Asynchronous Parallel Schemes: A Survey

Eric Jui-Lin Lu
Michael G. Hilgers
Bruce McMillin

November 1993

CSC 93-19

Abstract

It is well known that synchronization and communication delays are the major sources of performance degradation of synchronous parallel algorithms. It has been shown that asynchronous implementations have the potential to reduce the overhead to minimum. This paper surveys the existing asynchronous schemes and the sufficient conditions for the convergence of the surveyed schemes. Some comparisons among these schemes are also presented.

Department of Computer Science
University of Missouri-Rolla
Rolla, Missouri 65401

1 Notation

A $m \times n$ matrix A can be represented by $(a_{i,j})$ where $1 \le i \le m, 1 \le j \le n$ and $a_{i,j} \in \mathbb{R}$ is an element of A. Similarly, x_i, $1 \le i \le n$, is an element of the vector $x \in \mathbb{R}^n$. To avoid confusion, a sequence of vectors in $\mathbb{R}^n$ will be denoted by $x(k)$, $k = 0, 1, \ldots$. The absolute value of A (or x) will be a matrix $|A|$ (or a vector $|x|$) whose components are the absolute values of A (or x). A vector x is said to be positive (nonnegative) if each of its components is positive (nonnegative). The spectral radius of A is denoted as $\rho(A)$. $\mathbb{N}$ is the set of nonnegative integers.

The mathematical model of a fixed point problem is of the form:

$$x = F(x) \tag{1.1}$$

where $F : \mathbb{R}^n \to \mathbb{R}^n$ is a function and $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$. When $F(x)$ represents a system of equations, Equation (1.1) is usually written in the form of

$$x_i = f_i(x_1, \ldots, x_n) \quad \text{for all } i = 1, \ldots, n \tag{1.2}$$

where $F(x) = (f_1(x), \cdots, f_n(x))$ and $f_i(x)$ is a function from $\mathbb{R}^n$ to $\mathbb{R}$.

2 Introduction

There is a great deal of research devoted to solving a system of equations iteratively using parallel computers. In most of the work done in this area, the solutions of the system are obtained by distributing computational load among processors while maintaining coordination among processors by global shared memory or by message passing. The parallel algorithms of this type are called *synchronous parallel algorithms*.

Let $x_i(t)$ be the value of x_i residing in the memory of the ith processor at time t. A synchronous iterative execution of Equation (1.2), as defined in [1, 2], can be described mathematically by the formula

$$x_i(t+1) = \begin{cases} f_i(x_1(t), \ldots, x_n(t)), & \text{if } t \in T^i \\ x_i(t), & \text{otherwise.} \end{cases} \tag{2.1}$$

where t is an integer-valued variable used to index different iterations, not necessarily representing real time, and T^i, an infinite subset of $\{0, 1, 2, \ldots\}$, is the set of time indices at which x_i is updated.

In a synchronized iterative execution, each processor has to wait for all the other components to be updated before starting next iteration. For example, to numerically solve a system of two equations using a parallel computer of two processors, the second iteration of processor 1 cannot be executed until the updated component $x_2(1)$ is received from processor 2, and vice versa. This is shown in Figure 1.

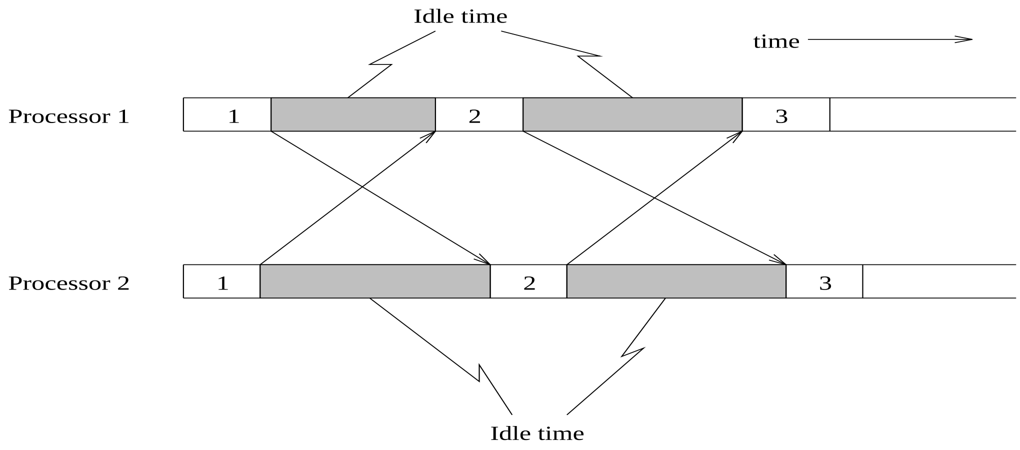

Figure 1: Synchronized Iterative Execution.

The shadow areas between two consecutive iterates are idle times. Thus, the performance of a synchronous algorithm is largely affected by communication channels and the slowest processor.

Asynchronous iterative execution, on the other hand, allows processors to continue computation without requiring them to wait for all the other components to be updated. Once a updated component arrives, this new value is incorporated in the next evaluation. This can be seen in Figure 2.

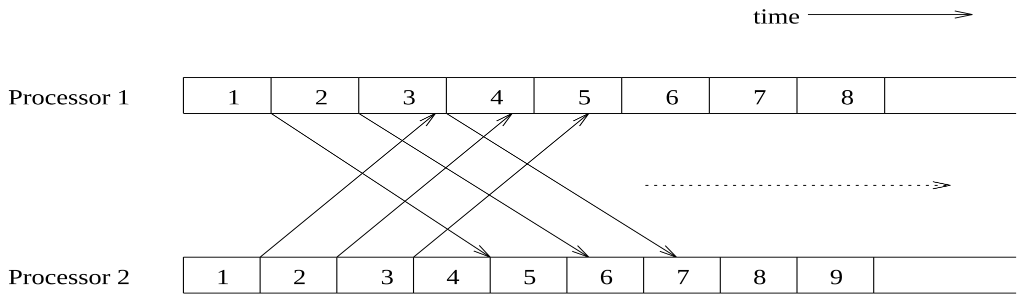

Figure 2: Asynchronous Iterative Execution.

