-14.741	0.126	39.591
IS:OC:NU:HP	0.506	0.733	0.871	0.108	0.431	-1.690	-1.742	2.439	-61.483	53.997
LC:IS:OC:NU:HP	0.058	0.774	0.546	0.958	0.688	-4.832	1.468	-9.059	-2.020	27.487

Figure 8: Event Rate: Pr(F) and Coefficients for the 2^k Factorial Design Fit

5.1 Performance influence of factor TW

From the event rates attainable with TW (Figure 7) it can already be seen that partitioning the simulation model, thus increasing the number of LP's (and by that also the number of physical processors) yields a considerable performance increase. This is the case for any combination of factors, but at different increments: A 16-fold (from 2065.1 to 32958.1) acceleration is gained when using 32 processors instead of 2 for a nonuniform virtual time progression (+NU) at a high model parallelism degree (+HP) when lazy cancellation (+LC) is used at full state saving (-IS) and optimism control disabled (-OC). The highest possible event rate is 41063.3, achieved with lazy cancellation (+LC) together with incremental state saving (+IS) and optimism control disabled (-OC) for a simulation model with nonuniform virtual time progression (+NU) and a low degree of model parallelism (-HP) on 32 processors. The worst is 2052.1 at ([+LC -IS +OC +NU +HP]) with 2 processors. Other observations are that the event rate “seems” to benefit from incremental state saving in almost every case, that optimism control does not necessarily help to increase the event rate, and that e.g. the uniformity factor (NU) can have diametral effects, depending on the other factors.

Clearly, looking just at the event rates does not suffice for judging the general validity of the performance results, or the relative importance of factors. As an example, if a software designer is interested in the relative merit of LC and IS to give priority to certain implementation efforts, the event rate table does not give a direct answer. A more detailed analysis is required to answer these kinds of questions, specifically, the “amount” of empirical evidence in the observations needs to be quantified in order to be able to give reliable decision parameter values. This quantification of empirical evidence is derived by the analysis of variance within and between groups of factors, as summarized in the table of



TW Case	Pr(F)				
	2 LP	4 LP	8 LP	16 LP	32 LP
(Intercept)	-	-	-	-	-
LC	● 0.000	● 0.000	● 0.000	● 0.000	● 0.000
IS	● 0.000	● 0.000	● 0.000	● 0.000	● 0.000
OC	0.521	0.328	0.343	0.699	0.490
NU	● 0.000	● 0.000	● 0.000	● 0.000	● 0.000
HP	● 0.000	● 0.001	● 0.000	● 0.000	● 0.000

TW Case	Coefficients				
	2 LP	4 LP	8 LP	16 LP	32 LP
(Intercept)	3581.345	6498.233	12333.370	18912.950	23332.040
LC	13.360	1135.601	3635.617	7911.227	8604.754
IS	1052.739	1262.830	1355.210	919.802	540.820
OC	-1.632	-4.998	-14.217	-14.758	-47.311
NU	-210.260	-342.050	-307.352	292.470	1885.607
HP	-11.787	-17.719	-256.353	-598.615	-512.461

