Proceedings of the 1995 Winter Simulation Conference
ed. C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman

A PARALLEL DISTRIBUTED SIMULATION OF A LARGE-SCALE PCS NETWORK: KEEPING SECRETS

Brian A. Malloy

Department of Computer Science
Clemson University
Clemson, South Carolina 29634, U.S.A.

Albert T. Montroy

Harris Information Systems Division
PO Box 98000
Melbourne, Florida 32902, U.S.A.

ABSTRACT

Recently, research and development of complex PCS (personal communication service) networks has increased due to the rise in demand for mobile cellular communications. The efficiency of a PCS network is crucial in minimizing cost while maintaining quality service to mobile subscribers. Simulation is used extensively to facilitate the development of an efficient network. In this paper we present a conservative distributed simulation of a large-scale PCS network. The conservative approach that we propose permits large simulations using the PVM software to configure the network into a parallel machine. Using a unique approach to exploit lookahead, we are able to induce speedups comparable to those produced in Carothers et al. (1994).

1 INTRODUCTION

One approach to providing large-scale simulation studies is through the exploitation of parallelism. Currently, techniques do exist for parallelizing a PCS network. In Carothers et al. (1994), a parallel simulation is presented that uses an optimistic protocol on a distributed network of workstations to obtain excellent speedup in the range of 2.8 to 7.8, on eight processors. The speedup for the optimistic protocol is especially impressive since communication cost is high in a distributed approach. However, the optimistic protocol can be demanding on the memory hierarchy so that large simulations for more than 1024 cells may be prohibitive.

In Greenberg et al. (1994), a parallel simulation is presented that uses the conservative protocol on a MasPar, a tightly-coupled, synchronous multiprocessor equipped with 16k processors. Using the MasPar, a speedup of 120× was achieved over the sequential execution on a fast workstation. Since the conservative approach is less demanding of the memory hierarchy, an order of magnitude increase in the size of networks previously simulated using the optimistic approach was achieved. However, the MasPar multiprocessor may not be available to many simulationists interested in the study of PCS network performance.

In this paper, we present a parallel simulation that uses the conservative protocol on a distributed network of workstations. Using the approach outlined in Greenberg et al. (1994), we were not able to induce appreciable speedup into our parallel distributed simulation. However, by configuring the network to reduce communication and by exploiting lookahead in a unique fashion, we obtained speedups in the range of 3.4 to 7.5, on eight processors, over the sequential executions. The parallel architecture for our executions was *parallel virtual machine* (PVM) (Geist et al. 1993) and timings for our simulation were measured using *wall clock* time. Timings of the sequential executions that we used as a comparison to the parallel executions were the fastest timings obtained.

In the next section we present background material followed by Section 3 which overviews the model and the approach that uses *secrets* to obtain good speedup. In Section 4, we present implementation details. Section 5 highlights the results of our experiments. In the final section we draw some conclusions.

2 BACKGROUND

In this section we provide background for our work beginning with a description of a typical personal communication service network. We then overview the two important protocols for parallelizing simulations and give a brief summary of PVM.

2.1 What is PCS

A personal communication service (PCS) network (Cox 1990) is a wireless communication network, which provides service for mobile phone users or PCS subscribers. The communication area covered by a

PCS is partitioned into areas called **cells** with a set of radio channels assigned to each cell. A mobile phone, or **portable**, resides in the signal range of a particular cell for a period of time and then moves to another cell.

There are two important channel allocation schemes, *fixed channel assignment* (FCA) and *dynamic channel assignment* (DCA). We now describe FCA, the allocation scheme used in this work.

When a subscriber makes a phone call, the status of the destination portable is determined. If the portable is currently involved in another call then the line is busy and the call is **dropped**; the cell does not proceed past this point. If the line is not busy then the cell in which the portable resides attempts to allocate a radio channel to connect the call. If the cell is unable to find a free channel to connect the call then the call is **blocked**. The arrival of a new call is not the only way that a channel may be requested. If a portable is involved in a call when it is moving out of signal range of the current cell, it frees the channel that was allocated to the call and requests a channel from the cell into which it is moving. This action of passing a call-in-progress from one cell to another is called a **call handoff**. If no channel is available in the destination cell then this is termed a **handoff block** and the call is terminated. If a channel is available in the destination cell then it is allocated and the call continues with no perceivable interruption. Channels are released only when a portable with a call-in-progress moves out of the current cell's signal range or the call completes.