Since synchronization is the major source of performance degradation, asynchronous iterations have the potential to outperform their synchronous counterparts. The developed results have been applied to a wide variety of iterative algorithms to solve problems such as dynamic programming, shortest path problems, network flow problems, unsupervised pattern clustering, consistent labeling, and artificial neural networks [3, 4, 5, 6].

The purpose of this paper is to survey the existing asynchronous schemes and their sufficient conditions for the convergence. Although this survey is by no mean complete, it gives readers a fairly good overview of work from the past three decades.

Additionally, we will analyze the relationship among the sufficient conditions for the convergence of the surveyed asynchronous schemes. For the convenience of such analysis, we present some definitions and theorem in next section. Then the surveyed models are presented in chronological order.

In the final section, we summarize the pros and cons of asynchronous models. With the knowledge of the asynchronous scheme, we also summaries the properties of asynchronous models. As indicated in the surveyed papers, there are some difficulties that need to be solved. These difficulties and the possible future research are illustrated in this section.

3 Definitions and Examples

In this section, we will present some definitions and theorem for the convenience of the analysis that we will make in the following sections. We also present some examples which have been proved in the literature.

Definition 3.1 *Let X be a set and d a function from $X \times X$ to the set $\mathbb{R}^+$ of non-negative real numbers satisfying the following properties. For all x, y, z in X,*

$$d(x,y) = 0 \quad \text{if and only if} \quad x = y; \tag{3.1}$$

$$d(x,y) = d(y,x); \tag{3.2}$$

$$d(x,z) \leq d(x,y) + d(y,z). \tag{3.3}$$

Then d is called a metric *or* distance function *on X and $d(x,y)$ is called the* distance *from x to y. The set X with metric d is called a* metric space *and is denoted by (X,d).*

It is easy to verify that the following functions are distance functions.

1. If $x, y \in \mathbb{R}$, $d(x,y) = |x-y|$.
2. Any norm can induce a metric by setting $\|x-y\| = d(x,y)$. Particular examples are:
   - $d(x,y) = \|x-y\|_1 = \sum_{i=1}^{n} |x_i - y_i|$.
   - $d(x,y) = \|x-y\|_2 = [\sum_{i=1}^{n} (x_i - y_i)^2]^{1/2}$
   - $d(x,y) = \|x-y\|_\infty = \max_{i=1,\dots,n} |x_i - y_i|$.

Definition 3.2 *Let (X,d) be a metric space and $F : X \to X$ a function. The F is* contractive *with respect to the metric d provided that there is a positive number $\alpha < 1$ such that, for all x, y in X,*

$$d(F(x), F(y)) \leq \alpha d(x,y). \tag{3.4}$$

Definition 3.3 *(X,d) is a* complete *metric space if every Cauchy sequence $\{x_n\}$ in X converges to a point in X. $\{x_n\}$ is a* Cauchy sequence *in X if, for any $\epsilon > 0$, there exists an $N \in \mathbb{N}$ such that $d(x_m, x_n) < \epsilon$, for all $m, n \geq N$.*

Theorem 3.1 (Banach's fixed point theorem) *Let (X,d) be a complete metric space and $F : X \to X$ a contractive function. Then F has a unique fixed point in X.*

Definition 3.4 *A normed space X with the norm $\|\cdot\|$ is called a Banach space if X is a complete metric space for the metric d defined on X by the formula,*

$$d(x,y) = \|x-y\|.$$

As proved in [10], both $(\mathbb{R}, |\cdot|)$ and $(\mathbb{R}^n, \|\cdot\|)$ are complete metric spaces and Banach spaces.

4 Chaotic Iteration

The idea of a chaotic iteration was introduced by Chazan and Miranker in 1969 [7]. The proposed model is used to solve systems of linear equations of the form

$$Ax = b \tag{4.1}$$

where A, an $n \times n$ matrix where $a_{i,j} \in \mathbb{R}$, is symmetric and positive definite, and $x, b \in \mathbb{R}^n$. Let D be a diagonal matrix where $d_{i,i} = a_{i,i}$ and $E = D - A$. We have $x = D^{-1}Ex + D^{-1}b$. Then let $B = D^{-1}E$ and $C = D^{-1}$, so we have

$$x = Bx + Cb.$$

Since B, C, and b are given, the operator F of Definition 4.1 is defined as:

$$F(x) = Bx + Cb. \tag{4.2}$$

Definition 4.1 *A chaotic iterative scheme $(F, \mathfrak{I})$ is a class of sequences of n-vectors, $x(j)$, $j = 0, 1, \ldots$. Each sequence in this class is defined recursively by*

$$x_i(j+1) = \begin{cases} x_i(j), & i \neq k_{n+1}(j) \\ f_i(x_1(j-k_1(j)), \cdots, x_n(j-k_n(j))), & i = k_{n+1}(j) \end{cases} \tag{4.3}$$

x(0) is given. $F : \mathbb{R}^n \to \mathbb{R}^n$ is a function of the form $F(x) = (f_1(x), \cdots, f_n(x))$. $\mathfrak{I}$ is a sequence of $n+1$-vectors such that $\mathfrak{I} = \{k_1(j), k_2(j), \ldots, k_{n+1}(j)\}$, $j = 0, 1, \ldots,$ with the following properties:

For some fixed integer $s > 0$

(a) $0 \leq k_i(j) < s$, *for* $i = 1, \ldots, n$ *and* $j = 0, 1, \ldots$

(b) $1 \leq k_{n+1}(j) \leq n$, *for* $j = 0, 1, \ldots$

Moreover, $k_{n+1}(j) = i$ *infinitely often for each* i, $1 \leq i \leq n$.

This definition may be interpreted as follows: At each instant of time j, the $k_{n+1}(j)$th component of $x(j)$ is updated while the remaining $n-1$ components are unchanged. The updating uses the first component of $x(j - k_1(j))$, the second component $x(j - k_2(j))$, etc. Every component is updated infinitely often, and no update uses a value of a component which was produced by an update s or more steps earlier.

Note that Definition 4.1 implies that no processor can drop out of computing forever. Furthermore, since $k_{n+1}(j)$ can be any value from 1 to n, the update sequence $\{k_{n+1}(1), k_{n+1}(2), \ldots\}$ is arbitrary as long as the condition (a) of Definition 4.1 is satisfied. For example, for a system of two equations, $\{k_3(1), k_3(2), \cdots\}$ can be $\{1, 2, 1, 2, \ldots\}$, $\{2, 1, 2, 1, \ldots\}$, $\{1, 1, 2, 1, \ldots\}$, etc. As a consequence, the problem is solved, using the chaotic relaxation, in chaotic appearance. In addition, any numerical method that follows a specific update sequence is a special case of the chaotic iteration. Gauss-Seidel and Jacobi methods, for examples, are two special cases of the chaotic iteration schemes.

