Journal of Parallel and Distributed Computing **58**, 68–91 (1999)
Article ID jpdc.1999.1543, available online at http://www.idealibrary.com on IDEAL®

Stochastic Performance Prediction for Iterative Algorithms in Distributed Environments

Henri Casanova,*, [1] Michael G. Thomason,[†] and Jack J. Dongarra[‡]

**Department of Computer Science and Engineering, University of California at San Diego, La Jolla, California 92093-0114;*
[†]Department of Computer Science, 104 Ayres Hall, University of Tennessee, Knoxville, Tennessee 37996-1301;
[‡]Department of Computer Science, 104 Ayres Hall, University of Tennessee, Knoxille, Tennessee 37996-1301 and Mathematical Science Section, P.O. Box 2008, Building 6012, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37821-6367
E-mail: casanova@cs.ucsd.edu, thomason@cs.utk.edu, dongarra@cs.utk.edu, dongarra@msr.emp.ornl.gov

Received April 20, 1998; accepted February 24, 1999

The parallelization of iterative algorithms is an important issue for efficient solution of large numerical problems. Several theoretical results concerning sufficient conditions for, and speed of, convergence of parallel iterative algorithms are available. However, those results usually do not take into account the processor workloads and network communications at the application level. The approach in this paper develops a Markov chain based on random variables which describe aspects of the multiuser, distributed-memory environment and the phases of the algorithm. The performance characterization addresses stochastic characteristics of the algorithmic execution time such as mean values and standard deviations. We present simulation results as well as experimental results over different time periods. The results provide information about the impact of distributed environment and implementation style on long-run, expected execution time characteristics. © 1999 Academic Press

Key Words: convergence; distributed-memory; iterative algorithm; Markov chain; multiuser; parallel; performance prediction; stochastic modeling.

1. INTRODUCTION

Iterative algorithms are widely used in different areas of science and engineering, e.g., medical imaging [22] and network flow in electrical networks, communication networks, and financial models [5]. A broad class of iterative algorithms aims at finding a fixed point of a given operator. Many well-known numerical methods use

[1] Corresponding author.

such an algorithm with linear or nonlinear operators. For problems with large dimension and/or extensive numerical computation for each component of the solution vector at each iteration (e.g., gradient approximations or Hessian computations for nonlinear operators), it is natural to consider parallel implementations of iterative algorithms.

Analyzing the behavior and, thus, the performance of a parallelized iterative algorithm in a distributed environment is not an easy task. The amount of computation performed by a processor to update a component of the solution vector often is not known a priori and in nonlinear cases may depend on the operator. For instance, in popular algorithms to compute approximations of the gradient of the operator at each iteration (e.g., gradient descent algorithms), the amount of computation required to approximate the value of a component of the gradient (say the ith component) depends on the *shape* (*or geometry*) of the operator along direction i at the current solution vector. Furthermore, if distributed iteration is implemented on a nondedicated system with other users, the computation and communication of those users impact the availability of processor and network resources to the iterative algorithm.

In this research, we focus on nondedicated, distributed-memory environments such as clusters of processors on a network. There are existing convergence results that indirectly support a quantitative assessement of the parallel algorithm convergence rate, but almost all these results are purely theoretical and do not take into account the nature of the multiuser, distributed environment itself. Commonly available results are lower bounds on the algorithm theoretical rate of convergence.

Due to nondeterminism (randomness) both in network communications and in computational workloads at processors, stochastic methods appear to be a natural way to move towards more comprehensive models. These models should capture the fluctuations of the distributed environment and the algorithm and their impact on the user's implementation, at least in terms of average or "expected" characteristics of execution time conditioned on the assumptions made to obtain tractable models. In the following sections, we give definitions and assumptions concerning parallel iterative algorithms in multiuser, distributed-memory environments, introduce application-level models of the distributed environment and the algoritm which lead to a finite-state, time-homogeneous Markov chain; discuss performance characterizations related to convergence, and describe results of simulations and experiments.

