Determining Initial States for Time-Parallel Simulations

Jain J. Wang and Marc Abrams

TR 92-53

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

October 22, 1992

Determining Initial States for Time-Parallel Simulations

Jain J. Wang and Marc Abrams

Department of Computer Science, Virginia Tech
Blacksburg, VA 24061-0106
{jaywang,abrams}@vtopus.cs.vt.edu

Abstract

Time-parallel simulations exploit parallelism by partitioning the time domain of a simulation model. Exploiting temporal parallelism requires predicting future states of a simulation model. A poor prediction of future states may cause extensive recomputation so that a time-parallel simulation requires more real time to execute than a corresponding sequential simulation. Recurrent states of a simulation model provide potential temporal parallelism. In this paper, we propose a time-parallel simulation method which uses a pre-simulation to identify recurrent states. An approximation technique is suggested to extend the the class of simulation models which can be simulated efficiently using our time-parallel simulation. A central server system and a virtual circuit of a packet-switched data communication network modeled by closed queueing networks are experimented with the proposed time-parallel simulation. Experiment results suggest that the proposed approach can exploit massive parallelism while yielding accurate results.

1 Introduction

Time-parallel simulations decompose the time domain of simulation models. The maximum degree of parallelism in space-parallel simulations, such as the Time Warp and Chandy and Misra approaches, is often constrained by the topology of a simulation model. Time-parallel simulations, in contrast, poses no inherent upper bound on parallelism. However, partitioning the time domain of a simulation model is often difficult and application dependent [14].

Time-parallel simulations partition the simulation time interval into a number of sub-intervals. A processor is assigned to simulate the simulation model for each sub-interval with a *guessed* initial state. The simulation terminates when the initial state of each sub-interval and the final state of its preceding sub-interval have met a pre-defined *matching condition*. A simulation which requires all states to be matched is referred to as a *full state matching* simulation, otherwise, a *partial state matching* simulation [18].

Therefore, exploiting temporal parallelism requires prediction of future simulation states. For some time-parallel simulation approaches, a poor prediction of future states may cause extensive recomputation and results in a time-parallel simulation that requires more real time to execute than a corresponding sequential simulation [19].

Lin and Lazowska [15] propose a time parallel approach which partitions the time domain at time points where a pre-determined *recurrent* state occurs. A state s_r is said to be recurrent if it is guaranteed that the simulation model when starting from s_r will eventually return to this state. Let P denote the number of processors available. Lin and Lazowska's algorithm can achieve linear speedup given that the pre-determined recurrent state occurs no less than (P-1) times and these occurrences are evenly distributed in the simulation time interval. However, Lin and Lazowska do not discuss how recurrent states are identified.

In this paper, we propose using Markovian modeling and a *pre-simulation* to identify recurrent states to obtain temporal parallelism. The rest of this paper is organized as follows. In section 2, we discuss a mapping of simulation models to Markov chains. Section 3 describes the use of Markov chain models and a pre-simulation to explore temporal parallelism by identifying the most frequently occurring states. Section 4 presents the algorithm of the proposed time-parallel simulation. Experiments with a central server system and a virtual circuit in a communication network with sliding-window flow control using the proposed time-parallel simulation are described in section 5. Experiment results are also compared with sequential simulations as well as multiple-replica simulations. Finally, a summary is given in section 6.

2 Markovian Modeling

It is well known that if $\{X(t), t \geq 0\}$ is an irreducible, aperiodic, and positive recurrent Markov chain with a state space $S \in \{1, 2, \ldots\}$, then there exists a random variable X with a probability distribution $\pi = \{\pi_i, i \in S\}$, such that $X(t) \Rightarrow X$, where $\Rightarrow$ denotes weak convergence [5, 8]. Here, π represents the *stationary* distribution of $\{X(t), t \geq 0\}$. A discrete event simulation can be viewed as a stochastic process $\{X(t), t \geq 0\}$ with state space S and a discrete time parameter t and each event is represented by a state transition. If the purpose of the simulation is to study the long term behavior of the target system, then the goal of the simulation is to estimate the expected value $Ef(X)$, where $f : S \rightarrow R$ is some real valued function on S.