Example 1 (Gauss-Seidel). Let $\mathcal{J}$ be defined by $k_1(j) = k_2(j) = ... = k_n(j) = 0$ and $k_{n+1}(j) \equiv (j \bmod n) + 1$. Here $s = 1$ and $k_{n+1}(j) = i$, for each $i = 1, ..., n$, exactly once in every n updates. This scheme is precisely the Gauss-Seidel relaxation procedure. For $n = 2$, we can see the execution sequence from the following table.

j	$k_3(j)$	$j - k_1(j)$	$j - k_2(j)$	$x_1(j+1)$	$x_2(j+1)$
0	1	0	0	$x_1(1) = f_1(x_1(0), x_2(0))$	$x_2(1) = x_2(0)$
1	2	1	1	$x_1(2) = x_1(1)$	$x_2(2) = f_2(x_1(1), x_2(1))$
2	1	2	2	$x_1(3) = f_1(x_1(2), x_2(2))$	$x_2(3) = x_2(2)$
3	2	3	3	$x_1(4) = x_1(3)$	$x_2(4) = f_2(x_1(3), x_2(3))$

Example 2 (Jacobi). Let $\mathcal{J}$ be defined by $k_1(j) = ... = k_n(j) = [k_{n+1}(j) - 1]$ and $k_{n+1}(j) \equiv (j \bmod n) + 1$. Here $s = n$ and $k_{n+1}(j) = i$, for each $i = 1, ..., n$, exactly once in every n updates. This scheme is precisely the Jacobi

relaxation procedure. For $n = 2$, we can see the execution sequence from the following table.

j	$k_3(j)$	$j - k_1(j)$	$j - k_2(j)$	$x_1(j+1)$	$x_2(j+1)$
0	1	0	0	$x_1(1) = f_1(x_1(0), x_2(0))$	$x_2(1) = x_2(0)$
1	2	0	0	$x_1(2) = x_1(1)$	$x_2(2) = f_2(x_1(0), x_2(0))$
2	1	2	2	$x_1(3) = f_1(x_1(2), x_2(2))$	$x_2(3) = x_2(2)$
3	2	2	2	$x_1(4) = x_1(3)$	$x_2(4) = f_2(x_1(2), x_2(2))$

In [7], Chazan and Miranker showed that

Theorem 4.1 *Let $(F, \mathfrak{I})$ be a chaotic iteration scheme. The sequence of iterates $x(j)$ generated by $(F, \mathfrak{I})$ converges to the solution of Equation (4.2) if and only if $\rho(|B|) < 1$.*

Now, considering the chaotic iteration scheme corresponding to over and under relaxation with parameter ω, we have

$$x = \omega(Bx + Cb) + (1 - \omega)x.$$

Or,

$$x = (I - \omega D^{-1}A)x + \omega Cb.$$

If $B^\omega = I - \omega D^{-1}A$, $C^\omega = \omega C$, then $F^\omega(x) = B^\omega x + C^\omega b$. The chaotic iteration with ω is denoted as $(F^\omega, \mathfrak{I})$.

Theorem 4.2 *The scheme $(F^\omega, \mathfrak{I})$ converges for all $\mathfrak{I}$ on assumptions of Definition 4.1 when $\rho(|B|) < 1$, and $0 < \omega < 2/(1 + \rho(|B|))$.*

In [8, 9], Miellou extended the chaotic iteration scheme. The extensions include (1) more than one component of $x(j)$ can be updated at each instant of j, and (2) the operator F can be non-linear.

Let $\{J_j\}$ be a sequence of non-empty index set where $j \in \{0, 1, \dots\}$ and $J_j \subset \{1, \dots, n\}$. Equation (4.3) is rewritten as

$$x_i(j+1) = \begin{cases} x_i(j), & i \notin J_j, \\ f_i(x_1(j - k_1(j)), \cdots, x_n(j - k_n(j))), & i \in J_j, \end{cases} \tag{4.4}$$

where i occurs in $\{J_j\}$ an infinite number of times. Moreover, it is assumed that every component of x will be updated at least once in every consecutive s iterates.

The extended model can be interpreted as follows: At each iterate j, every ith component of $x(j)$, for all $i \in J_j$, is updated while the remaining components are unchanged. All updates occur concurrently and use the first component of $x(j - k_1(j))$, the second component $x(j - k_2(j))$, etc. Every component is updated infinitely often, and no update uses a value of a component which was produced by an update s or more steps earlier.

Definition 4.2 *Let $F : D(F) \to E$ where $D(F) \subset E$ and $E = \prod_{i=1}^{n} E_i$ where $\{E_i\}$ is a set of Banach spaces. For any given point $u \in D(F)$, we say that F is contracting in u for the vector norm η if*

(a) *there exists a nonnegative $n \times n$ matrix T such that*

$$\eta(F(u) - F(v)) \leq T\eta(u - v), \quad \forall v \in D(F).$$

(b) $\rho(T) < 1$.

where $\eta(x) = (\|x_1\| \ \cdots \ \|x_n\|)$ for any $x \in E$.

In general, we say F is *pseudocontraction*. Note that Miellou defined the vector norm η as a canonical vector norm (la norme vectorielle canonique). For $E_i = \mathbb{R}$, $1 \leq i \leq n$, the vector norm $\eta(x) = (|x_1| \cdots |x_n|) = |x|$ for $x \in E = \mathbb{R}^n$. Thus, the vector norm η is still a vector when $E = \mathbb{R}^n$ and the condition (a) can be rewritten as

$$|F(u) - F(v)| \leq T|u - v|, \quad \forall v \in \mathbb{R}^n.$$

With the above definition, Miellou showed that

Theorem 4.3 *If F has a fixed point $x^* \in D(F)$ and is contracting in x^* for the vector norm η, the sequence $\{x(j)\}$, defined in (4.4), convergences to the fixed point x^*.*

Note that if F is contracting on $D(F)$ (ie. for all $x \in D(F)$), the above theorem also holds.

