Modeling and Control of Distributed Asynchronous Computations

Longsong Lin and John K. Antonio
School of Electrical Engineering
Purdue University
West Lafayette, IN 47907-1285

Abstract

A stochastic model for a class of distributed asynchronous fixed point algorithms is presented and a methodology for optimizing the rate of convergence is introduced. An important parameter in our model, called the degree of synchronization, quantifies the average amount of time each processor is willing to wait for information from other processors (before beginning computation of its update variable based on the available estimates of variables from other processors). The main focus of the paper is to analyze the relationship between the convergence rate and the degree of synchronization for a class of iterative fixed point algorithms. Preliminary analysis indicates that significant improvements in convergence rates can be achieved by proper control of the parameters in our model.

1 Introduction

As high-speed computer networks (such as gigabit fiber optic networks) begin to emerge, there is promise of delivering a substantial increase in computing power for many important and computationally intensive problems [1]. Perhaps the most basic concern in large-scale distributed computation is that of synchronization. Imagine the overhead associated with trying to coordinate and schedule the order of computations in a large-scale distributed computer network. Factors contributing to the difficulty of synchronization include the following: (i) there is no global clock, (ii) the computational resources vary from one node to the next, (iii) the network is prone to failures, and (iv) the demand patterns of the traffic are difficult to predict.

Fortunately, it turns out that a large class of computational problems arising from enginnering applications can be solved in a virtually totally asynchronous distributed computing environment. (By virtually totally asynchronous, we mean the processors can basically compute and exchange information in any order.) Two of the founding contributors to the area of asynchronous computation are Bertsekas and Tsitsiklis. Many of the fundamental concepts found in the present paper come from their recently published book [3].

In this paper, our focus is on types of iterative computational problems which can be solved in a distributed asynchronous computing environment. We propose a computational model which bridges the gap between *purely asynchronous* algorithms (in which each processor computes a new update in a totally uncontrolled manner) and *purely synchronous* algorithms (in which each processor waits for updated variables from all other processors before computing a new update). The main idea of the approach is to control the idle waiting time of the processors so as to achieve a superior overall convergence rate. For the class of problems considered, our model can provide a compromise somewhere between the purely asynchronous and synchronous modes of computation.

Consider, for example, a purely asynchronous mode of operation in which the processors are first updating and then transmitting their results as often as possible. In such a scenario, if the service rates (i.e., bandwidths) of the communication channels are small relative to the rate at which the updates are computed, then there will be large time delays in receiving updated variables from other processors; because whenever the transmissions are done as often as possible, the communication network may become congested, causing delays in transmissions increase. Therefore, even though all of the processors are busy all of the time (high processor throughput), it is not clear whether they will make much progress toward converging, because they are computing their updates based on relatively out-of-date information.

At the other extreme is a purely synchronous mode of operation in which each processor waits for all updated variables before beginning a new update. In this case, the processors may be idle for relatively long periods of time, and therefore the potential may exist to improve the convergence rate by allowing computation to begin some instant *before* all updates are received from all other processors.

The paper is organized as follows. In Section 2 we define the class of iterative fixed point problems to be studied. Section 3 describes the network model employed to estimate the delay times associated with distributed asynchronous computations. The model we develop is a closed queuing system. In Section 4 we study (analytically) the case of two interconnected processors working on a system of two linear iterative update equations—our goal being to estimate rates of convergence in terms of the parameters of our model. We then describe a graphical method for estimating optimal values for the parameters of our model (in terms of parameters of the network and those of the update equations).

0-8186-2672-0/92 $03.00

2 The Basic Problem

We restrict our attention to iterative fixed point problems, which in the synchronous case have the form

$$x_i(k+1) = f_i(x_1(k), x_2(k), \cdots, x_n(k)), \ i = 1, 2, \cdots, n, \quad (1)$$

where $x_i(k+1)$ is the so-called "updated" value of the variable x_i, which is computed according to the given function of the previous iterates, i.e., $f_i(x_1(k), x_2(k), \cdots, x_n(k))$. It is further assumed that the vector $x(k) = [x_1(k), x_2(k), \cdots, x_n(k)]^T$ converges to a unique fixed point, say $x^* = [x_1^*, x_2^*, \cdots, x_n^*]^T$, as $k \to \infty$. A primary objective of this paper is to estimate the amount of time required for various types of distributed "asynchronous" algorithms for updating equation (1) to converge to within a small neighborhood of x^*. As described in [2,3], many classical algorithms solve fixed point problems, including gradient search methods for nonlinear optimization, iterative techniques for solving linear equations, and dynamic programming methods for solving shortest path problems.

