Faster Discrete-event Simulation Through Structural Caching[1]

Gianfranco Ciardo
Dept. of Computer Science, College of William & Mary
Williamsburg VA, USA
ciardo@cs.wm.edu

Yingjie Lan
Dept. of Computer Science, College of William & Mary
Williamsburg VA, USA
lanyjie@yahoo.com

Abstract

We develop a structural caching strategy to improve the performance of simulation for a wide class of models expressed in high-level formalisms. By imposing a Kronecker consistent *partition of a model into submodels, we compute once, and cache for future use, the effect of each event on each submodel. This greatly reduces the cost of processing events, updating the current state, and collecting statistics.*

INTRODUCTION

A discrete-event simulation based on a discrete-state model fires the event with the earliest scheduled time among all scheduled events enabled in the current state. As a result, the current state may change, in turn causing some formerly enabled events to become disabled, and some disabled ones to become enabled. The process starts with the initial state, which is normally an input specified by the model itself, and, along the way, it accumulates state-dependent measures, or rewards, which in turn are used to compute statistics such as means and variances. The simulation ends according to various criteria, such as maximum number of events or runtime reached, or desired precision of some measures met.

While the system model can have an enormous or even infinite state space, only a finite but large subset of states is of course explored on any finite run (with some states likely to be visited multiple times). Analogously, the number of possible state-to-state transitions is also large. In practice, this makes it impossible to use a low-level formalism (i.e., one where all states and transitions are explicitly enumerated). For this reason, high-level formalisms, i.e., those where a *global* state is described as a vector of *local* (sub)states, such that events affect some of these components, are quite popular. Examples include Petri nets and queuing models: while their description is compact, they can define complex underlying low-level stochastic processes. A standard simulation engine can then interact with such a high-level model just as with a low-level model, provided we specify a well-defined interface that maps such a *structured* model onto the underlying flat view of the low-level process. Since the underlying state space of the modeled system is still the same, though, the runtime remains large; in fact, it is made even worse by the additional overhead introduced by a high-level model.

In traditional simulation, each state along the simulated path is generated and, once measures are calculated and the next event fired, it is forgotten. This is one major advantage of simulation with respect to analytical methods such as numerical solution of Markov models, which suffer from the curse of state-space explosion. Simulation does not require the storage of the state space although, of course, it is often the case that the larger the state space, the more event firings will be needed to obtain similar statistical accuracy for the output measures. For high-level formalisms, computing the set of enabled events and the state reached by firing one of them in the current state is an expensive operation. The reduction of these costs is the focus of this paper.

In our new strategy, high-level formalisms not only provide a more intuitive and convenient way for modeling, but also present opportunities to improve performance by exploiting the structure of the model with a Kronecker-based caching technique. In the following sections, we briefly introduce *Petri nets* (but our ideas are equally applicable to other structured formalisms such as queuing models) and our concept of *Kronecker-consistent decomposition*, then we present our *caching strategy* and present an *analysis of the performance advantages* exhibited by our technique.

STOCHASTIC PETRI NETS

Stochastic Petri nets (SPNs) are a popular formalism to describe systems, due to their ability to capture complex properties and behavior in a concise and intuitive manner, and to the existence of techniques and tools for their solution. For illustration purposes, we assume SPNs with exponentially-distributed firing times, but, provided analogous structural requirements hold, our approach is applicable even in the presence of general distributions, where simulation is often the only practical solution method. An SPN $(\mathcal{P}, \mathcal{T}, \mathbf{i}^{(0)}, I, O, H, \lambda)$ is a finite, directed, bipartite graph with annotations:

- $\mathcal{P} = \{p_1, \ldots, p_{|\mathcal{P}|}\}$ is a set of *places*.
- $\mathcal{T} = \{t_1, \ldots, t_{|\mathcal{T}|}\}$ is a set of transitions, $\mathcal{P} \cap \mathcal{T} = \emptyset$.
- $\mathbf{i}^{(0)} \in \mathbb{N}^{|\mathcal{P}|}$ is the *initial marking* (a marking, or state, $\mathbf{i} \in \mathbb{N}^{|\mathcal{P}|}$ assigns a number of *tokens* to each place).
- $I : \mathcal{P} \times \mathcal{T} \times \mathbb{N}^{|\mathcal{P}|} \to \mathbb{N}$, $O : \mathcal{P} \times \mathcal{T} \times \mathbb{N}^{|\mathcal{P}|} \to \mathbb{N}$, and $H : \mathcal{P} \times \mathcal{T} \times \mathbb{N}^{|\mathcal{P}|} \to \mathbb{N} \cup \{\infty\}$, are the (state-dependent) cardinalities of the input, output, and inhibitor arcs.
- $\lambda : \mathcal{T} \times \mathbb{N}^{|\mathcal{P}|} \to \mathbb{R}$ is a (state-dependent) transition rate.

A transition t is *enabled* in state $\mathbf{i}$ iff $\forall p \in \mathcal{P}, I(p,t,\mathbf{i}) \leq \mathbf{i}_p < H(p,t,\mathbf{i})$, where $\mathbf{i}_p$ denotes the number of tokens in place p for state $\mathbf{i}$. If t becomes enabled upon entering $\mathbf{i}$ at time θ, its *firing time* τ is sampled from the exponential distribution with parameter $\lambda(t,\mathbf{i})$, and t is scheduled to fire at time $\theta + \tau$. If $\theta + \tau$ is the earliest scheduled firing time, t will be fired, leading the SPN to a new state $\mathbf{j}$ given

[1]Work supported in part by the National Science Foundation under grants CCR-0219745 and ACI-0203971 and by the National Aeronautics and Space Administration under grant NAG-1-02095.

by $\forall p \in \mathcal{P},\ \mathbf{j}_p = \mathbf{i}_p - I(p,t,\mathbf{i}) + O(t,p,\mathbf{i})$, we write $\mathbf{i} \xrightarrow{t} \mathbf{j}$. However, if another transition t' fires at time $\theta' < \theta + \tau$ and changes the state to $\mathbf{i}'$, the *remaining firing time* $\tau - (\theta' - \theta)$ for t is is discarded, if t becomes disabled in $\mathbf{i}'$, or it is scaled[2] by a factor $\lambda(t,\mathbf{i})/\lambda(t,\mathbf{i}')$. Due to the memoryless property of the exponential distribution, this is statistically equivalent to resampling a new firing time for each enabled transition after each firing of any transition; however, this is not only computationally less efficient, but also not generalizable to the non-memoryless case.

The reachability set, or *(global) state space* $\mathcal{S}$ is defined as the smallest set containing $\mathbf{i}^{(0)}$ and such that, if $\mathbf{i} \in \mathcal{S}$ and there exists a transition t such that $\mathbf{i} \xrightarrow{t} \mathbf{j}$, $\mathbf{j} \in \mathcal{S}$ as well. In simulation, we usually visit only a subset $\mathcal{S}^v$ of $\mathcal{S}$. The concept of a *state indexing function* is fundamental to our work. This is a function that maps the already visited states $\mathcal{S}^v$ to the first $|\mathcal{S}^v|$ natural numbers and all other states to null: $\psi : \mathbb{N}^{|\mathcal{P}|} \to \{0, 1, ..., |\mathcal{S}^v| - 1, \mathsf{null}\}$; in practice, states can be assigned increasing indices in the order they are found.

CACHING TO SPEED-UP SIMULATION

One way to speed up discrete-event simulation is to "cache" computations. When visiting state $\mathbf{i}$ for the first time, we assign it an index $\psi(\mathbf{i})$; then, for each enabled transition t, we compute its rate $\lambda(t,\mathbf{i})$ and the identity of the corresponding state $\mathbf{j}$ reached by firing t in $\mathbf{i}$; then, we search for state $\mathbf{j}$ among the already visited portion of the state space $\mathcal{S}^v$ and, if found, we retrieve its index $\psi(\mathbf{j})$; otherwise, we insert it in $\mathcal{S}^v$ and assign it a new index $\psi(\mathbf{j})$; finally, we store this information in entry $\psi(\mathbf{i})$ of vector R_t, which is of size $|\mathcal{S}^v|$: $R_t[\psi(\mathbf{i})] \leftarrow \langle \lambda_t(\mathbf{i}), \psi(\mathbf{j}) \rangle$. Note that this is similar to the algorithm required to build the transition rate matrix for a numerical solution, except that we don't recursively explore $\mathbf{j}$, we explore it only if the simulation actually visits it.

