Journal of Applied Mathematics and Stochastic Analysis, **11**:3 (1998), 397-409.

PERFORMANCE LIMITATIONS OF PARALLEL SIMULATIONS[1]

LIANG CHEN
Lucent Technologies
Wireless Networks Group
67 Whippany Road, Room 14D-270
Whippany, NJ 07981 USA

RICHARD F. SERFOZO
Georgia Institute of Technology
School of Industrial and Systems Engineering
Atlanta, GA 30332 USA

(Received November, 1997; Revised February, 1998)

This study shows how the performance of a parallel simulation may be affected by the structure of the system being simulated. We consider a wide class of "linearly synchronous" simulations consisting of asynchronous and synchronous parallel simulations (or other distributed-processing systems), with conservative or optimistic protocols, in which the differences in the virtual times of the logical processes being simulated in real time t are of the order $o(t)$ as t tends to infinity. Using a random time transformation idea, we show how a simulation's processing rate in real time is related to the throughput rates in virtual time of the system being simulated. This relation is the basis for establishing upper bounds on simulation processing rates. The bounds for the rates are tight and are close to the actual rates as numerical experiments indicate. We use the bounds to determine the maximum number of processors that a simulation can effectively use. The bounds also give insight into efficient assignment of processors to the logical processes in a simulation.

Key words: Parallel Simulation, Distributed Processing, Speedup Bounds, Time Warp, Virtual-Time Conservation Principle, Linearly Synchronous, Random Time Transformation.

AMS subject classifications: 68U20, 65Cxx.

[1] This research was supported in part by NSF grants DDM-922452 and DMI-9457336.

1. Introduction

Analysts rely on discrete-event simulations to evaluate the performance of large-scale, complex systems, such as telecommunications networks and computer systems. Existing simulation packages based on serial processing of events, however, are often inadequate for large, realistic systems. The alternative is to use simulations based on parallel processing. Several protocols for parallel simulations have been developed for general systems as well as for special purpose applications. For a survey of these protocol, see Fujimoto [5]. Each protocol has its strengths and weaknesses depending on the application at hand and the mechanisms and techniques used for synchronizing the parallel processors. There have been several studies of the speedup of parallel simulations for particular protocols and applications; see for instance [1, 6, 8, 10]. The approach in these studies is to model both simulator and simulated system as a single Markovian stochastic system at a detailed level. Another approach is to capture the major characteristics of a parallel simulation protocol by using coarser performance measures based on macro-level assumptions that are not too sensitive to detailed properties of the simulation protocol and the simulated system.

The present paper is such a macro-level study of parallel simulations. The aim is to give insights into the following issues:

- What is the maximum number of processors that can be usefully employed in a parallel simulation?
- Does the structure of the system being simulated limit the maximum potential processing rate of the simulation?
- How do non-homogeneous processors differ from homogeneous ones in affecting a simulation's execution rate?
- How is the maximum potential processing rate of a simulation affected by processor scheduling: the way in which processors are assigned to execute events of the processes?
- How is the processing rate (in real time) of a simulation related to the throughput rates (in virtual time) of the system being simulated?

In this paper, we study the potential performance of a parallel simulation of a general discrete-event system consisting of several interacting logical processes. Events associated with the logical processes are executed by a set of processors over an infinite time horizon. The simulation may be synchronous or asynchronous, and conservative or optimistic. Although we present our results in the setting of discrete-event simulations, they also apply to other types of discrete-event systems using distributed computations.

The evolution of a logical process i in the simulation is presented by its virtual time (simulated time) $T_i(t)$. This is a random function of the events it processes in the real time (simulation time) t. In a synchronous parallel simulation, the virtual times of all processes are equal ($T_i(t) = T_j(t)$ for all i, j and t). We consider systems with a significantly weaker *virtual-time conservation principle* that all processes have the same long-run average virtual speeds (as defined in the next section). This does not mean that all logical processes or processors in a simulation must be homogeneous, but only that their virtual time flows have the same rate in the long run. We show that this principle is satisfied for a wide class of simulations that are *linearly synchronous*. In such a simulation, the virtual times of the processes may vary considerably as long as their difference from each other (or from the simulation's global virtual time) is of the (linear) order $o(t)$ as t tends to infinity. In an aggressive Time Warp simulation, for instance, it is often the case that the difference between a processor's virtual time and the global virtual time is bounded by a constant (this may even be a constraint inherent in the protocol). Such simulations are therefore linearly synchronous. We also show that the linearly synchronous property is equivalent to the virtual-time conservation principle.