Before addressing the issues involved with computing the update equations in a distributed asynchronous computing environment, we first review the obvious class of single processor serial algorithms.

2.1 Serial Algorithms

A single processor serial algorithm for equation (1) would operate, for example, by first computing $x_1(k+1)$, followed by the computation of $x_2(k+1)$, and so on down the line until $x_n(k+1)$ is computed. The procedure would then be repeated with the index k incremented by one. Figure 1 depicts the order of computations for this type of serial algorithm. This is the familiar Jacobi-type updating scheme.

2.2 Distributed and Asynchronous Algorithms

Now, assume we have n interconnected processors available to compute the required updates of equation (1). As suggested in reference [3], an obvious way to take advantage of having multiple processors is to assign one update equation to each of the n processors, i.e., use processor i to compute the update associated with the i^{th} update equation. Figure 2 depicts the order of computations associated with each processor for this type of distributed processing algorithm. At the end of each computation time block, note that each processor broadcasts its updated variable to all other processors (denoted by the arrowed links emanating from the end of each computation time block). Also note that after computation the processors wait for a certain amount of time (the idle time) before starting another computation. Observe that if the idle time for all processors is made large enough, then the distributed computation will become synchronous, as shown in Figure 3. So, a synchronous updating scheme can be viewed as a special case within the class of asynchronous updating schemes.

A question naturally arises regarding the importance of having each processor wait until updates from all other processors are received before starting to compute its next update. Due to the work of Bertsekas *et al.*, see references [2,3], it is known that a large class of iterative fixed point problems (of the same form we shall assume) will converge (eventually) to the unique fixed point in virtually any type of asynchronous computing environment. In the present paper, we are interested in analyzing and optimizing the *rate of convergence* for these types of algorithms.

In our asynchronous computing model, we assume processor i updates the i^{th} variable according to

$$x_i\left(t_i + T_P^i(t_i)\right) = f_i(x_1(t_i - D_{1i}(t_i)), \cdots, x_i(t_i - D_I^i(t_i)), \cdots, x_n(t_i - D_{ni}(t_i))), \quad (2)$$

where t_i is a time instant when processor i begins computing an update, $D_I^i(t_i)$ is the idle period before time instant t_i, $T_P^i(t_i)$ is the amount of time require to compute the update (i.e., the processing time, starting at time t_i), and $D_{ji}(t_i)$ indicates delay associated with sending processor j's variable to processor i (i.e., it is the "age" of processor i's version of the j^{th} variable).

As we shall describe in the later sections, the quantities $T_P^i(t_i)$, $D_I^i(t_i)$, and $D_{ji}(t_i)$ actually denote expectations of random variables. In order to simplify the notational burden, throughout the rest of the paper we shall assume that assume $D_I^i(t_i) = D_I$, $T_P^i(t_i) = T_P$, and $D_{ji}(t_i) = D_{ji}$, for all i and t_i.

2.3 The Degree of Synchronization

Intuition suggests that the algorithm depicted in Figure 2 is "less synchronous" than the one shown in Figure 3. In order to quantify this insight, we introduce a parameter called the *degree of synchronization*, defined as

$$s = \frac{D_I}{\widetilde{D}_I}, \quad (3)$$

where $\widetilde{D}_I$ is defined as the minimum value of D_I such that $D_{ji} \leq D_I$.[1] The algorithm depicted in Figure 2 has a degree of synchronization, $s < 1$, while the one shown in Figure 3 has a degree of synchronization, $s = 1$. Because D_I and D_{ji} actually represent expected values of random variables, a value of $s = 1$ does not actually imply that the distributed algorithm is "synchronous" in the strictest sense of the word, but rather that the "expected" behavior is synchronous. Our focus shall be to study the relationship between the overall convergence rate of the algorithm and the value of the degree of synchronization. We will discover that the value of the optimal degree of synchronization depends on the structure of the update equations and the relative speeds of the processors and communication channels.

2.3.1 Intuitive Observations

Referring back to Figure 2, note that for a given value of T_P, the total number of message transmissions in

[1] As mentioned earlier, the value of D_{ji} actually depends of the value of D_I. For instance, if D_I is decreased, then the value of D_{ji} increases (due to network congestion).

a fixed time interval increases whenever D_I is decreased. In particular, the number of broadcasts done by each processor in a long time interval of length say τ, is given by approximately $\frac{\tau}{D_I+T_P}$. Therefore, by decreasing the idle time, the total amount of traffic on the network increases, which in turn implies (intuitively) that the average delay in sending messages through the network should increase. Figure 4 depicts a delay curve for a "typical" buffered communication channel. As depicted in the figure, the average delay on a communication channel increases rapidly as the amount of offered traffic is increased past a moderate level.