2. PARALLEL ITERATIVE ALGORITHMS

We use two, well-established paradigms for parallel implementation of iterative algorithms: *synchronous and asynchronous*. In [27], it is shown that asynchronous implementations have "good" communication complexity as compared to synchronous ones, but it is difficult to use these results to obtain quantitative estimates of actual performance in a given distributed environment. A performance comparison between the two paradigms is provided in [8, Section 6.3.5], but the model is nonrandom and does not describe nondeterministic systems.

A unique reference that proposes a stochastic approach is [30] which gives an analysis of asynchronous iteration with expected values based on [18]. This work is of interest for shared-memory homogeneous environments and *age scheduling* strategy for which processors-execution times are described by a specific family of probability distributions (increasing failure rate (IFR) functions) with simulation used to approximate some parameters. In contrast, this paper focuses on distributed-memory systems that may be nonhomogeneous and assumes *static scheduling* (which simulations in [30] show to be superior) to develop a Markov chain model.

We consider iterative algorithms in which an operator is applied repetitively to a vector of real-valued data until some convergence criteria are met. Let $\mathbb{R}$ denote the reals and $\mathbb{N}$ the nonnegative integers $\{0, 1, 2, ...\}$. The computation of the algorithm is a sequence of vectors $\{x(t)\}$ in $\mathbb{R}^m$, and the iteration can be written as

$$\begin{aligned} &x(0) \in \mathbb{R}^m \\ &x(t+1) = Op(x(t)) \in \mathbb{R}^m \qquad \text{for all} \quad t \in \mathbb{N}. \end{aligned} \tag{1}$$

If the algorithm converges, the sequence $\{x(t)\}$ converges to a fixed point of operator Op. Much work has been devoted to finding useful operators for specific problems and finding operators that provide the highest convergence rates (cf. [24]).

In this research, we consider the general iterative equation Eq. (1) without looking at details of the numerical method(s) implemented to compute Op. The iteration to update the components of the solution vector $x(t)$ is distributed among a collection of processors. As previously stated, we consider only *static scheduling*, meaning that each processor updates one *piece* of $x(t)$ which is a preassigned, fixed subset of the components of vector $x(t)$.

2.1. Synchronism vs. Asynchronism

Synchronous implementations of iterative algorithms are straightforward parallelizations of sequential implementations. This makes them attractive as their convergence properties are thus well known. It is often easy to convert a sequential implementation of a given algorithm into a synchronous parallel implementation. Assume that the distributed environment used to execute the algorithm consists of p processors. Each processor can access its local memory and communicate with any other processor via a network. Each processor starts each iteration with the entire current solution vector in its memory and updates its subset of $x(t)$ by applying part of the operator Op to the entire vector. The processors then perform an *all-to-all* communication, exchange their up-to-date subsets of the solution vector, and proceed to the next iteration. More formally, if the components of the solution vector $x(t)$ are denoted by $x_i(t)$, $i = 1, ..., m$ and if the components of $Op(x(t))$ are denoted by $Op_i(x_1(t), ..., x_m(t))$, the synchronous iteration can be written as:

$$x_i(t+1) = Op_i(x_1(t), ..., x_m(t)) \qquad \text{for} \quad 1 \leqslant i \leqslant m; \quad \text{all } t \in \mathbb{N}. \tag{2}$$

The most obvious performance bottleneck in synchronous implementations is the all-to-all communication phase. First, for slow networks, having to exchange $(p-1)^2$ messages at each iteration can be prohibitive [22]. Improved network technologies make this less a concern in many applications today, e.g., for implementations that transmit reasonably small messages on a fast local area network, but it remains a factor. Second, and more importantly for this paper, the possible lack of synchronization among the processors [25, 19, 10, 15] can lead to serious performance losses because relatively fast processors may be idle for large percentages of real-time while awaiting slower processors. This lack of synchronization tends to become particularly prominent when the iterative algorithm is run on a nondedicated cluster of workstations with multiple users. This phenomenon is a clear motivation for studying asynchronous implementations.