For the class of simulations that are linearly synchronous, we show that their simulation processing rates have a natural, simple representation in terms of the throughput rates of the simulated systems. The proof of this is based on the fundamental property that the number of events the simulation executes at real time is a random time transformation of the number of events in the simulated system (see expression (2)). This time transformation idea relating the simulation to the system being simulated is implicit in studies of parallel simulations, but it has not been exploited explicitly as we do here. The analysis also uses sample-path properties of stochastic processes and strong laws of large numbers.

After characterizing linearly synchronous processes, we study their maximum potential processing rates under the following processor scheduling strategies.

Autonomous processor assignments. Each processor is assigned to a subset of processes, and the subsets are disjoint.

Group processor assignments. Disjoint groups of processors are assigned to disjoint subsets of processes (a special case is global scheduling – all processors are assigned to all processes).

Using the relation between the processing rate of the simulation and the system throughputs, we derive upper bounds on the simulation's processing rate under these processor scheduling strategies. The bounds are tight and our numerical examples indicate that they tend to be close to the actual processing rates. We describe how the bounds can be used to obtain efficient processor scheduling strategies.

The main interest in the bounds is that they show how a simulation may be limited by the structure of the system being simulated. A conventional view is that if K homogeneous processors are employed in a parallel simulation, then in an ideal situation, there would be a simulation protocol such that the speedup of the simulation is approximately K. Our bounds show, however, that for most applications, the speedup is much less than K no matter what simulation protocol is used. The reason is that the efficiency of a parallel simulation may be limited by the structure of the simulated system. We give expressions for the maximum number of processors that can be usefully employed in a simulation; more processors beyond this maximum will not add to the speed of the simulation.

In a related study of parallel simulations of queueing networks, Wagner and Lazowska [11] show that, under a conservative protocol, the structure of the network limits the parallel simulation speed. They also gave an upper bound on the speedup of a specific queueing network simulation. Another study on speedup bounds for self-initiating systems is Nicol [9]. Also, Felderman and Kleinrock [4] give several performance bounds for asynchronous simulations of specific applications under the Time Warp protocol. The ideas and analysis techniques used in these studies are geared to the particular models they consider and do not apply to the general setting herein covering both conservative and optimistic protocols and a wide class of protocols that obey the virtual-time conservation principle.

The rest of the paper is organized as follows. Section 2 describes the class of linearly synchronous parallel simulations that satisfy the virtual-time conservation principle. Section 3 contains upper bounds on the simulation speed under autonomous and group processor assignments. Section 4 gives a numerical example, and Section 5 gives insight into obtaining efficient processor assignments.

2. Conservation Principle and Linearly Synchronous Simulations

In this section, we discuss a conservation principle that is satisfied by a large class of what we call linearly synchronous simulations. For such a simulation, we derive a simple relation between the simulation's processing rate and the throughput rates of the system being simulated.

We shall consider a parallel simulation of a discrete-event system, denoted by S, consisting of n logical processes that are simulated by a set of processors over an infinite time horizon. We also write $S = \{1, \ldots, n\}$. Associated with a logical process is a stream of events for the system (e.g., a stream of service times, waiting times and departure times for a node in a queueing network). The events of one process might depend on events from other processes. Each event contains several items of information including its virtual timestamp – the virtual time that it is scheduled to occur in the simulated system. In this section, we do not specify how the processors in the simulation are assigned to the processes, such assignments are the subject of the next section. Messages are exchanged between processes (or processors) for synchronizing or coordinating their executions of events in the simulation. Following the standard convention, we also call these messages events. An event in the simulation is said to be a *true event* if it actually occurs in the simulated system S. The other events are called *synchronizing events*; they are generated only for synchronizing the executions of the processes. The simulation may be asynchronous or synchronous and its protocol may be conservative (like the Chandy-Misra protocol) or optimistic (like an aggressive Time Warp protocol). Aside from its processing rate information, our analysis does not require other details of the protocol.