From the previous discussion, we know that the scheme proposed by Chazan and Miranker is a special case of Miellou's model. It is interesting to verify that

Corollary 1 *Let $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$ be an operator of the form $F(x) = Bx + C$. If $\rho(|B|) < 1$, then*

(a) *F has a unique fixed point x^*.*

(b) *F is contracting in x^* for the vector norm η.*

[**Proof:**] Since $(\mathbb{R}^n, \|\cdot\|_\infty^\omega)$, where

$$\|x\|_\infty^\omega = \max_{i=1,\dots,n} |\frac{x_i}{\omega_i}|, \quad \omega_i > 0$$

is a weighted maximum norm, is a complete metric space [10], it is sufficient to show that F is contracting on $\mathbb{R}^n$ with respect to $\|\cdot\|_\infty^\omega$ to prove (a).

Let $x, y \in \mathbb{R}^n$. Then,

$$\eta(F(x)-F(y)) = |B(x-y)| \leq |B||x-y| = |B|\eta(x-y), \quad \forall x, y \in \mathbb{R}^n.$$

Since $|B|$ is non-negative and $\rho(|B|) < 1$, then, see [9],

$$\|F(x) - F(y)\|_\infty^\omega \leq \alpha \|x - y\|_\infty^\omega$$

where $\alpha < 1$.

Let $d(x, y) = \|x - y\|_\infty^\omega$. By Banach's fixed point theorem, F has a unique fixed point $x^* \in \mathbb{R}^n$.

It has been proved in part (a) that F is contracting on $\mathbb{R}^n$. Thus, F is contracting in $x^* \in \mathbb{R}^n$ for the vector norm η. $\square$

5 Asynchronous Iterative Scheme

Motivated by [7, 8, 9], Baudet proposed an asynchronous iterative scheme [12] in 1978. Unlike the chaotic iteration scheme which does not allow use of the values which was produced by an update s or more step earlier, the asynchronous iterative scheme has no restriction on the choice of the antecedent values used in the evaluation of an iterate. Furthermore, the operator F considered in the asynchronous iterative scheme can be linear or non-linear. The formal definition of the scheme is presented below.

Definition 5.1 *Let F be an operator from $\mathbb{R}^n$ to $\mathbb{R}^n$. An asynchronous iteration corresponding to the operator F and starting with a given vector $x(0)$ is a sequence $x(j)$, $j = 0, 1, \dots,$ of vectors of $\mathbb{R}^n$ defined recursively by*

$$x_i(j+1) = \begin{cases} x_i(j) & \text{if } i \notin J_j \\ f_i(x_1(\tau_1(j)), \dots, x_n(\tau_n(j))) & \text{if } i \in J_j \end{cases} \tag{5.1}$$

where $\mathfrak{J} = \{J_j | j = 0, 1, 2, \dots\}$ is a sequence of nonempty subsets of $\{1, \dots, n\}$ and $\mathfrak{I} = \{(\tau_1(j), \dots, \tau_n(j)) | j = 0, 1, 2, \dots\}$ is a sequence of elements in $\mathbb{N}^n$.

In addition $\mathfrak{I}$ and $\mathfrak{J}$ are subject to the following conditions, for each $i = 1, \dots, n$:

(a) $0 \leq \tau_i(j) \leq j$, $j = 0, 1, \dots;$

(b) *$\tau_i(j)$, considered as a function of j, tends to infinity as j tends to infinity;*

(c) *i occurs infinitely many often in the set J_j, $j = 0, 1, 2, \dots$.*

An asynchronous iteration corresponding to F, starting with $x(0)$ and defined by $\mathfrak{I}$ and $\mathfrak{J}$, will be denoted by $(F, x(0), \mathfrak{J}, \mathfrak{I})$.

In the definition of the chaotic iteration, there exists a fixed integer s such that $\tau_i(j) \geq j - s \geq 0$ for $j = 0, 1, 2, \dots$ and $i = 1, 2, \dots$. Clearly, this condition implies the condition (b) of Definition 5.1, and, in the sense, asynchronous iterations provide a generalization of the chaotic iteration.

Baudet defined the contracting operator F as follows:

Definition 5.2 *An operator F from $\mathbb{R}^n$ to $\mathbb{R}^n$ is a contracting operator on a subset D of $\mathbb{R}^n$ if*

(a) *there exists a nonnegative $n \times n$ matrix T such that*

$$|F(x) - F(y)| \leq T|x - y|, \quad \forall x, y \in D.$$

(b) $\rho(T) < 1$.

Then, he proved the following theorem:

Theorem 5.1 *If F is a contracting operator on a closed subset D of $\mathbb{R}^n$ and if $F(D) \subset D$, then any asynchronous iteration $(F, x(0), \mathfrak{J}, \mathfrak{I})$ corresponding to F and starting with a vector $x(0)$ in D converges to the unique fixed point of F in D.*

Thus, for the sequence $\{x(j)\}$ generated by Equation (5.1), it converges to the unique fixed point if the operator F is contractive on the whole domain of the operator. On the other hand, for the sequence $\{x(j)\}$ generated by Equation (4.4), it converges to the unique fixed point only if the operator F is contractive in a point of the domain. Thus, the condition required for Equation (5.1) to converge is stronger than the one required by Miellou's model.

6 Asynchronous Fixed Point Algorithms

In 1983, Bertsekas proposed an algorithmic model for distributed computation of fixed points. As indicated in [13], the computation model is similar to the models presented by Chazan and Miranker[7], Miellou [8, 9], and Baudet[12]. This model has been further refined by Bertsekas and Tsitsiklis in [1, 2].

The asynchronous model in [1, 2] is defined as

$$x_i(t+1) = \begin{cases} f_i(x_1(\tau_1^i(t)), \ldots, x_n(\tau_n^i(t))), & \forall t \in T^i, \\ x_i(t), & \forall t \notin T^i. \end{cases} \tag{6.1}$$

where $\tau_j^i(t)$ are times satisfying

$$0 \leq \tau_j^i(t) \leq t, \quad \forall t \geq 0.$$

Note that, unlike the chaotic iteration [7, 8, 9] and the asynchronous iterative schemes [12] which all processors have the same set of $\{\tau_j(t)\}$ for $j \in \{1, \ldots, n\}$, the asynchronous model allows each processor i has its own set of $\{\tau_j^i(t)\}$. So, the chaotic iteration and the asynchronous iterative schemes are special cases of Equation (6.1). Any particular choice of the sets T^i and the values of the variables $\tau_j^i(t)$ is called a *scenario*.

