EFFICIENT MASSIVELY PARALLEL SIMULATION OF DYNAMIC CHANNEL ASSIGNMENT SCHEMES FOR WIRELESS CELLULAR COMMUNICATIONS

Albert G. Greenberg
AT&T Bell Laboratories

Boris D. Lubachevsky
AT&T Bell Laboratories

David M. Nicol*
College of William and Mary

Paul E. Wright
AT&T Bell Laboratories

January 7, 1994

Abstract

Fast, efficient parallel algorithms are presented for discrete event simulations of dynamic channel assignment schemes for wireless cellular communication networks. The driving events are call arrivals and departures, in continuous time, to cells geographically distributed across the service area. A dynamic channel assignment scheme decides which call arrivals to accept, and which channels to allocate to the accepted calls, attempting to minimize call blocking while ensuring co-channel interference is tolerably low. Specifically, the scheme ensures that the same channel is used concurrently at different cells only if the pairwise distances between those cells are sufficiently large. Much of the complexity of the system comes from ensuring this separation.

The network is modeled as a system of interacting continuous time automata, each corresponding to a cell. To simulate the model, we use conservative methods; i.e., methods in which no errors occur in the course of the simulation and so no rollback or relaxation is needed. Implemented on a 16K processor MasPar MP-1, an elegant and simple technique provides speedups of about 15x over an optimized serial simulation running on a high speed workstation. A drawback of this technique, typical of conservative methods, is that processor utilization is rather low. To overcome this, we developed new methods that exploit *slackness* in event dependencies over short intervals of time, thereby raising the utilization to above 50% and the speedup over the optimized serial code to about 120x.

*David Nicol's research was carried out in part during sojourns at AT&T Bell Laboratories. It was supported in part by NASA Grant NAG-1-1132 and NSF Grant CCR-9210372. Research was also supported in part by the National Aeronautics and Space Administration under NASA Contract No. NAS1-19480 while Nicol was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681.

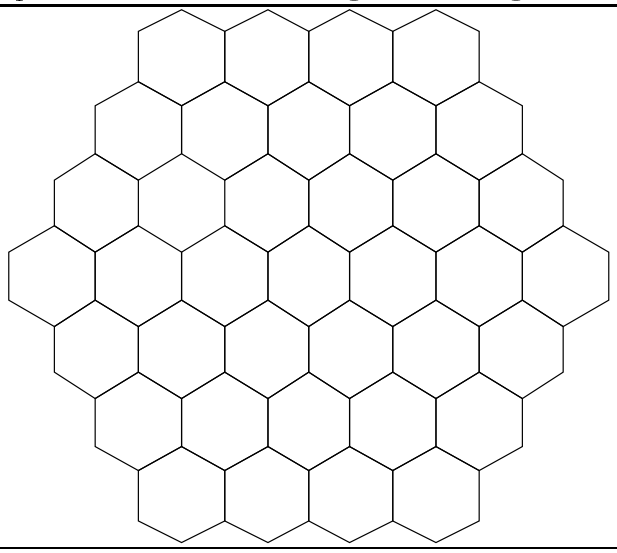
1 Introduction

Dynamic channel assignment (DCA) schemes provide flexible and efficient access to bandwidth in cellular communication networks. To date, mathematical analysis has brought insight into the design of such schemes, but only through *discrete event* simulation can system engineers obtain crucial numerical performance characterizations. In these simulations, the driving events are call arrivals and departures, distributed geographically across the area covered by the cellular network. A DCA scheme decides which call arrivals to accept and which channels to allocate to accepted calls. A rejected call is blocked; that is, denied a channel. Simulations are used to estimate call blocking statistics as functions of the network parameters and load.

The central idea of cellular networks is that a given channel can be used concurrently in geographically distinct cells, if and only if those cells are sufficiently far apart. This geographic separation ensures that the level of interference between calls assigned to the same channel is low. A DCA scheme dynamically allocates more channels to cells where the load is currently greatest. At a high level, we can view DCA as a type of resource allocation scheme under distributed control. In many DCA schemes, when resource sharing is not advantageous (say, in light or heavy load conditions), each cell allocates channels from a set of channels that the cell nominally "owns." When resource sharing is advantageous, cells borrow channels from each other, in response to demand. Owing to these interactions, all cells must be simulated to evaluate performance, not a single "typical" cell. Unfortunately, this implies slow and burdensome simulation runs, see e.g. [1], using conventional serial simulation techniques on high speed workstations.