The simulation can then use these $|\mathcal{T}|$ cached vectors R_t by operating on the integer indices $\psi(\mathbf{i})$ instead of the actual states $\mathbf{i}$: if t fires, we go from "state" $\psi(\mathbf{i})$ to "state" $\psi(\mathbf{j})$, and this last quantity, an integer, is obtained with a fast $O(1)$ lookup in $R_t[\psi(\mathbf{i})]$.

Each time a transition fires, we also need to determine which enabled transitions become disabled (their entry must be removed from the event list), which disabled transitions become enabled (a new firing time must be sampled for them and an entry for them must be inserted into the appropriate place of the event list), and which enabled transitions have their remaining firing time scaled (their entry in the event list must be updated and moved to the appropriate place). If we want to exploit the caching idea, we should perform these operations without examining the new state $\mathbf{j}$, but instead use only its index $\psi(\mathbf{j})$. We can use boolean $|\mathcal{T}| \times |\mathcal{T}|$ matrices Δ^d, Δ^e, and Δ^c that (conservatively) specify when the firing of a transition t might disable, enable, or change the rate of a transition u, for $(t,u) \in \mathcal{T}^2$. Then, after the firing of a transition t, we scan the rows $\Delta^d[t,\cdot]$, $\Delta^e[t,\cdot]$, and $\Delta^c[t,\cdot]$, and perform the needed operations on the event list. If we store these three matrices in sparse row-wise format, the number of operations is $O(\eta^d + \eta^e + \eta^c)$, likely to be much smaller than not only $|\mathcal{T}|$ but even of the size of the event list (η^x is the number of nonzero entries in row $\Delta^x[t,\cdot]$). To scale a remaining firing time, we simply multiply it by the rate in the previous state $\mathbf{i}$ and divide it by the rate in the new state $\mathbf{j}$ (for simplicity, the "current rate" can be stored in the corresponding entry of the event list). Finally, a simulation is carried on to compute a set $\mathcal{M}$ of statistical measures on the model. For each measure m, this requires to accumulate the products of the *reward rate* $\rho_m(\mathbf{i}^{(n)})$ of the n^{th} visited state by the length of that visit. Again, we must evaluate the reward function ρ_m on each visited state $\mathbf{i}$ once, and cache its value in the entry $\psi(\mathbf{i})$ of a vector.

This caching approach requires $O((|\mathcal{T}| + |\mathcal{M}|) \cdot |\mathcal{S}^v|)$ memory, too much to be of practical use. Furthermore, while the time spent for the actual simulation is greatly reduced, the time to search each state in $\mathcal{S}^v$ when building the entries of the R_t vectors could be relatively large, e.g., $O(\log |\mathcal{S}^v|)$ if a search tree is used to store $\mathcal{S}^v$; this cost would be amortized only if states are revisited many times, which may not be the case in practice when the state space is large. In the following, we will show how the idea of caching can nevertheless be practical, if implemented in conjunction with a *structural decomposition* of the high level model. First, however, we need to discuss the type of decomposition we require.

KRONECKER CONSISTENT DECOMPOSITION

Given an SPN, *partition* its set of places $\mathcal{P}$ into K mutually exclusive subsets, $\mathcal{P} = \mathcal{P}_K \cup \cdots \cup \mathcal{P}_1$, effectively defining K subnets having common transitions but disjoint *local* substates. Then, a state $\mathbf{i}$ is partitioned as $(\mathbf{i}_K, \ldots, \mathbf{i}_1)$ and the *local state space* $\mathcal{S}_k$ of the k^{th} subnet is the projection of the state space $\mathcal{S}$ onto the places $\mathcal{P}_k$ of the k^{th} subnet. Clearly, the cross-product of the local state spaces, or *potential state space*, $\widehat{\mathcal{S}} = \mathcal{S}_K \times \cdots \times \mathcal{S}_1$ includes $\mathcal{S}$.