For the algorithm to make any progress at all, it is not allowed to have $\tau_j^i(t)$ remain forever small. Furthermore, no processor should be allowed to drop out of the computation and stop iterating. Thus, certain assumptions must be imposed. And based on the following assumptions, the asynchronous model can be classified as total asynchronism and partial asynchronism.

Assumption 6.1 (Total asynchronism) *The sets T^i are infinite and if $\{t_k\}$ is a sequence of elements of T^i which tends to infinity, then $\tau_j^i(t_k)$ tends to infinity, for every j, when k tends to infinity.*

Assumption 6.2 (Partial asynchronism) *There exists a positive constant s such that:*

1. *For every $t \geq 0$ and every i, at least one of the elements of the set $\{t, t+1, \ldots, t+s-1\}$ belongs to T^i.*
2. *There holds*
$$t - s < \tau_j^i(t) \leq t, \quad \forall t \in T^i.$$
3. *There holds $\tau_i^i(t) = t$, for all i and $t \in T^i$.*

The constant s, called *asynchronism measure*, bounds the amount by which the information available to a processor can be outdated.

Total Asynchronism The sufficient conditions for the totally asynchronous algorithms to converge are presented in the following theorem.

Theorem 6.1 *Let $X = \prod_{i=1}^{p} X_i \subset \prod_{i=1}^{p} \mathbb{R}^{n_i}$. 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 $k \geq 0$.*

(b) *The sets $X(k) = \prod_{i=1}^{p} X_i(k)$ have the property $f(x) \in X(k+1)$, for all $x \in X$.*

(c) *Every limit point of a sequence $\{x(k)\}$ with the property $x(k) \in X(k)$ for all k, is a fixed point of f.*

Then, under total asynchronism, and if $x(0) \in X(0)$, every limit point of a sequence $\{x(t)\}$ generated by the asynchronous iteration is a fixed point of f.

The key idea behind Theorem 6.1 is that eventually $x(t)$ enters and stays in the set $X(k)$; furthermore, it eventually moves into the next set $X(k+1)$. But will the following situation, as illustrated in [14], happen?

> [**Example**] Suppose that D is defined by the following inequalities in $\mathbb{R}^2$:
> $$0 \leq x_1 \leq 2, \quad 0 \leq x_2 \leq x_1,$$
> and let the value of the function at the point $(2, 2)$ be $(1, 1)$, i.e.,
> $$F(2, 2) = (1, 1).$$

> If the initial value $x(0) = (2, 2)$ and the computation of x_1 is faster than x_2, then $x(1) = (1, 2)$ which is not a member of D. Thus, we cannot tell whether or not the sequence converges since $x(1)$ is out of the domain of F.

Due to the condition (b) in Theorem 6.1 which assures that $x(1)$ must be in $X(1) \subset X(0) \subset D$, the stated situation will be excluded. Bertsekas and Tsitsiklis call the condition (b) the *Box Condition*.

Partial Asynchronism As implied by the assumption of partial asynchronism, the value of $x(t+1)$ depends only on $x(t), x(t-1), \ldots, x(t-s+1)$, and not on any earlier values $x(\tau)$, $\tau \leq t-s$. Thus, old values are purged from the system after at most s time units. Now, let $z(t) = (x(t), x(t-1), \ldots, x(t-s+1))$ and $z(t) \in Z$ for all $t \geq 0$. Also, we denote by X^* the set $\{x \in X | x = f(x)\}$ of fixed points of f, and by Z^* the set of all elements of Z of the form $(x^*, \ldots, x^*)$, where x^* is an arbitrary element of X^*

Theorem 6.2 (Lyapunov Theorem) *Consider the asynchronous iteration. Suppose that f is continuous and that the partial asynchronism assumption holds. Suppose also that there exist a positive integer t^* and a continuous function $d : Z \mapsto [0, \infty)$ with the following properties:*

(a) *For every $z(0) \notin Z^*$ and for every scenario, we have $d(z(t^*)) < d(z(0))$.*

(b) *For every $z(0) \in Z$, for every $t \geq 0$, and for every scenario, we have $d(z(t+1)) \leq d(z(t))$.*

Then, we have $z^* \in Z^*$ for every limit point $z^* \in Z$ of the sequence $\{z(t)\}$.

7 Generalized Asynchronous Iterations

The variables manipulated in all of the asynchronous schemes discussed in the previous sections are continuous and defined in the vector space $\mathbb{R}^n$. However, many relaxation techniques involve the manipulation of symbolic or discrete-valued data. To solve the problems of discrete-valued data asynchronously, Uresin and Dubois, in 1986, proposed their generalized asynchronous scheme. In [15, 14, 6], they showed that, if the mapping is contractive, a generalized asynchronous relaxation converges to a unique fixed point in an arbitrary set, finite or infinite, countable or not. The results are used to prove the convergence of three asynchronous algorithms for the all-pairs shortest path problem, the scene-labeling problem, and a neural net model.

Definition 7.1 *Let S be an arbitrary set, finite or infinite, countable or not. Let F be an operator from S^n to S^n. An asynchronous iteration corresponding to the operator F and starting with a given vector $x(0)$ is a sequence $x(j)$, $j = 0, 1, \ldots$, of vectors of S^n defined recursively by*

$$x_i(j+1) = \begin{cases} x_i(j) & \text{if } i \notin J_j \\ f_i(x_1(\tau_1(j)), \ldots, x_n(\tau_n(j))) & \text{if } i \in J_j \end{cases} \tag{7.1}$$

where $\mathfrak{J} = \{J_j | j = 0, 1, 2, \ldots\}$ is a sequence of nonempty subsets of $\{1, \ldots, n\}$ and $\mathfrak{I} = \{(\tau_1(j), \ldots, \tau_n(j)) | j = 0, 1, 2, \ldots\}$ is a sequence of elements in $\mathbb{N}^n$.