Let N be the number of events to be simulated and let $t_i, 1 \leq i \leq N$, be the simulation time at which event i occurs. If the simulation model has a discrete state space and the probability distribution of $X(t_i), 1 < i \leq N$, depends only on $X(t_{i-1})$, then the simulation model can be directly modeled by a discrete-time Markov chain. A mapping from such a simulation model to a discrete-time Markov chain is described as follows. For each state of the simulation model, create a corresponding state for the Markov chain. Let u and v be any two states in the Markov chain which correspond

to state u' and v' in the simulation model, respectively. Then the occurrence of each event can be modeled by a single state transition in the Markov chain. Let $e_{u',v'}$ denote an event upon whose occurrence the simulation model changes its state from u' to v' and $p(e_{u',v'})$ denote the probability that $e_{u',v'}$ occurs when the simulation model is in state u'. Then transition probability of the corresponding Markov chain from state u to v is given by:

$$p_{u,v} = p(e_{u',v'}).$$

Let n denote the number of states in the resulting Markov chain. If $N \geq cn$ for any integer $c > 1$, then some state(s) will occur at least c times in a sequence of N state transitions. Assume that the resulting Markovian chain is ergodic (i.e. each state of the Markov chain is positive recurrent and aperiodic). Let A be the $m \times m$ transition probability matrix of the Markov chain. Then the stationary distribution of the Markov chain π can be determined by solving the following equations [2]:

$$\begin{array}{ll} \pi A = \pi & \text{and} \\ \sum_{k=1}^{m} \pi_k = 1. & \end{array} \tag{1}$$

For ergodic Markov chains, the limiting distribution is asymptotically equivalent to the stationary distribution and can be determined by the following:

$$\lim_{n \to \infty} A^n = \begin{bmatrix} \pi \\ \pi \\ \bullet \\ \bullet \\ \bullet \\ \pi \end{bmatrix}. \tag{2}$$

Here, (1) provides a set of linear equations. Parallel algorithms for solving linear equations are given in [1]. Let $A_{i,j}$ denote the j^{th} element of row i in A. Then for (2), a lower bound n', where n' is a positive integer, can be found such that for all $k > n', 1 \leq i, j \leq m$, $|A^k_{i,j} - A^\infty_{i,j}| \leq \epsilon$ for some small value $\epsilon > 0$ [2, pp.90–92]. Computing $A^{n'}$ is a *prefix* problem [12]. Greenberg *et al.* have proposed efficient parallel algorithms for prefix problems [7].

When the state space of the target system is very large (or even infinite in many cases), using equations (1) and (2) to solve the stationary distribution is too computationally prohibitive. For this case, simulation becomes an attractive alternative to estimate $E(f(X))$.

3 Temporal Decomposition Using Recurrent States

Direct solution of Markov chains to obtain stationary distribution discussed in the last section is feasible for small simulation models. In this section, we describe tem-

poral decomposition of a simulation model using recurrent states and propose using a simulation approach to obtain an estimated state distribution.

Let $T_j(s), 1 \leq j \leq N+1$, denote the simulation time of the j^{th} occurrence of state $s \in S$ and let $r_m(s)$ denote $T(s)_{m+1} - T(s)_m, 0 < m \leq N$, which is the simulation time between the m^{th} and the $(m+1)^{th}$ occurrence of state s. Assume that for each $s \in S$, $r_m(s)$ are identically and independently distributed random variables for all m. Then the *recurrence period* of state s is defined to be:

$$E(r_m(s)) = \frac{1}{\pi_s}. \tag{3}$$

State s is said to be *positive recurrent* when $E(r_m(s)) < \infty$. If there exist some $c > 1$ such that $E(r_m(s)) < \frac{N}{c}$, then temporal parallelism may be obtained through partitioning the simulation time interval into sub-intervals at the time points where state s occurs. Hence, the initial and the final states (in this case, state s) of these sub-intervals are determined. The states used for partitioning the simulation time interval are called *matching states* (since they determine the matching condition between sub-intervals). To obtain a high degree of parallelism, we are interested in finding the *most frequently occurring state(s)* (MFOS), which is the state with the minimum recurrence period.