A partition is *Kronecker consistent* iff, for each $t \in \mathcal{T}$:

(logical independence of substates)
$\mathbf{i} \xrightarrow{t} \mathbf{j} \wedge \mathbf{i}' \xrightarrow{t} \mathbf{j}' \wedge \mathbf{i}'' \in \{\mathbf{i}_K, \mathbf{i}'_K\} \times \cdots \times \{\mathbf{i}_1, \mathbf{i}'_1\} \Rightarrow \mathbf{i}'' \xrightarrow{t} \mathbf{j}''$, where $\mathbf{j}''_k = \mathbf{j}_k$ if $\mathbf{i}''_k = \mathbf{i}_k$, and $\mathbf{j}''_k = \mathbf{j}'_k$ if $\mathbf{i}''_k = \mathbf{i}'_k$, and

(product-form expression of rates)
there exist functions $\lambda_k : \mathcal{T} \times \mathbb{N}^{|\mathcal{P}_k|} \to \mathbb{R}$, for $K \geq k \geq 1$, such that, $\forall \mathbf{i} \in \widehat{\mathcal{S}}$, $\lambda(t,\mathbf{i}) = \lambda_K(t,\mathbf{i}_K) \cdots \lambda_1(t,\mathbf{i}_1)$.

We have employed the first condition on Kronecker consistent partitions to speed-up symbolic state-space generation. Through K *local indexing functions* $\psi_k : \mathbb{N}^{|\mathcal{P}_k|} \to \{0, 1, ..., |\mathcal{S}_k| - 1, \mathsf{null}\}$, a substate $\mathbf{i}_k$ can be mapped to its index $i_k = \psi_k(\mathbf{i}_k)$, or to null if it has never been seen before. Then, any subset of states in $\widehat{\mathcal{S}}$ or, rather, its indicator function, can be stored as a (multi-way) decision diagram [4]. Instead of also using a decision diagram to encode the *transition relation* specifying the possible state-to-state transitions

[2] A semantic where the remaining firing time is not scaled when the state changes is also possible; this simplifies simulation but complicates numerical solutions.

as traditionally done with symbolic methods, we stored a boolean incidence matrix $\widehat{\mathbf{N}} = \sum_{t\in\mathcal{T}} \bigotimes_{K\geq k\geq 1} \mathbf{N}_{t,k}$, where "$\otimes$" indicates a *Kronecker product* and the $|\mathcal{T}| \cdot K$ boolean incidence matrices $\mathbf{N}_{t,k} \in \{0,1\}^{|\mathcal{S}_k|\times|\mathcal{S}_k|}$ describe the effect of each transition t on each subnet [2] ($\mathbf{N}_{t,k}$ is the identity if the k^{th} subnet does not affect the enabling of t nor is affected by its firing, i.e., if the two are *independent*). This idea was initially developed in [5] to encode the *transition rate matrix* $\mathbf{R}$ of a *stochastic automata network* defining an underlying continuous-time Markov chain (CTMC). The additional second condition on the rates for a Kronecker consistent partition allows us to write $\mathbf{R}$ as the sub-matrix corresponding to $\mathcal{S}$ of the (real) Kronecker expression $\widehat{\mathbf{R}} = \sum_{t\in\mathcal{T}} \bigotimes_{K\geq k\geq 1} \mathbf{R}_{t,k}$, where $\mathbf{R}_{t,k}$ is a real matrix capturing the values of $\overline{\lambda}_k(t,\cdot)$. Both [2] and [5] assumed that each local state space $\mathcal{S}_k$ can be built, stored, and indexed in isolation, prior to performing any type of analysis. Simulation, however, may be applied when the state space $\mathcal{S}$ is too large even for symbolic methods, or infinite. In [3], we demonstrated an interleaving algorithm where the local state spaces $\mathcal{S}_k$ and the global state space $\mathcal{S}$ are built at the same time, alongside the Kronecker matrices $\mathbf{N}_{t,k}$. The key idea is to classify the *potentially reachable local states* $\widehat{\mathcal{S}}_k$ as *confirmed* or *unconfirmed*. The symbolic global state space generation operates on *structured states* i.e., vector of K integer indices $(i_K, ..., i_1)$ of the K substates $(\mathbf{i}_K, ...\mathbf{i}_1)$ forming a global state $\mathbf{i}$, while the K local state space generations operate on substates $\mathbf{i}_k$, $K \geq k \geq 1$. During execution, a local state $\mathbf{i}_k \in \mathcal{S}_k$ is classified as confirmed if a reachable global state $(i_K, ..., i_1)$ has been previously found such that $\psi(\mathbf{i}_k) = i_k$. When an unconfirmed $\mathbf{i}_k$ is determined to be part of a globally reachable state, it is confirmed by immediately building the corresponding row $i_k = \psi(\mathbf{i}_k)$ of each matrix $\mathbf{N}_{t,k}$ (of course, only for those transitions t that depend on the k^{th} subnet). Since it is possible that $\mathbf{i}_k \xrightarrow{t} \mathbf{j}_k$ in isolation, but, at least so far, no global state with k^{th} component equal to $\psi(\mathbf{j}_k)$ has been found, entries in these rows might refer to an unconfirmed local state $\mathbf{j}_k \in \widehat{\mathcal{S}}_k \setminus \mathcal{S}_k$, or, rather, to its index $j_k = \psi(\mathbf{j}_k)$.