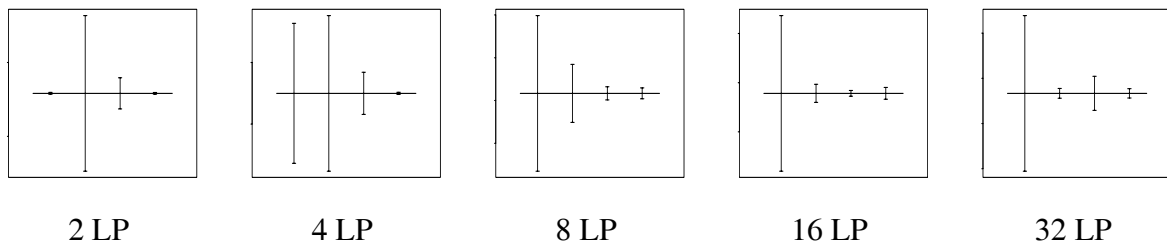


Figure 9: TW, Response Event Rate: The Relative Importance of Factors

Figure 8. As for the question of the relative importance of LC and IS we can see from the F-values' probabilities that both LC and IS have significant influence on the response *event rate* (Pr() values in the lines LC and IS are all zero). More than that, the coefficients (same lines) quantify the "amount" of influence on the response, namely an increasing amount for LC when the number of processors are increased, but, surprisingly, a decreasing "amount" of influence induced by IS as the number of LP's increases: While LC "contributes" about 13.4 to the event rate for 2 LPs, IS contributes about 1052.7; at 32 LP's LC contributes 8604.8, while the contribution of IS has declined to 540.8. This phenomenon is better presented graphically as in the plots of Figure 9: The potential "range" of contribution that LC "induces" at small numbers of LPs is relatively small to that induced by IS. The situation changes as we go to 8 LPs. Here the contribution of LC to the response *event rate* takes over, and continues to dominate over the contribution of IS. Yet we cannot say whether this is due to the simulation model attributes NU and HP, but will come back to this issue. Going further into the effects of factors, we cannot find significance for the influence of the optimism control OC in the data set, but identify the negative effects of the simulation model attributes (NU, HP) on the response. The negative contribution of a simulation model progressing time in a nonuniform way (+NU) in TW is intuitive (-210.3 at 2 LPs, -342.1 at 4 LPs etc.). But why the positive coefficient 1885.6 at 32 LPs? At this point, indeed, one or more of the other factors (it is LC as will be seen) finds the chance to effectively make use of the irregularities in the time progression, thus – overall – yielding a positive contribution. The negative contribution of a simulation model with high model parallelism (+HP) in TW is counterintuitive! In the particular data set, although the negative effect is small, a large event population is the consequence of a high model parallelism, extending communication overheads which – overall – overwhelm the acceleration potentials of high model parallelism. To investigate in which way this happens, we need to look at the higher level interactions among factors.

Besides the effects that LC and IS (on their own) induce on the response, their combination also explains a certain contribution. This effect is called the *first level interaction* among LC and IS, denoted as LC:IS, and reveals an interesting tendency in the data set: the interaction among LC and IS increases until a break even point is reached (at 8 LPs), from where on it decreases. This strengthens our argument from above, that from an implementation viewpoint no absolute priority can be given to either LC or IS (in general LC is easier to implement than IS), but will depend on (besides other things) the number of LPs involved for a particular simulation model. Note at this point, that our methodology has detected a nontrivial performance phenomenon of TW, which presumably would not have been seen using a standard analytical approach.

Yet another issue of general interest is the influence of the simulation model on TW performance. In the case study we have identified the two simulation model attributes NU and HP. Using *first level interactions* we can identify this impact for both LC and IS. As far as lazy cancellation (LC) is concerned, we see that interactions with the simulation model become significant in simulations involving more than 2 LPs. For a larger number of LPs, LC:NU contributes in a positive way to the event rate (3235.3 with 32 LPs), whereas LC:HP contributes negatively (with -1683.4 at 32 LPs) (but note that LC on its own (8604.8 with 32 LPs) overshadows this effect). We can therefore conclude, that lazy cancellation benefits more noticeably from nonuniform models than from models with high parallelism. The interaction among the state saving policy (IS) and the simulation model (NU and HP) is significant at some points, but contributes only very little, relatively speaking. Finally note, that also at the higher level interactions (LC:IS:NU, LC:IS:HP, LC:NU:HP) we find significance whenever LC is involved, allowing for the conclusion, that in this data set lazy cancellation together with incremental state saving was of utmost importance for performance, and, although one might have had high expectations, the optimism control mechanism could not really improve performance.