A large class of DCA schemes can be modeled as a system of automata interacting in continuous time, each corresponding to a cell (Section 2). In these models, interactions are complex, but local. In our experiments, we consider hexagonal networks (see Figure 1) of about 100×100 cells, which is reasonable for evaluating large metropolitan area designs. A given cell interacts with all cells within a given discrete distance r. In a typical design, $r = 2$, and it turns out that a given cell interacts with 18 others. In this paper, we present efficient parallel simulation techniques, and report preliminary results on their performance on a 16K processor MasPar MP-1.

Figure 1 A 37 cell region of a hexagonal cellular network. In the dynamic channel assignment schemes considered here, a cell interacts with all $3r(r+1)$ cells in the r ring *macrohexagon* centered at that cell, where r is a parameter. If $r = 1$ then a cell interacts with the ring of 6 touching cells. If $r = 2$ then a cell interacts with that ring of 6 cells and the 12 additional cells touching the first ring. The entire region depicted here is an $r = 3$ ring macrohexagon.



Our first simulation method is based on the elegant, simple conservative synchronization method of [4]. We map the cellular array onto the processors, with one cell mapped to each processor. At each moment during the simulation, each cell maintains the time of its next event. A cell simulates its next event when the associated time is the minimum among the next-event-times of all cells in its neighborhood. Performance results obtained on the MasPar are promising. In particular, if the interaction parameter $r = 2$, then a MasPar code implementing this simulation method runs about 15 times faster than the best known serial algorithm implemented on a Silicon Graphics workstation that uses the MIPS R3000 microprocessor. This order of magnitude increase in simulation speed allows for the simulation of systems which are an order of magnitude larger than those typically studied before.

Although this approach is quite general and powerful, it suffers from rather low processor utilization, typical of conservative simulation methods. When the interaction parameter $r = 1$ (a cell has 6 neighbors), processor utilization is about 8.8%, when the parameter is 2 (18 neighbors), utilization is 3.2%, and when the parameter is 3 (36 neighbors), utilization is 1.4%. To overcome this, we use new methods that exploit *slackness* in the interactions between cells. By slackness, we mean that events at different potentially interacting subsystems are often uncorrelated over short intervals of time. Though the simulation remains conservative, it allows a cell to simulate its next event when the associated time exceeds the next-event time of a neighboring cell if this neighbor can have no unforeseen effect on the simulated event. To make that determination requires looking ahead into the future events of that neighbor. Though the look-ahead computations add overhead to the simulation, they pay off in much higher utilization: about 60% for $r = 1$, 56% for $r = 2$, and 54% for $r = 3$. Accordingly, the speedup is substantial; for typical runs with $r = 2$, the MasPar code runs about 120x faster than the optimized serial code.

In our experiments, we make standard Markovian assumptions, which trade some realism for

simplicity. Call arrivals are described by a Poisson process, with cell i receiving calls at rate λ_i. Holding times of accepted calls are assumed to be exponentially distributed. To date, we have not incorporated mobility and intra-cell call hand-off into the simulation, though this is intended for future work. In both serial and parallel simulations, we exploit the Markovian assumptions. In the serial simulation, we use an efficient alias method to generate the next event, rather than an event-list. In the parallel simulation, we use a uniformization technique to support the lookahead calculations in the high utilization code. This use of uniformization is helpful in streamlining the code, but is orthogonal to the idea of exploiting event slackness.

In this paper, we consider the class of *Markov* dynamic channel assignment schemes (Section 3), introduced by Raymond [6]. Here, the term "Markov" might be misleading because the schemes themselves have nothing to do with statistical assumptions about the traffic. The defining characteristic of Markov schemes is that the criteria for accepting a call depends only on the number of calls in progress at each of the cells. In our experiments, we focused on a novel (unpublished) efficient class of Markov schemes under investigation at AT&T Bell Laboratories.

