Proceedings of the 1992 Winter Simulation Conference
ed. J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson

APPROXIMATE TIME-PARALLEL SIMULATION OF QUEUEING SYSTEMS WITH LOSSES

Jain J. Wang
Marc Abrams

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061-0106

ABSTRACT

This paper presents partial state matching simulation using approximation for time-parallel simulation. The notion of degree of freedom in time-parallel simulation is introduced. Two partial state matching algorithms are proposed to simulate FCFS G/G/1/K and G/D/1/K queues in which arriving customers that find the queue full are lost. The performance of the algorithms is studied. Experiment results with M/M/1/K and M/D/1/K models show that potential speedup and simulation accuracy of the algorithms are good in general cases.

1 INTRODUCTION

Computer simulation is a process of computing a *sample path*, or *trajectory*, of a target simulation model, consisting of the time evolution of the state of the model over a period of simulation time. The state space and time domain of a simulation model form a *space-time* region (Chandy and Sherman 1989). The state space is typically defined as the set of components, processes (in the process-oriented world view), or state variables comprising a simulation model. For parallel distributed simulations, parallelism can be obtained through decomposing the state space (space-parallel) or the time domain (time-parallel). In this paper, the term *simulation* refers to discrete event simulation.

1.1 Space-Parallel Simulation

Many space-parallel algorithms have been proposed (Fujimoto 1990). These algorithms decompose a simulation model into components based on the model's state space. Each component is modeled by a *logical process*. Logical processes communicate with each other through messages (Chandy and Misra 1981; Jefferson 1985). In this approach, speedup is bounded by the number of logical processes. For example, in a queueing network model, each queue is usually modeled by a logical process. When the network consists of fewer queues than available processors, speedup is limited to the number of queues. In addition, speedup may be further limited by the overhead involved in coordinating multiple processors to enforce execution of events in chronological order (Fujimoto 1990), by structural dependencies (Wanger and Lazowska 1989), and by time scale differences (Ammar and Deng 1991) that exist within some models.

1.2 Time-Parallel Simulation

Time-parallel simulations (Ammar and Deng 1991; Bagrodia, Chandy, and Liao 1992; Chandy and Sherman 1989; Greenberg, Lubachevsky, and Mitrani 1990; Jones 1986; Lin and Lazowska 1991; Reiter, Bellenot, and Jefferson 1991) exploit parallelism through temporal decomposition of a simulation model. Some of these simulations combine spatial and temporal decomposition (Ammar and Deng 1991; Bagrodia, Chandy, and Liao 1992; Chandy and Sherman 1989; Jones 1986; Reiter, Bellenot, and Jefferson 1991). In this paper, a time-parallel simulation refers to one that obtains parallelism solely through temporal decomposition. A time-parallel simulation partitions the trajectory into a number of segments, or *batches*, along the time domain. Each batch is assigned to one processor. Each processor computes the batch assigned to it by simulating the entire system independently (and possibly asynchronously) for the time interval of the batch. Therefore, multiple processors can simultaneously simulate the system at different points in simulation time.

The *initial state* and the *final state* of a batch are the initial and final values of all the state variables, respectively. The simulation proceeds in an iterative manner. Initially, each batch is assigned a *guessed* initial state. The batch is computed using the guessed initial state. After computation of one or more batches has completed, the guessed initial states may be modified. Then the batches are *corrected* using a state correction technique with the updated initial states. The process is repeated until no changes occur in any initial state, at which point the simulation *converges* or reaches a *fixed* point. We use *estimated trajectory* and *true trajectory* to refer to an intermediate trajectory before convergence and the final trajectory after convergence, respectively.

The efficiency of time-parallel simulations rests on the ability to select the initial states of each batch to minimize the number of iterations required for convergence and the availability of efficient state correction mechanisms. In the worst case, a time-parallel simulation can take longer to execute than a sequential simulation.

In this paper we study time-parallel simulation using partial state matching with approximation. This approach sacrifices simulation accuracy in exchange for shorter convergence time. In this paper, two approximation techniques are introduced for simulating FCFS G/G/1/K queues and FCFS G/D/1/K queues, respectively. In both cases, arriving customers that find the queue full are lost rather than block.

The rest of this paper is organized as follows. In section 2, we review two related time-parallel simulation algorithms. Section 3 introduces the proposed partial state matching simulation. Two partial state matching algorithms and experiment results for FCFS G/G/1/K and G/D/1/K queues with losses are presented in sections 4 and 5. Conclusions are given in section 6.

2 RELATED WORK IN TIME-PARALLEL SIMULATION

In this section, we review two related time-parallel simulation algorithms: (1) Greenberg, Lubachevsky, and Mitrani's (GLM) recurrence algorithm (Greenberg, Lubachevsky, and Mitrani 1990) and (2) Lin and Lazowska's time-division algorithm (Lin and Lazowska 1991). The GLM algorithm is discussed with more detail because it forms the basis of one of our approximation algorithms.

2.1 The GLM Algorithm

The GLM algorithm provides an efficient way to simulate a class of queueing network models which can be expressed as recurrence relations and transformed into a parallel prefix problem. Let D be a domain and $\circ$ be any associative operator on that domain. Let N be any positive integer. A prefix problem is to compute each of the products $a_0 \circ a_1 \circ \ldots \circ a_k$ for $1 \leq k \leq N$ (Hills and Steele 1986; Lander and Fischer 1980). We first review how the GLM algorithm is applied to a FCFS G/G/1/∞ model.