When a numerical or an analytical solution to find the MFOS is not feasible, we propose to use a *pre-simulation* to obtain an estimated MFOS. The pre-simulation algorithm is described by the following steps.

Algorithm PS (Pre-Simulation):

1. For each processor $p_i, 1 \leq i \leq P$, simulate the entire system with a randomly chosen initial state until N/P events have been simulated. During simulation, record the number of occurrences of each state that p_i generates. (Each p_i chooses a different random state.)

2. For each p_i, identify the MFOS and its occurrence rate among the states that p_i has generated. If there are more than one MFOS, pick one randomly.

3. Sum the occurrence rates for state(s), if any, which is (are) selected by more than one processor. Identify the state which has the highest over-all occurrence rate. If this results in more than one state, choose one randomly.

The state obtained in step 3 will be used as the matching state for the time-parallel simulation. This heuristic algorithm does not guarantee selecting the actual MFOS which has the minimum recurrence period. However, for our time-parallel simulation, it is not crucial that the real MFOS be selected as long as the selected state has a relatively high occurrence rate. Note that, in this algorithm we use only one matching state. For a simulation model which has a large state space and the occurrences of

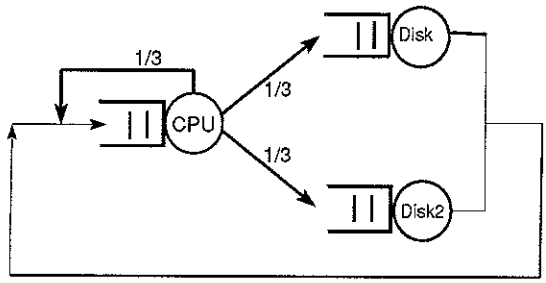

Figure 1: A Central Server System. The number associated with each arc represents the probability of the transition corresponding to the arc.

states are uniformly distributed, multiple matching states can be used to obtain a higher degree of parallelism.

4 Combinatorial Method

The pre-simulation algorithm is applicable only to those simulation models with a discrete-state space. In this section, we present a time-parallel simulation algorithm (algorithm C) which allows simulation models with continuous-state space by combining the imbedded Markov chain method [11] and the partial state matching technique [18].

Consider a FCFS M/G/1 queue model. Let Q and $FRST$ denote the queue length and the first job remaining service time of queue at simulation time t, respectively. Then the simulation model is described by a stochastic process: $\{Y(t), t > 0\}$, where Y is a 2-tuple: $\langle Q, FRST \rangle$. Kleinrock shows that the model can be described equivalently by the imbedded (queue length) Markov chain [11, pp.174-180]. However, computing the transition probabilities for the imbedded Markov chain is not always possible. In the next section, we discuss an approximation technique which allows the method of imbedded Markovian chain to be applicable to a larger class of models.

4.1 Approximate Markovian Modeling

Consider a central server system (Figure 1) which consists of a CPU, two disks, and a set of jobs which circulate around the system. Each job is associated with a *length*, which is a constant defined to be the product of the job's service time and the device's service rate. The lengths of the jobs follow some probability distribution and remain unchanged as the jobs travel around the system. Thus, this model does not have a product form solution. Each job after being served by the CPU is routed to one of the two disks or back to the CPU for service. The buffer sizes of the devices are infinite and the time required for the job to move between devices is ignored.