Lubachevsky presented the basic conservative synchronization scheme (Section 4) and a variant that exploits aggregation (Section 5) in [4], where it was applied to simulating Ising models. In the simulation context, Ising models differ from the cellular models in that in the cellular models the interactions between neighboring sites are more complex, and that there is potential for event slackness in these interactions. In [5], Lubachevsky presented a notion of *opacity*, which is related to slackness (Section 5); similar concepts are often exploited by conservative synchronization schemes when subsystems enter states where they are guaranteed not to affect other subsystems for computable periods of time. Precise details tend to be application dependent. Lubachevsky [5] and Heidelberger and Nicol [3] apply uniformization (Section 5) in parallel simulations of Markovian systems.

2 A Dynamic Channel Assignment Model

At a high level, the channel assignment problem is a resource allocation problem. The resources are communication channels in the radio time-frequency domain. Users and *base stations* are distributed geographically across the service area. A user's request to make a call is handled at the nearest base station, which decides whether to accept or to block the call. As a rule, in making that decision the base station communicates with other stations, typically using an independent, wired signaling network. An accepted call is allocated a channel for communication with its base station, which separately manages the connection (wireless or wired) to the call's destination.

In wireless parlance, the communication from users to base stations is known as the *reverse* link. Communication from base stations to users is the *forward* link. Communication in the two directions are essentially independent. In particular, the two directions are usually assigned disjoint channel sets, and the engineering of the two directions is to a large degree mutually independent. In our experiments to date, we have considered just the reverse link. As the transmission method, it is simplest to think of TDMA (time division multiple access) or FDMA (frequency division multiple access), where a channel corresponds to a time or frequency slot, respectively.

In wireless networks, physical phenomena, such as channel interference, are continuous. However, dynamic channel assignment schemes can be modeled as discrete event systems by appropriate discretization. First, the service area is partitioned into cells (as in Figure 1), with each cell having a base station at its center. Second, channel interference constraints are discretized taking into ac-

count the worst case propagation of signal power with distance. By maintaining these constraints, the system ensures that the interference between two users of the same channel at different cells is negligible. Channel interference constraints may vary from network to network. However, the basic constraint is that a channel in use a given cell cannot be used at any other cell within the r ring *macrohexagon* centered at that cell, where the macrohexagon is obtained by adding r rings of cells about the central cell, as described in Figure 1, and r is a fixed parameter related to signal power and attenuation. A stronger restriction limits concurrent use of adjacent channels; for example, while channel n is in use at a cell, channels $n-1$, n, and $n+1$ cannot be used in the macrohexagon centered at this cell. Other restrictions may deal with the details of time and frequency slotting that define the channels; under certain conditions, two cells within the same macrohexagon cannot use different time slots falling within the same frequency band, even though those time slots define logically distinct channels.

What's common among all the restrictions is *locality*, as this follows from the central idea of channel reuse in cellular systems. In additional, practical dynamic channel assignment schemes are also local in that the decision to accept or to reject a call at a given cell involves only the states of those cells in the neighborhood of that cell. The extent of the cell's neighborhood depends on the DCA scheme and the restrictions in force. The DCA scheme might require a search of some sort over all channels that could already be in use in the neighborhood. In addition, the decision to accept a call might require some rearrangements in the existing assignment of channels to calls within the neighborhood.

By this discretization of the physical network and the assumption that under the the controlling DCA scheme a cell's action depends only on its state and the states of its neighbors, the network behaves as a system of interacting, continuous time automata. As stated in the Introduction, in our experiments we assume call arrivals are Poisson and call holding times are exponentially distributed. It follows that the system is Markovian, with each cell corresponding to a Markov chain interacting with its counterparts in the neighborhood.

There has been some theoretical work on centrally controlled, maximal packing strategies, which always accept a new call whenever there exists an admissible rearrangement of the assignment of channels to the calls in progress that leaves a channel free for the new call [2]. This is not practical, if only because finding the rearrangements needed is a complex online graph-coloring problem, and is NP-hard.