The speed of the simulation is described in terms of general processing rates of events as follows. Simulation time (the time the simulation has been running) is referred to as *real time* and is denoted by t. The simulated time (elapsed time) of process i in the simulation is called its *virtual time*, and is denoted by $T_i(t)$. This time is usually the minimum of the timestamps of the events of the process that may have been executed but are still subject to being canceled or rolled back (uncommitted events). This $T_i(t)$ is a random function of real time t that may decrease periodically but it eventually tends to infinity. Under a conservative protocol, the $T_i(t)$ will typically be nondecreasing in real time, but under an optimistic protocol, this virtual time may decrease periodically due to rollbacks. The simulation speed of process i is measured by its *virtual speed*

$$v_i = \lim_{t \to \infty} t^{-1} T_i(t).$$

This limit and the others herein are with probability one. We assume that the limit v_i exists and is a positive constant. This is a rather natural law of large numbers for an infinite horizon simulation. The v_i can also be interpreted as the average amount of virtual time simulated for process i in one unit of real time. Let $N_i(\tau)$ denote the number of (true) events in the system S associated with process i in the virtual time interval $(0, \tau]$. We define the *virtual throughput* of the process $i \in S$ as

$$\lambda_i = \lim_{\tau \to \infty} \tau^{-1} N_i(\tau). \tag{1}$$

This λ_i is the rate in virtual time of true events passing through process i. We assume this limit exists and is a positive constant for each process i. Keep in mind that the throughput rate λ_i is a parameter of the simulated system S while v_i is a parameter of the simulation.

The relevant parameters of the simulation are as follows. We shall view the parallel simulation of the system S also as a discrete-event system and denote it by $\bar{S}$. The two systems S and $\bar{S}$ have the same network structures and routing probabilities for the true events, but the systems run on different time scales. Another difference is that $\bar{S}$ may contain synchronizing events while S does not. Let $\bar{N}_i(t)$ denote the number of true events simulated at process i in the real time interval $(0,t]$. (The bar over a parameter denotes that it is associated with the simulation $\bar{S}$). This quantity is related to the variables of the system S by

$$\bar{N}_i(t) = N_i(T_i(t)). \tag{2}$$

In other words, the simulation's cumulative work stochastic process $\bar{N}_i$ is a random time transformation of the system's throughput process N_i. This is a fundamental relation for parallel simulations that allows one to express performance parameters of the simulation in terms of parameters of the system being simulated. Although this time transformation is implicit in some studies, we have not seen it exploited as explicitly as we do here. The rate of the stochastic process $\bar{N}_i$ is called the *simulation processing rate* or execution rate of process i. This rate exists and is given by

$$\bar{\lambda}_i = \lim_{t \to \infty} t^{-1} \bar{N}_i(t) = v_i \lambda_i. \tag{3}$$

This expression follows since, from the existence of v_i and λ_i, we have

$$\begin{aligned} \lim_{t \to \infty} T^{-1} \bar{N}_i(t) &= \lim_{t \to \infty} t^{-1} T_i(t) T_i(t)^{-1} N_i(T_i(t)) \\ &= \lim_{t \to \infty} t^{-1} T_i(t) \lim_{t \to \infty} T_i(t)^{-1} N_i(T_i(t)) \\ &= v_i \lambda_i. \end{aligned}$$

The last limit uses the facts that $T_i(t) \to \infty$ as $t \to \infty$ since $v_i > 0$ exists.

The following is a rather natural conservation principle for the simulation described above.

Definition 1: The simulation satisfies a *virtual-time conservation principle* if $v_i = v_j$ for each $i, j \in S$ (the virtual speeds of all of its processes are equal).

This principle says that the long-run average virtual speeds are equal, although in the short run they may vary considerably. To see what type of simulation satisfies this principle, consider the simulation's *global virtual time*

$$GVT(t) = \min_{1 \leq i \leq n} T_i(t).$$

This minimum of the processes' virtual times at the simulation time t is a measure of the simulation's progress. Typically, the $GVT(t)$ is nondecreasing in t and an event that is executed at a virtual time less than the GVT is committed or put in the fossil collection forming the simulation output; such events are never rolled back. The following is another notion related to the conservation principle.

Definition 2: The simulation described above is *linearly synchronous* if

$$t^{-1}[T_i(t) - GVT(t)] \to 0, \text{ as } t \to \infty, \text{ for each } i \in S. \tag{4}$$

This says that the virtual times never get too far away from each other – their derivation is of the order $o(t)$ on the time interval $[0, t]$. This is satisfied, for instance, when the deviation of the virtual times is bounded by some constant; this may even be a natural or desired constraint built into the simulation protocol. Note that at the termination of a simulation, all virtual times are equal (otherwise, the simulation would not be complete). This suggests that for an infinite time horizon representation of a simulation, the virtual times should be relatively close to each other for a large t (as in a linearly synchronous system).

The following is a characterization of the conservation principle. It says, in particular, that a system satisfies this principle if and only if it is linearly synchronous.

Theorem 3: *The following statements are equivalent.*

(*a*) *The simulation satisfies the virtual-time conservation principle.*

(*b*) $\bar{\lambda}_i/\lambda_i = \bar{\lambda}_j/\lambda_j$, *for any processes* $i, j \in S$.

