CONVERGENCE RATE AND TERMINATION OF ASYNCHRONOUS ITERATIVE ALGORITHMS

Dimitri P. Bertsekas
John N. Tsitsiklis
Laboratory for Information and Decision Systems
Massachusetts Institute of Technology
Cambridge, MA 02139, U.S.A.

Abstract

We consider iterative algorithms of the form $x := f(x)$, executed by a parallel or distributed computing system. We focus on asynchronous implementations whereby each processor iterates on a different component of x, at its own pace, using the most recently received (but possibly outdated) information on the remaining components of x. We provide results on the convergence rate of such algorithms and make a comparison with the convergence rate of the corresponding synchronous methods in which the computation proceeds in phases. We also present results on how to terminate asynchronous iterations in finite time with an approximate solution of the computational problem under consideration.

Keywords: Iterative methods, asynchronous algorithms, parallel algorithms, distributed algorithms, termination detection.

1. INTRODUCTION

This paper deals with iterative algorithms of the form $x := f(x)$, where $x = (x_1, \ldots, x_n)$ is a vector in $\Re^n$ and $f : \Re^n \mapsto \Re^n$ is an iteration mapping defining the algorithm. In many interesting applications, it is natural to consider distributed executions of this iteration whereby the ith processor updates x_i according to the formula

$$x_i := f_i(x_1, \ldots, x_n), \tag{1.1}$$

while receiving information from other processors on the current values of the remaining components. Iteration (1.1) can be executed ***synchronously*** whereby all processors perform an iteration, communicate their results to the other processors, and then proceed to the next iteration. It can also be executed ***asynchronously***, whereby each processor computes at its own pace while receiving (possibly outdated) information on the values of the components updated by the other processors. In several circumstances, asynchronous methods can have certain advantages over their synchronous counterparts (see Section 2) and can be a desirable alternative. On the other hand, the mathematical properties of asynchronous iterations are quite different from those of their synchronous counterparts. Even though a fairly comprehensive theory is available [BT2], there are certain issues (pertaining to the convergence rate and termination of asynchronous iterations), that have not been sufficiently studied and this is the subject of the present paper.

Outline of the paper.

In Section 2, we present a mathematical model of synchronous and asynchronous iterations, discuss the possible advantages of asynchronous methods, and present the basic convergence results that are available. In Section 3, we concentrate on asynchronous methods in which the iteration

Research supported by the NSF under Grants ECS-8519058 and ECS-8552419, with matching funds from Bellcore and Dupont, and by the ARO under Grant DAAL03-86-K-0171.

mapping f is monotone and compare the convergence rates of the synchronous and asynchronous variants. Section 4 is similar, except that attention is shifted to the case where the iteration mapping f is a contraction with respect to a maximum norm. Finally, in Section 5, we consider modifications whereby an asynchronous algorithm can be made to execute for a finite amount of time and terminate with an approximate soulution of the computational problem under consideration. This is essentially a problem of detecting the validity of certain termination conditions which is rather trivial in the context of synchronous methods. We indicate that this issue becomes much more difficult in the context of asynchronous methods and we identify certain conditions under which our aim can be accomplished. We note that the literature on the subject is rather large. For this reason, we do not provide a comprehensive list of references, and we refer the reader to [BT2].

2. THE ALGORITHMIC MODEL AND BASIC CONVERGENCE RESULTS.

Let $X_1, \ldots, X_p$ be subsets of Euclidean spaces $\Re^{n_1}, \ldots, \Re^{n_p}$, respectively. Let $n = n_1 + \cdots + n_p$, and let $X \subset \Re^n$ be the Cartesian product $X = \prod_{i=1}^p X_i$. Accordingly, any $x \in \Re^n$ is decomposed in the form $x = (x_1, \ldots, x_p)$, with each x_i belonging to $\Re^{n_i}$. For $i = 1, \ldots, p$, let $f_i : X \mapsto X_i$ be a given function and let $f : X \mapsto X$ be the function defined by $f(x) = \big(f_1(x), \ldots, f_p(x)\big)$ for every $x \in X$. We consider an iteration of the form

$$x := f(x), \tag{2.1}$$

and we call f the *iteration mapping* defining the algorithm. We assume that there are p processors, with the ith processor assigned the responsibility of updating the ith component x_i according to the rule $x_i := f_i(x) = f_i(x_1, \ldots, x_p)$. We say that an execution of iteration (2.1) is *synchronous* if it can be described mathematically by the formula

$$x(k+1) = f\big(x(k)\big),$$

where k is an integer–valued variable used to index different iterations, not necessarily representing real time. Synchronous execution is certainly possible if the processors have access to a global clock, if each processor initiates an update at each "tick" of the clock, and if the results of an update can be reliably transmitted to other processors before the next "tick". Barring the existence of a global clock, synchronous execution can be still accomplished by having each processor perform the $(k+1)$st update as soon as its kth update has been completed and the results of the kth update of all other processors have been received.

In an *asynchronous* implementation of iteration (2.1), processors are not required to wait until they receive all messages generated during the previous iteration. Rather, each processor is allowed to keep updating its own component at its own pace. If the current value of the component updated by some other processor is not available, then some outdated value (received at some time in the past) is used instead. Furthermore, processors are not required to communicate their results after each iteration but only once in a while. We allow some processors to compute faster and execute more iterations than others, we allow some processors to communicate more frequently than others, and we allow the communication delays to be substantial and unpredictable. We also allow the communication channels to deliver messages out of order, *i.e.*, in a different order than the one they were transmitted.

There are several potential advantages that can be gained from asynchronous execution (see e.g., [K], [BT1], [BT2]). On the other hand, a major potential drawback is that asynchronous algorithms cannot be described mathematicallly by an equation of the form $x(k+1) = f\big(x(k)\big)$. Thus, even if the latter difference equation is convergent, the corresponding asynchronous iteration could diverge, and indeed this is sometimes the case. Even if the asynchronous iteration converges, such a conclusion often requires rather difficult analysis. Nevertheless, there is a large number of results stating that certain classes of important algorithms retain their desirable convergence properties in the face of asynchronism [BT2]. A very general result of this form will be presented soon, following a precise description of our model of computation.