3 Markov Dynamic Channel Assignment Schemes

We consider regular hexagonal networks, as illustrated in Figure 1, with one channel use restriction: no two cells in the same r ring macrohexagon can concurrently use the same channel, where r is a parameter. We refer to the r rings of cells centered at a given cell as the neighbors of the central cell. Let M denote the total number of channels, indexed 1, ..., M. All channels are available for use in all cells. In Markov dynamic channel assignment schemes, a permutation π_i of the integers 1, ..., M controls channel assignment at cell i, where the index i runs through some total ordering of the cells. Designing these permutations for peak performance across a wide range of loads leads to challenging combinatorial problems. As noted above, in our experiments we use new designs under investigation at AT&T Bell Laboratories.

In a Markov DCA scheme, the state of cell i relevant to channel allocation is just the number of calls in progress at cell i, which we denote as $state(i)$. Channel allocation follows a stack discipline,

meaning that if $state(i) > 0$ then the channels in use at cell i are $\pi_i(1)$, ..., $\pi_i(state(i))$. Suppose that $state(i) < M$ at the time of a call arrival to cell i, and let channel index $n = \pi_i(state(i) + 1)$. The arrival is accepted on channel n if channel n is not already in use at any neighboring cell; that is, by the stack discipline,

$$\pi_j^{-1}(n) > state(j), \text{ for all neighboring cells } j \text{ of cell } i, n = \pi_i(state(i) + 1), \quad (1)$$

where π_j^{-1} denotes the inverse of permutation π_j. Accordingly, the call is carried at cell i on channel n, and $state(i)$ is incremented by 1. On the other hand, if either $state(i) = M$ or $state(i) < M$ but channel $n = state(i) + 1$ is in use at some neighboring cell then the call is blocked (rejected and lost).

A call departure at cell i momentarily introduces a hole in the the stack of assigned channels, $\pi_i(1)$, ..., $\pi_i(n)$, if the departing call is carried on channel $\pi_i(m)$, $m < n = state(i)$. To maintain the stack discipline, on departures the assignment is immediately "repacked," say by reassigning the call carried on channel $\pi_i(n)$ to the channel freed by the departure. As a result, it is as if the call assigned to channel n departs, and the result is that $state(i)$ is decremented by 1.

Figure 2 describes the code for a serial simulation of a Markov DCA policy, assuming that cell i receives calls at Poisson rate λ_i, and each accepted call departs after holding an exponentially distributed length of time with mean $1/\mu$ (equal means are not essential). In the implementation, we used an alias method to choose the cell receiving the next call arrival (line 7) or call departure (line 6), in $O(logN)$ time where N is the total number of cells. The sums in lines 2 and 3 were also updated in $O(logN)$ time. We implemented and tested the algorithm on a Silicon Graphics workstation running the MIPS R3000 microprocessor. The code speed varied from 8,000 to 20,000 events per second, depending on the values of parameters λ_i, μ, r, etc.

Figure 2 A serial algorithm for simulating a Markov DCA policy. The code is executed repeatedly until the termination condition is met.

1. Compute the total call arrival rate, $rate_a := \sum_{\text{all cells } i} \lambda_i$.
2. Compute the total call departure rate, $rate_d := \sum_{\text{all cells } i} \mu state(i)$.
3. Toss a coin, and with probability $rate_a/(rate_a + rate_d)$, next event type := arrival, and with the complimentary probability, next event type := departure.
4. Generate an exponentially distributed random variable Δ_t with mean $1/(rate_a + rate_d)$.
5. $time := time + \Delta_t$.
6. if next event type = departure then { Toss a coin, selecting cell i with probability $\mu state(i)/rate_d$. $state(i) := state(i) - 1$. }
7. if next event type = arrival then { Toss a coin, selecting cell i with probability $\lambda_i/rate_a$. If $state(i) < N$ and if condition (1) holds then $state(i) := state(i) + 1$. }

4 Basic Parallel Simulation Approach

Our first parallel simulation method maps the network to the parallel architecture, with one cell mapped to each processor. Figure 3 describes at a high level the code for each cell. Here, i is a cell index, $time(i)$ is the time of occurrence (timestamp) of the next event scheduled at cell i, and $neighbors(i)$ is the set of cells in the neighborhood of cell i, as determined by the network modeling assumptions (Section 3). The task of the i^{th} processor is to execute this code, until reaching the termination condition; e.g., $time(i) > end\_time$, where end_time is an input parameter.