In addition $\mathfrak{I}$ and $\mathfrak{J}$ are subject to the following conditions, for each $i = 1, \ldots, n$:

1. *$\tau_i(j) \leq j$, $j = 1, 2, \ldots$;*
2. *$\tau_i(j)$, considered as a function of j, tends to infinity as j tends to infinity;*
3. *i occurs infinitely many often in the set J_j, $j = 1, 2, \ldots$.*

An asynchronous iteration corresponding to F, starting with $x(0)$ and defined by $\mathfrak{I}$ and $\mathfrak{J}$, will be denoted by $(F, x(0), \mathfrak{J}, \mathfrak{I})$.

The above asynchronous scheme is directly derived from Baudet's asynchronous scheme. However, since the considered operator is defined from S^n to S^n instead of from $\mathbb{R}^n$ to $\mathbb{R}^n$, Uresin and Dubois defined the contracting operator as follows:

Definition 7.2 *An operator F from S^n to S^n is an asynchronously contracting operator on a subset $D = D_1 \times D_2 \times \cdots \times D_n$ of S^n if and only if there is a sequence $\{A(k)\}$ of subsets of S^n, such that*

(a) *$A(0) = D$,*

(b) *$x \in A(k)$ implies $F(x) \in A(k+1)$ for all $k = 0, 1, \ldots$,*

(c) *$A(k+1) \subseteq A(k)$ for all $k = 0, 1, \ldots,$*

(d) *$\lim_{k\to\infty} A(k) = \{\xi\}$,*

where ξ is the point of convergence.

With the generalized definitions of asynchronous iteration and contracting operator, they proved that

Theorem 7.1 *If F is an asynchronously contracting operator on a subset $D = D_1 \times \cdots \times D_n$ of S^n, then an asynchronous iteration $(F, x(0), \mathfrak{J}, \mathfrak{I})$ corresponding to F and starting with a vector $x(0)$ in D converges to a unique fixed point of F in D.*

Except for the above theorem, Uresin and Dubois also proposed an interesting convergence criteria which is applicable to all cases where the synchronized version of the iteration is known to converge.

Theorem 7.2 *An asynchronous iteration $(F, x(0), \mathfrak{J}, \mathfrak{I})$ corresponding to F, which is defined in S^n, converges to a unique fixed point, ξ, if the following conditions hold:*

(a) *there exists an ordering relation such that if $x \leq x'$, then $F(x) \leq F(x')$, and also $F(x) \leq x$ for all $x, x' \in S^n$, and*

(b) *the synchronous iteration corresponding to F and starting with $x(0) \geq \xi$ converges to ξ.*

8 Summary

According to the definitions in [1, 2], the models proposed in [7, 8, 9, 16] are partial asynchronous algorithms. On the other hand, the models proposed in [12, 15, 14, 6] are totally asynchronous algorithms.

It is well known that $\rho(A) < 1$ is a necessary and sufficient condition for a system of linear equations, $F(x) = Ax + b$, to converge sequentially (or synchronously). It can be easily shown that, if $\rho(A) < 1$ and A is nonnegative, then F is contractive on $\mathbb{R}^n$ with respect to weighted maximum norm $\|\cdot\|_\infty^\omega$. From the discussions in Sections 4 and 5, we know that the asynchronous algorithms converge if the operator F is contractive with respect to the weighted maximum norm. Thus, we can conclude that

Proposition 1 *For solving a system of linear equations $F(x) = Ax + b$ where A is nonnegative, an asynchronous algorithm (either partial or total) converges if its counterpart synchronous or sequential algorithm converges.*

Note that if A is negative, then the above proposition may be not true because $\rho(A) \leq \rho(|A|)$. Even when $\rho(A) < 1$, it is possible that $\rho(|A|) \geq 1$. An example can be seen in pages 435-437 [1].

8.1 Pros and Cons

From the previous discussions, we can now summarize the potential advantages that may be gained from asynchronous execution:

- *Reduction of the synchronization penalty.* The reduction is obtained by allowing processes to continue evaluation without requiring them to wait for all the other components to be updated.
- *Reduction of programming complexity* The chaotic form of the relaxation eliminated considerably programming time in coordinating the processes.
- *Reduction of the effect of bottlenecks.* In the synchronous executions, processes have to wait for the slowest process among them at a synchronization point. This is not true in the asynchronous execution.
- *Convergence acceleration due to the the incorporation of the newest values into the execution whenever available.* Whenever a new value is evaluated or received from other processor, asynchronous algorithms incorporate such new values into next computation. This, in general, will accelerate the convergence.

Though the asynchronous algorithms have several advantages over the synchronous algorithms, it has some drawbacks:

- Due to the chaotic ordering, the asynchronous iteration could be divergent even if the corresponding synchronous (or sequential) iteration converges. One example can be found in [1] at page 438.

- The analysis of convergence, convergence rate, and stability is often difficult even if the convergence of the asynchronous iteration can be established.

- An asynchronous algorithm may have converged (within a desired accuracy) but the algorithm does not terminate because no processor is aware of this fact. This results from the fact that, for asynchronous execution, each processor has only partial information on the progress of the execution. However, the determination of whether termination conditions are satisfied is based on the global information.

8.2 Implementation

In this section, we give an example to illustrate some situations which may occur when the asynchronous algorithms are implemented. We use the model proposed by Bertsekas and Tsitsiklis. With minor modification, other models also apply.

[**Example**] Consider an asynchronous algorithm involving two processors and a system of two linear equations defined as follows:

$$\begin{cases} x_1 = 0.2x_1 + 0.6x_2, \\ x_2 = 0.6x_1 + 0.4x_2. \end{cases} \tag{8.1}$$

Processor i keeps local information of $x^i = (x_1^i, x_2^i)$, updates x_i^i at times $t \in T^i$, and transmits x_i^i to the other processor.

As shown in Figure 3, the following events occur:

t=0: Processor 1 updates x_1^1 using $(x_1^1(0), x_2^1(0))$ and transmits it to processor 2, where it is received at time between $t = 1$ and $t = 2$. $x^1 = (x_1^1(1), x_2^1(0))$, denoted as $(1, 0)$ in Figure 3, between $t = 0$ and $t = 1$. Processor 2 updates x_2^2 using $(x_1^2(0), x_2^2(0))$ and transmits it to processor 1, where it is received at time between $t = 1$ and $t = 2$. $x^2 = (x_1^2(1), x_2^2(0))$, denoted as $(0, 1)$ in Figure 3, during $t = 0$ and $t = 1$.

