identical to the starting state of the next PE, or it can be any of the states from which transitions are allowed to the latter. A more restrictive condition applied in [5] is to pick the single most likely state j^* as the starting state for all the processors, in which case the end-states are always sewn to starting-states without transition.

As before, storing the system path can be avoided if we further impose the condition that the buffer must empty before sewing in. In this case, the set of possible starting states is further restricted, since the buffer cannot be empty if the source rate exceeds the trunk capacity. Thus, we need only consider the "under-load" states, which satisfy $R(j^*) \leq c$. Again, in the case of low loss probabilities, we found that this additional condition is easily met during the simulation, leading to considerable savings in time and memory.

The above arguments can also be extended to the case of statistical multiplexing of several sources of this type through a single buffered trunk.

4 Conclusion

We have presented our experiences with massive parallel simulations of specific ATM systems, an endeavor in which few general methods are in existence. Certain problematic issues specific to massive parallelism and the SIMD nature of our simulations have been discussed, with possible recipes to overcome them. Further research aims at generalizing these techniques to wider classes of communication and queueing systems.

References

[1] D. Anick, D. Mitra, and M. M. Sondhi. Stochastic theory of a data handling system with multiple sources. *Bell Systems Technical Journal*, 61:1871–1894, 1982.

[2] K. M. Chandy and R. Sherman. Space-time simulation. *Proceedings of the 1989 SCS Multiconference on Distributed Simulation*, pages 53–57, March 1989.

[3] Anwar Elwalid, Debasis Mitra, and Robert H. Wentworth. A new approach for allocating buffers and bandwidth to heterogenous, regulated traffic in an atm node. *IEEE Journal on Selected Areas in Communications*, 13(6):1115–1127, Aug. 1995.

[4] R. M. Fujimoto. Parallel discrete event simulation. *Commun. ACM*, 33(10):31–53, Oct. 1990.

[5] Richard M. Fujimoto, C. Anthony Cooper, and Ioanis Nikolaidis. Parallel simulation of statistical multiplexers. *32nd IEEE Conference on Decision and Control*, 1993.

[6] A. G. Greenberg, B. D. Lubachevsky, and I. Mitrani. Unboundedly parallel simulations via recurrence relations. In *Proceedings of the Conference on Measurement and Modeling of Computer Systems.*, pages 1–12, Boulder, Colorado, May 1990.

[7] F. J. Kaudel. A literature survey on distributed discrete event simulation. *Simuletter.*, 18(2):11–21, June 1987.

[8] Y.B Lin and E. A Lazowska. Time-division algorithm for parallel simulation. *ACM TOMACS*, 1(1):73–83, Jan. 1991.

$$\begin{aligned} e_i^p &\rightarrow e_{e_i^p}^{(p+q)} \\ q &\rightarrow 2 * q \end{aligned}$$

The advantage of using the zero-buffer regeneration point is that it is not necessary to store the entire history of the source process. But the disadvantage is that, this condition may or may not occur often, depending on the system parameters, namely the source rates and trunk capacity. In the case of systems designed to allow only very low probabilities of buffer violation ($\approx 10^{-9}$) the buffer is empty for a sizeable fraction of time and this regeneration point is easily reached. In our simulations, we observed that this remarkably simple procedure worked rather well for systems with small n for which the regeneration condition was reached reasonably often. A discrete time version of this simulation is discussed in detail by Fujimoto et al [5], but, in their case, step 2 is not an inefficiency as they have far fewer, but much faster, processors. Hence the PEs that finish early do not need to wait for the slower on account of the MIMD discipline in [5].

3.1.3 End State Matching

When n is large, it is no longer economical for each PE to simulate n distinct paths, of which only one is finally used. We then have to confront the complete challenge of end-state matching. This would involve simulating just one system path from each PE, starting from a chosen initial state, until a designated final state is reached. It is hence important to choose these initial states appropriately so that the simulation is efficient as well as accurate. Since the algorithm in this case is straightforward, it suffices to address the issue of how to choose the distribution of initial states among the PEs.

Thus, we must now consider *full regeneration* of the system, as opposed to the partial regeneration of the previous section. As before, one can avoid storing the entire history of the system when possible, by computing the desired statistics *on line*. This is possible if, for example, the desired statistics are *local* and the regeneration is frequent.

Distribution of initial states : We now discuss heuristic methods to choose a good distribution of initial states in the context of this Markov source system. Here it is important to have some theoretical or experimentally determined idea of the steady state distribution of the system. In particular, we would like to assign initial random initial states to each PE from a subset of states that are most often revisited, which must be determined before-hand either analytically or through a pilot simulation.