A queueing model with feedback causes both optimistic and conservative algorithms to perform poorly [13]. For this central server model, Wanger and Lazowska [19] have shown that for any conservative parallel algorithms, 3.67 is the upper bound of speedup. A number of researchers [6, 10, 16, 20] have used this model as a benchmark for their parallel simulation protocols and none of them could achieve a speedup better than 3.

Let $\{Q_1(t), Q_2(t), Q_3(t), FRST_1(t), FRST_2(t), FRST_3(t), t > 0\}$ be the stochastic process which describes the central server model. Then $Q(t) = \{Q_1(t), Q_2(t), Q_3(t)\}$ is an imbedded queue length process. Let μ_i denote the job service rate of queue i, and $Q_i(n)$ denote the queue length of queue i immediately after the occurrence of event n, respectively. To simplify $Q(t)$, we make the following assumptions:

Assumption 1: The departure processes of all queues are mutually independent such that the probability that the next job departure will occur at queue i, denoted p_i, is given by:

$$p_i = \begin{cases} 0 & Q_i(n) = 0, \\ \frac{\mu_i}{\sum_{j=1}^{M} \mu_j,\ Q_j(n)>0} & Q_i(n) > 0, \end{cases} \tag{4}$$

where M is the number of queues in the system.

Assumption 2: The *state holding time* (i.e. simulation time between any two consecutive departure events) is the same.

With assumption 1, $Q(t)$ can be easily modeled by a Markov chain. With assumption 2, the computation of job arrival and departure times is not required. Also, the pre-simulation does not require an event list because the newly created event is always the next event to be executed. These assumptions largely simplify and will reduce the real execution time required for the pre-simulation.

4.2 Algorithm C (Combinatorial)

The proposed combinatorial algorithm can be described by the following steps:

The Combinatorial Algorithm:

1. Model the target simulation model with a stochastic process using the mapping algorithm discussed in section 2.

2. If the process produced by step 1 has a continuous-state space, identify an imbedded Markov chain for the process, possibly using the approximation technique of section 4.1.

3. If the state space of the discrete-space process produced in step 1 or 2 is small, use (1) or (2) to solve the MFOS; else execute algorithm PS to obtain an estimated MFOS.

4. Implement a time-parallel simulation with partial state matching using the (estimated) MFOS obtained from step 3 as the matching state.

The execution time of the pre-simulation is $O(N(a+1)/P)$, where a is the cost for recording the system state and is a function of the number of state variables of the system. The execution time of the time-parallel simulation (Step 4) is $O(N/P)$.

5 Applications

In this section we apply algorithm C to two examples, a central server system and a virtual circuit of a data communication network with a sliding-window flow control.

5.1 Central Server System

We use the central server system described in section 4.1 as the simulation model for our first experiment. We first let the system contain only 3 jobs. The job lengths are generated by an exponentially distributed random variable with a mean of 0.01. All three devices have the same job service rate. Since there are only 3 jobs, the service times are effectively 3 small numbers. Using the approximation technique of section 4.1, the embedded queue length process $Q(n) = \{Q_1(n), Q_2(n), Q_3(n), n = 1, 2, \ldots, N\}$ can be modeled by a Markov chain of 10 states. Figure 2 shows the state transition matrix of the resulting Markov chain. Solution of equations (1) or (2) yields the stationary distribution: $\langle \frac{3}{10}, \frac{2}{10}, \frac{2}{10}, \frac{1}{10}, \frac{1}{15}, \frac{1}{15}, \frac{1}{45}, \frac{1}{45}, \frac{1}{90}, \frac{1}{90} \rangle$. State 0 (all three jobs are in the CPU's queue), which has the the smallest recurrence period, is hence chosen as the matching state.

To validate the accuracy of the occurrence frequencies obtained from the approximate Markov chain, we execute the original simulation model 10 times using a sequential simulation. Each run uses a different seed and simulates 10^5 arrival events. A comparison of these state probability distributions is given in Table 1. It shows that solution of the approximate Markov chain generates a very close probability distribution to the results of an exact simulation.