Let t be a time variable, representing (global) real time. Even though t should be viewed as a continuous variable, the presentation, the notation, and the proofs are simplified if we introduce a small constant Δ, which is viewed as the unit of time, and analyze the behavior of the algorithm at times that are integer multiples of Δ. For such an analysis to be possible, we *only need* to assume that no processor can execute more than one update during a time interval of length Δ. Clearly, such an assumption should be valid in practice if Δ is taken very small. Still, even though Δ is supposed to be small, it is notationally convenient to scale the time axis so that we can assume that $\Delta = 1$. (This entails no loss of generality.) To conclude, our model will be cast in terms of an integer time variable t, which is proportional to real time.

Let $x_i(t)$ be the value of x_i residing in the memory of the ith processor at time t. We assume that there is a set of times T^i at which x_i is updated. To account for the possibility that the ith processor may not have access to the most recent values of the components of x, we assume that

$$x_i(t+1) = f_i\big(x_1(\tau_1^i(t)), \ldots, x_p(\tau_p^i(t))\big), \quad \forall t \in T^i, \tag{2.2}$$

where $\tau_j^i(t)$ are integer times[†] satisfying $\tau_j^i(t) \le t, \quad \forall t$. At all times $t \notin T^i$, $x_i(t)$ is left unchanged and

$$x_i(t+1) = x_i(t), \qquad \forall t \notin T^i. \tag{2.3}$$

The difference $t - \tau_j^i(t)$ is related to the communication delay of the message x_j sent from processor j to processor i, and which is used in an update of x_i that starts at time t. In a synchronous execution, we have $t - \tau_j^i(t) = 0$. As $t - \tau_j^i(t)$ increases, we can say that the amount of asynchronism in the algorithm is larger. Of course, for the algorithm to make any progress at all, we should not allow $\tau_j^i(t)$ to remain forever small. Furthermore, no processor should be allowed to drop out of the computation and stop iterating. For this reason, the following assumption is introduced:

Assumption 2.1. The sets T^i are infinite and if $\{t_k\}$ is a sequence of elements of T^i which tends to infinity, then $\lim_{k\to\infty} \tau_j^i(t_k) = \infty$ for every j.

Asynchronous convergence under Assumption 2.1 has been established by several authors for a large variety of choices of the iteration mapping f, starting with the work of Chazan and Miranker [CM] (see [BT2] and the references therein). The following result originally given in [B] and reformulated in [BT2], seems to be the most general one.

Proposition 2.1. Suppose that for each $i \in \{1, \ldots, p\}$, there exists a sequence $\{X_i(k)\}$ of subsets of X_i such that:
(a) $X_i(k+1) \subset X_i(k)$, for all $k \ge 0$.
(b) The sets $X(k) = \prod_{i=1}^p X_i(k)$ have the property $f(x) \in X(k+1)$, for all $x \in X(k)$.
(c) All limit points of a sequence $\{x(k)\}$ with the property $x(k) \in X(k)$ for all k, are fixed points of f.
Furthermore, assume that $x(\tau) \in X(0)$ for all $\tau \le 0$. Then, under Assumption 2.1, all limit points of a sequence $\{x(t)\}$ generated by the asynchronous iteration (2.2)–(2.3) are fixed points of f.

We discuss briefly the assumption $x(\tau) \in X(0)$ for $\tau \le 0$. In the most common case, the algorithm is initialized at time 0 with some $x(0) \in X(0)$, and we have $\tau_j^i(t) \ge 0$ for all $t \ge 0$. In this case, the values of $x(\tau)$, $\tau < 0$, have no effect on the algorithm, they can be assumed without loss of generality to belong to $X(0)$, and the proposition applies. Another possible situation is the following. Suppose that until some time t^* the processors had been executing some other asynchronous iteration $x := g(x)$ and that at time t^* they start executing the asynchronous iteration $x := f(x)$ using the values $x(\tau)$ produced by the iteration $x := g(x)$ as initial conditions. As long as the original iteration was initialized with a vector in the set $X(0)$ and if the mapping g maps $X(0)$ into $X(0)$, we have $x(\tau) \in X(0)$ for all $\tau \le t^*$. We can then replace the time origin by t^* and use Prop. 2.1 to establish convergence.

The conditions of Prop. 2.1 can be easily verified in two important cases that are the subjects of Subsections 2.1 and 2.2, respectively.

2.2. Monotone mappings

Assumption 2.2. The iteration mapping $f : X \mapsto X$ has the following properties:
(a) f is continuous.
(b) f is monotone [that is, if $x \le y$ then $f(x) \le f(y)$].[†]
(c) f has a unique fixed point x^*.
(d) There exist vectors $u, v \in X$, such that $u \le f(u) \le f(v) \le v$.

Let f^k be the composition of k copies of f (f^0 is the identity mapping) and let

$$X(k) = \{x \mid f^k(u) \le x^* \le f^k(v)\}.$$

It is easily shown that $f^k(u)$ and $f^k(v)$ converge to x^*, as k tends to infinity. As a result, Prop. 2.1 applies and establishes asynchronous convergence, provided that the algorithm is initialized at some $x(0)$ satisfying $u \le x(0) \le v$.

Assumption 2.2 can be verified for a variety of algorithms, such as linear iterations involving nonnegative matrices, the Bellman–Ford algorithm for the shortest path problem, the successive approximation algorithm for infinite horizon dynamic programming, and dual relaxation algorithms for linear and nonlinear network flow problems [BT2].

2.2. Maximum norm contractions.

Let $X = \Re^n$, and consider a norm on $\Re^n$ defined by

$$\|x\| = \max_i \frac{\|x_i\|_i}{w_i},$$

[†] The values of the variables $\tau_j^i(t)$ for $t \notin T^i$ are of no importance. Still, it is sometimes convenient to assume that these variables are defined for all t. We interpret $x_j(\tau_j^i(t))$ as the value of x_j available to processor i at time t, even if $t \notin T^i$ and this value is not used in an update.