Consider, for example, the aggregate output from n on/off sources, each with a probability $p_o n$ of being in the "on" state. This can be modeled as a single Markov source with $n+1$ states, with the following steady state distribution of states.

$$P(j) = \frac{n!}{j!(n-j)!} p_o n^j (1 - p_o n)^{n-j} \tag{13}$$

Among the set of possible states $\{j : j \in [0, n]\}$, we can choose the subset of states $\{j^* : P(j^*) \geq P_{min}\}$, from which the starting states can be chosen with the same distribution. In this case, each PE would simulate a large number of source transitions and proceed till its end-state can be sewn with the starting state of its neighbor in the future. In the case of this Markov source, the starting state of the neighbour can be j^* or $j^* \pm 1$. In a general case, the ending state of each PE can be

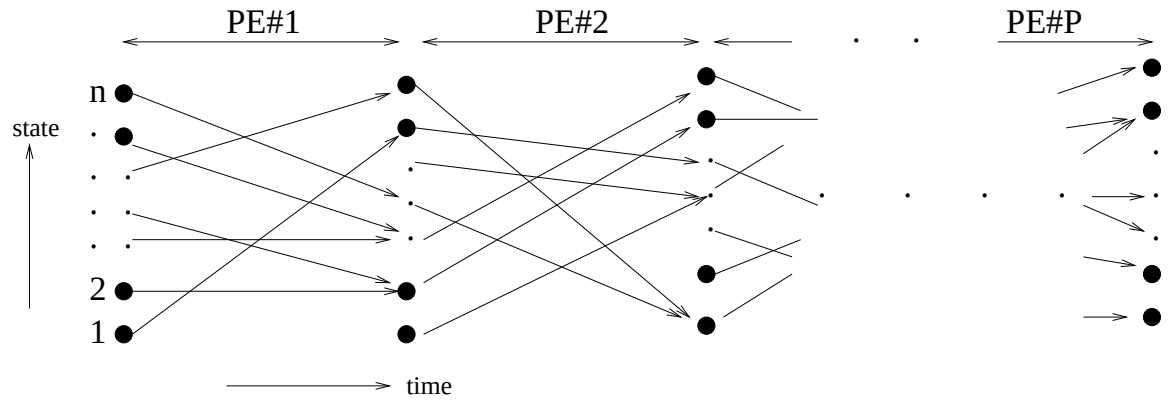

Figure 6: Schematic representation of the simulation required for the parallel prefix method.

3.1.2 Regenerative Simulation

We now adapt the method described above to take advantage of *partial regeneration points* of the system. In particular, we consider the times at which the buffer empties, i.e $B = 0$. It can be shown that these points on the systems state-path are *partial regeneration points*, i.e, given the long range stability of the system, this condition occurs infinitely. It would hence be desirable to have each PE start and end at such a regeneration point, while independently computing the desired statistics, such as buffer violation intervals, waiting times, sojourn and state statistics etc. In our case, it turned out that the statistics of interest were purely *local*, i.e. the computations necessary involved only the current and immediate past state of the system, owing to the Markovian property. This fact eliminates the need to store the entire history of the system between regeneration points, and thus enables much longer simulations in a single shot.

All of the above is contingent on the frequency of partial regeneration, which depends on the system parameters - the source rates, the trunk capacity and the buffer size. Here we were lucky, as we were often dealing with systems in which the loss probabilities were expected to be very small, and hence the buffer was empty most of the time. This contributed to the success of the procedure. In addition, we required long enough paths to be able to "amortize" the overhead caused by different PEs reaching their respective regeneration points after different amounts of computation.

1. As before, on each PE we simulate n system paths, each with a distinct starting state and empty buffer. Each path consists of a fixed, but large (say K), number of source transitions while simultaneously computing desired statistics (including the buffer, from eq.10).

2. After these K transitions, each PE inspects the buffer value and continues only paths with non-empty buffers till empty buffer is reached.

3. The statistics of interest, denoted by ψ, can be accumulated along each of the n long simulations using parallel-prefix as follows.

$$\psi_i^p \quad \rightarrow \quad \psi_i^p + \psi_{e_i^p}^{(p+q)}$$

$$B^p_{k+1} \quad = \quad max(0, B^p_k + a^p_{k+1}) \tag{10}$$

With the redefinition $A + B \equiv max(A, B)$ and $AB \equiv A + B$, the above equations can be expressed in the following matrix form.

$$\begin{pmatrix} B^p_{k+1} \\ 0 \end{pmatrix} = \begin{pmatrix} a^p_{k+1} & 0 \\ -\infty & 0 \end{pmatrix} \begin{pmatrix} B^p_{k+1} \\ 0 \end{pmatrix} \tag{11}$$

or, simply $\mathcal{B}^p_{k+1} = \mathcal{M}^p_k \mathcal{B}^p_k$, which can be written recursively as

$$\mathcal{B}^p_{k+1} = \{\prod_{k=1}^{K} \mathcal{M}^p_k\} \mathcal{B}^p_0 \tag{12}$$

Thus, each PE independently computes, using it's set of matrices $\mathcal{M}$, the set of partial products

$$\mathcal{N}^p_k = \prod_{r=1}^{k} \mathcal{M}^p_r.$$

This set of partial products is then independently accumulated by each PE by the parallel prefix method. In other words, each PE performs the following of iterative operation starting with $q = 1$ for $k \in [1, K]$.

$$\begin{aligned} \mathcal{N}^p_k \quad &\rightarrow \quad \mathcal{N}^p_k \mathcal{N}^{(p-q)}_K \\ q \quad &\rightarrow \quad 2 * q \end{aligned}$$

This procedure hence generates the set of partial products $\mathcal{N}$ for each event of the entire "sewn" path, thus enabling the evaluation of the buffer at any instant along the path, and from it the buffer violation time as described in 2.

4. Carry out above steps starting with 0 initial buffer, generating as many transitions as allowed by the memory requirements to store the state and sojourn time arrays. This constitutes one iteration of the algorithm. Compute final buffer and repeat above steps with this as initial buffer value to proceed with second iteration.

