Time-Parallel Simulation Using Partial State Matching for Queueing Systems

Jain J. Wang and Marc Abrams

TR 92-50

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

October 2, 1992

Time-Parallel Simulation Using Partial State Matching For Queueing Systems

Jain J. Wang and Marc Abrams

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

TR 92-50

September 30, 1992

Abstract

This paper describes partial state matching for approximate time-parallel simulation. The notion of degree of freedom in time-parallel simulation is introduced. Two partial state matching algorithms are proposed to simulate acyclic networks of FCFS G/G/1/K queues in which arriving customers that find the queue full are lost. The algorithms are suitable for SIMD as well as MIMD architectures. The performance of the algorithms is studied. Experiment results with M/M/1/K and M/D/1/K queueing networks show that the potential speedup and simulation accuracy of the algorithms are good. The worst performance of both algorithms occurs when traffic intensity is one. Arguments are made to explain this phenomenon.

Categories and Subject Descriptions: D.1.3 [**Programming Techniques**]: Concurrent Programming; D.4.1 [**Operating Systems**]: Process Management–*synchronization*; F.1.2. [**Computation by Abstract Devices**]: Modes of Computation–*parallelism*; I.6.8. [**Simulation and Modeling**]: Type of Simulation–*discrete event*

General Terms: Algorithms, Experimentation, Performance

Additional Key Words and Phrases: discrete event simulation, space-parallel, time-parallel, trajectory

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 [4]. 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 [5]. 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 [3, 8]. 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 [5], by structural dependencies [15], and by time scale differences [1] that exist within some models.

1.2 Time-Parallel Simulation

Time-parallel simulations [1, 2, 3, 4, 6, 9, 12, 14] exploit parallelism through temporal decomposition of a simulation model. Some of these simulations combine spatial and temporal decomposition [1, 2, 3, 4, 9, 14]. 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. Two partial state matching algorithms 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. These queues are of great importance to modeling packet switched communication networks, which drop packets when switching node buffers are full. The proposed algorithms are suitable for SIMD as well as MIMD architectures.

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 acyclic FCFS G/G/1/K networks 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 [6] and (2) Lin and Lazowska's time-division algorithm [12]. 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 $1 \leq k \leq N$ [7, 11]. 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 [13]:

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

$$D_i = \begin{cases} A_i + \delta_i & i = 1, \\ max(D_{i-1}, A_i) + \delta_i & 1 < i \leq N. \end{cases} \tag{2}$$

It is assumed that job interarrival and service times are continuous random variables whose values can be pre-sampled. From equation 1, $A_i = A_1 + \alpha_1 + \ldots + \alpha_{i-1}$. Thus, solving $\langle A_1, A_N \rangle$ is a prefix problem. For the departure time sequence, equation 2 can be rewritten by a matrix recurrence relation [6] to which a prefix algorithm can be applied directly. In the rest of this paper, a sequence $X_a, X_{a+1}, \ldots, X_{a+(b-a)-1}, X_b$ will be represented by $\langle X_{a,b} \rangle$.

In a G/G/1/∞ model, an *event* is a job arrival or a departure. An *event time* sequence $\langle E_1, E_{2N} \rangle$ is the result of merging sequence $\langle A_1, A_N \rangle$ and $\langle D_1, \ldots, D_N \rangle$ in time order. Because there are N jobs, a total of $2N$ events will be simulated. The queue length sequence $\langle L_0, L_{2N} \rangle$, 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 $\langle a_0, a_N \rangle$ into P batches evenly. Then batch l, $1 \leq l \leq P$, will contain $\langle a_{((l-1)*N/P)+1}, a_{l*N/P} \rangle$.

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:

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

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/∞ queue requires $O(N/P + logP + logN)$ [6].

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 predefined *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 M denote the number of state variables in a simulation model, which are denoted $v_1, v2, \ldots, v_M$, and let $S = \{v_k | k = 1, \ldots, M\}$. 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: $S_j(t) = (v_{1,j}(t), v_{2,j}(t), ...v_{M,j}(t))$. For

discrete event simulations, the simulation models change states only at certain discrete simulation times. Let N be the number of such times, and denote the times as $t_1, t_2, \ldots, t_N$. The *trajectory* of a simulation after j iterations is represented by the sequence $S_j(t_0), S_j(t_1), \ldots, S_j(t_N)$.