An important criteria used to judge the quality of a PCS network is the **blocking probability** or the ratio of the number of blocked calls to the number of attempted calls. Intuitively, the blocking probability is the probability that a call will not be connected due to channel availability. To provide quality service to subscribers, it is important to engineer the PCS network so that the blocking probability is low, typically less than 1 percent (Carothers et al. 1994).

Blocking probability can be controlled in a PCS network simulation by adjusting several of the parameters that define the network. These parameters are **average call length**, **average call interarrival time**, **number of channels per cell**, and **number of portables per cell**. As the ratio of portables per cell to channels per cell decreases, so does the blocking probability. Likewise as the average call length decreases and the average call inter arrival time increases the blocking probability decreases.

Performance modeling of large high-capacity personal communication service (PCS) networks is often accomplished through discrete event simulation. Hexagonal or square cells are used to represent the network in a PCS simulation. To avoid obtaining inaccurate statistics from the simulation, experiments should ideally model large networks consisting of thousands of cells. Since mobile phones move from cell to cell, all cells must be simulated to evaluate network performance. Using conventional sequential algorithms results in time consuming and burdensome simulation runs. As a result of these slow simulation runs, most studies only examine small-scale PCS networks containing less than 50 cells, and output statistics of the boundary cells are generally discarded to avoid the boundary effect (Kuek and Wong 1992, Zhang and Yum 1989). This approach may lead to biased output statistics, but simulating a large network with a wrap-around topology can be used to achieve reliable results (Lin and Mak 1994).

2.2 Protocols for Parallel Simulation

The two general categories of protocols used in parallelizing simulation programs are the conservative approach and the optimistic approach. In conservative simulations a processor does not execute an event scheduled for time t until all messages with time stamp less than t have been processed. This sequencing of event processing is known as the **local causality constraint** . Adherence to this constraint ensures that execution of all events is in chronological order. In optimistic simulations, the chronological processing of events is not necessary. A processor may execute events in any order and when the local causality constraint is violated the processor returns to a previous state where the constraint holds. This action of state recovery is known as *roll back*. The optimistic approach can produce substantial speedup due to parallelism (Carothers et al. 1994).

In the conservative approach, success is typically measured by the amount of lookahead that can be achieved to allow a window of opportunity in which processors can execute in parallel. In the optimistic approach, success is typically measured by the predictability of the events so that rollbacks, to recover from violations of the local causality constraint, are kept to a minimum. The PCS network simulation contains a high degree of predictability.

2.3 Parallel Virtual Machine

Parallel Virtual Machine (PVM) (Geist et al. 1993) is a software package that provides support for the construction of a parallel computer using a network of workstations. PVM supports a message passing communication paradigm that can accommodate more than 25 platforms, ranging from a Cray/YMP to an

80386 personal computer running the Unix operating system. Messages may be passed between any of the machines supported; data conversions, for platforms which use different data representations, are transparent to the user.

The cost of communication in PVM is high. Furthermore, many machines may be contending for the use of the network and this contention can have serious impact on performance. Thus, reducing communication in programs running in PVM is therefore a crucial consideration.

3 DESIGN OF FCA MODEL

Our PCS model is composed of two main object types, a cell and a portable. The cell represents a cellular tower and the communication area covered by that tower. Each cell has a certain number of channels associated with it that may be allocated to calls in that cell. The portable represents a portable phone unit. Each portable resides in a cell for a period of time and then moves to one of four neighboring cells. The important events of the simulation are the actions of the portables. These events are movements, call arrivals, and call completions.

We represent the PCS network as a square mesh of cells. In the parallel version of the PCS network the square is partitioned into stripes of equal dimensions where each stripe is assigned to a processor. Figure 1 illustrates a PCS network composed of four stripes where each stripe has sixteen cells and the overall simulation area covers sixty-four cells. We chose to partition the square in this manner in order to minimize the number of communication edges per processor, thus reducing the amount of inter-processor communication needed in the parallel version.