[†] Vector inequalities are to be interpreted componentwise throughout the paper.

where $x_i \in \Re^{n_i}$ is the ith component of x, $\|\cdot\|_i$ is a norm on $\Re^{n_i}$, and w_i is a positive scalar, for each i. (We call such a norm a *block-maximum norm.*) Suppose that f has the following contraction property: there exists some $\alpha \in [0,1)$ such that

$$\|f(x) - x^*\| \leq \alpha\|x - x^*\|, \qquad \forall x \in \Re^n, \quad (2.4)$$

where x^* is a fixed point of f. Given a vector $x(0) \in X$ with which the algorithm is initialized, let

$$X_i(k) = \left\{x_i \in \Re^{n_i} \mid \|x_i - x_i^*\|_i \leq \alpha^k\|x(0) - x^*\|\right\}.$$

It is easily verified that these sets satisfy the conditions of Prop. 2.1 and asynchronous convergence to x^* follows.

Iteration mappings f with the contraction property (2.4) are very common. We list a few examples:
(a) Linear iterations of the form $f(x) = Ax + b$, where A is an $n \times n$ matrix such that $\rho(|A|) < 1$ [CM]. Here, $|A|$ is the matrix whose entries are the absolute values of the corresponding entries of A, and $\rho(|A|)$, the *spectral radius* of $|A|$, is the largest of the magnitudes of the eigenvalues of $|A|$. As a special case, we obtain totally asynchronous convergence of the iteration $\pi := \pi P$ for computing a row vector π with the invariant probabilities of an irreducible, discrete-time, finite-state Markov chain specified in terms of the stochastic matrix P, provided that one of the components of π is held fixed throughout the algorithm [BT2, p. 435].
(b) Gradient iterations of the form $f(x) = x - \gamma\nabla F(x)$, where γ is a small positive stepsize parameter, $F : \Re^n \mapsto \Re$ is a twice continuously differentiable cost function whose Hessian matrix is bounded and diagonally dominant ([B], [BT2, p. 437]).

Other examples are the projection and other algorithms for the solution of variational inequalities (under certain diagonal dominance conditions), and waveform relaxation methods for the solution of ODEs or boundary value problems ([BT2], [M], [S]).

3. CONVERGENCE RATE COMPARISONS: MONOTONE ITERATIONS.

Throughout this section, we assume that Assumption 2.1 is in effect and that the iteration mapping f satisfies the monotonicity Assumption 2.2. The monotonicity assumption is very convenient for making convergence rate comparisons between different variants of the same algorithm. A classical example concerns the comparison of the Jacobi and Gauss-Seidel variants of the linear iteration $x := f(x) = Ax + b$ when A is a nonnegative matrix of spectral radius less than 1. In particular the Stein-Rosenberg Theorem [V] asserts that, in a serial computing environment, the Gauss-Seidel iteration converges at least as fast as its Jacobi counterpart. The result in the following subsection states that exactly the opposite is true in a parallel computing environment.

3.1. Comparison of synchronous Jacobi and Gauss-Seidel methods.

Let us restrict ourselves for the moment to a synchronous computing environment. In particular, we assume that component updates and the delivery of the results to every other processor can be accomplished within one time unit. A *Jacobi* iteration is described by the equation

$$x^J(t+1) = f\big(x^J(t)\big). \qquad (3.1)$$

In a *Gauss-Seidel* iteration, components are updated one at a time and the update of a component x_i uses updated values of the preceding components $x_1, \ldots, x_{i-1}$. In practice, the mapping f is usually sparse (that is, each function f_i depends only on a few of the components x_j) and in this case, the Gauss-Seidel iteration can be somewhat parallelized by having more than one (but usually not all) components being updated simultaneously. (This is accomplished by means of the well-known coloring procedure [BT2, Section 1.2.4]). Let $U(t)$ be the set of components that are updated at time t. Then, the Gauss-Seidel iteration is described by

$$x_i^U(t+1) = x_i^U(t), \qquad \text{if } i \notin U(t), \qquad (3.2)$$

and

$$x_i^U(t+1) = f_i\big(x_i^U(t)\big), \qquad \text{if } i \in U(t). \qquad (3.3)$$

The following result is proved in [T], generalizing an earlier result of [SW]:
Proposition 3.1. If $x^J(0) = x^U(0) = x(0)$ and the property $f\big(x(0)\big) \leq x(0)$ holds, then $x^* \leq x^J(t) \leq x^U(t)$ for all t.

Proposition 3.1 establishes the faster convergence of the Jacobi iteration, at least for special choices of initial conditions. [A symmetrical result holds if $x(0)$ satisfies $x(0) \leq f\big(x(0)\big)$]. It can also be shown [T] that for any initial conditions satisfying $x^* < x(0)$ or $x^* > x(0)$, there exists some constant K [depending on $x(0)$], such that $x^* \leq x^J(t+K) \leq x^U(t)$ for all t. (In words, the convergence rate of the Jacobi iteration cannot be worse than the convergence rate of the corresponding Gauss-Seidel iteration. A related effect has also been observed experimentally in the context of a specific example [ZL].) In the next subsection, these results are extended to obtain a much more general convergence rate comparison result. In particular, it will be shown that if the number of components updated at each time step is increased or if the size of the "communication delays" $t-\tau_j^i(t)$ is reduced, then the convergence rate can only improve.

3.2. Comparison between alternative asynchronous iterations.

We consider two alternative executions of the asynchronous iteration $x := f(x)$. We distinguish between them by putting a "hat" on the variables associated with the second execution. Thus, the first execution is decsribed by Eqs. (2.2) and (2.3), while the second by

$$\hat{x}_i(t+1) = \hat{x}_i(t), \qquad \text{if } t \notin \hat{T}^i, \tag{3.4}$$

$$\hat{x}_i(t+1) = f_i\big(\hat{x}_1(\hat{\tau}_1^i(t)),\ldots,\hat{x}_p(\hat{\tau}_p^i(t))\big), \ \text{ if } t \in \hat{T}^i, \tag{3.5}$$

Assumption 3.1. (a) For each i,j, and $t \geq 0$, we have $\tau_j^i(t+1) \geq \tau_j^i(t) \geq 0$ and $\hat{\tau}_j^i(t+1) \geq \hat{\tau}_j^i(t) \geq 0$.
(b) For each i, we have $T^i \supset \hat{T}^i$.
(c) For each i, j, and $t \in \hat{T}^i$, we have $\tau_j^i(t) \geq \hat{\tau}_j^i(t)$.