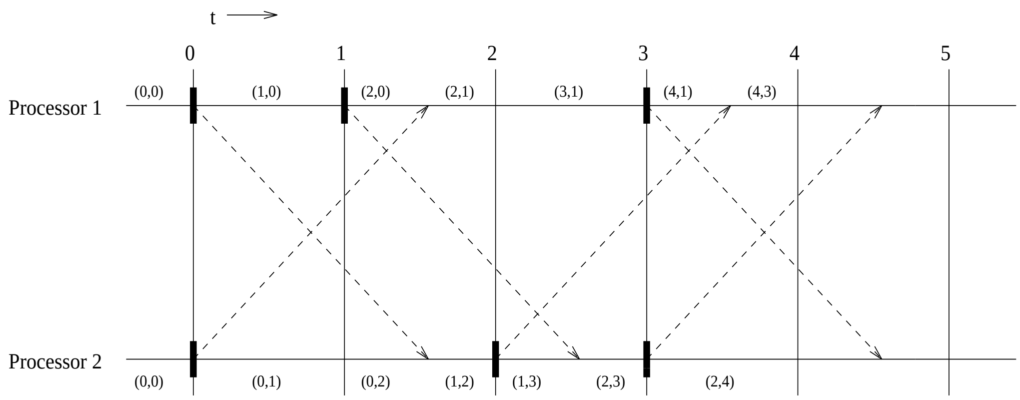

Figure 3: Asynchronous Iterative Execution.

t=1: Processor 1 updates x_1^1 using $(1,0)$ and transmits it to processor 2, where it is received at time between $t = 2$ and $t = 3$. $x^1 = (2,0)$ after $t = 1$ and before $x_2^2(1)$ is received. $x^1 = (2,1)$ after $x_2^2(1)$ is received. Processor 2 does not update. However, according to Equation (6.1), $x^2 = (0,2)$ after $t = 1$ and before $x_1^1(1)$ is received. After $x_1^1(1)$ is received, $x^2 = (1,2)$. Note that $x_2^2(1) = x_2^2(2)$.

t=2: Processor 2 updates x_2^2 and transmits it to processor 1 where it is received at time between $t = 3$ and $t = 4$.

t=3: Processor 1 updates x_1^1 and transmits it to processor 2 where it is received at time between $t = 4$ and $t = 5$. Processor 2 updates x_2^2 and transmits it to processor 1 where it is received at time between $t = 4$ and $t = 5$.

As a result, $T^1 = \{0, 1, 3, \dots\}$ and $T^2 = \{0, 2, 3, \dots\}$.

When the above asynchronous algorithm is implemented, it is likely that the message $x_2^2(1)$ is received by processor 1 at time between $t = 0$ and $t = 1$. The situation is shown in Figure 4.

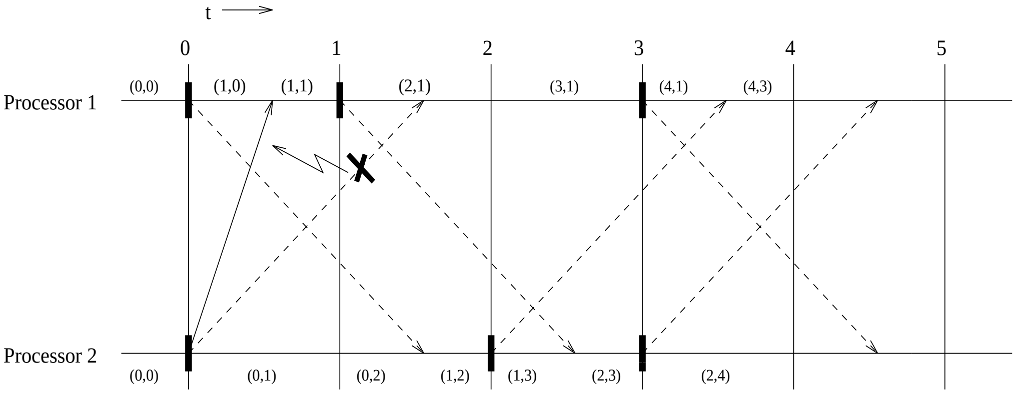

Figure 4: Asynchronous Iterative Execution.

As you can see from Figure 4, $x_1^1(2)$ is evaluated using $x^1 = (1,1)$ instead of $x^1 = (1,0)$ as shown in Figure 3. Consequently, the running result may vary unless there is only one unique solution. For a system as Equation (8.1), there is a unique solution; in other words, the running results are identical in both cases shown in Figures 3 and 4. However, for a system as Equation (8.2), the running result may vary.

$$\left\{ \begin{array}{l} x_1 = 0.4x_1 + 0.6x_2, \\ x_2 = 0.6x_1 + 0.4x_2. \end{array} \right. \tag{8.2}$$

Readers are encouraged to verify this with the initial value $x(0) = (1,2)$ and the asynchronous execution shown in Figure 5.

Furthermore, since each processor has only partial (or local) information of the whole system, the following conditions may occur.

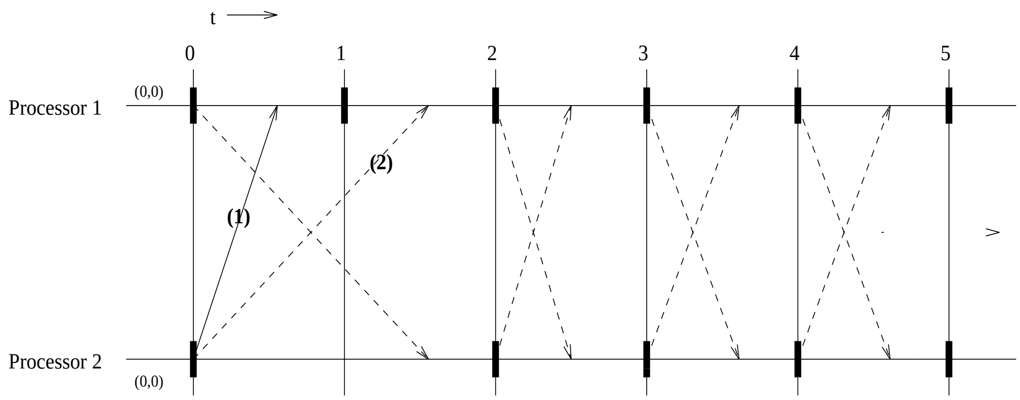

Figure 5: Asynchronous Iterative Execution.