In our experiment, we implement the time-parallel simulation on a single processor workstation such that each processor is emulated by a process which executes sequentially. That is, a new process starts only when the current process has reached its termination condition and stopped. In the rest of this paper, the term *process* is equivalent to *processor* in a real multiprocessor implementation.

Initially, each device has a job in its queue. Because state 0 is chosen to be the matching state, it serves as the final state of each simulation sub-interval as well as the initial state of each simulation sub-interval except the first one. Therefore, a final state is considered to be reached when all three jobs are in the CPU's queue regardless of the remaining service times (hence it is partial state matching as described in [18]).

In the experiment, we let $P = 10^2$ and $N = 10^5$. Therefore, for a perfectly balanced loading, each processor should simulates 10^3 events. Because the expected number of

$A =$

	0	1	2	3	4	5	6	7	8	9
0	1/3	1/3	1/3	0	0	0	0	0	0	0
1	1/2	1/6	0	1/6	1/6	0	0	0	0	0
2	1/2	0	1/6	1/6	0	1/6	0	0	0	0
3	0	1/3	1/3	1/9	0	0	1/9	1/9	0	0
4	0	1/2	0	0	1/6	0	1/6	0	1/6	0
5	0	0	1/2	0	0	1/6	0	1/6	0	1/6
6	0	0	0	1/2	1/2	0	0	0	0	0
7	0	0	0	1/2	0	1/2	0	0	0	0
8	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	1	0	0	0	0

0 : <3,0,0> 1 : <2,1,0> 2 : <2,0,1> 3 : <1,1,1> 4 : <1,2,0>
5 : <1,0,2> 6 : <0,2,1> 7 : <0,1,2> 8 : <0,3,0> 9 : <0,0,3>

Figure 2: The State Transition Matrix of the Central Server System. Each state of the Markov chain corresponds to a state $\langle Q_1, Q_2, Q_3 \rangle$, where Q_1, Q_2, and Q_3 are the queue lengths of the CPU, Disk1, and Disk2, respectively.

Table 1: A comparison of the stationary probability distributions obtained from solution of the approximate Markov chain and an exact simulation.

States	Approximate Markov Chain	Exact Simulation
0	30%	28.6%
1	20%	21.4%
2	20%	21.1%
3	10%	9.9%
4	6.7%	6.8%
5	6.7%	6.8%
6	2.2%	1.7%
7	2.2%	1.6%
8	1.1%	1.0%
9	1.1%	1.1%

Table 2: A comparison of average queue lengths of the central server model. Each number is an average of 10 runs.

Algorithm	CPU	Disk1	Disk2	Arrivals Simulated
Parallel	2.160	0.422	0.425	100264
Sequential	2.188	0.406	0.406	100000

events in each sub-interval is only 3.33 (i.e. $\frac{1}{3/10}$), each process is expected to simulate a large number of sub-intervals. Therefore, for each process, the termination condition is defined as, "state 0 occurs and at least 10^3 arrival events have been simulated by the process". Although this simple algorithm will cause more than N arrival events to be simulated, because state 0 occurs with a very high frequency, experiment results show that only an average of 2.64 more arrival events are simulated by each processor (Table 2).

The experiment executes the model 10 times using the combinatorial simulation and a sequential simulation, respectively. Table 2 compares the expected queue lengths. The differences for the CPU, Disk1, and Disk2 queues are 1.2%, 3.8%, and 4.2%, respectively. Because the load of each processor is nearly perfectly balanced, and the cost of computing the stationary distribution (using equation (1) or (2)) is constant, a linear speedup can be achieved.

To illustrate that the combinatorial simulation can achieve massive parallelism and still maintain high accuracy, we vary the value of P, ranging from 10 to 10^5, and compare the results with a sequential simulation as well as a multiple-replica simulation in which each processor simulates the entire simulation model independently and the results of these independent runs are averaged together [9]. For the multiple-replica simulation, each simulation run simulates N/P events.