The study of asynchronous implementations started as early as 1969 [12] and has been the object of many extensions and generalizations. As for the synchronous case, we assume that there are p processors in the distributed environment and that the solution vector is segmented in pieces (subsets of components of $x(t)$) assigned to each processor, i.e. static scheduling. By contrast with the synchronous implementation, there is no all-to-all communication phase to synchronize the processors. Instead, a processor may perform more than one update between communications, possible using *out-of-data* for the subsets from the other processors. Each processor must at times communicate its most up-to-date values for its subset to other processors.

A formal description of the asynchronous iteration is given in [3] and is inspired by the definition of chaotic relaxations in [12]. The definition we give here is very similar: for $1 \leqslant i \leqslant m$ and $t = 1, 2, ...,$

$$x_i(t) = \begin{cases} x_i(t-1), & \text{if } \; i \notin J_t \\ Op_i(x_1(t)), ..., x_m(s_m(t))), & \text{if } \; i \in J_t, \end{cases} \tag{3}$$

where J_t is a subset of $\{1, ..., m\}$, and $s_i(t)$ is in $\mathbb{N}$. We adopt three additional conditions for asynchronous iteration proposed in [3].

Condition 2.1. *For* $1 \leqslant i \leqslant m$:

(i) $s_i(t) \leqslant t$ *for all* $t = 1, 2, ...;$

(ii) $\lim_{t \to \infty}(s_i(t)) = \infty;$

(iii) *i occurs infinitely often in the sets J_t, $t = 1, 2, ...$*

Condition (i) states that when a processor updates a component of the solution vector, it can only use previously computed computed components. Condition (ii) states that the same value for a component cannot be used indefinitely when computing updates. Condition (iii) requires that a processor not abandon a component forever. In the formal definition of asynchronous iterations that we have given so far, there is no limit on the amount by which a component used in an update can be out-of-date. If there is no upper bound on this amount, the implementation is referred to as *totally asynchronous* [8]; otherwise, the implementation is said to be *partially asynchronous*. Actual implementations are often partially asynchronous since it is often practical to fix some kind of bound on the asynchronism for implementation purposes.

2.2. Convergence

The definition of the asynchronous iteration shows clearly that the algorithm can be as "asynchronous as needed" to take advantage of the very phenomena that are performance bottlenecks for a synchronous implementation. However, convergence of the algorithm is no longer implied by the same conditions as for a sequential implementation and its convergence rate must also be reexamined.

Work in analyzing the convergence of asynchronous parallel iterative algorithms includes [8, 7, 12, 21, 20, 3, 6, 4, 26, 14, 28, 29]. Some of the earliest work focused on specific iterative algorithms or on specific implementations. A sufficient condition for convergence for linear operators is available in [12], only for partially asynchronous implementations. In [21, 20], this sufficient condition is generalized to the case of certain nonlinear operators, still in a partially asynchronous setting. A recent and general theorem in [8] gives a sufficient condition for convergence of asynchronous iterative algorithms based on a sequence of subsets of $\mathbb{R}^m$ (a "box condition"). The applications given in [8] are contractions or pseudo-contractions with respect to a weighted maximum norm (traditional "Lipschitz-like" properties detailed in [8]), and it may be dificult to fully exploit the generality of the theorem in practical situations. A lower bound on rate of convergence is obtained under additional assumptions.

A fundamental reference on which to develop a stochastic approach is Baudet's work [3]. That work contains a theorem establishing the convergence of asynchronous iterations for *contracting operators* defined by the following "Lipschitz-like" condition.