- Let asynchronism measure $s = 2$. Then $1 < \tau_2^1(3) \leq 3$; in other words, $\tau_2^1(3)$ can be 2 or 3. So, processor 1 cannot update at $t = 3$ until $x_2^2(2)$ or $x_2^2(3)$ arrives. However, in practice, since processor 1 has no global information, the local t value on processor 1 is 2 and, thus, processor 1 updates at $t = 3$. In other words, each processor increases its local t value by one whenever there is an update. Since processor 1 does not know there is an update on processor 2 at $t = 2$, the local t value on processor 1 is 2 at $t = 3$ and processor 1 will update x_1^1 without knowing it has to wait until $x_2^2(2)$ arrives. This may break the assumptions made for partial asynchronous algorithms. Though the stated situation may be solved by employing vector clock and rollback [17, 18], the overhead is too huge to make asynchronous parallel algorithms be attractive.

- Processors may converge to some point which is not the solution. For example, to solve Equation (8.1), assume the initial value is $x(0) = (1, 2)$. Then processor 1 may converge to $(1.5, 2)$ if it does not receive any new value from processor 2. This situation can be corrected by dedicating a node to check if the global solution is reached.

Since it is not practical to specify the scenario when you implement an asynchronous algorithm, it is natural to evaluate x^i by using the latest updated components. However, it implies the update sequence may vary from one run to another and be non-reproducible.

8.3 Future Research

As seen in Figures 1 and 2, it is intuitive to assume that asynchronous algorithm runs faster than its corresponding synchronous algorithm. However, is it possible that asynchronous algorithm converges slower than synchronous one? Also, in the synchronous algorithm shown in Figure 6, will asynchronous algorithm run at least as fast as synchronous one? Miellou obtained the asymptotic rate of convergence for the chaotic iterative scheme [9]. Bertsekas and Tsitsiklis showed that, for solving system of linear equations, partial asynchronous algorithm converges geometrically [2]. It is interesting to know the convergence rate for totally asynchronous algorithm and, if possible, how to optimize the convergence rate.

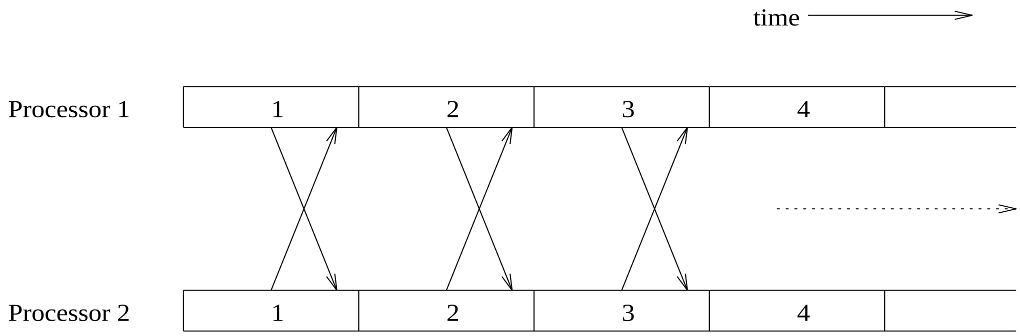

Figure 6: Synchronous Iteration with Overlapping Communication and Computation.

There have been some works done in solving systems of non-linear equations asynchronously. They all assumed that there is a unique fixed point and proved that the asynchronous scheme converges to that fixed point. However, considering the errors introduced by computers when they are used to solve the system numerically, it will be interesting to see if the asynchronous scheme indeed converges to the expected fixed point.

At present, there is very strong evidence suggesting that asynchronous iterations converge faster than their synchronous counterparts. However, this evidence is principally based on analysis and simulation. There is only a small number of related experimental works. Furthermore, the proper implementation of asynchronous algorithms in real parallel machines can be quite challenging and more experience is needed in this area.

References

[1] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods.* Prentice-Hall, Inc., 1989.

[2] D. P. Bertsekas and J. N. Tsitsiklis, "Some aspects of parallel and distributed iterative algorithms – a survey," *Automatica*, vol. 27, no. 1, pp. 3–21, 1991.

[3] D. P. Bertsekas and D. A. Castanon, "Parallel synchronous and asynchronous implementations of the auction algorithm," *Parallel Computing*, vol. 17, pp. 707–732, 1991.

[4] E. M. Chajakis and S. A. Zenios, "Synchronous and asynchronous implementations of relaxation algorithms for nonlinear network optimization," *Parallel Computing*, vol. 17, pp. 873–894, 1991.

[5] P. Tseng, D. P. Bertsekas, and J. N. Tsitsiklis, "Partially asynchronous, parallel algorithms for network flow and other problems," *SIAM Journal of Control and Optimization*, vol. 28, pp. 678–710, May 1990.

[6] A. Uresin and M. Dubois, "Parallel asynchrnous algorithms for discrete data," *Journal of ACM*, vol. 37, pp. 588–606, July 1990.

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

[8] J. C. Miellou, "Iterations chaotiques a retards, etudes de la convergence dans le cas d'espaces partiellement or donnes," *CRAS, serie A t.278*, pp. 957–960, 1974.

[9] J. C. Miellou, "Algorithmes de relaxation chaotique a retards," *RAIRO-R1*, pp. 55–82, 1975.

[10] F. H. Croom, *Principles of Topology.* Saunders College Publishing, 1989.

[11] R. L. Burden and J. D. Faires, *Numerical Analysis.* PWS-KENT Publishing Company, fourth ed., 1989.

[12] G. M. Baudet, "Asynchronous iterative methods for multiprocessors," *Journal of the Association for Computing Machinery*, vol. 25, pp. 226–244, April 1978.

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

[14] A. Uresin and M. Dubois, "Sufficient conditions for the convergence of asynchronous iterations," *Parallel Computing*, vol. 10, pp. 83–92, 1989.

[15] A. Uresin and M. Dubois, "Generalized asynchronous iterations," in *Proceedings of the Conference on Algorithms and Hardware for Parallel Processing*, pp. 272–278, September 1986.

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

[17] F. Mattern, "Virtual tiem and global states of distributed systems," in *Procedings of International Workshop on Parallel and Distributed Algorithms*, pp. 215–226, Oct. 1988.

[18] C. Fidge, "Logical time in distributed computing systems," *Computer*, pp. 28–33, Aug. 1991.