We now make two fundamental observations regarding the relationship between the degree of synchronization and the rate of convergence.

1. If s is chosen sufficiently small (i.e., close to zero), then the processors are busy computing updates most of the time. However, these updates are computed based on relatively out-of-date estimates from the other processors' variables; because the average communication delay is large (generically) whenever s is small. So, even though the processors are "busy" most of the time, it is not clear how much progress each computed update will make toward converging to the solution.

2. If s is large (close to one, for example), then the processors may not compute updates very often. However, these updates are computed based on recently computed iterates from the other processors; because the communication delay is small whenever s is large. Thus, there is the potential for each of these updates to make good progress toward converging to the solution (however, these updates are not computed very often).

Figure 5 depicts the tradeoffs between choosing small versus large values of s for the simple case of two interconnected processors. Note that the delay time in sending a message is large whenever s is small (and small whenever s is large). For the small s case, the updates are computed based on variables that are three iterates old, while for the large s case the updates are computed based on variables that are two iterates old. Also note that in any fixed time interval, approximately 20% more updates can be done for the small s case than for the large s case.

We now pose the following question: "What value of s produces the best convergence rate?" In Section 4 we address this question for the case of two interconnected processors. First, however, we formally describe a closed queuing model for distributed computation.

3 A Queuing Model for Distributed Computation

We view a distributed computing system as a network of processors (processing nodes) connected by buffered communication links. A standard $M/M/1$ queue is employed for modeling the delay associated with the communication links. A queuing system equipped with a flow control facility is used to characterize delay associated with the processors at each node. Together, these two components are used to form a general network comprised of any number of processing nodes interconnected by communication links. The model should provide a level of detail sufficient for estimating convergence rates in a meaningful fashion. Our model is modular and scalable, and applicable to a large class of distributed computing systems.

3.1 Modeling the Communication Links

The delay characteristics of the communication links are modeled by a standard $M/M/1$ queue. The difficulty associated with dependent interarrival times and packet lengths is overcome by recalling the well-known Kleinrock independence assumption, which basically states that the merging of several packet streams on a transmission line has an effect of restoring the independence of interarrival times and packet lengths [4,9]. Based on this assumption, our model assumes independent Poisson arrivals and exponential interarrival times. The capacity of link (i,j) is denoted by μ_{ij}^L (also called the service rate of the link). The messages (which in our case are updated variables) are exchanged with other processors in the form of packets which are sent into the link queue, where they wait for transmission through the associated communication link.

3.2 The Queuing Model for Local Computation

The queuing model for the computation at a node is basically an $M/M/1$ queue equipped with an extra feedback control mechanism which allows the flow of traffic to be controlled. The service rate of processor i is denoted by μ_i^P. Also, note that after a packet is serviced at processor i, it is routed out through link (i,j) with probability p_{ij}, and is routed back to processor i's queue with probability p_{ii}.

The rationale for modeling local computation with a queuing model is summarized as follows (borrowed primarily from reference [3]). First, the processor may be temporarily unavailable because other messages are currently being serviced or are scheduled for service ahead of a newly arrived message. Second, a contention resolution processes may be underway whereby the allocation of the processor to several contending applications is to be decided (i.e., the processor may be busy with other tasks besides computing the distributed updates). Because queuing times are generally hard to quantify precisely, our aim is to use the simplified models to obtain valuable qualitative and quantitative insights. Thus, we aggregate the effects of the aforementioned events with the proposed model, hereafter denoted as the processor queue.

Note from Figure 6 that the arrivals from distinct sources flow into the processor queue and wait for service. The two types of traffic serviced by the processor can be distinguished in the following sense. A packet destined for an outgoing communication link actually contains a newly updated value of the local processor's variable, and this information will be used by the other processors to compute new updates (at

other processors). On the other hand, a packet being fedback is conceptual in the sense that by the time this particular packet is serviced, the value of the local variable may have already been updated (due to another packet being serviced ahead of it). So, the value of the variable contained in the fedback packet is not actually used; instead, the most recent value of the local variable stored in local memory is used.