   This procedure can be iterated till the desired simulation path length is reached.

Note that this method requires storage for the states and sojourn times of the path during each iteration, and also involves the overhead of generating n paths while only one is finally used. Nevertheless, it is guaranteed to produce a single long simulation quite efficiently if sufficient memory is available in each PE and n is small. Even if the memory per PE is small this does not prevent long simulations, owing to massive parallelism.

This source feeds into a buffer, whose variation within each sojourn time is linear, except for a lower bound of 0 (assuming the infinite buffer approximation).

$$B^{new} = max(0,\ B^{old} + r_i t_i^s)$$

where r_i is the source output in state i. Thus, it is straightforward to simulate the above system serially, using appropriately distributed random numbers to generate transitions and sojourn times.

We now consider the time-parallel simulation of this system on an SIMD parallel computer with P Processing Elements(PEs).

3.1.1 Parallel-Prefix Method

The parallel prefix methods developed in [6] can be used to simulate this system when n, the number of possible source states, is small. On each processor $PE\#p : p \in [1, P]$, one can generate independent sample paths of the system starting from each possible initial state i to some ending state e_i^p (see figure 6). These sample paths can then be sewn end-to-end to form exactly P long paths, one from each starting state of $PE\#1$. Some of these long paths could have large segments in common, since sample paths starting from distinct initial states in each processor could have the same ending state. Thus, paths starting at distinct initial states on $PE\#1$ successively merge, after which they never separate, and are hence not statistically independent. Therefore, at the end of the simulation, only one randomly chosen path among the P generated ones can be considered for the computation of desired statistics. This overhead demands that the number of states n be small for the feasibility of the method. One can alleviate this problem to some extent by ignoring correlations between the n paths generated by individual PEs, since only one of them will finally be used. Thus, a single pair of random numbers can be used to generate all the n state transitions and sojourn times in each PE, which offers some time efficiency.

Thus, the algorithm consists of the following steps.

1. On each PE, generate n sample paths, one from each of possible starting state, each consisting of a fixed (say K) number of state transitions.

   Each transition requires a pair of random numbers - one to determine the destination state and the other for the sojourn time. This pair is used to advance each of the n paths by one transition. These values are stored in the arrays $\{s_{i,k}^p : (i, s_{i,k}^p) \in [1, n], p \in [1, P], k \in [1, K]\}$ and $\{\tau_{i,k}^p : \tau_{i,k}^p \in R_+, p \in [1, P], i \in [1, n], k \in [1, K]\}$.

2. Select an random initial state $\imath^1$ at $PE\#1$ determine the set $\{i^p : i^p \in [1, n], p \in [0, P]\}$ which constitute a single connected path starting with $\imath^1$ at $PE\#1$.

   This process takes a small amount of time even if done serially for a large P, and can be done using $i^{p+1} = e_i^p$.

3. For this path, the buffer variation can be computed using parallel-prefix as follows, given the inital buffer B_0.

   Denoting by $R(s)$ the rate of fluid generation of the source in state s, and by $a_k^p = R(s_{i^p,k}^p)\tau_{i^p,k}^p$ the total arrival in the kth inter-event interval of $PE\#p$, we can write the following recursive equation for the buffer variation.