Let [0,τ] be the interval of the simulation. The simulation 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 to a processor and all batches are computed asynchronously in parallel. 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 Queueing Networks with Losses

In this section we apply partial state matching to acyclic networks of FCFS G/G/1/K queues with losses in which the arriving jobs that find the queue full are lost. In the 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:

$$D_i = \begin{cases} max(D_1, \ldots, D_{i-1}, A_i) + \delta_i & Q_i < K, \\ 0 & Q_i = K. \end{cases} \quad (3)$$

Proposed below is an partial state matching algorithm to simulate acyclic queueing networks of FCFS G/G/1/K queues with losses using multiple processors. It is an open question whether there exists a linear recurrence equivalent to equation 3 that is solvable by a parallel prefix algorithm.

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

We first consider a model of a single FCFS G/G/1/K queue. For the model, the system state(s) 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. Except for the first job in the queue, the RST of each job is equal to the service time of the job. 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 partial state matching algorithm (PSM-D) 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. 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. Otherwise, the second phase which takes the finite buffer storage into account,transforms the estimated trajectory into a more accurately 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. However, sections 4.2 and 4.3 show that the output trajectory of phase 2 is close to the true trajectory.

For simplicity, we assume that the number of processors divides the number of events to be simulated evenly. For the G/G/1/K model, there are four types of events: *arrive, arrive-lost, depart*, and *depart-ignored*. Event types *arrive* and *arrive-lost* correspond to job arrivals of non-lost and lost jobs, respectively. Event type *depart* corresponds to those job departures that occur in both the G/G/1/∞ and the G/G/1/K models, while event type *depart-ignored* corresponds to departures that occur in the G/G/1/∞ model but are ignored in the G/G/1/K model. Let $L_{i,j}$ denote the queue length immediately after event i at iteration j in phase two. The PSM-D algorithm is shown as follows:

PSM-D Algorithm

Phase 1:

input: $\langle \alpha_{1,N-1} \rangle$ (interarrival time sequence);
$\langle \delta_{1,N} \rangle$ (service time sequence);
K (the buffer size); 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. 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/∞ queue.

end.

Phase 2:

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

output: $\langle M_{1,2N} \rangle$ (event type sequence);
$\langle D'_{1,m} \rangle$ (phase-two job departure time sequence);
$\langle L'_{0,2N} \rangle$ (phase-two queue length sequence).

begin

1. $j = 0$. For each $l, 1 \leq l \leq P$, compute steps 2 to 4 in parallel.
2. if j=0, then
$$b_l = \frac{2N(l-1)}{P} + 1; \; h_l = \frac{2Nl}{P}.$$
3. $$L_{b_l-1,j} = \begin{cases} min(L_{b_l-1}, K) & j = 0, \\ L_{b_l-1,j-1} & \text{otherwise.} \end{cases}$$
4. For $b_l \leq i \leq h_l$,
$$L_{i,j} = \begin{cases} min(L_{i-1,j} + 1, K) & L_i = L_{i-1} + 1, \\ max(L_{i-1,j} - 1, 0) & L_i = L_{i-1} - 1. \end{cases}$$

$$\mathrm{M_{i,j}} = \begin{cases} arrive & L_{i,j} > L_{i-1,j}, \\ depart & L_{i,j} < L_{i-1,j}, \\ arrive - lost & L_{i,j} = L_{i-1,j} = K, \\ depart - ignored & L_{i,j} = L_{i-1,j} = 0. \end{cases}$$

5. $j = j + 1$. 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 3.

6. $L'_i = L_{i,j}$ for $1 \leq i \leq 2N$.

7. Create an empty sequence $\langle D' \rangle$. For $1 \leq i \leq 2N$, if $M_{i,j} = depart$, append E_i to $\langle D' \rangle$; else if $M_{i,j} = depart - ignored$, append 0 to $\langle D' \rangle$

end.

In phase two, step 2 computes batch boundaries at the first iteration. Step 3 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 4 computes the queue length sequence and decides each event's type. A departure event is ignored in phase two if the queue is empty when the event occurs. The computation of the queue length sequence in step 4 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 4:

Property 4.1: For all $1 \leq i \leq 2N$ and $1 \leq j < j_{con}$, $L_i \geq L_{i,j} \geq L_{i,j_{con}}$.