CMB Case	Communication Overhead (Fraction)				
	2 LP	4 LP	8 LP	16 LP	32 LP
[-SI -GVT -NU -HP]	0.257	0.297	0.312	0.321	0.324
[-SI -GVT -NU +HP]	0.261	0.298	0.312	0.321	0.325
[-SI -GVT +NU -HP]	0.254	0.296	0.311	0.320	0.326
[-SI -GVT +NU +HP]	0.259	0.297	0.312	0.321	0.325
[-SI +GVT -NU -HP]	0.411	0.361	0.339	0.329	0.322
[-SI +GVT -NU +HP]	0.412	0.362	0.339	0.329	0.323
[-SI +GVT +NU -HP]	0.409	0.360	0.338	0.329	0.325
[-SI +GVT +NU +HP]	0.412	0.361	0.338	0.329	0.325
[+SI -GVT -NU -HP]	0.372	0.481	0.480	0.473	0.468
[+SI -GVT -NU +HP]	0.370	0.481	0.481	0.473	0.473
[+SI -GVT +NU -HP]	0.372	0.480	0.480	0.473	0.469
[+SI -GVT +NU +HP]	0.370	0.481	0.480	0.473	0.470
[+SI +GVT -NU -HP]	0.477	0.491	0.473	0.465	0.452
[+SI +GVT -NU +HP]	0.477	0.490	0.473	0.466	0.457
[+SI +GVT +NU -HP]	0.476	0.490	0.473	0.466	0.458
[+SI +GVT +NU +HP]	0.477	0.490	0.473	0.466	0.459

Figure 10: Mean Communication Overhead (Fraction) for CMB

CMB Case	Pr(F)					Coefficients				
	2 LP	4 LP	8 LP	16 LP	32 LP	2 LP	4 LP	8 LP	16 LP	32 LP
(Intercept)	-	-	-	-	-	0.379	0.407	0.401	0.397	0.394
SI	● 0.000	● 0.000	● 0.000	● 0.000	● 0.000	0.045	0.078	0.076	0.072	0.069
GVT	● 0.000	● 0.000	● 0.000	● 0.000	● 0.000	0.065	0.018	0.005	0.000	-0.004
NU	● 0.000	● 0.000	● 0.000	0.202	● 0.000	0.000	0.000	0.000	0.000	0.001
HP	● 0.000	● 0.000	● 0.000	0.082	● 0.000	0.001	0.000	0.000	0.000	0.001
SI:GVT	● 0.000	● 0.000	● 0.000	● 0.000	● 0.000	-0.012	-0.014	-0.009	-0.004	-0.003
SI:NU	● 0.000	● 0.000	● 0.001	○ 0.015	○ 0.017	0.000	0.000	0.000	0.000	0.000
SI:HP	● 0.000	● 0.000	● 0.001	○ 0.054	● 0.000	-0.001	0.000	0.000	0.000	0.001
GVT:NU	0.622	○ 0.026	○ 0.372	0.192	● 0.000	0.000	0.000	0.000	0.000	0.001
GVT:HP	0.544	● 0.000	0.726	○ 0.014	○ 0.519	0.000	0.000	0.000	0.000	0.000
NU:HP	● 0.000	● 0.000	0.832	● 0.004	● 0.000	0.000	0.000	0.000	0.000	0.000
SI:GVT:NU	● 0.000	● 0.003	0.791	0.056	● 0.000	0.000	0.000	0.000	0.000	0.001
SI:GVT:HP	● 0.000	0.145	0.954	● 0.002	0.316	0.001	0.000	0.000	0.000	0.000
SI:NU:HP	● 0.000	0.830	0.773	● 0.000	● 0.000	0.000	0.000	0.000	0.000	0.000
GVT:NU:HP	0.852	0.403	0.573	0.070	0.764	0.000	0.000	0.000	0.000	0.000
SI:GVT:NU:HP	● 0.000	0.564	0.526	○ 0.043	0.397	0.000	0.000	0.000	0.000	0.000