As illustrated in Figure 1, portable movement across stripe boundaries falls into two categories: communication movement and non-communication movement. When a portable moves in the direction East or West across stripe boundaries a message must be sent to a neighboring processor; therefore this movement represents communication between processors in adjacent stripes. In contrast, all movement across the North and South stripe boundaries requires no communication since the portable remains in the same stripe.

An intuitive mapping of the sequential square grid to processors may be to partition the square into smaller perfect squares. This mapping would result in each processor having four communication edges, a North, South, East, and West edge. With the stripe mapping illustrated in Figure 1, each processor has only two communication edges; one for the processor to the East and one for the processor to the West. This change in the mapping reduces the communication needed between processors by a factor of 2.

Our model of the PCS network processes call arrivals, call completions, and portable movement as described by Carothers et al. (1994). The PCS model simulates one end of the call. Thus, for any call currently in progress, each portable may be considered a call receiver or a call originator, but the PCS model does not actually maintain the connection for both ends of the same call.

3.1 Achieving Efficient Lookahead

The stripe configuration that we propose, not only reduces the amount of communication by minimizing the number of neighbors with which each processor interacts, it also provides an opportunity to exploit lookahead. The opportunity arises now because each stripe has only two neighbors and these neighbors are separated from each other by a fixed difference, the width of a stripe.

3.1.1 Taking Advantage of Stripe Width

We define the width of a stripe, *stripewidth*, as the number of columns of cells that compose the stripe. All stripes are of the same width, and the maximum lookahead that any given processor can give its neighbors is the width of a stripe. The stripe width is also used to generate *stripewidth* number of future moveout times and directions for each portable in the simulation. Therefore if *stripewidth* is four then we generate the next four moveout times and directions for each portable. A move is from one cell to an adjacent cell and may or may not be across a stripe boundary. When a portable moves into a new cell a new moveout time and new direction is generated to replace the move that was just processed. In addition to generating a new moveout time and direction upon moving into a cell, the portable inspects itself to determine the earliest time that it could be leaving this stripe. The portable simulates its location for the next *stripewidth* moves. It may know exactly what time and what direction it will be moving to another processor or it may simply know where within the current stripe it will be after the next *stripewidth* number of moves. If the portable knows exactly when it is moving out of this stripe it records its exact moveout time for either the East or West direction. This analysis is only required when the portable moves into a new cell and only if that portable does not already know exactly when and in what direction it will be leaving the current stripe. This information is used to determine the lowest possible time at which the

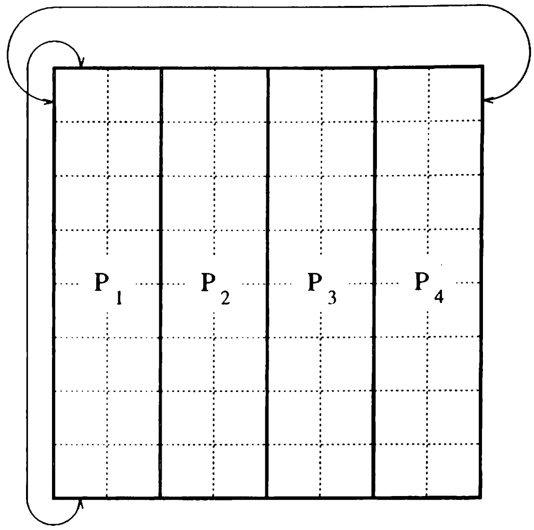


Figure 1: A PCS network consisting of four stripes where each stripe has 16 cells and the overall simulation area covers 64 cells. In the parallel simulation each of the four stripes are assigned to a processor. This figure also illustrates the wrapping around of portable movement across stripe boundaries.

processor may be sending a portable to either of its neighbors. The lookahead that any processor can give one of its neighbors in direction *d* is the minimum of the following: the lowest stripe moveout time for direction *d*, the lowest time a message is expected from another processor + *stripewidth*.