3.3 The Closed Queuing Model

Figure 6 depicts a generic node i in our distributed computing model. A node consists of two types of queues: (1) the queues associated with the outgoing communication links and (2) the queue which feeds the processor. Processor i's service rate is denoted by μ_i^P, and the capacity of the link which connects processor i to processor j is indicated by μ_{ij}^L. At node i, the routing probability p_{ij} indicates proportion of traffic routed through link (i,j), and p_{ii} indicates the routing probability of traffic fedback to itself. The rate of interprocess traffic from node k is λ_{ki}. Similarly, the intraprocess traffic is denoted by λ_{ii}.

Processor i is connected to neighboring nodes through outgoing and incoming links. These neighbor sets are defined by:

$$N_{\text{out}}(i) = \{j : \text{there is a link from } i \text{ to } j\} \quad (4)$$

$$N_{\text{in}}(i) = \{j : \text{there is a link from } j \text{ to } i\} \quad (5)$$

To ensure flow conservation, the routing probabilities must satisfy

$$p_{ii} + \sum_{j \in N_{\text{out}}(i)} p_{ij} = 1 \qquad \text{for all } i. \quad (6)$$

Furthermore, if we define the traffic rate flowing through processor i's queue to be λ_i, then we have

$$\lambda_i = \lambda_{ii} + \sum_{j \in N_{\text{in}}(i)} \lambda_{ji} = 1 \qquad \text{for all } i. \quad (7)$$

Finally, from the above definitions we have that

$$\lambda_{ii} = p_{ii}\lambda_i, \qquad \text{for all } i \quad (8)$$

and

$$\lambda_{ij} = p_{ij}\lambda_i, \qquad \text{for all } i, \text{ and } j \in N_{\text{out}}(i). \quad (9)$$

For more details about the general theory of closed queuing networks, see [5,6,7,10,11].

3.4 The Busy and Idle Periods of the Processor

The computation time (i.e., T_P from the previous section) at a node is exponentially distributed with rate μ_i^P. Thus, the expected computation time for each job is $T_P = 1/\mu_i^P$. There may be periods of time when the processor queue is empty. These represent idle periods (D_I from the previous section), meaning the processor is available but there are no jobs in the queue. Letting T_P and D_I denote the average computation time and idle time, respectively, we have that

$$\lambda_i = \frac{1}{D_I + T_P}. \quad (10)$$

As mentioned in the previous section, our focus is to characterize the effect of D_I on the convergence rate of the distributed fixed point algorithms. In the next section we carry out a "mean value analysis" for the case of two processors solving a linear two-dimensional fixed point problem. We should note that in a closed queuing system the value of D_I is actually adjusted by changing the number of packets circulating in the closed network. However, in the present paper we shall not derive the relationship between D_I and the total number of packets. Instead, for simplicity we shall assume that essentially any (continuous) value of D_I can be realized by properly selecting the number of packets in the network, even though this may not actually be possible because only integer number of packets are allowed. Nevertheless, our approximate analysis will yield some interesting insights.

4 Estimating Convergence Rates: The Two Processor Case

Figure 7 shows our closed queuing model for the case of two interconnected processors. For simplicity of notation, we have assumed both service rate and communication capacities are the same, i.e. $\mu_1^P = \mu_2^P = \mu^P$ and $\mu_{12}^L = \mu_{21}^L = \mu^L$. Also, we assume that $p_{12} = p_{21} = p$ and $p_{11} = p_{22} = q$ (where $p + q = 1$), and $\lambda_1 = \lambda_2 = \lambda$. Therefore, $\lambda_{12} = \lambda_{21} = p\lambda$ and $\lambda_{11} = \lambda_{22} = q\lambda$.

We assume linear iterative update equations of the form

$$x_1(t + T_P) = K\left[\zeta x_1(t - D_I) + (1 - \zeta)x_2(t - D)\right] \quad (11.1)$$

$$x_2(t + T_P) = K\left[(1 - \zeta)x_1(t - D) + \zeta x_2(t - D_I)\right], \quad (11.2)$$

where the constants K and ζ satisfy $0 < K < 1$ and $0 \le \zeta \le 1$. The parameter ζ characterizes the relative coupling of the equations. If ζ is close to one, we say the two equations are relatively loosely coupled (in the sense that computing the update $x_1(k+1)$ depends *mainly* on the value of $x_1(k)$ and not $x_2(k)$, and computing $x_2(k+1)$ only depends *mainly* on $x_2(k)$). Intuitively, the more loosely coupled the equations are, the less benefit there is in executing the update algorithm synchronously (or even close to synchronously), and the more benefit there may be in choosing a degree of synchronization substantially less than one. The reasoning behind this intuition is the notion that whenever the equations are relatively loosely coupled, then it is not very "important" that each processor have a recent estimate of the other processor's variable, because the influence of the other processor's variable on the update equation is relatively small. As we shall see, our analysis below justifies this intuition.