STRUCTURAL CACHING

We now discuss how, adapting the idea of [3], we can achieve the same caching effect with small memory requirements and, at the same time, greatly improve the amortization of the time spent for state lookups to compute their indices. As in [3], we will build the sets $\mathcal{S}_k^v$, where "v" stands for "visited", and the corresponding portions of the matrices $\mathbf{R}_{t,k}$, on-the-fly. Our goal is to perform the most frequent simulation computations on the structured state (a vector of K integer indices) instead of on the *model state* (a vector of $|\mathcal{P}|$ integers). Only the generation of the entries in the rows of the Kronecker matrices $\mathbf{R}_{t,k}$ requires "going all the way to the SPN model" and examining the composition of actual substates, i.e., vectors of $|\mathcal{P}_k|$ integers (which are anyway smaller than actual states, i.e., vectors of $|\mathcal{P}|$ integers).

The total storage requirements for the matrices $\mathbf{R}_{t,k}$, are just $O(|\mathcal{T}| \cdot (|\mathcal{S}_K^v| + \cdots + |\mathcal{S}_1^v|))$, since each row contains at most one entry and they can be stored as vectors; this is a logarithmic reduction with respect to $O(|\mathcal{T}| \cdot |\mathcal{S}^v|)$. Analogously, the cost of searching for a state $\mathbf{j}$ in the (global) state space $\mathcal{S}^v$ is now replaced by that of searching each $\mathbf{j}_k$ in the local state spaces $\mathcal{S}_k$. However, while we need to search $O(|\mathcal{T}| \cdot |\mathcal{S}^v|)$ states in the unstructured approach, we only need to search $O(|\mathcal{T}| \cdot (|\mathcal{S}_K^v| + \cdots + |\mathcal{S}_1^v|))$ local states in our structured approach, again a logarithmic reduction (furthermore, the searches are performed on smaller sets). This means that the search cost in our approach is amortized even if no global state is ever revisited, as long as local substates are revisited, no matter in what combination of global states.

A further improvement in time and memory is achieved by considering the *locality* of each SPN transition. When computing the (structured) state $(j_K, ..., j_1)$ reached when transition t fires in (structured) state $(i_K, ...i_1)$, we only need to look-up and update as many entries as the number of submodels that depend on t, usually much fewer than K (recall that, if t and the k^{th} submodel are independent, $\mathbf{R}_{t,k} = \mathbf{I}$ and is never used nor stored). To exploit locality, we set up a sparse data structure prior to starting the simulation, based on the connectivity of the SPN graph and the identities of the places appearing in the expressions for the transition rates.