(*c*) $\bar{\lambda}_A/\lambda_A = \bar{\lambda}_B/\lambda_B$, *for any subsets* A, B *of* S.

(*d*) *The simulation is linearly synchronous.*

Proof: The equivalence of (*a*) and (*b*) follows directly from $v_i = \bar{\lambda}_i/\lambda_i$, which was noted in (3). Clearly (*b*) is a special case of (*c*), and, (*c*) follows from (*b*) by using the fact that $\bar{\lambda}_A = \sum_{i \in A}\bar{\lambda}_i = \sum_{i \in A} v_i \lambda_i$ and $v_i = v_j$. Finally, from the definition $v_i = \lim_{t \to \infty} t^{-1} T_i(T)$, we have

$$t^{-1}GVT(t) = \min_j t^{-1}T_j(t) = \min_j v_j.$$

Using these limits in (4), it follows that (*d*) is equivalent to $v_i = \min_j v_j$, for each process i, which in turn is equivalent to (*a*). □

Statement (*c*) in Theorem 3 relates the processing rates of a linearly synchronous simulation to the throughputs of the system. The equivalence of (*b*) and (*c*) says that the equality in (*b*) for any pair of processes also applies to any pair of subsets of S. We will use these relations in the next section to derive performance bounds for simulations.

If the simulated system S is a closed system, it may be easier to express the simulation's processing rate in terms of visit ratios. Assume that there are a fixed number of true events that circulate in the closed system S. The visit ratio ρ_i of a process $i \in S$ is the average number of visits a true event makes to the process i between successive visits to process 1 (an arbitrary fixed process). For a large class of systems with Markovian routing of events, the visit ratios satisfy the equations

$$\rho_j = \sum_{i \in S} \rho_i p_{ij}, \quad j \in S, \tag{5}$$

where $\rho_1 = 1$ and p_{ij} is the routing probability that a (true) event in the system S moves from process i to process j. For these systems, the throughputs are related to visit ratios by $\lambda_i/\rho_i = \lambda_j/\rho_j$. The following immediate consequence of Theorem 3 is a relation between the processing rates of the simulation and the visit ratios.

Corollary 4: *If the simulation is linearly synchronous, then*

$$\bar{\lambda}_A/\rho_A = \bar{\lambda}_B/\rho_B, \textit{ for any subsets } A \textit{ and } B \textit{ of } S.$$

3. Upper Bounds of Simulation Speed

In this section we discuss upper bounds on the simulation speed under several processor-sharing scheduling strategies that are suggested for parallel simulations.

We shall consider a parallel simulation of a linearly synchronous system as described above. The system consists of n logical processes that are simulated by K processors. We will study the following strategies for assigning processors to the processes:

Autonomous Processor Assignments: The set of processes S is partitioned into K subsets $S_1,\ldots,S_K$ and processor k is assigned to simulate the events associated with the subset of processes S_k. Note that K cannot exceed the number of processes n. The special case in which each S_k is a single process is called *single-processor assignments.*

Group Processor Assignments: The set of processes S is partitioned into M subsets $S_1,\ldots,S_M$ and the set of processors $G = \{1,\ldots,K\}$ is partitioned into M subsets or groups $G_1,\ldots,G_M$. The group G_m is assigned to simulate the events for the set S_m. The idle processors of G_m reside in a pool and whenever a set of events need execution and the pool is not empty, then one processor is taken from the pool, and it executes one event in the event set with the smallest timestamp. The special case in which all K processors are assigned to all the processes ($M = 1$) is called *global processor assignments.*

To measure the efficiency or quality of the simulation, we will use the total *processing rate* $\bar{\lambda}_S = \sum_{j=1}^{n} \bar{\lambda}_j$ of the true events in the simulation. Recall that $\lambda_S = \sum_{j=1}^{n} \lambda_j$ is the total throughput of the system S being simulated. An important parameter affecting the simulation processing rate is the execution rate μ_k of processor k, which is the number of events that processor k can process continuously, including any overhead time. The μ_k is therefore the maximal processing rate of processor k or a bound on the rate of events processed by that processor. We also write

$$\mu_B = \sum_{k \in B} \mu_k, \quad B \subset G.$$

The following results give upper bounds for the simulation processing rate under the processor scheduling strategies described above.