We now focus on deriving an approximate expression for the value of D (in terms of the idle time, D_I).

First, based on the $M/M/1$ queuing assumptions of our model, we know that the average delay through a communication link is

$$\frac{1}{\mu^L - p\lambda}. \tag{12}$$

Similarly, the average waiting time in a processor queue is

$$\frac{1}{\mu^P - \lambda} - \frac{1}{\mu^P}. \tag{13}$$

Therefore, the average delay for an interprocess message is

$$D_{\text{inter}} \stackrel{\text{def}}{=} \frac{1}{\mu^L - p\lambda} + \frac{1}{\mu^P - \lambda} - \frac{1}{\mu^P}, \tag{14}$$

where

$$\lambda = \frac{1}{D_I + T_P} = \frac{1}{D_I + \frac{1}{\mu^P}}. \tag{15}$$

For the case where $p = 1$, we can estimate D as follows.[2] If $D_{\text{inter}} \leq D_I$, meaning the the degree of synchronization, $s \geq 1$, then we have that $D \approx D_I$. On the other hand, if $D_{\text{inter}} > D_I$, then D is approximately equal to D_{inter}. Thus, our approximation for D is

$$D \approx \max\{D_I, D_{\text{inter}}\}, \tag{16}$$

where, by substituting equation (15) into (14), we have

$$D_{\text{inter}} \stackrel{\text{def}}{=} \frac{1}{\mu^L - \frac{p}{D_I + \frac{1}{\mu^P}}} + \frac{1}{\mu^P - \frac{1}{D_I + \frac{1}{\mu^P}}} - \frac{1}{\mu^P}. \tag{17}$$

Now, for the case where $0 < p < 1$, we approximate D as

$$D \approx \max\{D_I, D_{\text{inter}}\} + \left(\frac{1}{p} - 1\right)(D_I + T_P). \tag{18}$$

Due to space limitations, the details justifying the above approximation are not included.

Based on an inductive proof technique introduced in [3, pp. 99–104], it can be shown that the convergence rate coefficient associated with our update equations, denoted by ρ, is governed by the solution of the following equation:

$$\rho = K\left[\zeta\rho^{-D_I} + (1-\zeta)\rho^{-D}\right]. \tag{19}$$

We note that in [3], the interdependence between D and D_I is not accounted for (instead, it is only assumed that D is bounded). Thus, by applying our approximation for D, i.e., equation (18), we provide a natural extension for the convergence rate result introduced in [3].

Figures 8 and 9 show surface plots of ρ versus D_I and p, for various values of μ^L (i.e., the capacities of the links). The value of μ^P (i.e., the speed of the processors) for all three cases was chosen as unity. The value of the link capacities associated with each plot are $\mu^L = 1.0$ and $\mu^L = 10.0$, respectively. We should recall that the closer the convergence rate coefficient is to zero, the faster the rate of convergence. In particular, the convergence rate coefficient is a scalar $0 \leq \rho < 1$ which satisfies $\| x(t) - x^* \| \leq \rho^t \| x(0) \|$. For the simple linear updates given in equation (11) the fixed point is clearly the zero vector.

Figure 8 depicts convergence rate behavior for the case where the service rates of the processors and the communication links are the same (i.e., $\mu^L = \mu^P = 1.0$). From the plot, note that if we set $p = 1$, then the "optimal" convergence rate of $\rho \approx 0.2$ is obtained by setting $D_I \approx 1.6$. In contrast, in Figure 9 where we have $\mu^L = 10.0$, a convergence rate of $\rho \approx 0.1$ is obtained at $D_I \approx 1.0$. Thus, as would be expected, if the capacities of the communication links are increased, then proper choice of D_I yields improved performance. Similar observations regarding the choice of p (for a fixed value of D_I) can be made.

4.1 Some Comments on Optimal Convergence Rates

Equation (19), which characterizes the behavior of the convergence rate, is an implicit function and must be solved numerically for given values of D_I and D. In order to determine the values of D_I and p which achieve the minimum (i.e., optimal) value of ρ, we are also faced with solving implicit functions. The surface plots provide a graphical means of choosing optimal values.