For a FCFS G/G/1/∞ model, let A_i and D_i denote the arrival time and the departure time, respectively, of job i for i=1,2,. . .,N. Let α_i denote the interarrival time between job i and job $i+1$, and δ_i denote the service time of job i. If A_1, the arrival time of the first job, is given, the arrival and departure time sequences $A_1, A_2, \ldots, A_N$ and $D_1, D_2, \ldots, D_N$ are the solution of the following recurrence relations (Mitra and Mitrani 1989):

$$A_i = A_{i-1} + \alpha_{i-1} \quad 1 < i \leq N, \qquad (1)$$

$$D_i = \begin{cases} A_i + \delta_i & i = 1, \\ max(D_{i-1}, A_i) + \delta_i & 1 < i \leq N. \end{cases}$$

It is assumed that job interarrival and service times are continuous random variables whose values can be pre-sampled. Therefore, sequences $\alpha_1, \ldots, \alpha_N$ and $\delta_1, \ldots, \delta_N$ can be computed in advance.

In a G/G/1/∞ model, an *event* is a job arrival or a departure. An *event time* sequence $E_1, E_2, \ldots, E_{2N}$ is the result of merging sequence $A_1, \ldots, A_N$ and $D_1, \ldots, D_N$ in time order. Because there are N jobs, a total of $2N$ events will be simulated. The queue length sequence $L_0, L_1, L_2, \ldots, L_{2N}$, where L_0 is the initial queue length and L_i, $1 \leq i \leq 2N$, is the queue length immediately *after* event i, is obtained by solving the following recurrence relation:

$$L_i = \begin{cases} L_{i-1} + 1 & \text{Event } i \text{ is an arrival,} \\ L_{i-1} - 1 & \text{Event } i \text{ is a departure.} \end{cases}$$

Therefore, the computation of the queue length sequence is again a prefix problem. Assume that the number of processors, denoted P, divides N evenly. To compute $a_0 \circ a_1 \circ \ldots \circ a_i$ for $1 \leq i \leq N$ in parallel, the GLM algorithm performs the following steps:

1. Partition the sequence of $a_0, a_1, \ldots a_N$ into P batches evenly. Then batch l, $1 \leq l \leq P$, will contain $a_{((l-1)*N/P)+1}, \ldots, a_{l*N/P}$.

2. Compute $f_l = a_{((l-1)*N/P)+1} \circ \ldots \circ a_{l*N/P}$ for all $1 \leq l \leq P$.

3. Assume there exists an $\iota \in D$, such that $\iota \circ a_i = a_i$, for $1 \leq i \leq N$. Compute:
$$\beta_l = \begin{cases} \iota & l = 1, \\ f_1 \circ \ldots \circ f_{l-1} & 1 < l \leq P. \end{cases}$$

4. Let s_i denote the product of $a_0 \circ a_1 \circ \ldots \circ a_i$ and q and r be the quotient and the remainder of $\frac{i}{N/P}$, respectively. For $1 \leq i \leq N$, compute:

$$s_i = \begin{cases} \beta_{q+1} & r = 0, q < P, \\ \beta_q \circ f_q & r = 0, q = P, \\ \beta_{q+1} \circ a_{q*(N/P)+1}, \ldots, a_i & r \neq 0, q < P. \end{cases}$$

Using the GLM algorithm to compute the job arrival time sequences, the job departure time sequence, the queue length sequence, and the event time sequence for a FCFS $G/G/1/\infty$ queue requires $O(N/P + logP + logN)$ (Greenberg, Lubachevsky, and Mitrani 1990). In the rest of this paper, a sequence $X_a, X_{a+1}, \ldots, X_b$ will be represented by $\langle X_{a,b} \rangle$.

2.2 Lin and Lazowska's Time-Division Algorithm

Lin and Lazowska's algorithm partitions the time-domain through state matching. The state of a system is defined by the values of the system's state variables. A simulation is *partial regenerative* if there exists a subset of the system state variables such that the subsystem represented by the subset can repeat its state for an unlimited number of times as the simulation proceeds indefinitely. Such subset is called a *regenerative substate.* Lin and Lazowska's algorithm partitions the trajectory at the points where the regenerative substate repeats its state.

For each batch, the simulation initializes the regenerative substate with a pre-defined *matching state* and gives the rest of the state variables arbitrary values. The batch is then computed based on this initial state until the regenerative substate matches the matching state of the following batch. Later, when the full information of the initial state is known, the batch is *fixed up* by applying a state correction mechanism. To exemplify the algorithm, consider computation of a G/G/1 model whose system state includes the queue length, the remaining service times of the jobs in the queue, and the current simulation time. The regenerative substate contains the queue length and the remaining service times. Assume that two processors, p_1 and p_2, are available. Each processor simulates a batch with the initial simulation time and queue length set to zero (i.e., an idle server). Computation of a batch stops when the server becomes idle again. If p_1 finishes first and the final simulation time of its batch is t, then t is added to the time of each event in the batch computed by p_2, and p_1 can initiate another batch whose correct initial time can be decided when p_2 finishes.