The code is correct because cell i stalls until the next-event-time of all its neighbors exceeds its own next-event-time. (Equal next-event-times introduce some non-determinism but do not impact correctness.) At that point, the neighbors are stalled and their current states together with cell i's own current state, determine the update (line 2) at cell i associated with the current event. The values of these states also enable the scheduling (line 3) of the next event for cell i.

Figure 3 The basic conservative synchronization skeleton. Each processor i executes this code repeatedly, until reaching a termination condition, such as $time(i)$ exceeding a given threshold.

1. wait_until $time(i) <= min\{time(j) : j \in neighbors(i)\}$
2. $state(i) := next\_state(state(i), state(neighbors(i)))$
3. $time(i) := next\_event\_time(time(i), state(i))$

The code of Figure 3 may be executed asynchronously, with no coordination between processors beyond that implied by the wait_until directive of line 1. In an asynchronous setting, the wait_until can be implemented by having each cell i cyclically poll the values $time(j)$ of its neighbors, at each cycle recomputing the minimal value. The system trajectory is guaranteed to be independent of the timing and the polling order, as long as no two of the next-event-times tested in an execution of line 1 are equal. If the code is executed synchronously, as in a typical SIMD architecture such as the MasPar, the effect of line 1 is to simultaneously mask out those cells whose next-event-times are not minimal in their neighborhood. The remaining cells execute lines 2 and 3 in lockstep.

Line 2, where the work of the simulation is done, is customized to simulate Markov dynamic channel assignment schemes as follows. If the current event is a call departure then $state(i)$, which denotes the number of calls in progress, is decremented by 1. On the other hand, if the current event is a call arrival, then the update depends on the states of the neighboring cells. As in the serial algorithm (Section 3), $state(i)$ is incremented by 1 if condition (1) holds, indicating the call is accepted. Otherwise, the state is unchanged. Figure 4 describes the remaining details of scheduling the next event for each cell i (line 3 of Figure 3).

Figure 4 Scheduling the next event for cell i (line 3 of Figure 3).

1. $rate := \mu state(i) + \lambda_i$.
2. Toss a coin; with probability $\lambda_i/rate$, next event type:= arrival, and with the complimentary probability, next event type:= departure.
3. Generate an exponentially distributed random variable Δ_t with mean $1/rate$.
4. $time(i) := time(i) + \Delta_t$.

This algorithm was implemented on a 16K processor MasPar MP-1, and its performance measured for a wide range of network parameters, and uniform loads ($\lambda_i = \lambda$). A substantial improvement in performance over the serial algorithm was obtained. In particular, taking the radius of the cell neighborhood (Section 3) $r = 2$, an important and typical value in design, we found the MasPar code ran about 15x faster than the serial code. For $r = 1$, 2, and 3, we measured execution rates of about 39, 8, and 4 million calls per minute, respectively. A fall off in execution rate with r is natural since the computational work per call arrival is proportional to the neighborhood size $S(r) = 3r(r + 1)$; $S(1) = 6$, $S(2) = 18$, $S(3) = 36$. However, the absolute execution rate is not commensurate with the raw processing power of the MasPar. It turns out that processor utilization degrades with r. For $r = 1$, 2, and 3, we measured the processor utilization (counting cycles where events are simulated, and not those where the processor is stalled at line 1 of Figure 3) of 8.8%, 3.2%, and 1.4%, respectively.

Note that when cell i passes the synchronization test at line 1 of Figure 3, none of its neighbors pass, which suggests (but does not prove) that the utilization should be no more than $1/(S(r)+1)$. In the next section, we alter the simulation method to do better.

5 Higher Utilization Approaches

In our DCA schemes, the set of states that a given cell may assume is large, which suggests that there may be considerable "slackness" in the interactions between neighboring cells. That is, often, the next change of state at a given cell might not depend on the detailed state of many of its neighbors. Examples of slackness for the present problem are given below.