Intuitively, achieving fast convergence involves the balancing of two competing factors: (1) reducing the amount of broadcast traffic (thus alleviating the communication delay) while (2) keeping the frequency of computation as high as possible. The former goal is achieved by adjusting the value of p, and the latter by adjusting D_I. We have derived (for a fixed p) an explicit solution for the optimal value of ρ, denoted as ρ^*. This optimal value is dependent on a nonlinear function of the optimal D_I, which, in turn, can only be solved numerically. Hence, the surface plots of ρ versus p and D_I are useful for locating optimal points without resorting to the task of solving the nonlinear equation. We have also derived an optimal solution in terms of a single intermediate parameter, namely E, which is a function of of both D_I and p. We can therefore adjust the values of D_I and p to obtain the minimum convergence rate within a specified set of constraints.

Next, we note that the value of $\widetilde{D}_I$ (used in equation (3) for defining the degree of synchronization) can be computed as an explicit function of parameters μ^L, μ^P, and p. Recall that $\widetilde{D}_I$ is defined to be the value of D_I such that $D_{\text{inter}} = D_I$. Therefore, we can solve for $\widetilde{D}_I$ by setting equation (17) equal to D_I, and solving for D_I. (Without loss of generality, we assume $\mu^P = 1$.) After some algebra, we get

[2] Recall from equations (11.1) and (11.2) that D represents the average "age" of processor i's estimate of processor j's variable, with $i \neq j$.

$$\widetilde{D}_I = \frac{(1+p) + p\sqrt{\left(1+\frac{1}{p}\right)^2 + \frac{4\mu^L}{p}\left(\frac{\mu^L}{p} - 1\right)}}{2\mu^L}. \quad (20)$$

This crucial idle time derived above plays an important role in constraining the value of D_I in formulating the optimization problem. Another critical value, that also has an influential effect on the convergence rate, is associated with the parameter ζ. We call it the *critical value of* ζ, denoted as ζ^c (and we have a closed form expression for its value). We can show that if $\zeta \geq \zeta^c$, then choosing the degree of synchronization as $s = 1$ (i.e., $D_I = \widetilde{D}_I$) will produce the optimal convergence rate. On the other hand, if $\zeta < \zeta^c$, then a degree of synchronization $s < 1$ will produce the optimal convergence rate. Also, given that $\zeta < \zeta^c$, we have derived an (implicit) equation for determining the optimal values of D_I and p. Due to space limitations, we shall not describe these results in detail. The family of curves shown in Figure 10 gives an indication that whenever ζ is close enough to one, then a degree a synchronization less than one will yield optimal performance. As would be expected, the value of ζ^c depends on the values of μ^L and μ^P.

5 Discussions and Future Work

Our preliminary analysis should serve well as a guide toward analyzing (and optimizing) convergence rates of iterative algorithms on large-scale computer networks. Our plan for future work includes an extension of our preliminary convergence rate analysis to include more general computer networks (with more than two processors). Also, in order to further justify the proposed analytical model, we plan to carry out experimental computations on an actual distributed/parallel system. Implementations of asynchronous fixed-point algorithms are currently underway using the NCube/2 hypercube multiprocessor computer.

References

[1] *IEEE Computer Magazine*, Special Report on "Gigabit Network Testbeds," Vol. 23, No. 9, pp. 77–80, Sept. 1990.

[2] D. P. Bertsekas, "Distributed asynchronous computation of fixed points," *Math. Programm.*, Vol. 27, pp. 107–120, 1983.

[3] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

[4] L. Kleinrock, *Queuing System. Volume 1 : Theory*. New York, NY: John Wiley & Sons, 1975.

[5] J. R. Jackson, "Job Shop Like Queuing Systems," Management Science, vol. 10, no. 1, 1963, pp. 131–142.

[6] W. L. Gordon and G. F. Newell, "Closed Queuing Systems with Exponential Classes of Customers," Operations Research, vol. 15., no. 2, 1967, pp. 254–265.

[7] P. Buzen, "Computational Algorithms for Closed Queuing Networks with Exponential Servers," Comm. ACM, vol. 16, no. 9, Sept. 1973, pp. 527–531.

[8] H. T. Kung, *Synchronized and Asynchronized Parallel Algorithms for Multiprocessors.* In J. F. Traub. (ed.), *Algorithms and Complexity.* New York, NY: Academic Press, 1976, pp. 153–200.