Theorem 5: *Under autonomous processor assignments*

$$\bar{\lambda}_S \le \min\{\mu_G, \lambda_S \min_{1 \le k \le K} \mu_k/\lambda_{S_k}\}. \tag{6}$$

Under group processor assignments,

$$\bar{\lambda}_S \le \min\{\mu_G, \lambda_S \min_{1 \le m \le M}\{\mu_{G_m}/\lambda_{S_m}, \alpha_{G_m} \min_{j \in S_m} \lambda_j^{-1}\}\}, \tag{7}$$

where

$$\alpha_{G_m} = \mu_{G_m}^{-1} \sum_{k \in G_m} \mu_k^2 \tag{8}$$

is the average maximal execution rate for the processor group G_m. In particular, under global processor assignments,

$$\bar{\lambda}_S \le \min\{\mu_G, \lambda_S \alpha_G \min_{1 \le j \le n} \lambda_j^{-1}\}. \tag{9}$$

Theorem 5 clearly reduces to the following when all K processors used in the simulation are homogeneous (here $\alpha_{G_m} = \mu$).

Corollary 6: *Suppose all processors are homogeneous with common execution rate*

μ. Then under autonomous processor assignments,

$$\bar{\lambda}_S \leq \mu \min\{K, \lambda_S \min_{1 \leq k \leq K} \lambda_{S_k}^{-1}\}. \tag{10}$$

And under group processor assignments,

$$\bar{\lambda}_S \leq \mu \min\{K, \lambda_S \min_{1 \leq m \leq M}\{\,|G_m|/\lambda_{S_m}, \min_{j \in S_m} \lambda_j^{-1}\}\}, \tag{11}$$

where $|G_m|$ denotes the size of the set G_m.

Remarks: (*a*) The inequalities in the preceding results are tight – there are elementary examples of systems in which the processing rate $\bar{\lambda}_S$ equals the specified upper bounds. Furthermore $\bar{\lambda}_S$ is apparently fairly close to these bounds for standard examples, as the next section illustrates.

(*b*) In the bounds on $\bar{\lambda}_S$ in (6), (7) and (9), the first term just states the obvious property that this rate is limited by the maximal processing rate μ_G of K processors. The second terms in these bounds, however, are more interesting. They reveal how the processing rate may be limited by the system parameters.

(*c*) Be mindful that the bounds do not consider idleness of processors that may be experienced, for instance, when a large subgroup of processors serves a small number of processors. In these cases, one should decrease μ_k to account for idleness. A reasonable compensation for idleness would be to multiply α_{G_m} by λ_{S_m}/μ_{G_m}, which represents the "traffic intensity" of events or the fraction of time that processors in G_m are busy (necessarily $\lambda_{S_m} \leq \mu_{G_m}$, otherwise the simulation would be unstable). This compensation may be good for space-division protocols, but it may not be needed for more efficient time and space division protocols that typically have small processor idleness.

(*d*) For closed networks as discussed in Corollary 4, the upper bounds on $\bar{\lambda}_S$ in the preceding results have obvious analogues in terms of visit ratios; just replace λ_j by ρ_j.

The major interest in the upper bounds of the simulation processing rate is that they give insight into the maximum number of processors that can be usefully employed in the simulation. To describe this, we say that K^* is the *maximum effective number of processors* for the simulation if its processing rate $\bar{\lambda}_S$ may be constrained by the system as the number of processors K exceeds K^*. For convenience, one might want to assume that the processors are ordered such that $\mu_1 \geq \mu_2 \geq \ldots$. It follows that the K^* is the smallest value of K for which the "processor constraint" μ_G, for all larger K, exceeds the "system constraint" as represented by the second terms in the bounds above. The following is a formal statement of this.

Corollary 7: *Under autonomous processor assignments*

$$K^* = \min\left\{K: \sum_{k=1}^{K'} \mu_k \leq \lambda_S \min_{1 \leq k \leq K'} \mu_k/\lambda_{S_k}, \text{ for } K' \geq K\right\}.$$

Under group assignments,

$$K^* = \min\left\{K: \sum_{k=1}^{K'} \mu_k \leq \lambda_S \min_{1 \leq m \leq M}\{\mu_{G_m}/\lambda_{S_m}, \alpha_{G_m} \min_{j \in S_m} \lambda_j^{-1}\}, \text{ for } K' \geq K\right\}.$$

In particular, for global processor assignments of homogeneous processors, the K^* is the smallest integer greater than $\min_{1 \leq j \leq n} \lambda_j^{-1}$.

We now prove the main result.