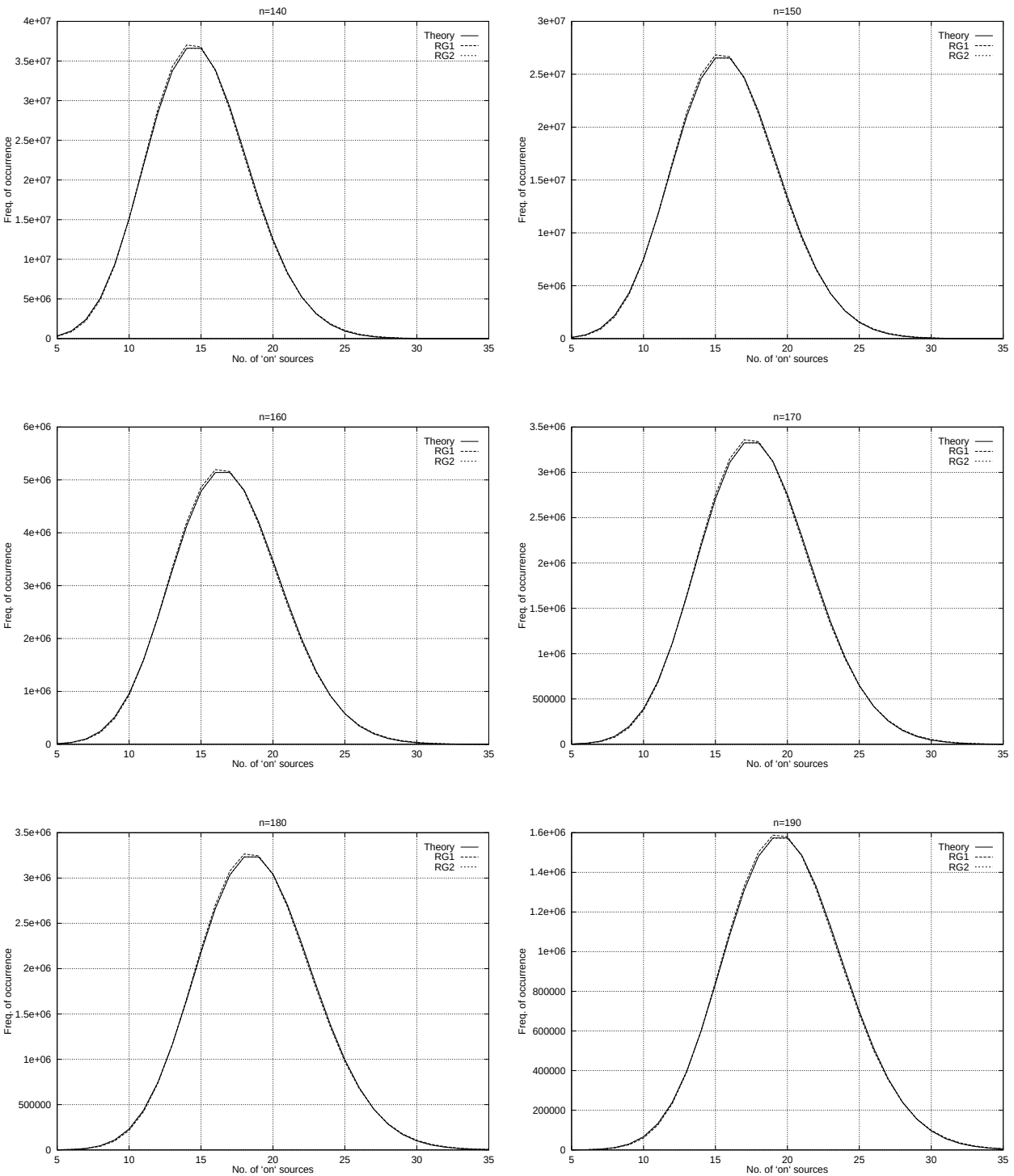

Figure 5: Comparison of theoretical and experimental (RG1 and RG2) source distibutions at the start of the period Note that RG2 is virtually identical to the theoretical prediction.

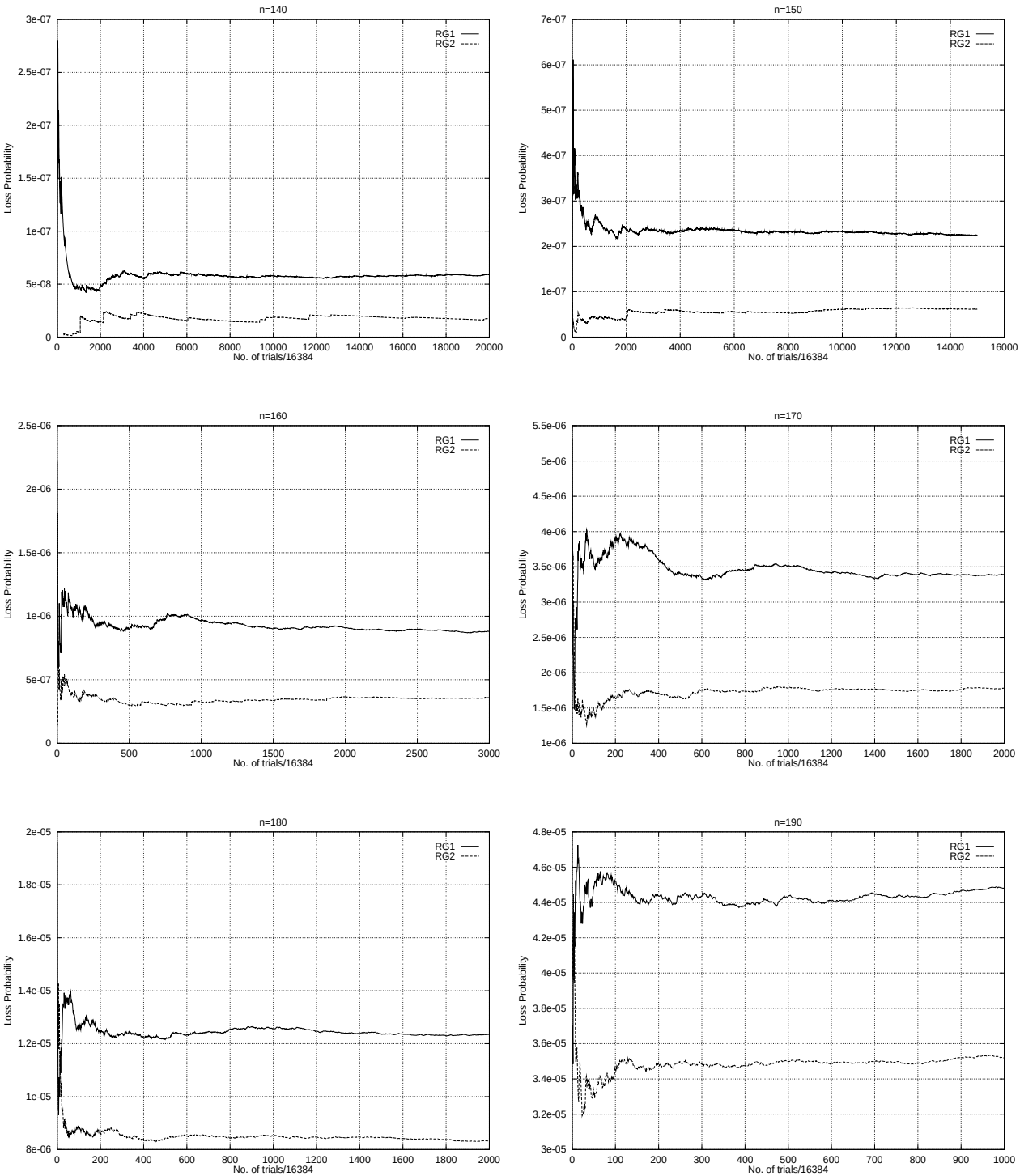

Figure 4: Loss Probabilities vs. Number of source configurations. RG1 and RG2 clearly converge to different values.