[9] M. Schwartz, *Telecommunication Networks: Protocols, Modeling and Analysis.* Reading, Mass: Addison-Wesley, 1987.

[10] K. S. Trivedi, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications.* New Jersey: Prentice-Hall, 1982.

[11] D. Bertsekas and R. Gallager, *Data Networks.* Prentice-Hall, New Jersey, 1987.

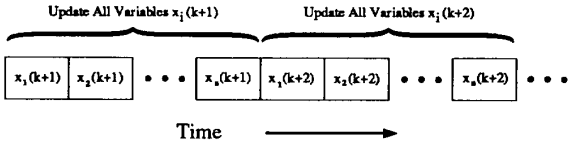


Figure 1: Order of computations in a serial algorithm.

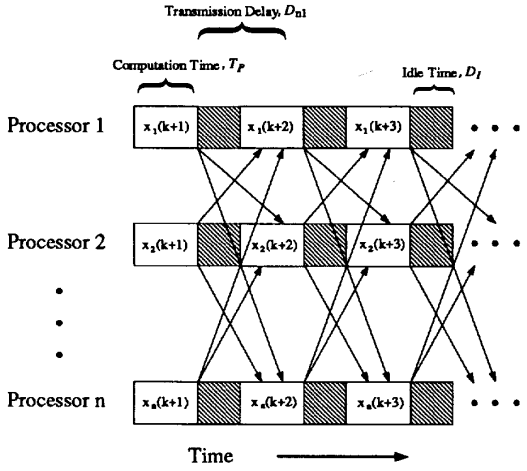


Figure 2: Order of computations in a distributed asynchronous algorithm.

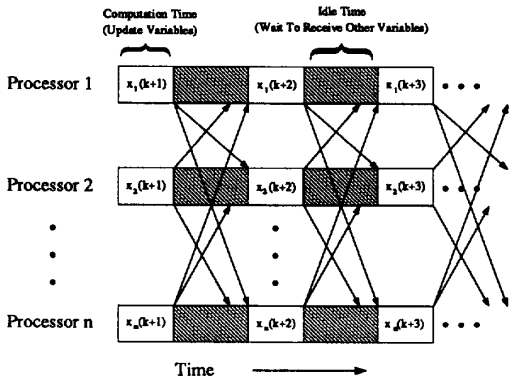


Figure 3: Order of computations in a distributed synchronous algorithm.

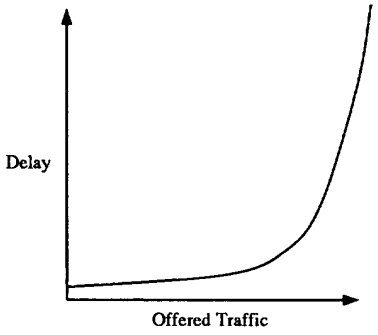
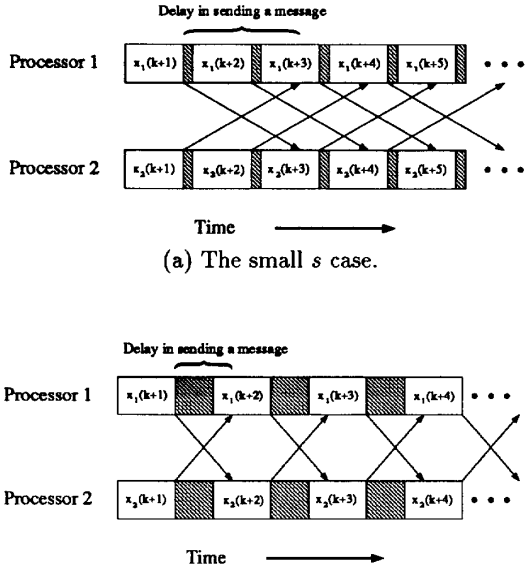


Figure 4: Typical delay characteristics for a communication channel.



(a) The small s case.

(b) The large s case.

Figure 5: Comparison between choosing small and large values of s.

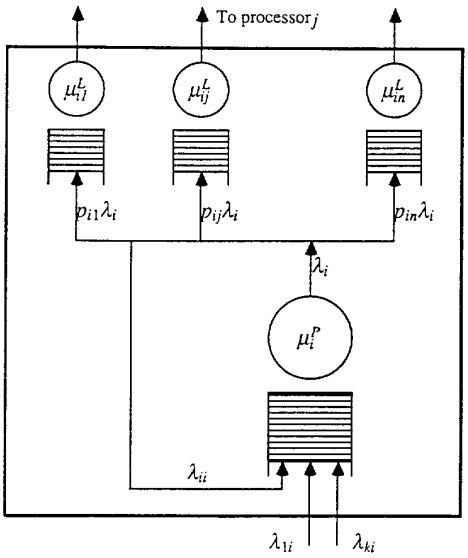


Figure 6: The interconnection of processor and link queues at node i.