We also increase the number of jobs and the number of events simulated from 3 and 10^5 to 300 and 10^7, respectively. Initially, each queue contains 100 jobs. The service times of the jobs are generated by an exponentially distributed random variable with a mean of 0.01. All other parameters and assumptions of the system are not changed from the first experiment.

The number of possible states of the imbedded queue length process of this model is about 4.5×10^4. Because of the large state space, we use the pre-simulation method of algorithm PS (Section 3) to estimate the MFOS; the MFOS is the state in which all 300 jobs are in the CPU queue. Figure 3 and Table 3 compare the expected queue lengths obtained from the three simulations. For the time-parallel simulation, each processor simulates no more than $N/P + 7$ events on the average for any value of P.

The time-parallel simulation outperforms the multiple replica simulation in simulation accuracy for any number of processors. The difference is significant when the number of processors exceeds 1000. We also use an uniformly distributed random

Table 3: A comparison of the average queue lengths of the central server model. In the Table, TP and MR represent the time-parallel and multiple-replica simulations, respectively. The average queue lengths obtained from the sequential simulation are 299.05, 0.473, and 0.473, respectively.

P	Simulation	CPU	Disk1	Disk2
10	TP	299.06	0.471	0.471
	MR	299.02	0.488	0.490
100	TP	299.06	0.467	0.468
	MR	298.55	0.719	0.729
1000	TP	299.06	0.470	0.469
	MR	293.62	3.180	3.193
10000	TP	299.06	0.470	0.469
	MR	177.18	61.817	61.306

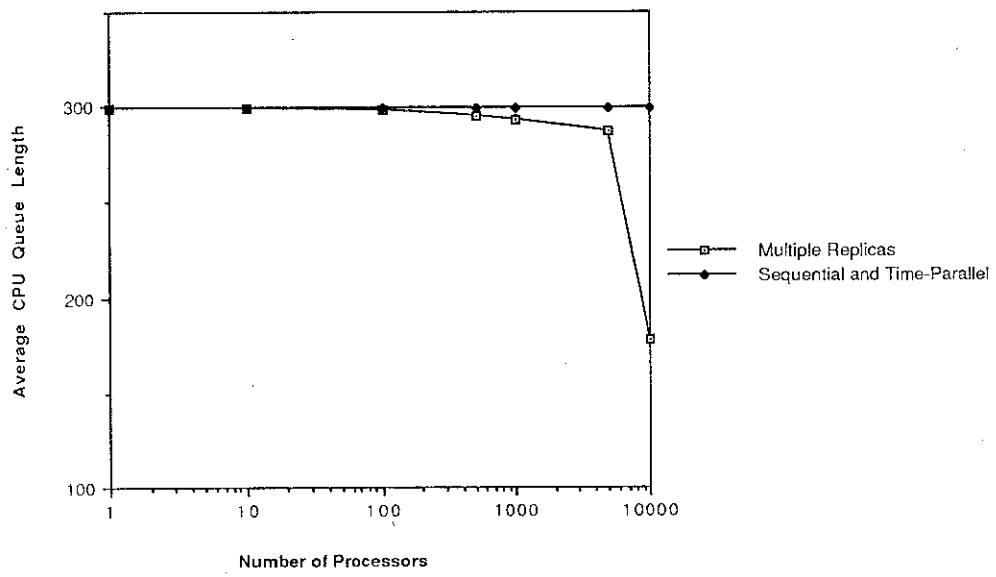

Figure 3: A comparison of queue lengths of the CPU for the sequential, time-parallel, and multiple replica simulations.