3 PARTIAL STATE MATCHING SIMULATION WITH APPROXIMATION

To apply the GLM algorithm, recurrence relations solvable as a prefix problem must be identified. For Lin and Lazowska's algorithm, a regenerative substate must be identified such that there exists a state correction mechanism which can fix up the batches efficiently. In addition, Lin and Lazowska's algorithm requires the matching states to occur at least (P-1) times to obtain P batches. Moreover, for balancing the load among processors, the occurrences of matching states have to be evenly distributed in the time interval of the simulation. Simulation models are not likely to fit these conditions. This paper proposes partial state matching with approximation to extend the class of models to which time-parallel simulation can be applied.

Before discussing the partial state matching simulation, some definitions are required. Let $S = \{v_k | k = 1, \ldots, M\}$ be the state space of a simulation model which contains M state variables. Let $v_{k,j}(t)$ denote the value of state variable v_k at simulation time t after j iterations (for $j = 0, 1, \ldots$) where $0 \leq t \leq \tau$ for some $\tau > 0$. The *system state* at time t after iteration j and before iteration $j+1$ is represented by an M-tuple: $S_j(t) = (v_{1,j}(t), v_{2,j}(t), \ldots v_{M,j}(t))$. For discrete event simulations, the simulation models change states only at certain discrete simulation times $t_1, t_2, \ldots, t_N$. The *trajectory* of a simulation after j iterations is represented by the sequence: $S_j(t_0), S_j(t_1), \ldots, S_j(t_N)$.

The simulation time interval $[0,\tau]$ is partitioned into P intervals: $[b_0, b_1], (b_1, b_2], \ldots, (b_{P-1}, b_P]$, where $b_0 = 0$ and $b_P = \tau$. The segment of the trajectory in the interval $(b_{l-1}, b_l]$ for $1 < l \leq P$, and $[b_{l-1}, b_l]$ for $l = 1$ is referred to as batch l. Each batch is assigned with a processor and all batches are computed simultaneously. For batch l, $1 \leq l \leq P$, $S_{j-1}(b_{l-1})$ and $S_j(b_l)$ are called the *initial* and the *final* state, respectively, of the batch at iteration j. That is, the initial state of a batch is obtained from the final state of the previous batch resulting from the previous iteration. The simulation is said to have *converged* on $v_k \in S$ if $v_{k,j}(b_l) = v_{k,j'}(b_l)$ for all $j' \geq j$ and $0 \leq l \leq P$. If the simulation has converged on all $v_k \in S' \subset S$, then the simulation is said to have *partially converged* on subset S'. Otherwise $S' \equiv S$, and we say the simulation has *exactly converged.* We call $S - S'$ the *unmatched set* of the simulation, where " $-$ " is the set difference operator. The number of state variables in the unmatched set is called the *degree of freedom* of the simulation. Therefore, the degree of freedom defines the number of states that must converge for the simulation to exactly converge. Note that the degree of freedom of a simulation may decrease as j increases because more state variables may converge as the simulation proceeds.

Let j_{con} be the smallest j such that the simulation exactly converges after j iterations. Then $S_j(t_0), S_j(t_1), \ldots, S_j(t_N)$ is a *true trajectory* for $j \geq j_{con}$ and is an *estimated trajectory* for $0 \leq j < j_{con}$. At worst, a simulation requires P iterations to complete because the correct initial state propagates at least one batch for every iteration.

We propose a partial state matching simulation which artificially *fixes* some state variables in the unmatched set with approximate values so that these state variables can be removed from the unmatched set. As a result, the simulation which converges on fewer variables will generally require fewer iterations for convergence. In the following sections we will discuss two partial state matching algorithms for simulating G/G/1/K queues and show some experiment results of both algorithms.

4 FCFS G/G/1/K QUEUES WITH LOSSES

In this section we apply partial state matching to a FCFS G/G/1/K model with losses in which the arriving jobs that find the queue full are lost. In the G/G/1/K model, we assume that job interarrival times and service times are modeled by independent, identically distributed random variables. Unless mentioned otherwise, in the rest of this paper a G/G/1/K queue refers to one with a FCFS queueing discipline.

The GLM method described in section 2 can not be applied directly to a G/G/1/K model with losses. Let Q_i denote the queue length immediately before job i arrives. If we define the departure time of a lost job to be 0, then the departure sequence satisfies the following relation:

$$D_i = \begin{cases} max(D_1, \ldots, D_{i-1}, A_i) + \delta_i & Q_i < K, \\ 0 & Q_i = K. \end{cases} \quad (2)$$

Proposed below is an partial state matching algorithm to simulate a FCFS G/G/1/K model with losses using multiple processors. It is an open question whether there exists a linear recurrence equivalent to (2) that is solvable by a parallel prefix algorithm.

4.1 The G/G/1/K Partial State Matching Algorithm

For a G/G/1/K model, the system state contains the next arrival time, the queue length, and the remaining service times (RST) of each job in the queue. The next departure time can be determined by the RST of the first job in the queue. Because the queue has room for K jobs, there are K RST's and $||S|| = K + 2$. By equation (1), job arrival times are constants and thus S' always contains the next arrival time. Therefore, the initial degree of freedom of the simulation is the initial value of $||S - S'||$ which is $(K + 1)$. In this section, we discuss an algorithm which reduces the initial degree of freedom to one.