P_l	5.777×10^{-8}	2.510×10^{-7}	8.942×10^{-7}	3.412×10^{-6}	1.236×10^{-5}	4.445×10^{-5}
n	140	150	160	170	180	190

Table 1: Loss probabilities P_l vs. Number of sources n, using random number generator $RG1$.

P_l	1.892×10^{-8}	5.563×10^{-8}	3.589×10^{-7}	1.779×10^{-6}	8.324×10^{-6}	3.519×10^{-5}
n	140	150	160	170	180	190

Table 2: Loss probabilities P_l vs. Number of sources n, using random number generator $RG2$.

3 Time-Parallel Simulation of an nM/D/1 ATM system

We now consider the simulation of ATM systems with Markovian sources statistically multiplexed through a single trunk providing deterministic service. Specifically, we adopt the formulation presented in [1]. In this case, the objective is to generate a single long simulation of the system by "sewing" together sub-paths generated using time-parallelism techniques [6].

We point out that massive parallelism requires consideration of several aspects of the simulation that are not of importance when only a few processors are used, as in [5].

3.1 Single Markov source: Representative Example

For illustration, consider the case of n identical on-off sources. Following the treatment in [1], this aggregate process from these sources can be modeled as a Markov chain with $n+1$ states in the fluid approximation. Assuming exponentially distributed sojourn times in on and off states with means α and β respectively, the probabilities of transition to neighboring states, from a state with i sources on, in an interval δt follows the distribution

$$\begin{aligned} P_{i,i+1} &\propto (n-i)\ \alpha\ \delta t \\ P_{i,i-1} &\propto i\ \beta\ \delta t \end{aligned} \tag{8}$$

while the average sojourn times are given by the inverse of the exit probabilities from the state

$$\bar{t}_i^s = \frac{1}{(n-i)\alpha + i\beta} \tag{9}$$

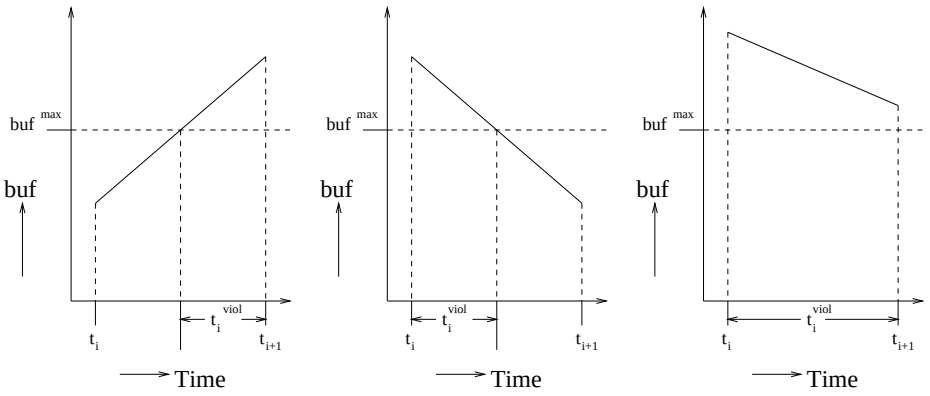

Figure 3: Schematic representation of the computation of t^{viol}.

Figure 4 depicts the variation of the loss probability versus number of configurations simulated for various values of n for two different random number distributions, while table 1 and table 2 list the final values. The generator RG1 was the manufacturer supplied MasPar parallel random number generator used directly with the sorting method described above, while RG2 used the MasPar generator to generate a pre-sorted ditribution using equation 1. As observed from the graph the loss probabilities undergo initial transient behaviour and asymptotically converge to fixed values in each case with increasing number of trials. The number of trials required for convergence is of the order of the inverse of the loss probability, which reduces monotonically, but non-linearly, as n is reduced.

Although convergence is always achieved, observe that the two different methods converge to different values in each case! In fact, RG1 consistently gives a higher loss probability compared to RG2. It also turned out that the results of RG1 were in better agreement with the conservative theoretical estimates presented in [3], which we were inclined to beleive.

We hence needed a measure to compare the performances of various methods for this simulation.

Statistical Comparison : In order to compare the relative statistical accuracies of these two conflicting results, we considered the following measure. The distribution of the number of sources that are "on" at any given instant, say the beginning of our chosen period, is binomial, and is given by

$$P_{n_{on}} = \frac{(n-1)!}{n_{on}!\,(n-n_{on})!}\ (\frac{t_{on}}{T})^{n_{on}}\ (\frac{T-t_{on}}{T})^{n-1-n_{on}} \tag{7}$$

Thus, when the distributions generated disagree with this theoretical prediction, they are faulty, while agreement does not imply correctness. Nevertheless, we argue that, this first order statistic bears significant effect on the loss probability. We compared the experimental distributions obtained in each method to this theoretical prediction, and the results are depicted in figure 5. RG1 deviates slightly from the expected distribution and is hence the worse of the two. It appears, from this criterion, that the second set of results, based on RG2, is more accurate.