The requirements $\tau_j^i(t) \geq 0$ and $\hat{\tau}_j^i(t) \geq 0$ basically mean that the algorithm is started at time 0. Furthermore, Assumption 3.2(a) states that subsequent iterations by the same processor are based on newer information. It is essentially equivalent to an assumption that messages are received in the order that they are transmitted. Part (b) states that in the first execution there are at least as many variable updates as in the second. Finally, part (c) states that the communication delays in the first execution are no larger than those in the first.

Proposition 3.2. Suppose that:
(a) Assumption 3.1 holds.
(b) $x^* \leq x(0) = \hat{x}(0)$.
(c) $f(x(0)) \leq x(0)$.
Then, $x^* \leq x(t) \leq \hat{x}(t)$ for all t. [A symmetrical result holds if $\hat{x}(0) = x(0) \leq x^*$ and $f(x(0)) \geq x(0)$.]
Proof.
Lemma 3.1. There holds $x(t+1) \leq x(t)$ for all t.
Proof of Lemma 3.1. We proceed by induction on t. If $0 \in T^i$ then $x_i(1) = f_i(x(0)) \leq x_i(0)$; if $0 \notin T^i$ then $x_i(1) = x_i(0)$. Thus, $x(1) \leq x(0)$.

Let us now assume the induction hypothesis $x(t) \leq x(t-1) \leq \cdots \leq x(1) \leq x(0)$. If $t \notin T^i$ then $x_i(t+1) = x_i(t)$. If $t \in T^i$, we first consider the case where t is the first element of T^i. Then, $x_i(t) = x_i(0)$. Furthermore,

$$\begin{aligned} x_i(t+1) &= f_i\Big(x_1\big(\tau_1^i(t)\big),\ldots,x_p\big(\tau_p^i(t)\big)\Big) \\ &\leq f_i\big(x(0)\big) \leq x_i(0) = x_i(t), \end{aligned}$$

where the first inequality follows from $x_j\big(\tau_j^i(t)\big) \leq x_j(0)$, $j = 1,\ldots,p$, which is a consequence of the induction hypothesis. Finally, let us suppose that t is not the first element of T^i and let t' be the previous element of T^i. Using Assumption 3.1(a), we have $\tau_j^i(t) \geq \tau_j^i(t')$, and the induction hypothesis implies that $x_j\big(\tau_j^i(t)\big) \leq x_j\big(\tau_j^i(t')\big)$. Using the monotonicity of f, we obtain

$$x_i(t+1) = f_i\Big(x_1\big(\tau_1^i(t)\big),\ldots,x_p\big(\tau_p^i(t)\big)\Big)$$

$$\leq f_i\Big(x_1\big(\tau_1^i(t')\big),\ldots,x_p\big(\tau_p^i(t')\big)\Big) = x_i(t'+1) = x_i(t).$$

Q.E.D.

We now complete the proof of the proposition. We proceed again inductively. We have $x(0) = \hat{x}(0)$, by assumption, which starts the induction. We assume the induction hypothesis that $x(s) \leq \hat{x}(s)$ for $s = 0,1,\ldots,t$. We consider three cases:
(i) If $t \notin T^i$, then Assumption 3.1(b) implies that $t \notin \hat{T}^i$. It follows that $x_i(t+1) = x_i(t) \leq \hat{x}_i(t) = \hat{x}_i(t+1)$, where the induction hypothesis was used to obtain the inequality.
(ii) If $t \in T^i$ and $t \notin \hat{T}^i$ then $x_i(t+1) \leq x_i(t) \leq \hat{x}_i(t) = \hat{x}_i(t+1)$, where we have used Lemma 3.1 for the first inequality and the induction hypothesis for the second.
(iii) If $t \in T^i$ and $t \in \hat{T}^i$, we have $\tau_j^i(t) \geq \hat{\tau}_j^i(t)$ [Assumption 3.1(c)]. We then use Lemma 3.1 and the induction hypothesis to obtain $x_j\big(\tau_j^i(t)\big) \leq x_j\big(\hat{\tau}_j^i(t)\big) \leq \hat{x}_j\big(\hat{\tau}_j^i(t)\big)$. The inequality $x_i(t+1) \leq \hat{x}_i(t+1)$ then follows from the monotonicity of f. **Q.E.D.**

Notice that Prop. 3.1 can be obtained as a corollary of Prop. 3.2, by imposing the additional assumptions that $\tau_j^i(t) = \hat{\tau}_j^i(t) = t$ for all i,j,t, and that $T^i = \{0,1,2,\ldots\}$ for all i. While Prop. 3.2 deals with special choices of the initialization $x(0)$, it also provides worst case convergence rate comparisons for other initial conditions, as we now discuss.

Let us compare three asynchronous executions which are identical except for the choice of initial conditions. These three executions generate sequences $\{\underline{x}(t)\}$, $\{\hat{x}(t)\}$, and $\{\overline{x}(t)\}$, respectively, and are initialized with $\underline{x}(0) = u$, $\overline{x}(0) = v$, where u and v are the vectors of Assumption 2.2. Furthermore, we assume that $u \leq \hat{x}(0) \leq v$. As a consequence of the monotonicity of f, it is easily shown (by induction on t) that $\underline{x}(t) \leq \hat{x}(t) \leq \overline{x}(t)$ for all t. It follows that over all possible choices of initial conditions $x(0)$ satisfying $u \leq x(0) \leq v$, the slowest convergence to x^* is obtained by letting either $x(0) = u$ or $x(0) = v$. Consequently, if one is interested in the worst case convergence rate of two alternative methods, only the initial conditions $x(0) = u$ and $x(0) = v$ need to be considered. However, these initial conditions have the properties $f(u) \geq u$ and $f(v) \leq v$ and Prop. 3.2 applies. Coming back to the context of Prop. 3.2, we conclude that the worst case convergence rate of $\hat{x}(t)$ is at least as bad as the worst case convergence rate of $x(t)$, where the worst case is taken over all choices of initial conditions satisfying $u \leq x(0) \leq v$.