Property 4.1 states that the queue length as a function of time of any iteration is a lower bound on the queue length of the previous iterations. Step 5 checks the convergence of the simulation on queue lengths and advances the simulation to the next iteration if the simulation is not converged. Step 6 creates a queue length sequence. Step 7 creates a departure time sequence $\langle D'_{1,m} \rangle$, in which $m \leq N$ is the number of jobs that enter the queue and are not lost. Thus, 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 time $O(j_{con}(N/P + logP))$ to execute with P processors because step 1 to 3 take a constant of time; step 4, 6, and 7 take time $O(N/P)$; step 5 requires time $O(logP)$. 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. Experiments which will be discussed in section 4.3 show that j_{con} is often a small constant, in which case the PSM-D algorithm has the same complexity as the GLM algorithm.

Figure 1: At simulation time t, both the $G/G/1/\infty$ queue and the $G/G/1/K$ queue are not empty. No approximation error is introduced in this case.

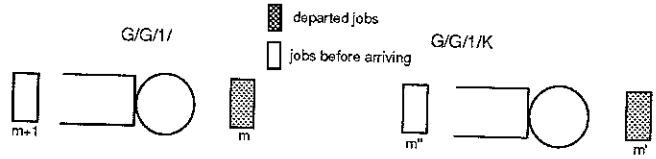

Figure 2: At simulation time t, both the G/G/1/∞ queue and the G/G/1/K queue are empty. No approximation error is introduced in this case.

4.2 Service Time Transformation of the PSM-D Algorithm

In this section, we discuss the transformation technique used in phase-two of the PSM-D algorithm to compute the departure time sequence. Recall that the PSM-D algorithm synchronizes the departure times of the G/G/1/∞ and G/G/1/K models such that departures of the two models occur at the same times. To explain this algorithm, let job m and job m' of the G/G/1/∞ and the G/G/1/K models, respectively, depart at some simulation time t and let job m'', where $m'' \geq m' + 1$, be the next job to enter the G/G/1/K queue that is not lost. To compute the next departure time (i.e. the departure time of job m'' denoted $D''_{m''}$) in the G/G/1/K model, there are four cases which are illustrated in Figures 1 to 3.

Case 1: At simulation time t, both the $G/G/1/\infty$ queue and the $G/G/1/K$ queue are not empty.

The next departure in the G/G/1/K model should occur at time $(t + \delta_{m''})$. However, the PSM-D algorithm will assign $(t + \delta_{m+1})$ to $D''_{m''}$. If $m'' \neq m + 1$, the algorithm effectively *reassigns* the service time of job $m + 1$ (i.e., δ_{m+1}) to job m''. Because job service times are modeled by identically distributed random variables, for sufficiently large N such re-association of service times will not change the probability distribution of service times. Thus, no approximation error is introduced in this case.

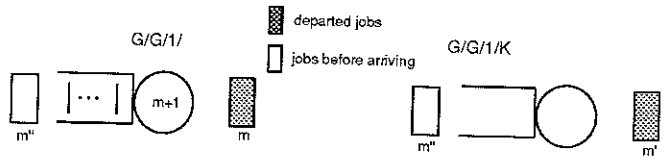

Figure 3: At simulation time t, the G/G/1/K queue is empty and the G/G/1/∞ queue is not empty. An approximation error occurs.

Case 2: At simulation time t, both the G/G/1/∞ queue and the G/G/1/K queue are empty.

At simulation time t, the queues of both models become empty. Based on property 4.1, we know that $m'' = m + 1$. The algorithm will assign $(A_{m+1} + \delta_{m+1})$ to $D''_{m''}$. In this case, $D''_{m''}$ is correctly computed. No approximation error is introduced.

Case 3: At simulation time t, the G/G/1/K queue is empty and the G/G/1/∞ queue is not empty.

At simulation time t, jobs $D_{m+1}, \ldots, D_{m''-1}, m'' > m + 1$, which are lost in the G/G/1/K queue, are in the G/G/1/∞ queue. When job m'' arrives, if the G/G/1/∞ queue is not empty and job $m + k$, for $m + 1 \leq m + k \leq m'' - 1$, is in service, the algorithm will *ignore* $D_{m+1}, \ldots, D_{m+k-1}$ and assign D_{m+k} to $D''_{m''}$ while the correct departure time of job m'' is $A''_m + \delta''_m$. Thus, an approximation error of $(A''_m + \delta_{m''}) - (D_{m+k-1} + \delta_{m+k})$ will occur. If the job service times are modeled by an identically distributed random variable, the expected approximation error is thus equal to $(A_{m''} - D_{m+k-1})$. The algorithm effectively *reassigns* a service time $(D_{m+k} - A_{m''})$ to job m''.