The algorithm consists of two phases. The first phase simulates a G/G/1/∞ model using the GLM algorithm and generates an *estimated* trajectory. The second phase which takes the finite buffer storage into account transforms the estimated trajectory into a *better* estimated trajectory. The phase-two trajectory is still estimated and not in general the true trajectory because the transformation process involves approximation of job departure times. Noted that if the queue length in the first phase never exceeds K, phase two is skipped because the trajectory generated by phase one is a true trajectory.

For simplicity, in our algorithm we assume that the number of processors divides the number of events to be simulated evenly. Let $L_{i,j}$ denote the queue length immediately after event i at iteration j. The algorithm is shown as follows:

Phase 1:

input: a G/G/1/K queue model and N (the number of jobs).

output: $\langle A_{1,N} \rangle$ (job arrival time sequence); $\langle D_{1,N} \rangle$ (phase-one job departure time sequence); $\langle L_{0,2N} \rangle$ (phase-one queue length sequence); $\langle E_{1,2N} \rangle$ (event time sequence).

```
begin
  1. $K' = K$.
  2. $K = \infty$.
  3. Apply the GLM algorithm of section 2.1 to compute $\langle A_{1,N} \rangle$, $\langle D_{1,N} \rangle$, $\langle L_{0,2N} \rangle$, and $\langle E_{1,2N} \rangle$ of the G/G/1/K queue.
end.
```

Phase 2:

input: $\langle L_{0,2N} \rangle$; $\langle E_{1,2N} \rangle$.

output: $\langle M_{1,2N} \rangle$ (an event type sequence; the type of an event can be *arrived*, *departed*, *lost*, or *ignored*); $\langle D'_{1,m} \rangle$ (phase-two job departure time sequence where $m \leq N$ is the number of jobs entering the queue and not lost; D'_i is the time of the

```
    $i_{th}$ departure); $\langle L'_{0,2N}\rangle$ (phase-two queue length sequence).
begin
    1. $K = K'$.
    2. $j = 0$. For each $l, 1 \leq l \leq P$, compute step 3 to 5 independently.
    3. if j=0, then
           $b_l = \frac{2N(l-1)}{P} + 1;\ h_l = \frac{2Nl}{P}$.
    4.
           $L_{b_l-1,j} = \begin{cases} min(L_{b_l-1}, K) & j = 0, \\ L_{b_l-1,j-1} & \text{otherwise.} \end{cases}$
    5. For $b_l \leq i \leq h_l$,
           $L_{i,j} = \begin{cases} min(L_{i-1,j}+1, K) & L_i = L_{i-1}+1, \\ max(L_{i-1,j}-1, 0) & L_i = L_{i-1}-1. \end{cases}$
           $M_{i,j} = \begin{cases} arrived & L_{i,j} > L_{i-1,j}, \\ departed & L_{i,j} < L_{i-1,j}, \\ lost & L_{i,j} = L_{i-1,j} = K, \\ ignored & L_{i,j} = L_{i-1,j} = 0. \end{cases}$
    6. $j = j + 1$. If there exists some l for $1 \leq l \leq P$ such that $L_{b_l,j} \neq L_{b_l,j-1}$ then go to step 4.
    7. Create an empty sequence $\langle D'\rangle$. For $1 \leq i \leq 2N$, if $M_{i,j} = departed$, append $E_i$ to $\langle D'\rangle$.
end.
```

In phase two, step 3 computes batch boundaries at the first iteration. Step 4 assigns each batch an initial queue length which is obtained from the final queue length of the preceding batch resulting from the previous iteration except for the first iteration at which a guessed initial queue length is given. The guessed initial queue length of each batch is obtained by computing the minimum of the corresponding phase-one queue length and the buffer size since the queue length can not exceed the buffer size. Step 5 computes the queue length sequence and decides each event's type. The computation of the queue length sequence in step 5 assumes that at any simulation time t, when a departure event occurs in the G/G/1/∞ queue, a departure event will also occur in the corresponding G/G/1/K queue at t. Obviously, this assumption is not true when losses occur. Therefore, the departure times used in phase two to compute a queue length trajectory are approximate. In the next section, we will show that this approximation in departure times will not cause significant errors in general. An important property follows the assumption in step 5:

Table 1: Numbers of iterations for an M/M/1/K model using the G/G/1/K partial state matching algorithm. Each data point is an average of 10 runs. Each run simulates 10^5 jobs which are divided into 2^{10} batches. For each entry (a, b), a is the average iteration number and $(a - b, a + b)$ is the 90-percent confidence interval for a.