3.3. Comparison of synchronous and asynchronous iterations.

Let us now compare a synchronous iteration in which processors wait to receive certain messages before proceeding to the next update, with an asynchronous iteration in which processors perform updates at every time unit. Of course, in order to make a fair comparison, we have to assume that the communication delays in the two algorithms are the same.

We use $\{x(t)\}$ and $\{\hat{x}(t)\}$ to denote the sequence generated by the asynchronous and the synchronous iteration, respectively. Let the notation $r_j^i(t)$ and $\hat{r}_j^i(t)$ be as in the preceding subsection. As the asynchronous iteration performs an update at each time unit, we let T^i be the set of all nonnegative integers. In the synchronous iteration, an update is performed only when certain conditions are satisfied (that is, when all the information needed for the next update is available). So, we have $\hat{T}^i \subset T^i$, the inclusion being proper, in general. The assumption that the communication delays are the same for the two algorithms, translates to the condition $r_j^i(t) = \hat{r}_j^i(t)$ for all $t \in \hat{T}^i$. Finally, we assume that $r_j^i(t)$ is nondecreasing in t. Thus, Assumption 3.1 is satisfied and Prop. 3.2 applies. It follows that for any common choice of initial conditions such that $x(0) = \hat{x}(0)$ and $f(x(0)) \leq x(0)$, the convergence of the sequence $\{x(t)\}$ corresponding to the asynchronous iteration is faster than that of the synchronous sequence $\{\hat{x}(t)\}$. By a symmetrical argument, the same conclusion is reached if $x(0) \leq f(x(0))$. We can then argue as in the preceding subsection, to conclude that the worst case [over all initial conditions satisfying $u \leq x(0) \leq v$] convergence rate of the asynchronous variant is better than that of the synchronous one.

Notice that the condition $r_j^i(t) = \hat{r}_j^i(t)$ was imposed only for $t \in \hat{T}^i$. We now discuss a choice of the variables $r_j^i(t)$, $t \notin \hat{T}^i$, that results in the most fair comparison between the synchronous and the asynchronous iteration. In particular, we are going to assume that a processor executing the asynchronous algorithm sends a message only when the corresponding processor executing the synchronous algorithm sends a message. Furthermore, we shall assume that the delays suffered by corresponding messages are the same in the two algorithms. As long as messages are delivered in the order that they are received, $r_j^i(t)$ and $\hat{r}_j^i(t)$ are nonincreasing in t and, furthermore, we will certainly have $r_j^i(t) = \hat{r}_j^i(t)$ for all i, j and $t \in \hat{T}^i$. We are therefore dealing with a special case of what was discussed earlier in this subsection. This shows that the superiority of the asynchronous method holds under the most fair comparison, whereby both algorithms send the same number of messages and the messages have the same delays. We may conclude that, in the case of monotone iterations, it is preferable to perform as many updates as possible even if they are based on outdated information and, therefore, asynchronous algorithms are advantageous.

All of the discussion in this subsection has been based on the premise that an update by some processor takes one time unit and that the delays $t - r_j^i(t)$ are integer. In particular, if the delays are nonzero, they must be an integer multiple of the time needed for an update. The analysis extends without change to the case where the communication delays are noninteger but larger than 1. In effect, our analysis captures those cases where communication is more time–consuming than computation (as is often the case in practice). In fact, if the communication delays are smaller than the update time, then the synchronous algorithm can be slowed down by the communication delays by at most a factor of 2, in which case there does not seem to be any good reason for considering an asynchronous algorithm.

The case where the communication delays are smaller than the time needed for an update can also be studied analytically and it can be shown that the convergence rate of the asynchronous iteration could be worse than that of its synchronous counterpart. This reinforces our earlier statement that asynchronous iterations should be considered primarily when the communication delays are substantial.

4. CONVERGENCE RATE COMPARISON: CONTRACTING ITERATIONS.

Throughout this section we assume that Assumption 2.1 is in effect, that $X = \Re^n$, and that the iteration mapping $f : \Re^n \mapsto \Re^n$ has the following contraction property [cf. Eq. (2.4)]:

$$\max_i \frac{1}{w_i}\|f_i(x) - x_i^*\|_i \le \alpha \max_i \frac{1}{w_i}\|x_i - x_i^*\|_i, \quad \forall x, \tag{4.1}$$

where each $\|\cdot\|_i$ is a norm on $\Re^{n_i}$, each w_i is a positive scalar, and $0 \le \alpha < 1$.

To simplify the discussion, we assume that the communication delay of any message is equal to D time units, where D is a positive integer, and that a variable update takes a single time unit. Then, a synchronous algorithm performs one iteration every $D+1$ time units and the contraction property (4.1) provides us with the estimate

$$\|x(t) - x^*\| \le A\alpha^{t/(D+1)}, \tag{4.2}$$

where $\|x\| = \max_i \|x_i\|_i/w_i$ and where A is a constant depending on the initial conditions. We define $\rho_S = \alpha^{1/(D+1)}$ and view ρ_S as the convergence rate of the synchronous iteration. This is meaningful if Eq. (4.2) holds with approximate equality at least for some initial conditions or if Eq. (4.2) is the only available convergence rate estimate.

We impose an additional assumption on f:
Assumption 4.1. There exists some β such that $0 \le \beta \le \alpha$ such that for all x and i,

$$\frac{1}{w_i}\|f_i(x) - x_i^*\|_i \le \max\left\{\frac{\alpha}{w_i}\|x_i - x_i^*\|_i,\ \max_{j\ne i}\frac{\beta}{w_j}\|x_j - x_j^*\|_j\right\}.$$

Notice that in the case where $\beta = \alpha$, Assumption 4.1 coincides with Eq. (4.1). The case where β is smaller than α can be viewed as a weak coupling assumption. In particular, when $\beta = 0$, then x_i^* can be computed from knowledge of f_i alone and interprocessor communication is unnecessary. It is intuitively clear that when β is very small, the information on the values of the variables updated by other processors is not as crucial and that the performance of an asynchronous algorithm should be comparable to its performance under the assumption of zero delays. We now develop some results corroborating this intuition.