The same matrices Δ^d, Δ^e, and Δ^c we discussed for the unstructured case are still applicable to our approach. If the rate of transition t depends on $L \leq K$ submodels, we need to perform $O(L)$ multiplications to obtain $\lambda(t, \mathbf{i})$, any time t becomes enabled upon entering $\mathbf{i}$. In the unstructured cached approach (which is, however, usually infeasible in practice due to its excessive memory requirements), the evaluation of the expression for $\lambda(t, \mathbf{i})$ would require at least $O(L)$ operations, but only the first time $\mathbf{i}$ is encountered, since a cache lookup would be used after that. If $\mathbf{i}$ is entered many times from states where t is disabled, our approach could spend L times more for these evaluations. Compared to a traditional (unstructured, uncached) simulation, however, it would still be more efficient, since L is small in practice.

Furthermore, when the rate of transition t needs to be scaled, we can exploit structural information even more. As an example, if the rate of t is the product of $L \leq K$ values obtained from the matrices $\mathbf{R}_{t,k}$, if the state changes from $\mathbf{i}$ to $\mathbf{i}'$ because of the firing of transition u, if t is enabled in both $\mathbf{i}$ and $\mathbf{i}'$, if $\Delta^c[u,t] = 1$, and if u changes only the k^{th} component, where the k^{th} subnet is one of the L affecting the values of the rate of t, then we can obtain the new value of $\lambda(t, \mathbf{i}')$ in $O(1)$ operations, as $\lambda(t, \mathbf{i}) \cdot \mathbf{R}_{t,k}[i'_k] / \mathbf{R}_{t,k}[i_k]$.

The computation of the reward rate in each state must also be performed in a structured fashion. In the common case where $\rho_m(\mathbf{i})$ if of the form $\sum_{K\geq k\geq 1} \rho_{m,k}(\mathbf{i}_k)$ or $\prod_{K\geq k\geq 1} \rho_{m,k}(\mathbf{i}_k)$, an analogous reasoning as for the rate computation applies: we only need to cache the value of $\rho_{m,k}(i_k)$ for each measure m, submodel k, and index $i_k = \psi(\mathbf{i}_k)$ of each local state in $\mathcal{S}^v$. In the general case, the expression for a measure ρ_m must instead be separated into *local sub-expressions* depend-

ing on a single local state $\mathbf{i}_k$ (multiple sub-expressions can refer to the same submodel k, while, on the other hand, some submodel might not be referenced at all). Then, a parse tree of the expression, where the basic elements are these sub-expressions, can be evaluated in the initial state and, as the model evolves, only portions of the parse tree must be re-evaluated, using cached values for the sub-expressions. Assume the reward rate $\rho_m = ((a_1 \cdot b_2) + (c_1 \cdot (d_3 + e_4)))$, where the subscripts indicate the submodel on which the sub-expression depends. If t fires in state $\mathbf{i}$, for which we have already computed the value of $\rho_m(\mathbf{i})$, and it changes only $\mathbf{i}_2$ into $\mathbf{j}_2$, we obtain $\rho_m(\mathbf{j})$ by retrieving the value of $b_2(\psi(\mathbf{j}_2))$ from the measure cache (or computing it from scratch if it is the first time we visit local state $\mathbf{j}_2$) and recomputing the value of the product $x = a_1 \cdot b_2$ (where the value of a_1 is unchanged) and of the sum $x + y$ (where the value of $y = (c_1 \cdot (d_3 + e_4))$ is unchanged).

PERFORMANCE ANALYSIS AND RESULTS

If, on average, m subnets depend on each transition and each $\mathcal{S}_k$ has s states, we need $m \cdot s$ cache entries for each transition and, in total, the entire size of the cache is $m \cdot s \cdot |\mathcal{T}|$ entries. As previously mentioned, the state space $\mathcal{S}$ is a subset of the potential state space $\widehat{\mathcal{S}} = \mathcal{S}_K \times \cdots \times \mathcal{S}_1$. If we assume that each $\mathcal{S}_k$ has the same cardinality, $s = O(|\widehat{\mathcal{S}}|^{1/K})$, where K is the number of partitions. If we define the *looseness* of a partition as $\ell = |\widehat{\mathcal{S}}|/|\mathcal{S}|$, then $|\widehat{\mathcal{S}}| = \ell \cdot |\mathcal{S}|$ and it follows that $s = O((\ell \cdot |\mathcal{S}|)^{1/K})$. This is usually extremely small in comparison to $|\mathcal{S}|$, also because ℓ can be large only when K is (in the limit, if $K = 1$, $\widehat{\mathcal{S}} = \mathcal{S}$ and $\ell = 1$).