The lookahead that we generate by using the width of a stripe does not ensure adherence to the local causality constraint. The lookahead may sometimes become invalid. Assume that a processor, P_i, has given a lookahead to processor P_j that allows P_j to continue processing until time t without the possibility of receiving a message from P_i. This lookahead becomes invalid if at some time less than t processor P_j sends a portable to P_i that will return to P_j before t. This means that processor P_i will need to send a message to P_j earlier than P_j expects to receive a message. To avoid deadlock and the possibility of losing portables due to the lookahead becoming invalid we use a method called *keeping secrets.*

3.1.2 Keeping Secrets

In order to deal with the possibility of a portable moving to one processor and then moving back to the original processor in less than *stripewidth* time, we introduce a concept that we call *keeping secrets.* If a processor is sending a portable to a neighbor that will return before the processor expects to receive a message from that neighbor then the processor must record the time at which the portable will return. This time is needed by the processor so that it will not violate the local causality constraint. The analysis to determine if this event is possible is done as the sending processor is preparing to send the portable. This means that the stripe sending the portable knows, at the time that the message is sent, that the portable will be returning before the recipient expects to send a message. This is the secret. The receiving stripe will not know that it will be sending this message until the portable is ready to move out.

4 IMPLEMENTATION

Our PCS network is a self-initiating simulation where the cells generate their own incoming calls through the portables and the call completion times along with moveout times are generated by the portables themselves.

Each cell has four queues to hold portables, one for each direction (North, South, East, and West). The queue into which a portable is inserted corresponds to the direction of the portable's next move. Each queue is a simple linked list that is $O(1)$ with respect to portable movement, and $O(n)$ with respect to portable insertion (where n is the number of portables residing in the cell). During the initialization phase of the simulation each cell initializes a predetermined number of portables to reside in that cell at the start of the simulation.

```
algorithm   Simulate PCS parallel
input       Simulationlength
output      Simulation of this stripe

begin Simulate PCS Parallel
    time = 1
    while time <= SimulationLength
        for each cell in this stripe
            complete calls scheduled to end at this time
        endfor
        for each cell in this stripe
            process all move outs scheduled for this time
            process all move ins local to this stripe
            hold portables destined for neighbors
        endfor
        for each direction, N, S, E, W
            if time to send message to neighbor or have
                or have portables to send to neighbor
                  send the message
            endif
        endfor
        while proceeding will violate causality constraint
            receive a message
        endwhile
        for each cell in this stripe
            process calls scheduled to arrive at this time
        endfor
    time++
    endwhile
end
```

Figure 2: Algorithm to simulate PCS network.

```
algorithm   Send Message
input       PortableHoldinglist, ReceiveTime, SendTime,
            direction, local clock time
output      Message to neighboring stripe (processor)

begin Send Message
    SendTime[direction] =MIN(LowestMoveout[direction],
    LowestReceiveTime + stripewidth)
    Pack SendTime[direction] /* the new lookahead */
    if PortablesHoldingList[direction] IS EMPTY
        msgtype = PMSG
        Pack msgtype
        /* pack the number of portables being sent */
        Pack HoldingList[direction].count
        for all portables in HoldingList[direction]
            if ReturnTime < ReceiveTime[direction]
                if ReturnTime <
                        SecretSendbackTime[direction]
                    SecretSendbackTime[direction] =
                            ReturnTime
                endif
            endif
            Pack portable and destination cell
        endfor
    else
        msgtype = NMSG
    endif
    send message to neighbor[direction]
      with time stamp of time
end
```

Figure 3: Algorithm to send a message.

Each portable has three independent time stamp fields using exponential distributed time stamps. These fields are the call completion time stamp, the next call time stamp, and the move time stamp. Call completion refers to the time at which the current call will end. If there is no call currently in progress, the call completion time stamp has the value of zero. Initially we assume that there are no calls in progress. The next call time stamp is the time at which the next call is scheduled to arrive to that portable. The move field is the time at which the portable will move from its current cell to one of the four neighboring cells. Move time stamps and the directions corresponding to each move are stored in an array implemented as a queue. The size of the array is the number of columns of cells in each stripe. At any time during the course of the simulation the head of the queue is positioned at the portables next move information. When new move times and directions are generated, the head of the move queue is advanced.

The simulation parameters such as *portables per cell*, *average call interarrival time*, *average call length*, *average cell time*, *simulation length*, and *simulation size* are defined by the user in an input file that is accessible by all processors. The parameters are defined as follows: *portables per cell* is the number of portables in each cell at the start of the simulation, *average call interarrival time* is the mean time at which new calls arrive, *average call length* is the mean duration of a call, *average cell time* is the mean time that a portable stays in a cell, and *simulation length* is the number of clock ticks to run the simulation.