This method is of complexity $O(n)$, although the multiplying constant could be large due to the algebraic operations of powers and roots. It was found that the two methods took about the same amount of time on the MasPar 2.1 system.

Computation of Loss Probabilities :

After generating the event times and source rates, we proceed to compute the buffer violation intervals t^{viol} for each phase configuration, and the loss probability P_l which is defined as

$$P_l = Avg(\frac{t^{viol}}{T}) \tag{6}$$

where the averaging in $Avg()$ is over the set of phase configurations. It is important that these values be computed only after the buffer variation reaches steady state. It can be shown that if one starts with 0 initial buffer, for a periodic source rate, steady state is reached after one time period. Thus our computation of the violation interval pertains to the second time period after the start of the simulation. The violation intervals are computed in parallel by each processor, in the following steps.

1. Note that within each time bin, the buffer variation is linear. Specifically,

   $$buf_{i+1} = max(0, buf_i + r_i(t_{i+1} - t_i)).$$

   Thus we start with 0 initial buffer and calculate the buffer at the end of the first period (B^{init}).

2. We now repeat the previous step for the second period starting with (B^{init}). Additionally, for each bin, we compute the length of time for which the buffer exceeded the limit B^{max}, which is successively accumulated in the variable t^{viol}. Figure 3 depicts this computation schematically.

3. We add the values of t^{viol} from each processor and repeat the above procedure, while accumulating the t^{viol} values after each run. Finally, the loss probability at the end of k such iterations is estimated as

   $$P_l = \frac{\Sigma(t^{viol})}{k.n_p.T}$$

2.1 Results

In the following simulations, which were carried out on the MasPar system mentioned before, the speed-up obtained almost the maximum possible, with negligible overhead. This is not surprising, since it is expected of replica-parallel simulations, where processors perform identical tasks with no communication with other processors except at the very end of the simulation. Since the complexity of each sample path is determined by the number of event times, which is twice the number of sources, the simulation time varies slightly with n, but the bulk of the simulation time is used to generate ordered random number sets.

generates n_p phase configurations in parallel. This method would be of complexity $O(n \log n)$ in the number of sources.

Alternatively, one can avoid the sorting step and generate a pre-sorted set of phases, with some additional computation, as follows. Given a set of n independent uniformly distributed random numbers
$\{x_i : 1 \leq i \leq n,\, x_i \in [0,1]\}$, one can use the well known distribution

$$y_{i+1} = y_i \; (x_{i+1})^{\frac{1}{n-i+1}} \tag{1}$$

to generate the set $\{y_i : 0 \leq i \leq n,\, y_i \in [0,1], y_{i+1} \leq y_i\}$ with $y_0 = 1$, which is a pre-sorted set of random numbers uniformly distributed in the the sorted simplex of the hypercube $[0,1]^n$. In other words, the distribution of the $\{y_i\}$ is identical to the distribution obtained when one first generates the $\{x_i\}$ and sorts them. This can be shown as follows. Consider the inverse of equation 1, given by

$$x_i = (\frac{y_i}{y_{i-1}})^{n-i+1} \tag{2}$$

The Jacobian Matrix for this transformation is lower triangular, and hence the Jacobian J is the product of the diagonal entries, i.e,

$$\begin{aligned} J &= \prod_{i=1}^{n} \frac{\partial x_i}{\partial y_i} && (3)\\ &= \prod_{i=1}^{n} \frac{n-i+1}{y_{i-1}} (\frac{y_i}{y_{i-1}})^{n-i} && (4)\\ &= n! && (5) \end{aligned}$$

which is as expected, since the PDF of a sorted set of n uniformly distributed random numbers must be uniform with value $n!$ in the simplex embedded in the hypercube satisfying the sortedness constraint $y_{i+1} \leq y_i$.

Thus, one can use the following alternative straight-forward procedure to determine the aggregate source rate.

1. Generate $n-1$ sorted random phases in the range $[0,T]$, using the relation

   $$t_i^{on} = T\, y_i$$

   where the y_i are generated as described above.

2. Compute the times when sources go off, which is given by $t_i^{off} = t_i^{on} + T_{on}$, while maintaining a list of the number of sources n_{mod} with offtimes larger than the period T.

3. Merge the three sorted lists $\{t_i^{on} : 1 \leq i \leq n\}$, $\{t_i^{off} : t_i^{off} \leq T\}$ and $\{t_i^{off} - T : t_i^{off} > T\}$ into one list $\{t_i : 1 \leq i \leq 2n\}$. The rate array is initialized as before, i.e, $r_i = +P$ if i a source turns on at t_i and $r_i = -P$ if a source turns off at t_i.

4. The rate at the start of the period is given by $r_0 = (1 + n_{mod})\, P$, while the other rates are determined cumulatively by $r_{i+1} \rightarrow r_{i+1} + r_i$.