Definition 2.1. An operator Op from $\mathbb{R}^m$ to $\mathbb{R}^m$ is said to be contracting on a subset D of $\mathbb{R}^m$ if there exists a nonnegative $m \times m$ matrix A such that

$$\forall x, y \in D, \qquad |Op(x) - Op(y)| \leqslant A\,|x-y|, \qquad \text{component-wise}$$

and $\rho(A) \leqslant 1$, where $\rho(A)$ denotes the spectral radius of A (i.e., the magnitude of A's largest magnitude eigenvalue).

Furthermore, [3] provides a lower bound on the convergence rate of the algorithm defined traditionally as

$$\mathscr{R} \triangleq \liminf_{t \to \infty} [(-\log \|x(t) - \zeta\|)/t], \tag{4}$$

where $\|\cdot\|$ denotes a norm of $\mathbb{R}^m$ and ζ is the fixed point of the operator. This definition of the rate of convergence has an immediate interpretation: if the logarithm is in base 10, then $1/\mathscr{R}$ measures the asymptotic number of *iterations* required to divide the initial error by a factor of 10, where an iteration is the computation described by Eq. (3) for all i. Whithout any additional assumptions, it is shown that

$$\mathscr{R} \geqslant -[\liminf_{t \to \infty} (k_t/t)] \log \rho(A), \tag{5}$$

where $\{k_t\}$ is a sequence of integers defined in [3]. With reference to [3] for details, we note here that this sequence $\{k_t\}$ is increasing and the more asynchronous the implementation, the less rapid the increase.

The insight into performance provided by Eq. (5) is in the form of a lower bound on the theoretical rate of convergence. Experiments in [3] indicate that the bound is very conservative, and the possibility for stochastic approaches is mentioned. Our goal is to introduce tractable stochastic models and obtain convergence results relevant to the user for a practical purpose, deciding which implementations is the best for a given multiuser, distributed-memory environment in terms of long-run, average performance.

3. APPLICATION-LEVEL MODELS

We introduce two models to describe the distributed environment and the algorithm at an application-level. Some of the low-level elements of the computer system are ignored or approximated to develop and analyze the models.

3.1. Modeling the Distributed Environment

We assume that the distributed environment is a computer network of p nodes connected by a communication facility. A node is composed of a processor, memory, and a network interface. Each node has its own memory accessed only by its processor. In this distributed-memory setting, nodes can exchange data via the communication facility, thanks to their network interfaces. We do not require that all the nodes be identical, i.e., the environment may be *heterogeneous*, but do make several strong assumptions below. The communication facility is seen as an abstract device that allows reliable point-to-point communication between any two nodes of the network and we do not make any assumptions about the network topology. Our model is therefore applicable in diverse environments, from a massively parallel processor (MPP) system to an Internet-wide collection of machines.

The performance of the network in terms of transmission speed is modeled by a random variable (RV) for each point-to-point data path (for a total of $p(p-1)$ RVs). Similarly, the performance of each node in terms of local computation is modeled by a RV that describes the time that node spends to perform one update of its subset of the solution vector (for a total of p RVs). The distributions of all these RVs describe the behavior of the algorithm execution in the distributed environment. In order to introduce tractable stochastic models, these RVs are assumed independent and stationary during the run of the iterative algorithm; however, any finitely specified, discrete probability distributions can be used—empirically estimated, analytically specified, or arbitrarily chosen.

3.2. Modeling the Algorithm

The algorithm is partitioned into *phases*. Figure 1 depicts one phase for three processors i, $i+1$, $i+2$. Note that real-time intervals for the same phase of the algorithm do not generally coincide at different processors. Each phase is composed of