Case 4: At simulation time t, the $G/G/1/K$ queue is not empty and the $G/G/1/\infty$ queue is empty.

From property 4.1, this case is impossible.

Therefore, only case 3 causes approximation errors in the departure time distribution. Thus the frequency with which case 3 occurs governs the accuracy of the proposed approximation. The next section presents experiments using the PSM-D algorithm for M/M/1/K and M/D/1/K models.

4.3 Experiment Results and Analysis for G/G/1/K queues

This section reports experiment results with the PSM-D algorithm on an M/M/1/K and M/D/1/K models. Table 1 shows some results of the experiment. In general, it

Table 1: Numbers of iterations for an M/M/1/K model using the PSM-D 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 table 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

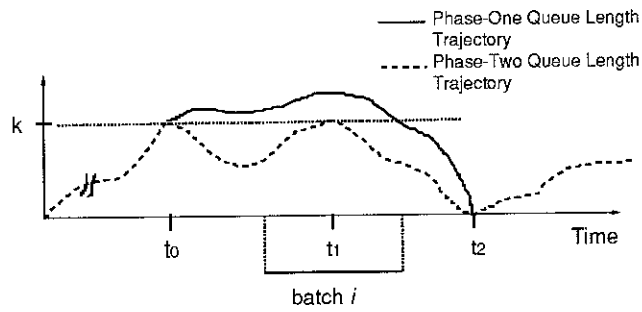

Figure 4: The first loss occurs at t_0. An E_F occurs at t_0 and t_1 and an E_E occurs at t_2. Sub-trajectories from t_0 to t_1 and from t_1 to t_2 are two propagation segments.

takes only a few iterations to converge. Many iterations are required only when both the *traffic intensity* (ratio of the mean arrival rate, denoted λ, to the mean service rate, denoted μ) is close to 1 and the buffer size is large. We discuss this phenomenon next.

Recall that phase two of the PSM-D algorithm is a process of transforming an initial estimated trajectory into a more accurate trajectory. When no job is lost, phase-two is unnecessary. Otherwise, the phase-one trajectory after the first loss has to be modified. This is illustrated in Figure 4. Let t_0 be a time at which 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. If at some 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 variable 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.

The number of iterations required for convergence is bounded by the number of batches spanned by the longest propagation segment. Therefore:

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

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 [10]:

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

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 occurring is:

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

Assume that $P_s(1, K) \simeq P_s(\rho, K)_{\rho \to 1}$. A proof that $P_s(\rho, K)$ has a minimum at $\rho = 1$ for any $K \geq 1$ is given in Appendix A.

The results of the PSM-D 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 when the buffer size is smaller than 5 for the M/D/1/K model. Because

Table 2: Normalized approximation errors of the M/M/1/K and the M/D/1/K model using the PSM-D algorithm. Each table 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
M/M/1/K	.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
M/D/1/K	.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

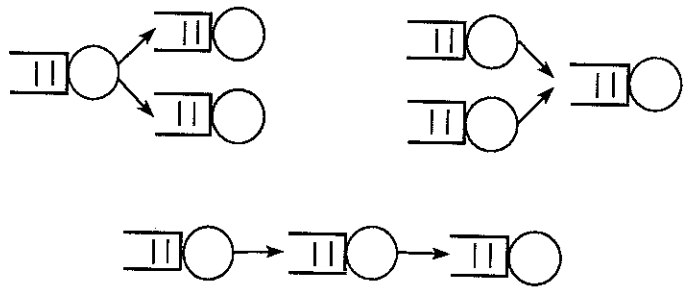

Figure 5: Three basic components of feed-forward queueing networks.

the approximation is *biased*, whenever case 3 occurs, the approximate service time is decreased by some amount. If the service time distribution has a small variance, when case 3 occurs it is likely that the approximate service time is smaller than the expected service time (i.e. $1/\mu$). When the buffer size is small, case 3 will occur more frequently and thus the resultant error will be more significant. Therefore, the algorithm causes more approximation errors when K is small for an M/D/1/K model, of which the service time distribution has a zero variance. In section 5, we study an alternative partial state matching algorithm for a G/D/1/K model whose buffer size is small.

4.4 Feed-Forward Networks of G/G/1/K Queues

The PSM-D algorithm can be applied to simulate acyclic feed-forward, finite buffer queueing networks. In an acyclic feed-forward queueing network, jobs enter the system from *source* queues and depart the system from *destination* queues. Because the network contains no cycles, each job visits each queue at most once. There are three basic components for feed-forward networks: fork, merge, and tandem (Figure 5). All feed-forward queueing networks are compositions of these three basic structures.