Figure 11: Communication Overhead: Pr(F) and Coefficients

5.2 Performance influence of factor CMB

In a second segment of arguments we want to investigate how the conservative approach (CMB) performed on the same simulation models studied with TW in the analysis above. Looking at the event rates (Figure 7) it is seen that CMB, although able to increase the event rate at a higher number of LPs, cannot catch up with the TW performance. Among the different variants of CMB protocols, the sender initiated nullmessage propagation approach (+SI) seems to have a clear advantage over the receiver initiated -SI version, with contributions to the event rate up to about 1737.3 using 32 LPs. The contribution of the GVT based nullmessage reduction technique (GVT) is negative. Apparently, the computational overhead cannot be compensated by the respective gain for these models, but GVT will definitely be more helpful for models containing low timestamp increment cycles (GVT was designed to break such cycles as early as possible). The sensitivity of all the CMB protocols to model attributes (NU, HP) is comparably small (274.5 for NU and -117.4 for HP at 32 LPs), verifying a general observation for conservative protocols [7] for this data set. The first level interactions show significance mostly when the sender initiated protocol (+SI) is involved, but the contribution to the event rate is of relative importance only. Since an intuitive explanation why +SI would perform better than -SI is that it induces less communication overhead, a view of the data set via a communication related response appears to better explain this conjecture. The tables in Figure 10 and 11 summarize the data set and the result of the variance analysis for the response *mean communication overhead* which represents the fraction of time the respective LP spent in the state of communication with another LP.

Figure 10 explains that comparably high communication overhead is induced by both protocols (ranging from 25 to almost 50 %). Moreover it tells that the previous conjecture does *not* hold: a CMB logical process with +SI spends (slightly) more of its time communicating than the receiver initiated counterpart. Nevertheless, +SI still benefits in terms of event rate (see line +SI in Figure 8). The reason for this is, that CMB with +SI can – due to the confluence with other factors – gain enough on the event rate on the investigated simulation models to still overshadow the slightly higher com-

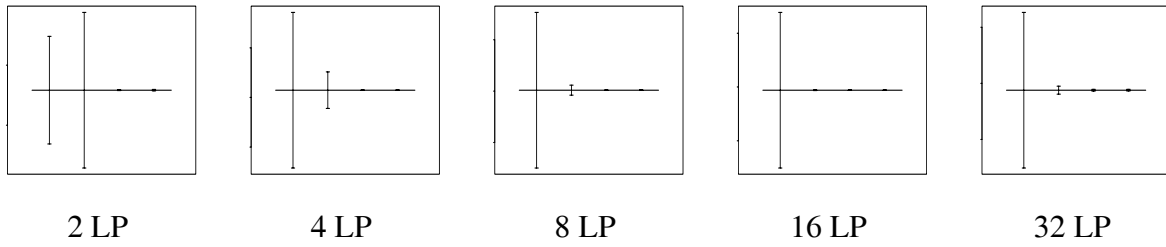


Figure 12: Sample Means of CMB Factors (Communication Overhead)