Consider two simulated subsystems, A and B, and let e be an event at simulated time t at subsystem A. We say that subsystem B is *moot* to e if the trajectory of subsystem B over some interval containing t is restricted to some set S_B of possible trajectories such that for each such trajectory the the simulation of e is the same. We refer to the time interval in this definition as a *moot period*.

In general and in the simulations considered here, moot periods arise in a random and state-dependent way, and the simulation itself must identify moot periods in order to exploit them. Leaving aside that computation for a moment, Figure 5 shows how to modify the basic synchronization skeleton of Figure 3 to exploit moot periods.

Figure 5 An improved conservative synchronization skeleton. Each processor i executes this code repeatedly, until reaching a termination condition, such as $time(i)$ exceeding a given threshold. In line 1, processor i does not stall if the minimization is empty, because all its neighbors are moot to the current event.

1. wait_until $time(i) \leq min\{time(j) : j \in neighbors(i)$ is not moot in an interval including $time(i)$, with respect to the event at i at time $time(i)$ $\}$
2. $state(i) := next\_state(state(i), state(neighbors(i)))$
3. $time(i) := next\_event\_time(time(i), state(i))$

Line 2 may require at least partial knowledge of the states of the moot neighbors. In practice, we cannot hope to identify on the fly all neighbors that are moot to the event in question, as indicated in line 1. Dependencies are complex and are known in full only after all events are simulated. However, if we include a cell $j \in neighbors(i)$ in the minimization that is moot to the event in question then the algorithm remains correct; at worst it stalls needlessly for cell j to update $time(j)$. Thus, the practical approach is to identify as many moot neighbors as possible at a small computational cost. Note that the algorithm in Figure 3 synchronizes on the premise that all cells j in cell i's neighborhood are not moot with respect to i's next event.

Situations where moot periods arise draw on the details of the model:

1. If the next event e at cell i is a call departure, then all cells are moot to e.
2. Suppose that the next event e at cell i is a call arrival at time t_i and that just before t_i cell i has $state(i)$ calls in progress. By the stack discipline controlling channel allocation, the arrival e is assigned to channel $n = \pi_i(state(i)+1)$, if no neighboring cell is using this channel at time t_i. Otherwise, the arrival is blocked. Now, consider a neighboring cell j whose next event time t_j is less than t_i. Of course, cell j is moot to e if channel n is not use at cell j at time t_i, but how might we detect this? Suppose further that in the time interval $[t_j, t_i]$ we know the times at which calls arrive to cell j and the holding times of all calls offered to cell j up to time t_i. This data is independent of interactions with other cells. Taking this into account, we can form a superset of the set of channels in use at cell j at time t_i. If channel n is not included in that superset then cell j is moot to the arrival e at cell i.

In our implementations to date, we have not tried to take advantage of all such inferences on moot periods. Instead, we opted for a simple implementation motivated by the second example, which exploits the Markovian structure of the model, so that we need only keep track of a few variables per cell. We do not exploit the fact that all cells are moot to departures of accepted calls, since integrating that inference into our implementation would require some searching through cell state histories. The details are as follows.

For each cell i, we keep track of the next event time $time(i)$ and a second, later time $\overline{time(i)}$. In addition to tracking the number of calls in progress $state(i)$ at each event we maintain bounding variables:

$$\underline{state(i)} \quad = \quad state(i) - \text{number of departures of offered calls at cell } i \text{ during } [time(i), \overline{time(i)}]$$

$$\overline{state(i)} \quad = \quad state(i) + \text{number of arrivals to cell } i \text{ during } [time(i), \overline{time(i)}]$$

Moot periods are identified as follows. Suppose the next event e scheduled at time t_i at cell i is a call arrival. A neighboring cell j is moot to this event if

$$time(j) < time(i) < \overline{time(j)},$$

and the decision to accept e is the same if we pretend the state of cell j at time $time(i)$ is $\underline{state(j)}$ as it would be if we pretend the state of cell j at $time(i)$ is $\overline{state(j)}$. Under the control of the Markov DCA scheme this means that either channel $n = \pi_i(state(i)+1)$ is in use at $time(i)$ at cell j under both pretenses or it is not in use under both pretenses. As noted above, for technical reasons we must also sometimes stall departures, treating neighboring cells as not necessarily moot. The rules are the same as for arrivals, except channel $\underline{m} = \pi_i(state(i))$ replaces channel $n = \pi_i(state(i)+1)$.