K	λ/μ	P=4	P=16	P=64	P=256	P=1024
	.1	1.2,0.2	1.7,0.3	2.0,0.0	2.0,0.0	2.0,0.0
	.3	1.9,0.2	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
	.5	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
	.7	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
	.9	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
1	.95	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
	1.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
	1.05	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
	1.1	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
	1.3	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
	.1	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0
	.3	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0
	.5	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0
	.7	1.1,0.2	1.6,0.3	2.0,0.0	2.0,0.0	2.0,0.0
	.9	1.9,0.2	2.0,0.0	2.0,0.0	2.0,0.0	2.2,0.2
10	.95	1.9,0.2	2.0,0.0	2.0,0.0	2.0,0.0	2.3,0.3
	1.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.7,0.3
	1.05	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.6,0.3
	1.1	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.6,0.3
	1.3	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.1,0.2
	.1	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0
	.3	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0
	.5	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0
	.7	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0
	.9	1.0,0.0	1.4,0.3	2.0,0.0	2.6,0.3	7.7,1.1
	.95	1.6,0.3	2.0,0.0	2.1,0.2	4.7,0.5	15.8,2.1
50	1.0	2.0,0.0	2.0,0.0	2.5,0.3	6.3,0.4	21.1,1.5
	1.05	2.0,0.0	2.0,0.0	2.1,0.2	5.1,0.5	18.2,1.2
	1.1	2.0,0.0	2.0,0.0	2.0,0.0	4.1,0.4	13.3,1.6
	1.3	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	3.8,0.4
	.1	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0
	.3	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0
	.5	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0
	.7	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0
	.9	0.1,0.2	0.1,0.2	0.2,0.4	0.3,0.5	1.0,1.8
	.95	1.1,0.3	1.3,0.4	2.1,0.5	4.8,1.6	17.0,6.1
100	1.0	2.0,0.0	2.2,0.2	4.9,0.6	17.3,1.7	66.6,6.6
	1.05	2.0,0.0	2.0,0.0	3.2,0.5	9.9,1.1	36.5,4.9
	1.1	2.0,0.0	2.0,0.0	2.0,0.0	4.9,0.5	16.5,1.5
	1.3	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	3.8,0.4

Property 4.1: For all $1 \leq i \leq 2N$ and $1 \leq j < j_{con}$, $L_i \geq L_{i,j} \geq L_{i,j_{con}}$. That is, the queue length as a function of time of any iteration is a lower bound on the queue length of the previous iterations.

Step 6 checks the convergence of the simulation on queue lengths and advances the simulation to the next iteration if the simulation is not converged. Step 7 creates a departure time sequence. Note that D'_i is the (estimated) i_{th} departure time rather than the (estimated) departure time of job i. Given $\langle M_{1,2N}\rangle$, it is easy to associate elements in $\langle D'_{1,m}\rangle$ with correct job indexes.

Phase two requires $O(j_{con}(N/P + logP))$ to execute with P processors because step 1 to 4 take a constant of time; step 5 and 7 take $O(N/P)$; step 6 requires $O(logP)$. Therefore, the total execution time of this algorithm is $O[(N/P + logP + logN) + j_{con}(N/P + logP)]$ in which the first term is the time required by the GLM algorithm in phase one.

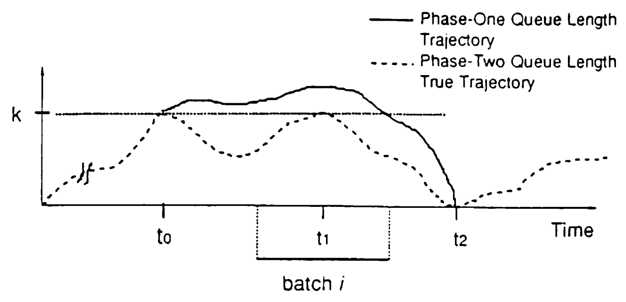
Phase-One Queue Length Trajectory
Phase-Two Queue Length True Trajectory
k
t0
t1
t2
Time
batch i

Figure 1: The first loss occurs at t_0. An E_F occurs at t_0 and t_1 and an E_E occurs at t_2. Sub-trajectories from t_0 to t_1 and from t_1 to t_2 are two propagation segments.

4.2 Experiment Results and Analysis

An experiment of the algorithm described above is carried out for an M/M/1/K model. Table 1 shows some results of the experiment. In general, it takes only a few iterations to converge. Many iterations are required only when both the *traffic intensity* (ratio of λ to μ) is close to 1 and the buffer size is large. We discuss this phenomenon next.

Recall that phase two of the G/G/1/K algorithm is a process of transforming an initial estimated trajectory into a more accurate trajectory. When there is no job being lost, phase-two computation is not necessary. Otherwise, phase-one trajectory after the first loss has to be modified. This is illustrated in Figure 1. At t_0 where the G/G/1/K queue is full and a job is lost. A change in queue length at t_0 has to be made and the change needs to be *propagated* along the queue length trajectory. At time t_1 the queue becomes full again. By property 4.1, the queue length at time t_1 will be changed to K in the first iteration in phase two regardless of the initial queue length of batch i which contains t_1. Therefore, no matter how many iterations it takes for a change to propagate from t_0 to t_1, it is guaranteed that the propagation *segment* between t_1 to t_2 would start with a correct initial queue length, namely K, at any iteration. Similarly, at t_2, where the G/G/1/∞ queues become idle the estimated queue length at any iteration in phase two will always be zero as well. Thus, the propagation starting from t_1 will not pass beyond t_2. We call time points such as t_1 and t_2 *synchronization* points. Formally, t is a synchronization point of a trajectory if for all state variables v_k in the trajectory, $v_{k,j}(t) = v_{k,j_{con}}(t)$ for all $1 \leq j \leq j_{con}$. A *propagation segment* is a trajectory fragment enclosed by two neighboring synchronization points. An event which leads to a synchronization point is called a *synchronization event*. Let E_F denote a (job arrival) event immediately after whose occurrence the G/G/1/K queue becomes full, and let E_E denote a (job departure) event immediately after whose occurrence the G/G/1/∞ queue becomes empty. Then, E_F and E_E are synchronization events of the queue length trajectory.