4.1 Simulating the PCS Network

Algorithm *SimulatePCS*, shown in Figure 2, overviews the simulation process that runs on each processor. The input to the algorithm is *SimulationLength*, which is specified in a common startup file that all processes may access. The simulation continues to process until the local clock of that processor, *time*, exceeds *SimulationLength*.

The three events possible, *call completion*, *movement*, and *call arrival* are processed in a specific order. At any time *t*. all calls scheduled for completion at *t* are completed, all movement at time *t* is processed, and all calls scheduled to arrive at *t* request a channel. With this ordering, when a channel is released at time *t*, it is also available to be reallocated at time t.

Each processor maintains a *ReceiveTime*, *SendTime*, and a *SecretSendbackTime* for each of its neighbors. For a direction, d, and a processor, p, $ReceiveTime_d$ corresponds to the last lookahead that was given to p by $neighbor_d$. Similarly, $SendTime_d$ of processor, p, corresponds to the last lookahead that p gave $neighbor_d$. The *SecretSendbackTime* for a given neighbor is a time at which that neighbor will be sending a message containing a portable. It is a secret that this processor is keeping. When a *SecretSendbackTime* for a neighbor is set to ∞, the processor is not keeping a secret with respect to that neighbor.

A processor sends messages to a neighboring processor only when the local clock time is equivalent to the *SendTime* for that neighbor or when the sending processor has portables to send in that direction. Likewise, a processor only receives messages from a neighbor when its local clock is equivalent to either the *ReceiveTime* or the *SecretSendbackTime* for that neighbor.

4.2 Message Exchanging Between Stripes

There are two types of messages exchanged by processors throughout the simulation. These message types are: (1) a portables message (PMSG), and (2) a null message (NMSG). Portable messages consist of portables destined for a neighboring processor and a new lookahead for that neighbor. Null messages contain only the new lookahead so that the recipient of the message may continue processing. Figure 3 shows the algorithm used by each processor in sending a message. Input to this algorithm is: *HoldingList*, *ReceiveTime*, *SendTime*, *direction*, and the local clock time,*time*. The *HoldingList[direction]* contains any portables that are moving out of this stripe to *neighbor[direction]*. The portables were inserted into the list during the movement processing phase of the simulation algorithm. The stripe sending the message first determines the maximum lookahead that it can give *neighbor[direction]*. This value is used to update *SendTime[direction]* and is then packed in the message buffer to be sent. After packing the new lookahead the message type, *msgtype*, is determined and packed. If *HoldingList[direction]* is not empty then *msgtype* is set to PMSG otherwise *msgtype* is set to NMSG. The stripe then packs *msgtype* so that he recipient of the message knows what actions to take upon receipt of the message. If *msgtype* is PMSG then the portables are packed in the message buffer along with the destination cell number. In the process of packing the portables, each portable is examined to determine if it will return to the sending stripe before the destination stripe expects to send a message. The time the portable will return is *ReturnTime*. If this situation is possible and *ReturnTime* is less than *SecretSendbackTime[direction]* then *SecretSendbackTime[direction]* is updated with *ReturnTime*. The message is time stamped with the current value of the local clock, *time*, and then sent to *neighbor[direction]*.

A stripe receives a message only if incrementing the local clock time would violate the local causality constraint. The status of the causality constraint is determined with respect to both the West and East neighbors. The stripe is in danger of violating the constraint with respect to direction d if the local clock time is equivalent to either *ReceiveTime[d]* or *SecretSendbackTime[d]*. The process of receiving a message is complimentary to the sending process.