Proof of Theorem 5: First, suppose the simulation uses autonomous processor assignments in which processor k is assigned to the set of processes S_k. The simulation's processing rate on any set of processes cannot exceed the maximum execution rate for that set; otherwise, the simulation would be unstable. Consequently,

$$\bar{\lambda}_{S_k} \leq \mu_k \quad \text{and} \quad \bar{\lambda}_S \leq \mu_G. \tag{12}$$

Next, note that the linearly synchronous property of the simulation implies by Theorem 3 that $\bar{\lambda}_S = \lambda_S \bar{\lambda}_{S_k} / \lambda_{S_k}$ for any k. This equality and the first inequality in (12) yield

$$\bar{\lambda}_S = \lambda_S \min_{k \leq K} \bar{\lambda}_{S_k} / \lambda_{S_k} \leq \lambda_S \min_{k \leq K} \mu_k / \lambda_{S_k}. \tag{13}$$

Combining this and the last inequality in (12), we obtain the assertion (6).

Next, suppose the simulation uses group processor assignments with processor set G_m assigned to the set of processes S_m. By Theorem 3, we have, similarly to the equality part of (13),

$$\bar{\lambda}_S = \lambda_S \min_{m \leq M} \bar{\lambda}_{S_m} / \lambda_{S_m}, \tag{14}$$

$$\bar{\lambda}_{S_m} = \lambda_{S_m} \min_{j \in S_m} \bar{\lambda}_j / \lambda_j. \tag{15}$$

Now the portion of events for the processes in S_m that are executed by processor k is μ_k / μ_{G_m}. Then the average maximum execution rate for any event from S_m is

$$\sum_{k \in G_m} (\mu_k / \mu_{G_m}) \mu_k = \alpha_{G_m},$$

(recall (8)). Consequently, similarly to (12), we have

$$\bar{\lambda}_j \leq \alpha_{G_m} \quad \text{for each } j \in S_m. \tag{16}$$

Applying this inequality to (15) and using $\bar{\lambda}_{S_m} \leq \mu_{G_m}$ (as in (12)) yields

$$\bar{\lambda}_{S_m} \leq \min\{\mu_{G_m}, \lambda_{S_m} \alpha_{G_m} \min_{j \in S_m} \lambda_j^{-1}\}.$$

Then using this in (14), we have

$$\bar{\lambda}_S \leq \lambda_S \min_{m \leq M} \{\min\{\mu_{G_m} / \lambda_{S_m}, \alpha_{G_m} \min_{j \in S_m} \lambda_j^{-1}\}\}.$$

Combining this with $\bar{\lambda}_S \leq \mu_G$ yields the assertion (7). Finally, note that the inequality (9) for global processor assignments is the special case of (7) with $M = 1$. □

4. Numerical Example

Our experience with numerical examples of Time Warp parallel simulations of queueing networks indicates that the upper bounds on simulation rates in the last section are fairly close to the actual ones. This is illustrated by the following example.

We considered the parallel simulation of a closed queueing network with eight single-server stations as shown in Figure 1. Each station has a general service time distribution with a common service rate and the service discipline is first-in-first-out. The customers moving along the stations are homogeneous and they move independently according to the probabilities shown on the arcs.

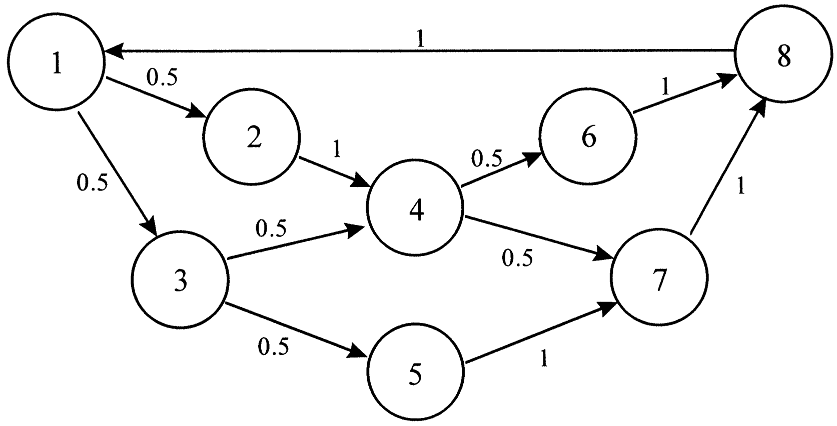

Figure 1. Queueing Network

We ran several simulations of the queueing network for which the number of customers was set at 8, 32, 64, 256, 512 and 1024. The efficiency of the simulation is measured by its *speedup* defined as the ratio $\bar{\lambda}_S/\mu$ (the simulation rate per unit of processing rate). According to Corollary 6,

$$\bar{\lambda}_S/\mu \leq \min\{8, \lambda_S \min_{j \leq 8} \lambda_j^{-1}\}.$$