It becomes evident that the number of iterations required for convergence is bounded by the number of batches spanned by the longest propagation segment. Therefore we have:

$$1 \leq j_{con} \leq \lceil \frac{max[d(E_F, E_E), d(E_F, E'_F)]}{l_p} \rceil + 1$$

where (E_F, E_E) and (E_F, E'_F) are any pair of neighboring synchronization events, l_P is the batch length, and $d(E_x, E_y)$ is the number of events between the occurrence of E_x and E_y.

Therefore, when some losses occur, if the traffic intensity is low or is very high, E_E and E_F tend to occur more frequently, respectively, and hence the maximum propagation length is shorter. As a result, the number of iterations required for convergence becomes small. When the traffic intensity is neither high or low, both synchronization events occur more sparsely and the longest propagation length increases. Thus, the number of iterations increases. Once the propagation length becomes longer than the batch length, adding more processors becomes useless because the number of iterations will grow linearly with the number of processors used. This situation can be seen when K is 50 and 100, and λ/μ is close to 1 in Table 1.

In Table 1, the worst M/M/1/K simulation performance always occurs when $\lambda/\mu = 1$, regardless of the number of processors and the buffer capacity. Actually, when $\lambda/\mu = 1$, it is least likely that the queue will become empty or full. This can be derived as follows.

Let $\rho = \lambda/\mu$. If P_0 is the probability that the M/M/1/∞ queue is idle (when an E_E occurs) then P_0 is given by:

$$P_0 = (1-\rho) \quad 0 \leq \rho < 1.$$

If P_K is the probability that the M/M/1/K queue is full (when an E_F occurs), then P_K is given by (Kleinrock 1975):

$$P_K = \frac{\rho^K - \rho^{K+1}}{1-\rho^{K+1}} = \frac{\rho^K}{\sum_{i=0}^K \rho^i}.$$

Because when the M/M/1/∞ queue is empty, the corresponding M/M/1/K must also be empty, the joint probability of E_E and E_F occurrence can be given as:

$$P_s(\rho, K) = \begin{cases} P_0 + P_K & 0 \leq \rho < 1, \\ P_K & \text{otherwise.} \end{cases}$$

Table 2: Normalized approximation errors of the M/M/1/K and the M/D/1/K model using the G/G/1/K partial state matching algorithm. Each entry is the value of $100 * (E(L) - A(L))/E(L)$, where $E(L)$ and $A(L)$ are the average queue lengths of 10 runs obtained from a sequential simulation and the partial state matching simulation, respectively.

	λ/μ	K=1	K=5	K=20	K=60	K=100
MM1K	.1	-0.03	0.0	0.0	0.0	0.0
	.3	0.29	-0.07	0.0	0.0	0.0
	.5	0.11	0.01	0.0	0.0	0.0
	.7	-0.24	0.06	0.07	0.0	0.0
	.9	0.07	0.83	-0.35	0.1	0.0
	1.0	-0.29	0.39	1.35	0.95	0.49
	1.1	0.46	0.56	-0.29	0.50	0.14
	1.3	0.43	0.38	0.08	0.26	0.16
MD1K	.1	0.88	0.0	0.0	0.0	0.0
	.3	4.11	0.0	0.0	0.0	0.0
	5	9.41	0.80	0.0	0.0	0.0
	.7	15.92	0.94	0.0	0.0	0.0
	.9	23.15	4.66	0.17	0.0	0.0
	1.0	26.64	8.0	2.22	0.85	0.36
	1.1	25.16	4.97	0.18	0.01	0.005
	1.3	22.35	1.73	0.02	0.01	0.003

It can be shown that $P_s(\rho, K)$ has a minimum at $\rho = 1$ for any $K \geq 1$ (Wang and Abrams 1992).

The results of the partial state matching simulation in average queue length are compared with a sequential simulation (Table 2). For a fair comparison, both simulations use the same sequence of random numbers. The approximation errors are very insignificant except for the M/D/1/1 model. Actually, when both the variance of job service times and the buffer size of the queue are small, the G/G/1/K algorithm becomes *biased* and will result in more significant errors. A detailed discussion of this approximation error is given in (Wang and Abrams 1992). In the following section, an alternative algorithm is proposed for this case.

5 THE G/D/1/K PARTIAL STATE MATCHING ALGORITHM

For a G/D/1/K queue, in which each job has a fixed service time, the system state contains the first job remaining service time (FRST), the queue length, and the next job arrival time. Let N be the number of arrivals, P be the number of processors, and β_l be the initial queue length of batch l. A G/D/1/K partial state matching algorithm is described by the following steps:

1. Partition the N arrivals evenly into P batches.

2. Compute the arrival time sequence using the GLM parallel prefix algorithm.

3. Set β_l to 0 for all $1 \leq l \leq P$.

4. Set FRST of each batch to 0.

Table 3: Numbers of iterations of the M/D/1/K model using the G/D/1/K partial state matching (PSM) algorithm and the full state matching (FSM) algorithm. Each data point is an average of 10 runs. Each run simulates 10^5 jobs, which are divided into 64 batches (i.e. P=64). For each entry (a,b), a is the average iteration number and $(a-b, a+b)$ is the 90-percent confidence interval for a.