In common simulation applications, the number N of events fired, thus the number of states calculated from the model, is usually proportional to the cardinality of the state space. If we assume that calculating a new local or global state directly from the model costs L and $G \approx L \cdot m$, respectively, and that accessing a local cache costs A, building our cache costs $m \cdot s \cdot |\mathcal{T}| \cdot L$ and so the ratio of time savings is

$$\frac{N \cdot G}{m \cdot s \cdot |\mathcal{T}| \cdot L + N \cdot m \cdot A} = \frac{N \cdot L}{s \cdot |\mathcal{T}| \cdot L + N \cdot A}.$$

For long simulations, the terms containing N dominate the above expression, and the ratio is close to L/A. In practice, A is just the cost of one direct access in a vector, while L is the cost of processing the effect of transition on a subset of places $\mathcal{P}_k$, so it is substantially larger.

Analogous reasoning leads us to conclude the the time saving ratio for event checking and measure computation is close to the cost ratio between performing these operations directly on the model vs. using the direct access provided by the cache. Thus, the entire simulation process can benefit from our structured caching strategy.

The following table shows the improvements achieved using our technique on three models: a flexible manufacturing system, an elevator, and the Aloha protocol. For each, we show the overall runtime and its "improvable" portion (state and event manipulation but not, e.g., random number generation), for both the traditional and the structural approaches. For the FMS, event processing is inexpensive, thus the traditional approach is slightly better. However, for the elevator and Aloha models, our caching strategy achieves substantial speedups: more than 4 and 6 overall, due to speedups in the improvable portion of almost 20 and 120, respectively.

Model	Traditional		Structural		Speedup	
	Overall	Impr.	Overall	Impr.	Overall	Impr.
FMS	28.41	3.91	30.47	5.97	0.93	0.65
Elevator	21.56	17.50	4.97	0.91	4.34	19.23
ALOHA	437.07	371.42	68.84	3.19	6.35	116.43

IMPLEMENTATION IN SMART

Our simulation engine in SMART [1] interacts with arbitrary discrete-state models expressed in a variety of formalisms through a well-defined interface. Our cache implements that same interface and intercepts calls from the simulation engine to the original model. Thus, when the caching option is on, the SMART simulation engine only interacts with the cache, while only the cache interacts with the original model.

While our presentation so far has been limited to the case of exponentially-distributed events, our implementation in SMART is more general. The caching idea is applicable to the general case, as long as the parameters for the distribution can be expressed as a function of "local functions" evaluated on local substates. In particular, the efficient scaling of the remaining firing times we exemplified in the previous section remains possible as long as the firing time distribution is characterized by an average expressed as the product of (up to) K factors, each depending on a different submodel. Of course, a no-scaling semantic is even simpler to implement.

REFERENCES

[1] G. Ciardo, R. L. Jones, III, A. S. Miner, and R. Siminiceaunu. Logical and stochastic modeling with SMART. *TOOLS*, Urbana-Champaign, IL, USA, Sept. 2003. Springer. To appear.

[2] G. Ciardo, G. Luettgen, and R. Siminiceanu. Saturation: An efficient iteration strategy for symbolic state space generation. *TACAS*, LNCS 2031, pages 328–342, Genova, Italy, Apr. 2001. Springer.

[3] G. Ciardo, R. Marmorstein, and R. Siminiceanu. Saturation unbound. *TACAS*, LNCS 2619, pages 379–393, Warsaw, Poland, Apr. 2003. Springer.

[4] T. Kam, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli. Multi-valued decision diagrams: theory and applications. *Multiple-Valued Logic*, 4(1–2):9–62, 1998.

[5] B. Plateau. On the stochastic structure of parallelism and synchronisation models for distributed algorithms. *SIGMETRICS*, pages 147–153, Austin, TX, USA, May 1985.