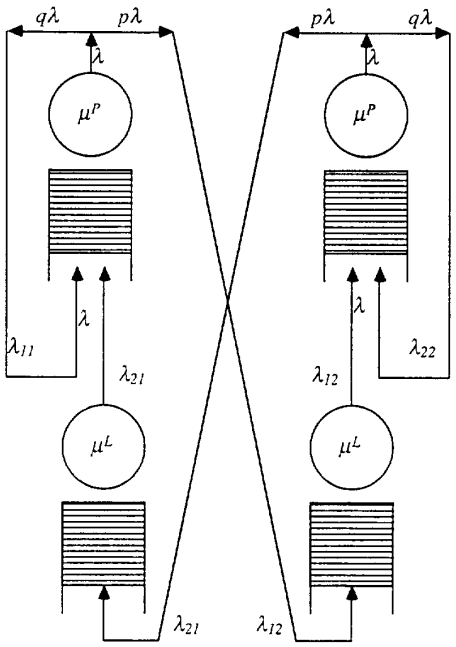


Figure 7: The closed queuing model for the two processor case.

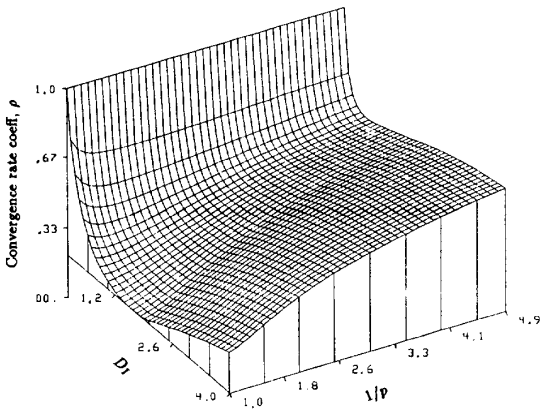


Figure 8: Convergence rate coefficient versus D_I and p. Case 1: $\mu^L = 1.0, \mu^P = 1.0$.

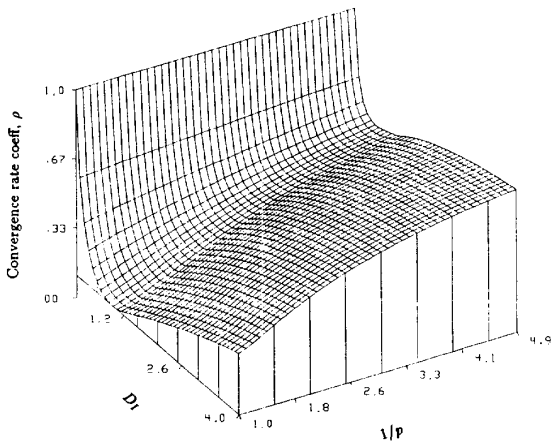


Figure 9: Convergence rate coefficient versus D_I and p. Case 2: $\mu^L = 10.0, \mu^P = 1.0$.

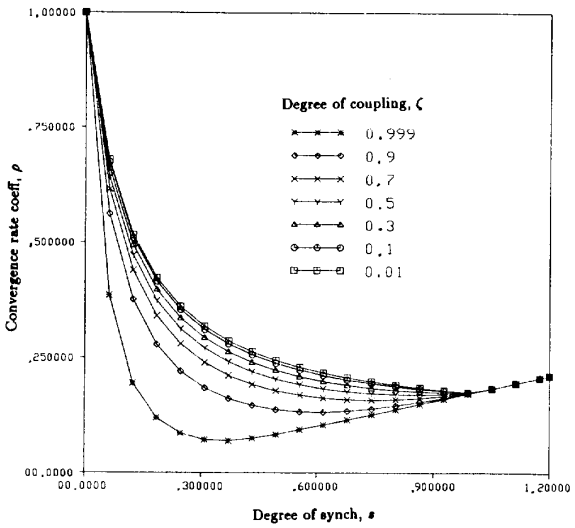


Figure 10: Convergence rate coefficient versus degree of synchronization for various values of ζ.