	λ/μ	K=1	K=5	K=20	K=60	K=100
PSM	.1	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
	.3	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
	.5	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
	.7	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
	.9	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
	1.0	2.0,0.0	2.0,0.0	2.1,0.2	3.5,0.4	7.9,1.1
	1.1	2.0,0.0	2.0,0.0	3.0,0.0	3.0,0.0	3.0,0.0
	1.5	2.0,0.0	2.0,0.0	3.0,0.0	3.0,0.0	3.0,0.0
	2.0	2.0,0.0	3.0,0.0	3.0,0.0	3.0,0.0	3.0,0.0
FSM	.1	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
	.3	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
	.5	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
	.7	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
	.9	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
	1.0	2.0,0.0	2.0,0.0	2.4,0.3	6.4,0.5	16.0,3.1
	1.1	2.0,0.0	2.0,0.0	7.0,0.9	61.2,5.1	64.0,0.0
	1.5	2.0,0.0	2.2,0.0	64.0,0.0	64.0,0.0	64.0,0.0
	2.0	9.2,0.9	64.0,0.0	64.0,0.0	64.0,0.0	64.0,0.0

5. For each batch l,$1 \leq l \leq P$, compute departure times and queue lengths independently via a sequential simulation.

6. Let f_l be the final queue length of batch l. If $\beta_{l+1} = f_l$, for all $1 \leq l < P$, exit; otherwise assign f_l to β_{l+1} for all $1 \leq l < P$ and go to step 4.

The degree of freedom of the simulation is one because job arrival times can be pre-computed using the GLM algorithm and the initial FRST's of all batches are fixed (i.e. 0). Thus, the initial unmatched set contains only the state variable for the queue length. Table 3 compares the convergence time in terms of iteration number of the proposed G/D/1/K partial state matching simulation with a *full state matching* simulation. The only difference between the two matching algorithms is that the full state matching does not use approximate FRST's and checks on both FRST and queue length for convergence. That is, for the full state matching simulation, except the first iteration, the initial FRST of each iteration is obtained from the final FRST of the preceding batch resulting from the previous iteration and the simulation completes only if both queue length and FRST converge.

Table 3 shows that full state matching requires many iterations to converge when $\lambda/\mu > 1$ and $K \geq 5$ and as λ/μ increases, the number of iterations converges to 64. In such case, no speed-up can be gained through using multiple processors. The partial state matching simulation, on the other hand, requires no more than 3 iterations to converge except when

Table 4: The normalized approximation errors for the M/D/1/K model using the G/D/1/K partial state matching algorithm. Each entry is the value of $100 * (E(L) - A(L))/E(L)$, where $E(L)$ and $A(L)$ are the average queue lengths of 10 runs obtained from the full state matching simulation and the partial state matching simulation, respectively.

	λ/μ	K=1	K=5	K=20	K=60	K=100
MD1K	.1	0.0	0.02	0.02	0.02	0.02
	.3	0.01	0.11	0.11	0.11	0.11
	.5	0.02	0.24	0.26	0.26	0.26
	.7	0.04	0.38	0.58	0.58	0.58
	.9	0.07	0.53	1.85	2.48	2.48
	1.0	0.06	0.65	2.38	4.86	10.26
	1.3	0.08	0.56	0.83	0.54	0.27
	1.3	0.10	0.29	0.06	0.03	0.01
	2.0	0.13	0.43	0.09	0.03	0.02

$\lambda/\mu = 1$. In fact, the number will converge to 2 as λ/μ increases and a linear speed-up can be obtained. In the worst case when $\lambda/\mu = 1$ and $K = 100$, the simulation takes an average of 7.9 iterations to converge.

The reason that the partial state matching outperforms the full state matching in convergence time is illustrated in Figure 2. It is not hard to see that for the FRST trajectory, a synchronization point occurs only when the G/D/1/∞ queue becomes empty. Therefore, as λ/μ increases, the possibility of the queue being empty decreases. When there exists no synchronization point in the FRST trajectory, linear convergence (i.e. $j_{con} = P$) will occur. For the partial state matching, the longest convergence times occur at $\lambda/\mu = 1$. The reason can be argued similarly as the M/M/1/K simulation discussed in section 4.2. The trade-off of execution time is simulation accuracy. Table 4 shows that the partial state matching simulation produces close results. The approximation error is less than 1% in general and is about 10% for the worst case.

6 CONCLUDING REMARKS

Partial state matching simulation artificially fixes some state variables in the unmatched set by using some approximate values so that these state variables can be removed from the unmatched set. As a result, the simulation needs to converge on fewer state variables and thus is likely to converge more quickly.