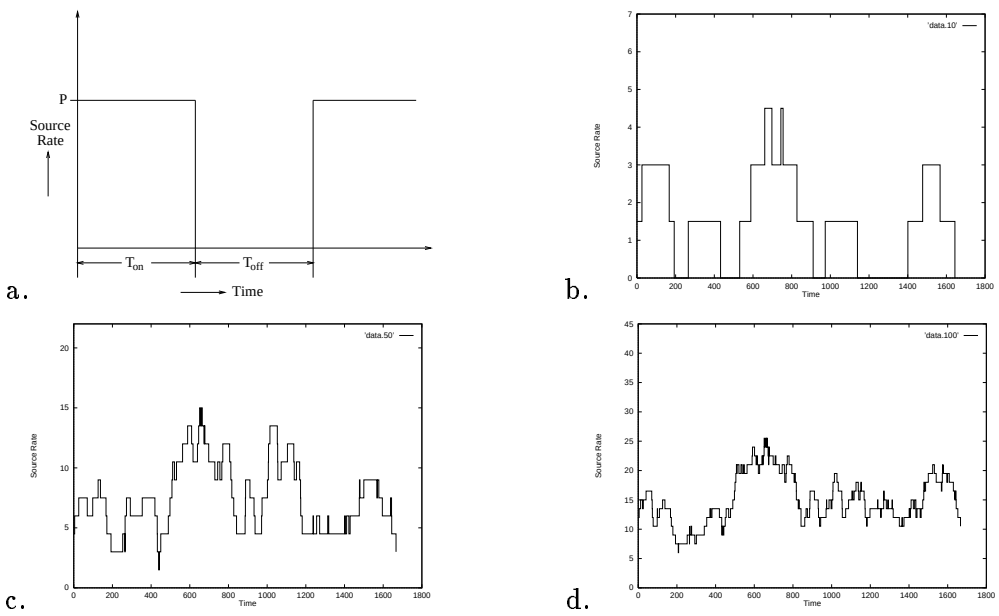

Figure 2: Rate process of (a) Single source (b) 10 sources. (c) 50 sources. (d) 100 sources. The parameters were $T_{on} = 166.67, T = 1666.67, P = 1.5$.

functions with jump discontinuities. The locations of the discontinuities are the event times of interest for simulations, as we describe below.

Thus, this simulation merely involves averaging the loss probabilities over a large number of replicas with different relative phases. Most of the complexity of the algorithm lies in determining the aggregate process, which is a computation repeated a very large number of times, and must be done efficiently. One straightforward procedure involving sorting is as follows. We maintain two lists - one for the event times $(t_i, 1 \leq i \leq 2n+1)$ and another for the source rates $(r_i, 1 \leq i \leq 2n)$ in the time bin immediately following t_i.

1. Generate $n-1$ random phases in the range $[0, T]$, assuming phase 0 for the first source. These are times at which sources turn on (t_i^{on}). These form the first n elements of t_i. The corresponding elements r_i are initially assigned the value $+P$.

2. Compute the times when sources go off, which is given by $t_i^{off} = (t_i^{on} + T_{on}) mod\, T$, while counting the number of sources n_{mod} with off-times larger than the period T. These times are placed in t_i, $i > n$, while the corresponding r_i are assigned the value $-P$.

3. Sort the array t_i, while performing the same rearrangements on r_i.[1]

4. The rate at the start of the period is given by $r_0 = (1 + n_{mod})\, P$, while the other rates are determined by cumulation, i.e $r_{i+1} \rightarrow r_{i+1} + r_i$.

Thus, on an SIMD machine with n_p processors, the above steps are carried out in parallel, which

[1] It was found that heapsort gave the best performance on SIMD parallel computers, since the speed is determined by the worst case. Quicksort has poor worst case performance.

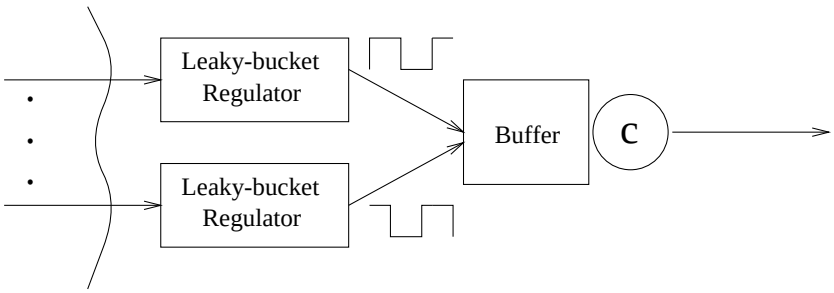

Figure 1: Regulated traffic into an ATM node.

the SGI we compared against, while the speed up values achieved were atleast by factors of 6 in all our simulations.

2 Replica-Parallel Simulation of a Leaky-bucket nD/D/1 ATM system

In the following, we discuss the simulation of ATM systems of the type described in [3](see figure 1), to which we refer the reader for a detailed mathematical analysis. Under certain extremal conditions, the arrival rates to the buffer from each of a set of n leaky-bucket regulators is a periodic, on-off signal, i.e. square wave (see figure 2a.), with parameters

$$T_{on} = \frac{B_T}{P - r},\ T_{off} = \frac{B_T}{r}$$

and the period given by

$$T = T_{on} + T_{off}.$$

Assuming the sources to be independent, the aggregate rate process is a superposition of n such square-waves with random relative phases. Figure 2 b., c. and d. depict this aggregate process for a specific choice of parameters. This incoming fluid accumulates in the buffer and is drained by the trunk C at a constant rate c, known as the channel capacity. The buffer has a maximum capacity B^{max}, and incoming "fluid" that sees the buffer full is lost. In our simulations, we assumed an infinite buffer approximation, but sought to determine the fraction of time for which the buffer level exceeds B^{max}, which is known as the "loss probability". It can be shown that such an approximation provides an upper bound on the actual loss probability for a finite buffer of this size.