Graphs of the actual simulation speedup and the preceding upper bound, as functions of the number of customers, are shown in Figure 2 below. Note that the actual speedup is very close to the bound. Although the speedup is not close to the maximum 8, its value near 3.4 is reasonable. In this case, the speedup bound is the constant $\lambda_S \min_{1 \leq j \leq 8} \min_{1 \leq j \leq 8} \lambda_j^{-1} = 3.45$. This bound does not vary with the number of customers in the network. Indeed, if λ'_j were another set of rates, then $\lambda'_S/\lambda'_j = \bar{\lambda}_S/\bar{\lambda}_j = \lambda_S/\lambda_j$.

We used an aggressive Time Warp parallel protocol with preemptive rollbacks to simulate the queueing network. The simulation consisted of eight processes representing the eight respective service stations. The main events of a process contain the arrival, service and departure times of customers at the node it represents. These processes were simulated on the KSR-1 parallel computer by eight homogeneous processors under single processor assignments. Upon finishing the processing of an event, a processor typically sends one or more events to other processors before it can execute another event. The processing time therefore includes communication overhead, which is a major cost for parallel processing.

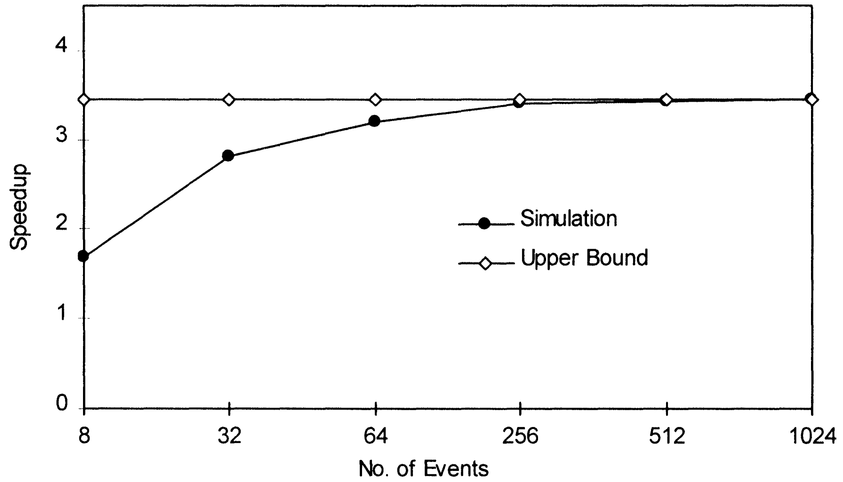

Figure 2. Speedup of Time Warp Simulation

5. Efficient Processor Scheduling Strategies

In this section, we use the simulation rate bounds to gain insight into efficient assignments of processors to processes.

For a linearly synchronous simulation under group processor assignments, the processing rate bound in Theorem 5 is

$$\overline{\lambda}_S \leq \min\{\mu_G, \lambda_S U_M\}, \tag{17}$$

where

$$U_M = \min_{1 \leq m \leq M} \{\mu_{G_m}/\lambda_{S_m}, \alpha_{G_m} \min_{j \in S_m} \lambda_j^{-1}\}. \tag{18}$$

For those simulations, as in the last section, in which the inequality (17) is close to being an equality, one could achieve the highest processing rate $\overline{\lambda}_S$ by maximizing U_M. In this regard, the following optimal processor assignment problem may be of interest.

Suppose the rates $\lambda_1, \ldots, \lambda_n$ and $\mu_1, \ldots, \mu_K$ are fixed and one can vary the processor assignments. Then one could ensure a maximal processing rate $\overline{\lambda}_S$ by choosing the integer M and sets $S_1, \ldots, S_M$ and $G_1, \ldots, G_M$ that maximize U_M. This optional processor assignment can be obtained by solving the following optimization problem successively for $M = 1 \ldots, \min\{ K , n \}$.