Two algorithms using this partial state matching approach to simulate FCFS G/G/1/K and G/D/1/K queueing models are proposed in this paper. The first algorithm uses approximate job departure times; the second algorithm uses approximate first job remaining service times. Experiment results of an M/M/1/K and an M/D/1/K model show that the speed-up

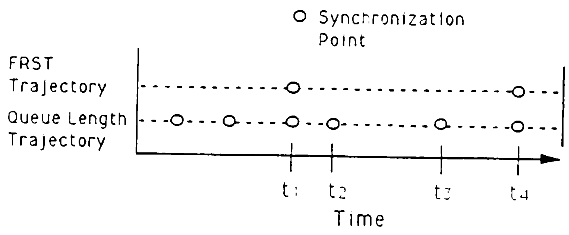

Figure 2: Without approximation, the full state matching simulation converges on both FRST and queue length. The longest propagation distance is $(t_4 - t_1)$. With approximation on FRST sequence, the partial state matching simulation converges only on queue length. The longest propagation distance is $(t_3 - t_2)$.

gained by using these partial state matching simulations against full state matching simulations becomes very significant as λ/μ exceeds 1. For simplicity, both algorithms do not use the processors optimally. In these algorithms, when a prefix of the trajectory being computed is converged so that no changes will be made in the following iterations to this prefix, the processor(s) assigned to this prefix remain(s) to compute the converged prefix until the simulation exactly converges. With a more carefully devised load distribution algorithm where batch partition is updated dynamically to exclude the converged prefix, speedup can further be enhanced.

Also, both partial state matching simulations produce small approximation errors in general cases. The worst performance for both simulations occurs when $\lambda/\mu = 1$. An argument is made to explain this phenomenon. The first algorithm introduces more significant errors when the model has a small buffer and has a small job service time variance. The second algorithm is introduced for this situation. Possible future work includes testing a broader class of probability distributions, and investigating networks of queues which contains G/G/1/K queues.

REFERENCES

Ammar H. H., Deng. S. 1991. Time Warp Simulation Using Time Scale Decomposition *Proceedings of the SCS Multiconference on Advances in Parallel and Distributed Simulation*. Jan., pp 15-22.

Bagrodia R., Chandy K. M., Liao W. T. 1992. An Experimental Study On the Performance of the

Space Time Simulation Algorithm. *Proceedings of the Sixth Workshop of the Parallel and Distributed Simulations.* pp 159-168.

Chandy K. M. and Misra J. 1981. Asynchronous Distributed Simulation via a Sequence of Parallel Computations. *Commun. ACM.* Vol. 24, No. 11, Nov. pp 198-205.

Chandy K.M. and Sherman R. 1989. Space-Time and Simulation. *Proceedings of the 1989 SCS Multiconference on Distributed Simulation.* March, pp 53-57.

Fujimoto R. M. 1990. Parallel Discrete Event Simulation. *Commun. ACM.* Vol. 33, No. 10, Oct. pp 31-53.

Greenberg A. G., Lubachevsky B. D., Mitrani I. 1990. Unboundedly Parallel Simulations via Recurrence Relations. *Proceedings of the Conference on Measurement and Modeling of Computer Systems.* Boulder, Colorado, May, pp 1-12.

Hills W. D., Steele G. L. 1986. Data Parallel Algorithms. *Commun. ACM.* Vol. 29, No. 12, Dec., pp 1170-1183.

Jefferson D. 1985. Virtual Time. *ACM Transactions of Programming Languages and Systems.* Vol. 7, No. 3, July, pp 404-425.

Jones D. W. 1986. Concurrent Simulation: An Alternative to Distributed Simulation. *Proceedings of the 1986 Winter Simulation Conference*, December. pp 417-423.

Kleinrock L. 1975. *Queueing Systems.* Vol. 1, Wiley-Interscience.

Lander R. E., Fischer M. J. 1980. Parallel Prefix Computation. *Journal of ACM.* Vol. 27, pp 831-838.

Lin Y. B., Lazowska E. 1991. A Time-Division Algorithm for Parallel Simulation. *ACM TOMACS*, Vol. 1, No. 1, Jan., pp 73-83.

Mitra D., Mitrani I. 1989. Control and Coordination Policies for System with Buffers. *ACM SIGMETRICS Performance Evaluation Review*, Vol. 17, No. 1, May, pp 156-164.

Reiter P., Bellenot S., Jefferson D. 1991. Temporal Decomposition of Simulations Under the Time Warp Operating System. *Proceedings of 1991 SCS Multiconference on Advances in Parallel and Distributed Simulation*, Jan., pp 47-54.

Wang J. J., Abrams M. 1992. Approximate Time-Parallel Simulation of Queueing Systems with Losses. Tech. Rep. TR92-08, Department of Computer Science, Virginia Tech.

Wanger D. B., Lazowska E. D. 1989. Parallel Simulation of Queueing Network: Limitation and Potentials. *Proceedings of 1989 ACM SIGMETRICS and PERFORMANCE*, May, pp 146-155.

AUTHOR BIOGRAPHIES

JAIN J. WANG is a Ph.D. student in the Department of Computer Science at Virginia Polytechnic Institute and State University. He received an engineering diploma from the National Taipei Institute of Technology, Taiwan, in 1983, and M.S. in computer science from the State University of New York in 1988. His research interests include parallel discrete event simulation and system performance analysis. He is a student member of ACM.

MARC ABRAMS is an assistant professor in the Department of Computer Science at Virginia Polytechnic Institute and State University. His research interests include parallel simulation, software performance analysis, and communication protocols. He received his Ph.D. from the University of Maryland in 1986. He serves as Program Chair for the 1992 SCS Parallel and Distributed Simulation (PADS) workshop.