It remains to specify how the variables $\overline{time(i)}$ and derived variables are computed. For this, we use a simulation lookahead technique, which is streamlined by the use of uniformization. A cell maintains a circular buffer of p events in chronological order, where p is a (small) parameter. Cell i's next event, scheduled at $time(i)$ is the first event in the buffer. After cell i passes the synchronization test (line 1 of Figure 5), it removes this event from its buffer, simulates it, adds a new event to its buffer, and updates its local variables. Figure 6 provides the details. The key concept arises in lines 4-9, which compute the next event. The idea is to use the upper bound $\overline{state(i)}$ on $state(i)$ to define a uniformizing rate $\overline{rate(i)}$ at which events occur at cell i. That is, the true rate at which these events occur never exceeds $\overline{rate(i)}$. The quantity $\overline{rate(i)}$ is the sum of two components: λ_i (the fixed rate at which calls arrive to cell i) and $\mu\overline{state(i)}$ (an upper bound on the rate at which calls depart cell i). The coin toss that fixes the type (arrival or departure) of the event is biased in proportion to these two components of the rate. In general, the component $\mu\overline{state(i)}$ of the rate oversamples departures, and so departures must be filtered so as to not deviate from the true probability space. Line 3 accomplishes this filtering with a rejection test, using the true rate $\mu state(i)$ at which events occur at the event's timestamp.

In our experiments with this algorithm implemented on the MasPar, we found that as a function of the lookahead parameter p, first performance increases very rapidly, then increases slowly, and finally eventually decreases very slowly towards the performance attained at $p = 1$ since the rate bounds become loose for large p. The decrease for large p could be removed by, for example, forming customized bounds at the cost of some search of the event buffer. In typical runs, we found a value of p near 10 to be optimal, but there was little to choose in performance between $p = 10$ and $p = 20$. As noted in the Introduction, utilization (cycles where true events are simulated, as opposed to cycles where the processor stalls or departures are filtered out) is much improved over the code of Section 4, to about 60% for $r = 1$, 56% for $r = 2$, and 54% for $r = 3$. This swamps the overhead of computing moot periods, leading to a speedup of about two orders of magnitude over the workstation code (a typical run for $r = 2$ is 120x faster than the workstation code) and about one order of magnitude over the code of Section 4.

Figure 6 Cell i executes this procedure after passing the synchronization test (line 1 of 5). To initialize the computation, all variables are zeroed, and then lines 4-9 of the procedure are executed p times. An event e in the event buffer is assigned a timestamp, a type attribute (arrival or departure), and a rate attribute, referred to as δ in line 1. It can be verified that the quantity $\mu state(i)/\delta$ controlling the coin toss of line 3 is at most one.

1. Remove the first event e from the local event buffer, which has timestamp $time(i)$, an event type attribute (arrival or departure) and a rate attribute δ.
2. If e is of type arrival, then $\overline{state(i)} := \overline{state(i)} - 1$ and decide as before whether to accept the call as a function of $state(i)$ and $state(j)$ for all neighbors j of cell i. If the call is accepted then $state(i) := state(i) + 1$.
3. If e is of type departure then $\underline{state(i)} := \underline{state(i)} + 1$. Toss a coin; with probability $\mu state(i)/\delta$ set $state(i) := state(i) - 1$.
4. $\overline{rate(i)} := \lambda_i + \mu\overline{state(i)}$
5. Toss a coin; with probability $\lambda_i/\overline{rate(i)}$,
6. $type$:= arrival, and with the complimentary probability $type$:= departure.
7. If $type$ = departure, then $\underline{state(i)} := \underline{state(i)} - 1$; otherwise $\overline{state(i)} := \overline{state(i)} + 1$.
8. Generate an exponentially distributed random variate Δ_t with mean $1/\overline{rate(i)}$.
9. $\overline{time(i)} := \overline{time(i)} + \Delta_t$.
10. Assign the event the type just computed, the rate attribute $\overline{rate(i)} - \lambda_i$, and timestamp $\overline{time(i)}$. Add the event to the end of the event buffer for cell i.