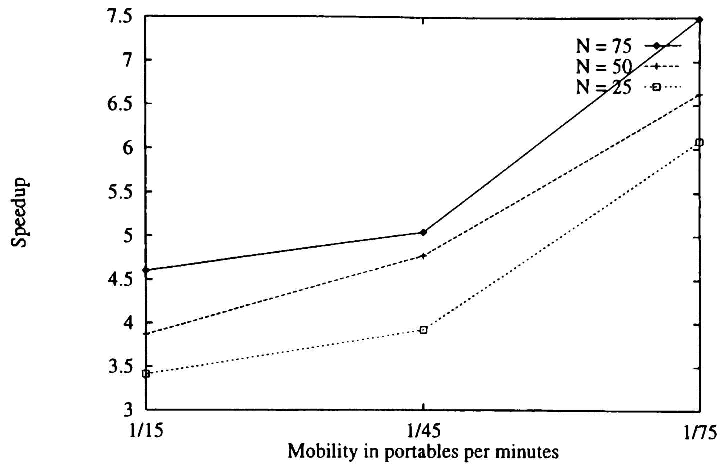
Table 1: Summary of the messages and contents for each simulation run, where N is the traffic density and M is the mobility.

Statistic N, M	*total messages*	*null messages*	*portable messages*	*portables /message*
75, 1/75	1081279	582084	499195	1.07
75, 1/45	1245808	449781	796027	1.12
75, 1/15	2072463	126179	1946284	1.37
50, 1/75	1017657	677452	340205	1.05
50, 1/45	1106954	555602	551352	1.08
50, 1/15	1677620	243214	1434406	1.24
25, 1/75	994034	819870	174164	1.02
25, 1/45	1003849	717697	286152	1.04
25, 1/15	1248336	449490	798846	1.12

5 EXPERIMENTS

In this section we describe the results of our experiments for a PCS network. The variables that are important to our experiments are *traffic density*, *mobility*, *average call length*, and *call arrival rate*. *Traffic density* is defined as the number of portables per cell and *mobility* is the rate at which portables move out of cells. The *average call length* is the mean duration of a call and the *call arrival rate* is the rate at which new calls are generated.

Our experiments measure speedup and investigate the effects of the *traffic density* and *mobility* with respect to speedup. We conducted experiments using *traffic densities* of 25, 50, and 75 portables per cell. Each *traffic density* was tested with *mobilities* of 1/(15 minutes), 1/(45 minutes), 1/(75 minutes). The *call interarrival rate* and the *average call length* were held constant across experiments. The call ar-



	N = 25			N = 50			N = 75		
	1/15	1/45	1/75	1/15	1/45	1/75	1/15	1/45	1/75
Seq	9.4	10.02	15.23	17.04	20.42	27.32	25.28	25.55	36.05
Par	2.75	2.56	2.5	4.4	4.28	4.12	5.5	5.07	4.82
Spdup	3.41	3.92	6.09	3.87	4.77	6.63	4.6	5.04	7.49

Figure 4: The graph illustrates the speedups achieved for different traffic densities with respect to mobility. *N* is the traffic density. The table displays the best sequential and parallel times for each experiment and the corresponding speedups. Times have been rounded to the nearest 100^{th} of an hour.

rival rate was set to 1/(10 minutes) and call length was exponentially distributed with a mean of 3 minutes. The ratio of portables per cell to channels per cell was fixed at 2.5. The simulation size was 1024 cells and all experiments ran for 2.5×10^5 simulation seconds. The distributed PCS network consisted of 8 workstations where each processor simulates a stripe containing 128 cells. Both the sequential and distributed experiments were conducted on a network of SUN SLC workstations.

To give a fair measure of speedup we used the fastest sequential time for specific portables per cell and *mobility*. Table 1 displays the number of portables per message, the number of portable messages, the number of null messages, and the total number of messages sent for each experiment conducted. Figure 4 illustrates the execution times of our experiments and the speedups that were achieved.

In the sections that follow, we illustrate the effects of *traffic density* and *mobility* on the performance of the parallel simulation.

5.1 Increasing Traffic Density

Our experiments indicate that Increasing the *traffic density* results in an increase in the number of portable messages and a decrease in the number of null messages sent during the course of the simulation. Figure 1 illustrates that the decrease in null messages is overshadowed by the increase in portable messages; thus, the total number of messages is increased.

Figure 4 shows that as the number of portables per cell increases so does speedup. This is largely due to an increase in processor workload.

5.2 Decreasing Mobility

Decreasing *mobility* greatly influences the number and types of messages sent. Figure 1 shows that as *mobility* decreases the number of portables per message decreases and the number of portables messages decreases significantly. The null messages increase, but the total number of messages decrease due to the significant decrease in the portables messages.

Speedup increases as *mobility* decreases (Figure