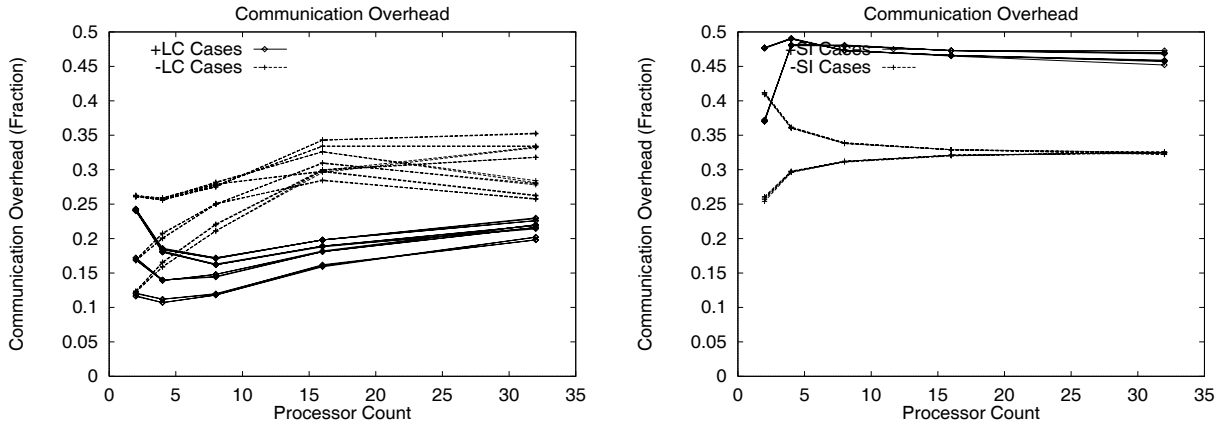


Figure 13: TW vs. CMB: Communication Overhead

munication overhead (looking at the data set via further responses will easily guide the analyst to a deeper understanding of factor coinfluences). The general observation of +SI outperforming -SI, by the way, could also be the reason for receiver initiated conservative protocols being less popular than sender initiated ones. Looking at the effect of the GVT based nullmessage reduction technique (GVT) on the response *communication overhead*, we find that +GVT starts reducing the communication overhead at and after 32 LPs (by only .5 % though). I.e. even in this “noncyclic” simulation model, GVT finds nullmessages which can effectively be absorbed. The first level interaction SI : GVT (as in Figure 11) explains that it makes more sense to use the reduction technique together with sender initiated protocol than with a receiver initiated one (the relative importance of SI and GVT is graphically represented in Figure 12). And also from the communication overhead viewpoint, the variation of the model attributes (NU, HP, Figure 11) has almost no impact.

Overall, the observations around the factor CMB has verified what is generally assumed by distributed simulationists: CMB is less sensitive to the particular structure of simulation models than TW, i.e. appears to be the more “robust” DS scheme. From the F-value’s probabilities and the coefficients of the fitted model in Figure 8 it can be concluded, that CMB does not really interact with the simulation model (NU, HP).

5.3 Performance interaction of the factors TW and CMB

The event rates obtained for TW and CMB (Figure 7) suggest that an optimistic simulation is to be preferred over the conservative one for the simulation models studied in the experiments. An argument supporting this hypothesis is that CMB generally causes higher communication overheads than TW,

	2 LP	4 LP	8 LP	16 LP	32 LP
Case	Pr(F)				
CMB (Intercept)	–	–	–	–	–
NU	• 0.000	• 0.000	• 0.000	• 0.000	• 0.000
HP	• 0.000	• 0.000	• 0.000	• 0.000	• 0.000
TW (Intercept)	–	–	–	–	–
NU	• 0.000	• 0.000	• 0.000	• 0.000	• 0.000
HP	• 0.000	• 0.001	• 0.000	• 0.000	• 0.000
Case	Coefficients				
CMB (Intercept)	1755.486	1571.399	2876.018	5260.535	10073.840
NU	10.872	11.399	25.426	63.445	274.541
HP	-27.601	-27.570	-40.084	-22.773	-117.414
TW (Intercept)	3581.345	6498.233	12333.370	18912.950	23332.040
NU	-210.260	-342.050	-307.352	292.470	1885.607
HP	-11.787	-17.719	-256.353	-598.615	-512.461

Figure 14: TW vs. CMB: Communication Overhead

irrespective of whether the sender or the receiver initiates lookahead propagation via nullmessages. The response *communication overhead* scattered between 25 and 50 % for CMB (Figure 10), while we observed overheads between 11 and 34 % for TW correspondingly.

A closer look at the rates, however, reveals that this argument is *not valid* in general. What has been discussed in the motivation of the paper as TW vs. CMB “in general” *not* being conclusive, is also supported in our experiments: We have already concluded, that CMB is less reactive to the simulation model than TW is; the coefficients for the model attributes NU and HP in Figure 13 and Figure 14 exhibit about the same tendency for TW and CMB for an increasing number of LPs (negative contribution of NU, while positive contribution of HP to the event rate), but in absolute terms, the contributions are larger in the TW case (e.g. +NU contributes 274.5 to the CMB event rate at 32 LPs, while +NU contributes 1885.6 to the TW event rate!).

“The choice of the simulation strategy is simulation model dependent” is an intuitive statement in the standard DS literature, and seems to be the case also for our experiment data set. Opposed to the traditional intuitive arguing, our methodology explains under which circumstances to prefer which strategy: Considering for example a simulation model that is non uniform (+NU), and has a low level of inherent parallelism (–HP), and looking at the 32 LP case, we find that receiver initiated conservative simulators are a bad choice (the achievable event rate is 8909.7 at [–SI –GVT] and 8411.5 at [–SI +GVT]) – this will be outperformed by any configuration of a TW simulator. In all other cases, the choice is not a trivial one. While a sender initiated conservative simulator with the nullmessage reduction technique enabled ([+SI +GVT], event rate 11446.5) outperforms an optimistic simulator with aggressive cancellation, full state saving and optimism control enabled ([–LC –IS +OC], event rate 11424.9), this would no longer be the case if lazy cancellation ([+LC –IS +OC], event rate 37988.6), or incremental state saving ([–LC +IS+OC], event rate 12098.2), or both ([+LC +IS +OC], event rate 40031.9) would be employed in the optimistic simulator. With the optimism control mechanism disabled, the situation becomes even better (for the particular simulation model), and an almost threefold acceleration of the execution is possible (event rate 41063.3 at [+LC +IS –OC]). If a receiver initiated conservative simulator did not use the nullmessage reduction technique for this simulation model ([+SI –GVT], event rate 12373.2) also