We present results of simulating three networks (a merge, fork, and tandem network) with the PSM-D algorithm. Each of these three models contains 7 homogeneous M/M/1/K queues (Figure 6). We assume that a job departing from a fork queue has equal probability of joining any of the queues connected to the server output. Also, the service times of a job at each server are independent. For any of these three models, because there are no cycles, the PSM-D algorithm can be applied queue by queue in an breadth-first order starting from any of the the source queues.

In section 4, it is shown that when no losses occur, the second phase of the PSM-D algorithm is not required. Recall from Theorem A.1 in Appendix A that the worst performance in convergence speed occurs when $\lambda/\mu = 1$. To study the worst case performance for the models in Figure 6, in our experiments $\lambda = \mu = 100(jobs/timeunit)$ for all source queues. For other parameters, we let $K = 10$ and $P = 64$.

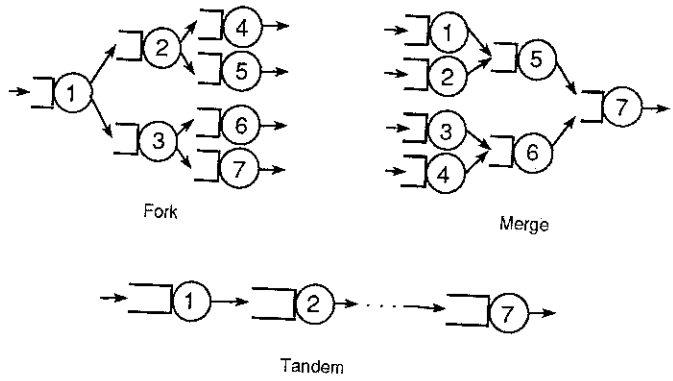

Figure 6: Feed-forward queueing models. Each model contains 7 queues.

Each model is executed 10 times. Each run simulates 10^5 external arrivals. In the merge network model, queues 1 to 4 are independent source queues each of which thus admits $10^5/4$ arrivals for each simulation run. Tables 3 to 5 compare the simulation results to a sequential simulation. They show that the PSM-D simulations produce very close results for all three models. Note that, differences in simulation results are inevitable because the sequential simulation uses an event-driven approach and different sequences of random variate and hence job service times and interarrival times are used in the sequencial and the time-parallel simulations.

In all three cases, the number of iterations for any queue is at most 2. Because the PSM-D algorithm requires processor synchronization only at the end of each iteration, we assume that the cost of processor synchronization is negligible. Then for the fork and merge network models, in which 2 iterations are required for each queue, the execution time is $O[7 * ((N/P + logP + logN) + 2 * (N/P + logP))] = O(33 * 10^3)$. A sequential simulation takes $O(7 * 10^5)$ to execute. Therefore, an order of 21-fold of speedup can be achieved. Similarly, for the tandem queue model in which only 4.1 iterations in average are required for 7 queues, an order of 40-fold of speedup can be achieved.

4.5 Circuit-Switched Communication Networks

In this section, we illustrate a pratical application of the PSM-D algorithm: a circuit-switched data communication network which uses time-division multiplexing (TDM) routing to select one of two paths for arriving messages. The network is shown in Figure 7. In the model, messages are transmitted from the source node (node 0) to the destination node (node 7) through two time-multiplexed circuits. The message buffer of each node except the source has only a finite capacity. A message loss will occur if the message that arrives finds the buffer full.

All nodes are assumed to have the same message service rate of 100 messages per

Table 3: A comparison of the queue lengths for the fork network model in which K=10 for all queues. Each data is an average of 10 runs. The processor number for the PSM-D simulation is 64.

Queue	Lost	Iteration	PSM-D	Sequential
1	Y	2.0	5.00	5.01
2	Y	1.1	0.82	0.82
3	Y	1.0	0.82	0.81
4	N	0	0.29	0.29
5	N	0	0.29	0.29
6	N	0	0.29	0.29
7	N	0	0.29	0.29

Table 4: A comparison of the queue lengths for the merge network model in which K=10 for all queues. Each data is an average of 10 runs. The processor number for the PSM-D simulation is 64.