Assume that M is a fixed integer. We can write

$$\mu_{S_m} = \sum_{j=1}^{n} \lambda_j x_{jm}, \text{ and } \mu_{G_m} = \sum_{k=1}^{K} \mu_k y_{km},$$

where $x_{jm} = 1$ or 0 according as j is or is not in S_m, and $y_{km} = 1$ or 0 according as k is or is not in G_m. Similarly,

$$\alpha_{G_m} = \sum_{k=1}^{K} \mu_k^2 y_{km} / \sum_{k=1}^{K} \mu_k y_{km}.$$

Then from its definition (18), U_M is maximized by the following mathematical program:

$$\max_{x_0, x_{jm}, y_{km}} x_0$$

subject to the constraints

$$x_0 \sum_{j=1}^{n} \lambda_j x_{jm} \leq \sum_{k=1}^{K} \mu_k y_{km}, \quad 1 \leq m \leq M,$$

$$x_0 \lambda_j x_{jm} \sum_{k=1}^{K} \mu_k y_{km} \leq \sum_{k=1}^{K} \mu_k^2 y_{km}, \quad 1 \leq j \leq n, \quad 1 \leq m \leq M, \tag{19}$$

$$\sum_{j=1}^{n} \sum_{m=1}^{M} x_{jm} = n, \quad \sum_{k=1}^{K} \sum_{m=1}^{M} y_{km} = K,$$

$$x_{jm} = 0 \text{ or } 1, \quad y_{km} = 0 \text{ or } 1, \; 1 \leq j \leq n, \quad 1 \leq m \leq M.$$

In this nonlinear programming problem, the variable x_0 plays the role of U_M and the two inequality constraints just guarantee that x_0 does not exceed the two terms on the right side of (18). The essence of the problem is to find the largest value of x_0 for which there are feasible x_{jm} and y_{km}. One could solve this problem by fixing x_0 at various values and running a standard linear integer programming package, for each fixed x_0, to find whether there are feasible x_{jm} and y_{km} (one can linearize the constraint (19) by introducing new variables in the usual way). Note that for the case in which the processors are homogeneous, one can replace the y_{km} with the variables $k_1, \ldots, k_M$ that denote the respective numbers of processors assigned to $S_1, \ldots, S_M$, where $k_1 + \ldots + k_M = K$. Then the two main constraints reduce to

$$x_0 \sum_{j=1}^{n} \lambda_j x_{jm} \leq k_m \mu, \quad x_0 \lambda_j x_{jm} \leq \mu.$$

Finally, to optimize the M, one would solve the preceding problem successively for $M = 1, \ldots, \min\{ K , n \}$.

Although the processor assignment optimization problem we just discussed is rather narrow, it may be useful for gleaning a little more speed from a simulation. Other ways to substantially increase the speed might involve varying the processing rates, assigning processors dynamically depending on the state of the simulation and using more efficient time and space divisions techniques. These approaches would lead to interesting optimization problems, but they would require more intricate details of the simulation structure than we have been discussing.

References

[1] Akylidiz, I.F., Chen, L., Das, S.R., Fujimoto, R.M. and Serfozo, R.F., The effect of memory capacity on Time Warp performance, *J. Parallel and Distributed Computing* 18 (1993), 411-422.

[2] Chen, L., Parallel simulation by multi-instruction, longest-path algorithms, (1993), *Queueing Systems: Theory and Applications* 27 (1997), 37-54.

[3] Chen, L., Serfozo, R.F., Das, S.R., Fujimoto, R.M. and Akyildiz, I.F., Perform-

ance of Time Warp parallel simulations of queueing networks, (1994), submitted for publication.

[4] Felderman, R.E. and Kleinrock, L., Bounds and approximations for self-initiating distributed simulation without lookahead, *ACM Trans. on Model. and Comput. Simul.* **1** (1991), 386-406.

[5] Fujimoto, R.M., Parallel discrete event simulation, *Commun. ACM* **33** (1990), 30-53.

[6] Gupta, A., Akyildiz, I.F. and Fujimoto, R.M., Performance analysis of Time Warp with multiple homogeneous processors, *IEEE Trans. of Softw. Eng.* **17** (1991), 1013-1027.

[7] Greenberg, A.G., Lubachevsky, B.D. and Mitrani, I., Algorithms for unboundedly parallel simulations, *ACM Trans. Computer Systems* **9**:3 (1991), 201-221.

[8] Lin, Y.-B. and Preiss, B.R., Optimal memory management for Time Warp parallel simulation, *ACM Trans. on Modl. and Comput. Simul.* **1** (1991), 283-307.

[9] Nicol, D.M., Performance bounds on parallel self-initiating discrete-event simulations, *ACM Trans. on Model. and Comput. Simul.* **1** (1991), 24-50.

[10] Dickens, P.M., Nicol, D.M., Reynolds, P.F. and Duva, J.M., Analysis of optimistic window-based synchronization, (1994), submitted for publication.

[11] Wagner, D.B. and Lazowska, E., Parallel simulation for queueing networks: Limitations and potentials, *Perf. Eval. Review* **17**:1 (1989), 146-155.