Consistently with our assumption that communication delays are equal to D, we assume that $\tau_j^i(t) = \max\{0, t-D\}$, for $j \ne i$, and that $\tau_i^i(t) = t$. (The latter equality reflects the fact that processor i need not send messages to itself and therefore no delay is incurred.) For the sake of some added generality, we actually make the assumption $\tau_j^i(t) \ge \max\{0, t-D\}$, for $j \ne i$. Under this assumption, we have the following result.
Proposition 4.1. Suppose that T^i is the set of all nonnegative integers for each i. Then, the sequence $\{x(t)\}$ of vectors generated by the asynchronous iteration satisfies

$$\|x(t) - x^*\| \le A\rho_A^t\|x(0) - x^*\|,$$

where ρ_A is a nonnegative solution of the equation

$$\rho = \max\{\alpha, \beta\rho^{-D}\}. \tag{4.3}$$

The proof of Prop. 4.1 is an easy inductive argument and can be found in [BT2, p. 441]. Notice that we either have $\rho_A = \alpha \le \alpha^{1/(D+1)} = \rho_S$ or $\rho_A = \beta\rho_A^{-D} \le \alpha\rho_A^{-D}$ which also yields $\rho_A \le \alpha^{1/(D+1)} = \rho_S$. In either case, the convergence rate of the asynchronous iteration is better.

We now consider two interesting limiting cases:
(a) Let us keep α and D fixed and suppose that β is small. (That is, we are considering the case where the iteration is very weakly coupled.) In particular, let us suppose that $\beta \le \alpha^{D+1}$. If $\rho_A = \beta\rho_A^{-D}$, then $\alpha^{D+1} \ge \beta = \rho_A^{D+1}$. On the other hand, $\rho_A \ge \alpha$, and we conclude that $\rho_A = \alpha$. Notice that the asynchronous convergence rate ρ_A is the same as the convergence rate α of the iteration $x(t+1) = f(x(t))$ which is a synchronous iteration without any delays. We conclude that when the "coupling strength" β is sufficiently small, then the communication delays have no effect on the asynchronous convergence rate. In particular, the asynchronous algorithm is $D+1$ times faster than its synchronous counterpart.
(b) Let us now consider the case where D tends to infinity (very large delays). It is clear that in this case ρ_S and ρ_A converge to 1. It is thus more meaningful to concentrate on the values of ρ_S^{D+1} and ρ_A^{D+1}. These can be viewed as the error reduction factors per phase of the synchronous iteration. For the synchronous iteration, ρ_S^{D+1} is of course equal to α. For the asynchronous iteration, ρ_A increases to 1 as D tends to infinity and, therefore, for D large enough, we will have $\rho_A > \alpha$. Then, Eq. (4.3) shows that $\beta\rho_A^{-D} = \rho_A$; equivalently, ρ_A^{D+1} is equal to β. Therefore, the convergence rate (per synchronous phase) is determined only by the coupling strength β. Once more we reach the conclusion that weakly coupled problems favor the asynchronous algorithm.

All of the above analysis can be carried through for the case where Assumption 4.1 is replaced by the related inequality

$$\frac{1}{w_i}\|f_i(x)-x_i^*\|_i \leq \frac{\gamma}{w_i}\|x_i-x_i^*\|_i+\max_{j\neq i}\frac{\beta}{w_j}\|x_j-x_j^*\|_j,$$

where $\alpha = \beta + \gamma < 1$. The main difference is that ρ_A is now a nonnegative solution of the equation

$$\rho = \gamma + \beta\rho^{-D},$$

as opposed to Eq. (4.3). It is easily shown that $\rho_A \leq \rho_S$, that ρ_A tends to γ when β is very small, and that ρ_A^{D+1} approaches $\beta/(1-\gamma) \leq \alpha = \rho_S^{D+1}$ as D increases to infinity. Thus, the qualitatitive conclusions we had derived under Assumption 4.1 remain valid for this case as well.

We have so far demonstrated the superiority of asynchronous iterations under the contraction condition. It can be argued, however, that the comparison is somewhat unfair for the following reason: we are assuming that communication delays are equal to D and that $\tau_j^i(t) = t - D$ for all $t \geq D$. This is equivalent to assuming that messages are transmitted by the processors executing the asynchronous algorithm at each time step. This corresponds to message transmissions at a rate $D+1$ higher than the message transmission rate in the synchronous algorithm. In order to make a more fair comparison, let us now consider an asynchronous iteration in which messages are transmitted only at integer multiples of $D+1$, that is, at the same times that the synchronous iteration is transmitting messages. Notice that processors will be receiving a message once every $D+1$ time units. Thus, at each update, the time elapsed since the last message reception can be at most D. Furthermore, messages carry information which is outdated by D time units. It follows that $t-\tau_j^i(t) \leq 2D$ for all t. We are therefore in the situation that was considered in Prop. 4.1, except that D is replaced by $2D$. In particular, if we assume that Assumption 4.1 holds, we obtain an asynchronous convergence rate estimate ρ_A, where ρ_A is a nonnegative solution of $\rho = \max\{\alpha, \beta\rho^{-2D}\}$. All of our earlier qualitative conclusions remain valid and, in particular, we have $\rho_A \leq \rho_S$, with the difference between $\rho_S - \rho_A$ being more pronounced in the case of weakly coupled iterations.

5. TERMINATION OF ASYNCHRONOUS ITERATIONS.

In practice, iterative algorithms are executed only for a finite number of iterations, until some termination condition is satisfied. In the case of asynchronous iterations, the problem of determining whether termination conditions are satisfied is a rather difficult problem because each processor possesses only partial information on the progress of the algorithm. We address this issue in this section.

While the general model introduced in Section 2 can be used for both shared memory and message–passing parallel architectures [BT2, Section 6.1], in this section we need to adopt a more explicit message–passing model. In particular, we assume that each processor j sends messages with the value of x_j to every other processor i. Processor i keeps a buffer with the most recently received value of x_j. We denote the value in this buffer at time t by $x_j^i(t)$. This value was transmitted by processor j at some earlier time $\tau_j^i(t)$ and therefore $x_j^i(t) = x_j(\tau_j^i(t))$. This model will be in effect throughout this section, and is easily seen to conform to the general model of Section 2.