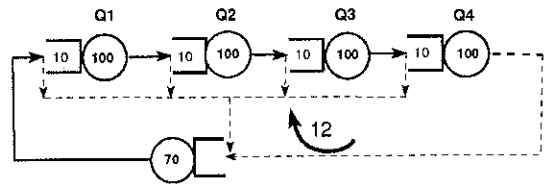

Figure 4: A closed queueing network model for a 4-node virtual circuit with sliding-window control. The window size of the VC is 12. The number appears in each queue and in the server is the buffer size (packets) and the service rate of the node, respectively. Dash lines are the paths for acknowledgments.

variable to generate the queue lengths. Results similar to those shown in Tables 2 and Figure 3 are observed for both cases (i.e., 3 or 300 jobs in the system).

5.2 Virtual Circuit of Communication Networks

In our second example, we use a virtual circuit in a packet-switched data communication network with sliding-window control (Figure 4). In a packet-switched data communication network, data to be transmitted across the network are grouped into *packets*. Before a packet is transmitted, a path from the source node to the destination node, called a *virtual circuit* (VC), has to be established. Sliding-window is a flow control mechanism which limits the number of packets simultaneously in transit in a VC [17, pp.171-191]. This number is called the *window size* of the VC. Each node, representing a switch or gateways of the network, in the VC has a finite size input queue. When an arriving packet to a node finds the buffer of the node full will be lost. We assume that when a packet is lost or arrives at the destination node, an acknowledgment will be sent back to the source node immediately and the delay of transmitting the acknowledgments is negligible. A virtual circuit with these assumptions can be modeled by a closed queueing network as shown in Figure 4. In our example, there are 4 nodes in the VC and the window size is 12. The extra node in the bottom (i.e. $Q5$) is artificially added to model the flow control mechanism and the job arrival process of the source node. Initially, all 12 jobs are in $Q5$.

Again, this VC model can be modeled by a continuous-space process in which there exists an imbedded discrete-space queue length process. Using assumptions 1 and 2, the imbedded process is simplified and can be modeled by a Markov chain. We apply algorithm C to the VC model and compare the results with a multiple replica simulation and a sequential simulation, in which $N = 10^7$. The packet lengths are generated by an exponentially distributed random variable. The MFOS obtained by algorithm PS for all value of P is $\langle 1, 0, 1, 0 \rangle$, where the i^{th} element in the 4-tuple is the queue length of Qi. The initial state of the system is thus $\langle 0, 0, 0, 0 \rangle$, since all jobs are in $Q5$ initially. A comparison of the expected queue lengths is shown in Table 4.

Table 4: A comparison of the average queue lengths of the virtual circuit model. In the Table, TP and MR represent the time-parallel and multiple-replica simulations, respectively. The average queue lengths obtained from the sequential simulation for queues 1 to 4 are 1.275, 1.708, 1.865, and 1.986, respectively.

P	Simulation	Q1	Q2	Q3	Q4
10	TP	1.275	1.706	1.863	1.984
	MR	1.275	1.711	1.870	1.99
100	TP	1.275	1.709	1.865	1.985
	MR	1.276	1.705	1.875	1.869
1000	TP	1.273	1.704	1.861	1.980
	MR	1.273	1.701	1.852	1.966
10000	TP	1.260	1.687	1.838	1.951
	MR	1.260	1.650	1.757	1.813

The results show that the combinatorial algorithm does only slightly better than the multiple replica simulation in simulation accuracy. This is because the expected first passage time from the initial state of the multiple replica simulation (i.e. $\langle 0,0,0,0\rangle$) to the matching state of the time-parallel simulation (i.e. $\langle 1,0,1,0\rangle$) is short. Therefore, these two simulations have close initial transient.

6 Summary and Final Remarks

A combinatorial time-parallel simulation which can achieve massive parallelism by identifying the recurrent states of the simulation model has been proposed. The degree of parallelism that can be obtained is proportional to the number of events and is not constrained by the topology of the simulation model. Approximate Markovian modeling is suggested to extend the the class of simulation models that the combinatorial simulation can be applied.