Queue	Lost	Iteration	PSM-D	Sequential
1	Y	2.0	5.02	5.01
2	Y	2.0	5.01	5.02
3	Y	2.0	4.95	5.04
4	Y	2.0	5.05	5.00
5	Y	2.0	8.79	8.79
6	Y	2.0	8.79	8.78
7	Y	2.0	8.96	8.97

Table 5: A comparison of the queue lengths for the tandem network model in which K=10 for all queues. Each data is an average of 10 runs. The processor number for the PSM-D simulation is 64.

Queue	Lost	Iteration	PSM-D	Sequential
1	Y	2.0	4.97	4.99
2	Y	2.0	4.05	4.09
3	Y	2.0	3.59	3.58
4	Y	2.0	3.30	3.30
5	Y	2.0	3.08	3.09
6	Y	2.0	2.93	2.92
7	Y	2.0	2.79	2.77

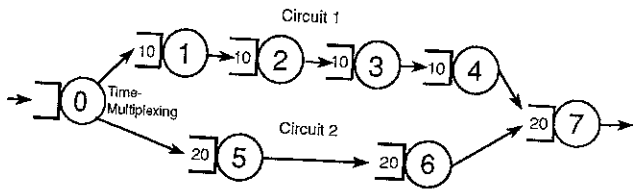

Figure 7: A model of a circuit-switched data communication network with time-division multiplexing. The number appeared in the buffer of each node is the maximum number of messages that the node can hold.

Table 6: A comparison of the results for the TDM network model using the PSM-S simulation and a sequential event-driven simulation. Each data is an average of 10 runs. Here, F, L, and γ denote fraction lost, length, and throughput of a queue, respectively. The number of processors for the PSM-D simulation is 64.

Queue	Iteration	F_{PSM}	F	L_{PSM}	L	γ_{PSM}	γ
0	0	0	0	-	-	100.2	99.80
1	2.0	.580%	.578%	1.10	1.10	33.11	33.06
2	2.0	.480%	.220%	1.10	1.10	32.95	32.98
3	2.0	.185%	.104%	1.10	1.10	32.89	32.94
4	1.8	.250%	.058%	1.10	1.10	32.81	32.92
5	0.3	1.7e-5	1.2e-5	2.62	2.64	66.48	66.54
6	0.3	1.5e-5	5.0e-6	2.63	2.64	66.48	66.54
7	2.0	3.53%	3.55%	9.82	10.82	95.76	95.92

time unit. The message arriving rate at the source node is set to be 100 messages per time unit as well. The lengths and interarrival times of the messages that arrive at the source node are assumed to be exponentially distributed. The lengths of the messages remain unchanged as the messages travel through the network; therefore service times at the servers are dependent.

Each time unit is partitioned into 3 equal-length time slots. The access to the circuits are time-multiplexed such that a message departing from the source node is routed to circuit 1 if it arrives at time slot 1, and circuit 2, otherwise.

The model is executed 10 times using the PSM-D and a sequential event-driven approach, respectively. Each simulation run admits 10^5 messages. The results are shown in Table 6.

5 THE G/D/1/K Partial State Matching Algorithm

In section 4.3 we show that for the G/D/1/K model, the PSM-D algorithm produces more significant approximation errors when K is small. In this section, we discuss an alternative algorithm for this case. For a FCFS G/D/1/K queue, in which each job has a fixed service time, the system state contains the queue length, the next job arrival time, and the first job remaining service time (FRST). The remaining service times of all other jobs other than the first one in the queue are the same as the fixed service time. Let N be the number of arrivals and P be the number of processors. Let β_l, F_{RST_l}, and f_l denote the initial queue length, the FRST, and the final queue length of batch l, respectively. A G/D/1/K partial state matching algorithm (PSM-S) is described by the following steps:

PSM-S Algorithm

input: $\langle \alpha_{1,N-1} \rangle$ (interarrival time sequence);
K (the buffer size); N (the number of jobs); D (the service time).

output: $\langle D_{1,N} \rangle$ (job departure time sequence);
$\langle L_{0,2N} \rangle$ (queue length sequence).

begin

1. Apply the GLM algorithm of section 2.1 to compute $\langle A_{1,N} \rangle$.
2. For each $l, 1 \leq l \leq P$,
$$b_l = \frac{N(l-1)}{P} + 1;\ h_l = \frac{Nl}{P}.$$
3. For all $l, 1 \leq l \leq P, \beta_l = 0$.
4. For each $l, 1 \leq l \leq P$, compute steps 5 to 6 in parallel.
5. $F_{RST_l} = 0$.
6. Compute departure time sequence $\langle D_{b_l,h_l} \rangle$ and queue length sequence $\langle L_{b_l,h_l} \rangle$ via a sequential simulation.
7. If $\beta_{l+1} = f_l$, for all $1 \leq l < P$, exit; otherwise $\beta_{l+1} = f_l$ for all $1 \leq l < P$ and go to step 3.

end.