5.1 Finitely terminating iterations.

We first consider asynchronous iterative algorithms that are guaranteed to terminate. For this to happen, we need to impose certain assumptions on the way that the algorithm is implemented:

Assumption 5.1. (a) If $t \in T^i$ and $x_i(t+1) \neq x_i(t)$, then processor i will eventually send a message to every other processor.
(b) If a processor i has sent a message with the value of $x_i(t)$ to some other processor j, then processor i will send a new message to processor j only after the value of x_i changes (due to an update by processor i).
(c) Messages are received in the order that they are transmitted.
(d) Each processor sends at least one message to every other processor.

According to Assumption 5.1(b), if the value of $x(t)$ settles to some final value, then there will be some time t^* after which no messages will be sent. Furthermore, all messages transmitted before t^* will eventually reach their destinations and the algorithm will eventually reach a quiescent state where none of the variables x_i changes and no message is in transit. We can then say that the algorithm has terminated.

We still assume that the sets T^i are infinite for each i. However, once the algorithm becomes quiescent any further updates will be inconsequential. It is not hard to see that the property $\lim_{t\to\infty} \tau_j^i(t) = \infty$ (cf. Assumption 2.1) follows from Assumption 5.1. This is because every processor eventually gets informed of the changes in the variables of the other processors [cf. Assumption 5.1(a)]. Also, if a processor i stops sending any messages (because x_i has stopped changing) then the last message received by processor j is the last message that was sent by processor i [due to Assumption 5.1(c)] and therefore processor j will have up-to-date information on x_i.

Let us now suppose that there exists a family $\{X(k)\}$ of nested sets with the properties introduced in Prop. 2.1. Furthermore, let us assume that there exists some k such that the set $X(k)$ consists of the single element x^*. Since we have just verified the validity of Assumption 2.1, it follows that we will eventually have $x(t) = x^*$; that is, the algorithm terminates in finite time. Notice that termination is equivalent to the following two properties:
(i) No message is in transit.
(ii) An update by some processor i causes no change in the value of x_i.
Property (ii) is really a collection of local termination conditions. There are several algorithms for termination detection when a termination condition can be decomposed as above (see [DS], [BT2, Section 8.1]). Thus termination detection causes no essential difficulties in this case.

5.2. Non-terminating algorithms.

Let us now shift our attention to the more interesting case of iterative algorithms that never terminate if left on their own. If we were dealing with the synchronous iteration $x(k+1) = f\big(x(k)\big)$, it would be natural to terminate the algorithm when the condition $\|x(t+1)-x(t)\| \leq \epsilon$ is satisfied, where ϵ is a small positive constant reflecting the desired accuracy of solution, and where $\|\cdot\|$ is a suitable norm. This suggests the following approach for the context of asynchronous iterations. Given the iteration mapping f and the accuracy parameter ϵ, we define a new iteration mapping $g : X \mapsto X$ by letting

$$g_i(x) = f_i(x), \qquad \text{if } \|f_i(x) - x_i\| \geq \epsilon,$$

$$g_i(x) = x, \qquad \text{otherwise.}$$

We will henceforth assume that the processors are executing the asynchronous iteration $x := g(x)$ and communicate according to Assumption 5.1. Once more, the termination condition for this iteration decomposes into a collection of local termination conditions and the standard termination detection methods apply. We will therefore concentrate on the question of whether eventual termination is guaranteed. One could argue as follows. Assuming that the original iteration $x := f(x)$ is guaranteed to converge to a fixed point x^*, the changes in the vector x will eventually become arbitrarily small, in which case we will have $g(x) = x$ and the iteration $x := g(x)$ will terminate. Unfortunately, this argument is fallacious, as demonstrated by the following example.

Example 5.1. Consider the function $f : \Re^2 \mapsto \Re^2$ defined by $f_1(x) = -x_1$, if $x_2 \geq \epsilon/2$, $f_1(x) = 0$, if $x_2 < \epsilon/2$, and $f_2(x) = x_2/2$. It is clear that the asynchronous iteration $x := f(x)$ is guaranteed to converge to $x^* = (0,0)$: in particular, x_2 is updated according to $x_2 := x_2/2$ and tends to zero; thus, it eventually becomes smaller than $\epsilon/2$. Eventually processor 1 receives a value of x_2 smaller than $\epsilon/2$ and a subsequent update by the same processor sets x_1 to zero.

Let us now consider the iteration $x := g(x)$. If the algorithm is initialized with x_2 between $\epsilon/2$ and ϵ, then the value of x_2 will never change, and processor 1 will keep executing the nonconvergent iteration $x_1 := -x_1$. Thus, the asynchronous iteration $x := g(x)$ is not guaranteed to terminate.

The remainder of this section is devoted to the derivation of conditions under which the iteration $x := g(x)$ is guaranteed to terminate. We introduce some notation. Let I be a subset of the set $\{1,\ldots,p\}$ of all processors. For each $i \in I$, let there be given some value $\theta_i \in X_i$. We consider the asynchronous iteration $x := f^{I,\theta}(x)$ which is the same as the iteration $x := f(x)$ except that any component x_i, with $i \in I$, is set to the value θ_i. Formally, the mapping $f^{I,\theta}$ is defined by letting $f_i^{I,\theta}(x) = f_i(x)$, if $i \notin I$, and $f_i^{I,\theta}(x) = \theta_i$, if $i \in I$. The main result is the following:

Proposition 5.1. Let Assumption 5.1 hold. Suppose that for any $I \subset \{1,\ldots,n\}$ and for any choice of $\theta_i \in X_i$, $i \in I$, the asynchronous iteration $x := f^{I,\theta}(x)$ is guaranteed to converge. Then, the asynchronous iteration $x := g(x)$ terminates in finite time.