The idea of statistical multiplexing involves combining of as many sources as possible through a single trunk while guaranteeing good quality of service in the form of low loss probabilities. Given the number of sources n, we would hence like to estimate the average loss probability over all possible relative phase configurations of the sources.

Aggregate rate process : The aggregate rate processes are periodic with period T, since the individual rate processes are assumed to be identical except for a possible random phase shift. One period of typical aggregate rate processes for different values of n are shown in figure 2 for a certain choice of the parameters and random relative phases. As expected, they are piecewise constant

1 Introduction

Emerging Asynchronous Transfer Mode (ATM) communication systems bring about new models and new demands for simulating these model. A new communication technology, ATM system, promises to deliver high volume traffic with very low failure rates. Consequently, simulating an ATM model involves collecting statistics over long time periods with an enormous number of "normal" events needed to be processed before a rare "fault" event occurs. Thus, the usual slowness drawback of serial simulations becomes especially noticeable when simulating ATM, eg. a single data point may require a week of computing on a fast workstation. It is here that parallel simulation, when applicable, has many advantages[7][2][4].

In this paper we present several different simulational models of ATM and different techniques to simulating these models on a massively parallel computer. In the course of this research, we encountered unexpected practical and theoretical difficulties, partly due to the massive-parallelism and SIMD aspects, and partly due to the fact that events of interest are rare.

First model we tried was an nD/D/1 ATM system with statistical multiplexing. The model easily lends itself to massively parallel processing via the obvious replication technique. The results of these simulations, however, were surprising: the statistics produced deviated quite substantially from those achieved on a serial computer. The relative difference was not noticeable for the non-interesting cases of frequent "fault" events but was becoming intolerably large for the interesting cases of rare "fault" events. It was quickly found out that the blame lies on the random number generator. However, the question remained which results were correct and what to do with the random number generator. Note that an efficient (that is, fast) generator on an SIMD computer must be inherently SIMD parallel and hence can not be arbitrarily substituted. We describe our way of statistical comparison of these results using secondary statistical measures and how we disambiguated and corrected the bias.

Another model we simulated was nM/D/1 ATM systems with statistically multiplexed Markovian sources. A similar (from the simulational point of view), discrete time version of this system was considered in [5] where an impressive speed of simulation was reported by using timing parallelization. We also decided to explore the time-parallel approach [2], [6], [8], but soon realized several important differences of our setup with that in [5]. Firstly, the bulk speed (along with the price) of our parallel computer is one-two order of magnitude lower than the speed of the computer employed in [5]. Secondly, there are substantial structural differences: our computer has a large number (16,384) of slow processors with relatively small memory and an SIMD mode of operation, while the simulation in [5] is arranged on a relatively modest number (32) of fast large-memory processors with an MIMD mode of operations. Thus, load-balancing and synchronization among processors, which are not of much importance in [5], are key factors contributing to the efficiency of our methods. And lastly, we discovered that an important feature that determines how easy or difficult would be to render the model time-parallel is the number of states of the simulated system. Ensuring proper end-state matching while maintaining efficiency in terms of load-balancing within the limited speed and memory per processor made our task more difficult and challenging. To overcome the challenge we tried several techniques, including parallel prefix, amortization of overheads, partial and full regeneration.

We found that, in all the models we considered, it was possible to obtain almost the theoretically maximum allowed speed-up using these methods, with negligible overheads. The combined processor speed of our MasPar system was about 2 Gflops, which is only about 7 times faster than

Massively Parallel Simulation of Asynchronous Transfer Mode (ATM) Systems

Abstract

Several models that arise in studying Asynchronous Transfer Mode (ATM) communication systems are simulated on a massively parallel SIMD computer. Fast simulations of ATM models are needed because the regimes of interest usually involve high volumes of traffic and low failure rates. In the course of this research, we encountered unexpected practical and theoretical difficulties, partly due to the massive-parallelism and SIMD aspects. These difficulties have not been reported in the previous accounts on the subject.

We found that, in a straightforward replica-parallel simulation of an nD/D/1 ATM system with statistical multiplexing, large variations in computed statistics are caused by apparently small differences in the distribution of various random number generators. Statistical comparison of these distributions using secondary statistical measures served to disambiguate the results.

We also found that time-parallel simulations of nM/D/1 ATM systems with statistically multiplexed Markovian sources can be efficiently performed using parallel prefix methods only when the sources have a small number of states, while more complex sources require end-state matching for efficient simulation. We discovered that, with the proper choice of initial state distributions and partial regeneration points, the time and memory requirements can be much improved.

Our simulations use the fluid model and infinite buffer approximations, and were carried out on the MasPar MP-1216 system with 16,384 processors, which was compared against an SGI workstation. We achieved almost the maximum possible speed-up (factor of about 7 compared to a fast workstation) with negligible overheads, thereby demonstrating the parallelizability of the models by these methods.