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

Table 7: Numbers of iterations of the M/D/1/K model using the PSM-S algorithm and the 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 table 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

fixed (i.e. 0). Thus, the initial unmatched set contains only the state variable for the queue length. Table 7 compares the number of iterations required for convergence of the proposed G/D/1/K partial state matching simulation with a *full state matching* (FSM) 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 7 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, or the number of processors. 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 $\lambda/\mu = 1$. 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

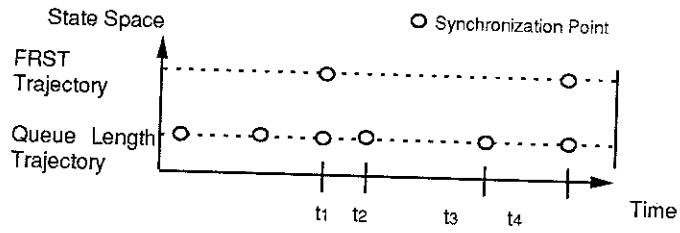

Figure 8: Without approximation, the full state matching simulation converges on both FRST and queue length. The longest propagation distance is $(t_4 - t_1)$. With approximation on FRST sequence, the partial state matching simulation converges only on queue length. The longest propagation distance is $(t_3 - t_2)$.

Table 8: The normalized approximation errors for the M/D/1/K model using the PSM-S algorithm. Each table 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
M/D/1/K	.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	4.88	10.28
	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

iterations to converge.

The reason that the partial state matching outperforms the full state matching is illustrated in Figure 8. In 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 4.3. The trade-off of execution time is simulation accuracy. Table 8 shows that the partial state matching simulation produces close results. The approximation errors of the PSM-S algorithm with respect to the FSM algorithm is less than 1% unless λ/μ is near 1 and $K \geq 20$.

6 Concluding Remarks

Partial state matching simulation artificially fixes some state variables in the unmatched set by using approximate variable 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 partial state matching algorithms, PSM-D and PSM-S, are proposed in this paper to simulate FCFS G/G/1/K queueing models. The PSM-D algorithm uses approximate job departure times through service time transformation; the PSM-S algorithm uses approximate first job remaining service times. Algorithm PSM-D introduces more significant errors when the model has a small buffer and has a small job service time variance. Algorithm PSM-S is introduced to alleviate this shortcoming of algorithm PSM-D.

Algorithms PSM-D and PSM-S both produce small approximation errors in general cases. The worst performance in convergence speed for both algorithms occurs when $\lambda/\mu = 1$. Arguments are made to explain this phenomenon. The PSM-D algorithm is applied to feed-forward queueing networks. The results suggest that the algorithm produces small approximation errors and can achieve significant speedup.

Algorithm PSM-D and PSM-S can be optimized to further reduce the time required for simulation. In particular, when a prefix of the trajectory being computed converges, the processor(s) assigned to this prefix continue(s) recomputing the converged prefix until the simulation exactly converges. The algorithms could be augmented by a load distribution algorithm where each batch partition is updated dynamically to exclude the converged prefix to achieve higher speedup.

REFERENCES

References

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

[2] Bagrodia, R., Chandy, K. M., Liao W. T. An Experimental Study On the Performance of the Space Time Simulation Algorithm. *Proceedings of the Sixth Workshop of the Parallel and Distributed Simulations* (1992), 159-168.

[3] Chandy, K. M. and Misra, J. Asynchronous Distributed Simulation via a Sequence of Parallel Computations. *Commun. ACM* 24, 11 (Nov. 1981), 198-205.

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

[5] Fujimoto, R. M. Parallel Discrete Event Simulation. *Commun. ACM* 33, 10 (Oct. 1990), 31-53.

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

[7] Hills, W. D., Steele, G. L. Data Parallel Algorithms. *Commun. ACM* 29, 12 (Dec. 1986), 1170-1183.

[8] Jefferson, D. Virtual Time. *ACM Transactions of Programming Languages and Systems* 7, 3 (July 1985), 404-425.