In this paper we apply the combinatorial simulation to two closed queueing network models. Experiment result show that our approach can obtain massive parallelism and still maintain high accuracy. We also compare the combinatorial simulation with multiple replica simulation. The combinatorial simulation always yields more accurate results than multiple replica simulation because in the combinatorial simulation, the use of the MFOS as the initial state reduces the initial transient of the simulation. However, if the initial transient is weak or if the ratio of N/P is large, the multiple replica simulation is more efficient due to the overhead of computing the MFOS required by the combinatorial simulation.

A concern of the combinatorial algorithm is that the pre-simulation has to record the number of state occurrences in order to identify the MFOS. The efficiency of the

recording may dominate the cost of the pre-simulation. Thus, an efficient state recording algorithm is essential for large simulation models. For acyclic queueing networks of any size, another time-parallel simulation method proposed in a previous paper [18] can be applied. In this paper, we only focus on queueing network applications, although other applications are also possible for the proposed time-parallel simulation.

References

[1] Aki, S. G. *The Design And Analysis of Parallel Algorithms.* (1989), Prentice Hall.

[2] Bhat, U. N. *Elements of Applied Stochastic Process.* 2nd ed. (1984), Jhon Wiley & Sons.

[3] Bertsekas, D., Gallager, R. *Data Networks.* 2nd ed. (1992), Prentice Hall.

[4] Chandy, K. M., Herzog, U., Woo, L.S. Parametric Analysis of Queueing Networks. *IBM J. Research and Development.* Vol 19, No. 1, (Jan. 1975), 43-49.

[5] Hordijk, A., Iglehart, D. L, Schassberger R. Discrete Time Methods for Simulating Continuous Time Markov Chains *Adv. Appl. Prob.* Vol 8, 772-778 (1976)

[6] Fujimoto, R. M. Lookahead in Parallel Discrete Event Simulation. *Proceedings of International Conference on Parallel Processing.* St. Charles, IL (August 1988)

[7] Greenberg, A. G., Lander, R. E., Paterson, M., Galil, Z. Efficient Parallel Algorithms for Linear Recurrence Computation. *International Proceeding Letters Computer.* Vol 15, No. 1, (Aug. 1982), 31-35.

[8] Heidelberger, P. Variance Reduction Techniques for the Simulation of Markov Process, I:Multiple Estimates *IBM J. Research and Development.* Vol 24, No. 5, (Sept. 1980), 570-581.

[9] Heidelberger, P. Statistical Analysis of Parallel Simulations. *Proceedings of the 1986 Winter Simulation Conference* (Dec. 1986), 290-295.

[10] Jones, D. W., Chou C., Renk, D., Bruell S. C. Experience with Concurrent Simulation *Proceedings of the 1989 Winter Simulation Conference* (Dec. 1989), 756-764.

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

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

[13] Leung, E., Cleary, J., Lomow, G., Baezner, D., Unger, B. The Effect of Feedback on the performance of conservative algorithms *Proceedings of 1989 SCS Multiconference on Advances in Distributed Simulation*, 44-49.

[14] Lin, Y. B. *Understanding the Limits of Optimistic and Conservative Parallel Simulation.* Technical Report 90-08-02, Department of Computer Science and Engineering, University of Washington, (1990).

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

[16] Reed, D. A., Malony, A. D., McCredie, B. D. Parallel Discrete Event Simulation Using Shared Memory. *IEEE Transaction on Software Engineering* Vol. 14, No. 4 (April), 541-553.

[17] Schwarz, M. *Telecommunication Networks.* (1987), Addison Wesley.

[18] Wang, J., Abrams, M. *Approximate Time-Parallel Simulation of Queueing Systems with Losses.* To be appeared in the *Proceedings of the 1992 Winter Simulation Conference.* An extension of this paper appears as Technical Report 92-50, Computer Science Department, Virginia Tech. (Sept. 92)

[19] 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.

[20] Wanger, D. B., Lazowska, E. D., Bershad B. N. Techniques for Efficient Shared-Memory Parallel Simulation. *Distributed Simulation 1989.* Society for Computer Simulation International, San Diego, CA (March 1989)