some aggressive cancellation optimistic simulators could be outperformed ($[-LC -IS -OC]$, event rate 12315.1) ($[-LC +IS +OC]$, event rate 12098.2), but not all of them ($[-LC +IS -OC]$, event rate 12495.5). In *no way* can a conservative simulator outperform a lazy cancellation TW simulator on this particular simulation model.

Note again that it is due to the factorial design methodology that the co-influences responsible for the non-conclusiveness among TW and CMB could be detected, and that very precise answers could be given with respect to an optimum factor combination. Particularly in the example, for a simulation model $[+NU -HP]$ the optimum DS is TW based with $[+LC +IS -OC]$, yielding the maximum event rate of 41063.3.

6 Conclusions

Performance prediction methods and tools for DS protocols are without any doubt critical for the future success and general acceptance of DS in practice. For a simulationist it is of utmost importance, to be able to evaluate the suitability of a certain DS protocol for a specific simulation task, for a certain multiprocessor system and a certain operational environment *before* substantial programming efforts are invested. A performance prediction methodology and set of tools has been developed (*i*) providing support for strategic decisions in DS projects, and (*ii*) easing performance engineering endeavors of DS protocols from the early design phase in order to avoid late and costly re-engineering. Appropriate levels of abstraction of logical process based DS protocols were introduced for performance prediction and incremental code development. A reasonable set of representative DS protocols was identified from the literature. Among them are conservative protocols with sender or receiver initiated nullmessage protocols and optimizations involving reduction operations, as well as Time Warp related protocols with different message cancellation policies (aggressive, lazy), state saving policies (full, incremental) and optimism control mechanisms. A conditional compile technique has been used to extract one of the DS protocol instances out of a generic skeleton, to preserve performance comparability across protocols. As a demonstration example, we have conducted a 2^k full factorial design upon these performance factors, and were able to uncover nontrivial factor coinfluences. With this approach, besides a performance ranking among the different DS protocols (conservative and optimistic strategies), also a quantification of factor influence becomes possible. Substantial and precise answers for optimum factor combinations could be given, demonstrating “decision support” based on early performance analysis in a convincing way. Both the skeleton and the protocol selection appear as a fundamental and easy to extend code base also for industrial DS projects.

Acknowledgements This work was mostly done while I was with the group of Günter Haring at the University of Vienna, together with Jim Johnson and Stephen J. Turner [11].

References

- [1] I. F. Akyildiz, L. Chen, R. Das, R. M. Fujimoto, and R. F. Serfozo. The effect of memory capacity on time warp performance. *Journal of Parallel and Distributed Computing*, 18(4):411–422, August 1993.