Proof. Consider the asynchronous iteration $x := g(x)$. Let I be the set of all indices i for which the variable $x_i(t)$ changes only a finite number of times and for each $i \in I$, let θ_i be the limiting value of $x_i(t)$. Since f maps X into itself, so does g. It follows that $\theta_i \in X_i$ for each i. For each $i \in I$, processor i sends a positive but finite number of messages [Assumptions 5.1(d) and 5.1(b)]. By Assumption 5.1(a), the last message sent by processor i carries the value θ_i and by Assumption 5.1(c) this is also the last message received by any other processor. Thus, for all t large enough, and for all j, we will have $x_i^j(t) = x_i\big(\tau_i^j(t)\big) = \theta_i$. Thus, the iteration $x := g(x)$ eventually becomes identical with the iteration $x := f^{I,\theta}(x)$ and therefore converges. This implies that the difference $x_i(t+1) - x_i(t)$ converges to zero for any $i \notin I$. On the other hand, because of the definition of the mapping g, the difference $x_i(t+1) - x_i(t)$ is either zero, or its magnitude is bounded below by $\epsilon > 0$. It follows that $x_i(t+1) - x_i(t)$ eventually settles to zero, for every $i \notin I$. This shows that $i \in I$ for every $i \notin I$; we thus obtain a contradiction unless $I = \{1,\ldots,n\}$, which proves the desired result. **Q.E.D.**

We now identify certain cases in which the main assumption in Prop. 5.1 is guaranteed to hold. We consider first the case of monotone iterations and we assume that the iteration mapping f satisfies Assumption 2.2. For any I and $\{\theta_i \mid i \in I\}$, the mapping $f^{I,\theta}$ inherits the continuity and monotonicity properties of f. Let u and v be as in Assumption 2.2 and suppose that $X = \{x \mid u \leq x \leq v\}$. Let θ_i be such that $u_i \leq \theta_i \leq v_i$. Since f satisfies Assumption 2.2(d), we have $f^{I,\theta}(u) \geq u$ and $f^{I,\theta}(v) \leq v$. We conclude that the mapping $f^{I,\theta}$ satisfies parts (a), (b), and (d) of Assumption 2.2. Assumption 2.2(c) is not automatically true for the mappings $f^{I,\theta}$, in general; however, if it can be independently verified, then the asynchronous iteration $x := f^{I,\theta}$ is guaranteed to converge, and Prop. 5.1 applies. Let us simply say here that Assumption 2.2(c) can be verified for certain network flow algorithms, as well as for successive approximation algorithms for discounted and (a class of) undiscounted dynamic programming problems. (These results will be reported in more detail elsewhere.)

Let us now consider the case where f satisfies the contraction condition of Eq. (4.1). Unfortunately, it is not necessarily true that the mappings $f^{I,\theta}$ also satisfy the same contraction condition. In fact, the mappings $f^{I,\theta}$ are not even guaranteed to have a fixed point. Let us strengthen Eq. (4.1) and assume that

$$\|f(x) - f(y)\| \leq \alpha\|x - y\|, \qquad \forall x, y \in \Re^n, \quad (5.1)$$

where $\|\cdot\|$ is again a block–maximum norm, as in Eq. (4.1), and $\alpha \in [0,1)$. We have $f^{I,\theta}(x) - f^{I,\theta}(y) = \theta_i - \theta_i = 0$ for all $i \in I$. Thus,

$$\|f^{I,\theta}(x) - f^{I,\theta}(y)\| = \max_{i \notin I} \frac{1}{w_i}\|f_i(x) - f_i(y)\|_i$$
$$\leq \|f(x) - f(y)\| \leq \alpha\|x - y\|.$$

Thus, the mappings $f^{I,\theta}$ inherit the contraction property (5.1). As discussed in Section 2.2, this property guarantees asynchronous convergence and therefore Prop. 5.1 applies again.

We conclude that the modification $x := g(x)$ of the asynchronous iteration $x := f(x)$ is often, but not always, guaranteed to terminate in finite time. It is an interesting research question to devise economical termination procedures for the iteration $x := f(x)$ for those cases where the iteration $x := g(x)$ does not terminate.

REFERENCES

[B] D. P. Bertsekas, "Distributed asynchronous computation of fixed points," *Mathematical Programming, 27*, 1983, pp. 107–120.

[BG] D. P. Bertsekas and R. G. Gallager, *Data Networks*, Prentice Hall, Englewood Cliffs, NJ, 1987

[BT1]D. P. Bertsekas and J. N. Tsitsiklis, "Parallel and distributed iterative algorithms: a selective survey," Technical Report LIDS–P–1835, Laboratory for Information and Decision Systems, M.I.T., Cambridge, Mass., November 1988.

[BT2]D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods,* Prentice Hall, Englewood Cliffs, NJ, 1989.

[CM] D. Chazan and W. Miranker, "Chaotic relaxation", *Linear Algebra and its Applications, 2,* 1969, pp. 199–222.

[DS] E. W. Dijkstra and C. S. Sholten, "Termination detection for diffusing computations", *Inf. Proc. Lett.*, 11, 1980, pp. 1–4.

[K] H. T. Kung, "Synchronized and asynchronous parallel algorithms for multiprocessors", in *Algorithms and Complexity,* J.F. Traub (Ed.), Academic, 1976, pp. 153–200.

[M] D. Mitra, "Asynchronous relaxations for the numerical solution of differential equations by parallel processors", *SIAM J. Scientific and Statistical Computing, 8,* 1987, pp. s43–s58.

[S] P. Spiteri, "Contribution a l'etude de grands systemes non lineaires", Doctoral Dissertation, L' Universite de Franche–Comte, Besancon, France, 1984.

[SW]D. Smart and J. White, "Reducing the parallel solution time of sparse circuit matrices using reordered Gaussian elimination and relaxation", *Proceedings of the 1988 ISCAS*, Espoo, Finland.

[T] J. N. Tsitsiklis, "A comparison of Jacobi and Gauss–Seidel parallel iterations, to appear in *Applied Mathematics Letters.*

[V] R. S. Varga, *Matrix Iterative Methods.* Englewood Cliffs, NJ: Prentice-Hall, 1962.

[ZL] S. A. Zenios and R. A. Lasken, "Nonlinear network optimization on a massively parallel connection machine", *Annals of Operations Research,* 14, 1988.