[9] Jones, D. W. 1986. Concurrent Simulation: An Alternative to Distributed Simulation. *Proceedings of the 1986 Winter Simulation Conference* (Dec.1986), 417-423.

[10] Kleinrock, L. *Queueing Systems.* vol. 1 (1975), Wiley-Interscience.

[11] Lander, R. E., Fischer, M. J. Parallel Prefix Computation. *Journal of ACM* 27 (1980), 831-838.

[12] Lin, Y. B., Lazowska, E. A Time-Division Algorithm for Parallel Simulation. *ACM TOMACS* 1, 1 (Jan. 1991), 73-83.

[13] Mitra, D., Mitrani, I. Control and Coordination Policies for System with Buffers. *ACM SIGMETRICS Performance Evaluation Review* 17, 1 (May 1989), 156-164.

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

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

Appendix A

Lemma A.1 For $0 \leq \rho < 1$, $P_s(\rho, K)$ is strictly decreasing.

Proof: We show that for $0 \leq \rho < 1$

$$\begin{aligned}\frac{\partial P_s}{\partial \rho} &= \frac{K\rho^{K-1}-(K+1)\rho^K}{1-\rho^{K+1}} + \frac{(K+1)\rho^K(\rho^K-\rho^{K+1})}{(1-\rho^{K+1})^2} - 1 \\ &= \frac{[K\rho^{K-1}-(K+1)\rho^K](1-\rho^{K+1})+(K+1)\rho^K(\rho^K-\rho^{K+1})-(1-\rho^{K+1})^2}{(1-\rho^{K+1})^2} < 0.\end{aligned}$$

Because $(1-\rho^{K+1})^2 \geq 0$, the above inequality holds if the numerator of the expression is less than zero. This is derived as follows:

$$\begin{aligned}&[K\rho^{K-1} - (K+1)\rho^K](1-\rho^{K+1}) + (K+1)\rho^K(\rho^K - \rho^{K+1}) - (1-\rho^{K+1})^2 \\ &= (K\rho^{K-1} - K\rho^K - \rho^K)(1-\rho^{K+1}) + (\rho^K + K\rho^K)(\rho^K - \rho^{K+1}) - (1 - 2\rho^{K+1} + \rho^{2K+2}) \\ &= -\rho^{2K+2} + \rho^{2K} + 2\rho^{K+1} - K\rho^K - \rho^K + K\rho^{K-1} - 1 \\ &= \rho^{2K}(1-\rho)(1+\rho) + K\rho^{K-1}(1-\rho) - \rho^K(1-\rho) - (1-\rho)(\textstyle\sum_{i=0}^{K}\rho^i) \\ &= (1-\rho)[\rho^{2K}(1+\rho) + K\rho^{K-1} - \rho^K - \textstyle\sum_{i=0}^{K}\rho^i] \\ &= (1-\rho)[(\rho^{2K+1} - \rho^K) + (\rho^{2K} - \rho^K) + (K\rho^{K-1} - \textstyle\sum_{i=0}^{K-1}\rho^i)] < 0. \ \square\end{aligned}$$

Lemma A.2 For $\rho > 1$, $P_s(\rho, K)$ is strictly increasing.

Proof: When $\rho > 1$, $P_s(\rho, K) = P_K$. Again, we take the first partial derivative of P_K with respect to ρ:

$$\frac{\partial P_K}{\partial \rho} = \frac{(K\rho^{K-1} - K\rho^K) + (\rho^{2K} - \rho^K)}{(1-\rho^{K+1})^2}.$$

Because $(1-\rho^{K+1})^2 > 0$ for $\rho > 1$, to prove the lemma it suffices to show that:

$$(K\rho^{K-1} - K\rho^K) + (\rho^{2K} - \rho^K) > 0 \quad \rho > 1.$$

Dividing both sides of the above inequality by ρ^{K-1} and letting $g(\rho, K)$ be the resulting left-hand-side expression, we have:

$$g(\rho, K) = (K - K\rho - \rho + \rho^{K+1}).$$

Because $g(1, K) = 0$ and $\partial g/\partial \rho = (K\rho^K - K) + (\rho^K - 1) > 0$, clearly, $g(\rho, K) > 0$ for all $\rho > 1$ and $K \geq 1$. $\square$

Theorem A.1 The minimum of $P_s(\rho, K)$ occurs at $\rho = 1$ for $\rho \geq 0$.

Combining Lemma A.1 and A.2 completes the proof of theorem A.1. $\square$