- [2] R. Ayani and B. Berkman. Parallel discrete event simulation on simd computers. *Journal of Parallel and Distributed Computing*, 18(4):501–508, August 1993.
- [3] W. L. Bain and D. S. Scott. An algorithm for time synchronization in distributed discrete event simulation. In B. Unger and D. Jefferson, editors, *Proceedings of the SCS Multiconference on Distributed Simulation*, 19 (3), pages 30–33. SCS, February 1988.
- [4] V. Balakrishnan, R. Radhakrishnan, D. Madhava Rao, N. Abu-Ghazaleh, and P.A. Wilsey. A performance and scalability analysis framework for parallel discrete event simulators. *Simulation Practice and Theory*, 8:529–553, July 2001.
- [5] K. M. Chandy and J. Misra. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering*, SE-5(5):440–452, September 1979.
- [6] S. R. Das and R. M. Fujimoto. An adaptive memory management protocol for time warp parallel simulation. In *Proc. of the 1994 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, Nashville, 1994*, pages 201–210. ACM, 1994.
- [7] A. Ferscha. Parallel and distributed simulation of discrete event systems. In A. Y. Zomaya, editor, *Parallel and Distributed Computing Handbook*, pages 1003 – 1041. McGraw-Hill, 1996.
- [8] A. Ferscha and J. Johnson. N-map: A virtual processor discrete event simulation tool for performance prediction in capse. In *Proceedings of the HICSS–28*, pages 276–285. IEEE Computer Society Press, 1995.
- [9] A. Ferscha and J. Johnson. Performance prototyping of parallel applications in n-map. In *ICA³PP96, Proceedings of the 2nd Internat. Conf. on Algorithms and Architectures for Parallel Processing, June 11-13, 1996, Singapore*, pages 84 – 91. IEEE CS Press, 1996.
- [10] A. Ferscha and J. Johnson. A testbed for parallel simulation performance prediction. In J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain, editors, *Proceedings of the 1996 Winter Simulation Conference*, pages 637 – 644, 1996.
- [11] A. Ferscha, J. Johnson, and St. Turner. Distributed simulation performance data mining. *Future Generation Computer Systems*, 18:157–174, 2001.
- [12] R. M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, October 1990.
- [13] R. M. Fujimoto. Performance of time warp under sythetic workloads. In D. Nicol, editor, *Proc. of the SCS Multiconf. on Distributed Simulation*, pages 23 – 28, 1990.
- [14] R. M. Fujimoto. Parallel discrete event simulation: Will the field survive? *ORSA Journal of Computing*, 5(3):218–230, 1993.
- [15] A. Ferscha G. Chiola. Adaptive distributed simulation of petri net models. In *Proceedings of 1995 Summer Computer Simulation Conference (SCSC '95)*, 1995.
- [16] Bjoern Gehlsen and Bernd Page. A framework for distributed simulation optimization. In *Proc. of Winter Simulation Conference (WSC 2001)*, pages 508–514, 2001.

- [17] A. Gupta, I. Akyildiz, and R. Fujimoto. Performance analysis of time warp with multiple homogeneous processors. *IEEE Transactions on Software Engineering*, 17(10):1013 – 1027, October 1991.
- [18] D. A. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, July 1985.
- [19] L. Lamport. Time, clocks, and the ordering of events in distributed systems. *Communications of the ACM*, 21(7):558 – 565, Jul 1978.
- [20] J. Lemeire and E. Dirckx. Performance factors in parallel discrete event simulation. In *Proc. of the 15th European Simulation Multiconference, Prague*, 2001.
- [21] Jason Yi-Bing Lin. Will parallel simulation come to an end? *Simulation Digest*, 25(3):11 – 12, 1996.
- [22] Y-B. Lin. Will parallel simulation research survive? *ORSA Journal of Computing*, 5(3):236–238, 1993.
- [23] Y-B. Lin, B. Preiss, W. Loucks, and E. Lazowska. Selecting the checkpoint interval in time warp simulation. In R. Bagrodia and D. Jefferson, editors, *Proc. of the 7th Workshop on Parallel and Distributed Simulation*, pages 3–10, Los Alamitos, CA, 1993. IEEE Computer Society Press.
- [24] B. D. Lubachevsky, A. Weiss, and A. Shwartz. An analysis of rollback-based simulation. *ACM Transactions on Modeling and Computer Simulation*, 1(2):154–193, April 1991.
- [25] D. M. Nicol and Ph. Heidelberger. On extending parallelism to serial simulators. In *Proc. of the 9th Workshop on Parallel and Distributed Simulation*, pages 60–67, Los Alamitos, CA, 1995. IEEE Computer Society Press.
- [26] Stephen Turner and M. Xu. Performance evaluation of the bounded time warp algorithm. In *Proceedings of the 6th Workshop on Parallel and Distributed Simulation*, pages 117–128. SCS, January 1992.
- [27] Jinsheng Xu and Moon-Jung Chung. Predicting the performance of synchronous discrete event simulation systems. In *Proceedings of ACM/IEEE International Conference on Computer Aided Design*, pages 18–24, 2001.