A general and independent approach to increasing utilization over the code of Section 4 is to use aggregation, saturating the processors by mapping a contiguous region of cells to each processor instead of a single cell. If the cellular network is sufficiently large then this can be very effective, since synchronization overhead grows roughly with the number of cells on the boundary of a region and useful work grows roughly with the number of cells in the interior of a region. If all neighbors of the next event for the i^{th} region lie within the i^{th} region, then the event is simulated without delay as in the serial algorithm. Otherwise, we apply the basic synchronization skeleton (line 1 of Figure 3), where variable $time(i)$ is now the time of the next event for the i^{th} region, and $neighbors(i)$ is the set of regions that include cells that are neighbors of cells in the i^{th} region. A further improvement is possible, recognizing that the neighbors of the cell receiving the next event may receive their next events much later than the next event for their respective regions. To exploit this, we need only keep track of next event times for individual cells and use these in the synchronization at line 1 of Figure 3.

6 Final Remarks

The serial and parallel codes described here were used to evaluate novel (unpublished) Markov dynamic channel assignment schemes under investigation at AT&T Bell Laboratories. Using the higher utilization codes, we were able to play and explore interactively the effects of network parameters, getting the results of 100 million call runs back in a few minutes. Though the models discussed here assume that the radio spectrum is channelized (by time or frequency division multiple access methods), it turns out that the same simulation approach applies to some models of spread spectrum multiple access methods, which extend an accepted call's signal across the spectrum. Examples are CDMA (code division multiple access) and FHSS (frequency hopped spread spectrum). The methods presented here, though quite successful, have plenty of room for improvement, and further research in increasing processor utilization and decreasing synchronization overhead is tantalizing.

7 Conclusions

We are investigating the use of massively parallel architecture for the discrete-event simulation of wireless cellular communication systems. The present study focuses on dynamic channel assignment schemes, implemented on SIMD machines, e.g., the MasPar MP-1 or -2. We observe that practical DCA schemes base their allocation decisions system state information that is geographically local. This feature allows us to simulate such systems on parallel architectures using synchronization schemes that are also geographically local.

The present study implemented two different synchronization schemes. One is very simple, causing a processor i to synchronize with all processors holding data state that is close enough geographically to affect i. This scheme achieves an order-of-magnitude speed increase on a 16K PE MasPar MP-1 over an optimized serial simulator on a high-performance workstation. Observing that this scheme suffers from low PE utilizations, we developed a more sophisticated scheme that uses run-time state information to reduce the number of synchronizations. This technique substantially boosts PE utilization, and achieves *two* orders of magnitude speed increase over the optimized serial implementation. Current extensions under study include model aggregation, and incorporation of more model features in the simulation.

References

[1] D.D. Dimitrijevic and J.F. Vucetic. Design and performance analysis of algorithms for channel allocation in cellular networks. Technical Report TM 0569-07-92-414-05, GTE Labs, July 1992. To appear in IEEE Transactions on Vehicular Technology.

[2] D.E. Everitt and D. Manfield. Performance analysis of cellular mobile communication systems with dynamic channel assignment. *IEEE Select. Areas Comm.*, 7:1172–1180, October 1989.

[3] P. Heidelberger and D.M. Nicol. Conservative parallel simulation of continuous time markov chains using uniformization. *IEEE Transactions on Parallel and Distributed Systems*, 4(8):906–921, August 1993.

[4] B.D. Lubachevsky. Efficient parallel simulations of asynchronous cellular arrays. *Complex Systems*, 1:1099–1123, 1987.

[5] B.D. Lubachevsky. Efficient distributed event-driven simulation of multiple-loop networks. *Communications of the ACM*, 32(1):111, January 1989.

[6] P.-A. Raymond. Performance analysis of cellular networks. *IEEE Trans. Comm.*, 39(12):1787–1793, 1991.