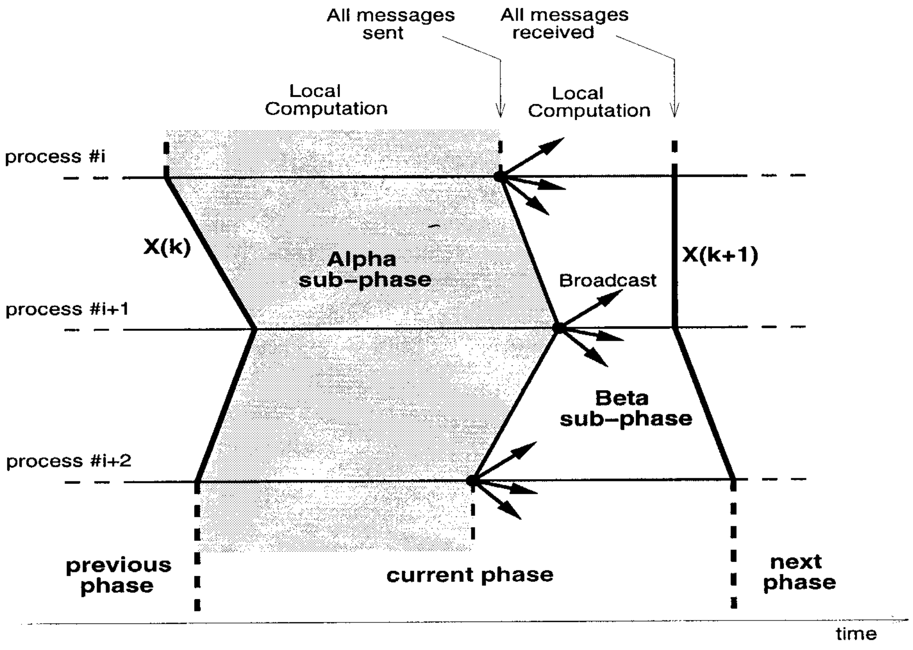

FIG. 1. Decomposition of the Algoritm into Phases.

two *subphases*, α and β. During the α *subphase*, a processor performs successive updates on its subset of the solution vector. All updates performed by a processor during its α subphase, beyond the first update, use out-of-data until the next messages are received from the other processors. At the end of the α subphase, a processor broadcasts its subset of the current solution vector to all the other processors. Just after this broadcast, its β *subphase* starts and the processor expects $p-1$ messages from the other processors. During its β subphase, a processor may perform additional updates (up to a preset number) on its subset of the solution vector. If any updates are performed during this subphase, they will also use out-of-date data. A processor finishes its β subphase when it has received all the $p-1$ messages; it then moves onto the next algorithm phase.

For each processor, the user must choose the number of updates to be performed during the α subphase and the maximum number of updates allowed during the β subphase. The larger those numbers, the more asynchronous the algorithm. A more asynchronous implementation usually implies a lower convergence rate but better use of computational resources (less processor idle-time). A method to evaluate this trade-off in terms of the expected execution time is an objective of this research.

Note that the model includes synchronism as the special case that, for each processor, there is a single update during its α subphase and no update during its β subphase. In the next section, we give several definitions and assumptions, then define a Markov chain (cf. [16]) of interests.

3.3. Underlying Markov Chain

Let us define two nonnegative, integer-valued constants and three nonnegative, real-valued RVs.

DEFINITION 3.1. (i) User-specified, integer constant $A_i > 0$ denotes the number of updates performed by processor i during its α subphase of each algorithm phase.

(ii) User-specified, integer constant $B_i \geqslant 0$ denotes the maximum number of updates that processor i is allowed to perform during its β subphase of each algorithm phase.

(iii) *RV* $\alpha^i(k) \in \mathbb{R}$ is the duration in seconds of the α subphase of the kth algorithm phase on processor i.

(iv) *RV* $n_{i \to j}(k) \in \mathbb{R}$ is the duration in seconds of the message transfer from processor i to processor j during the kth algorithm phase. By convention, processor i sends a message to itself at each phase and $n_{i \to i}(k) = 0$ for all i and k.

(v) *RV* $T^i(k) \in \mathbb{R}$ denotes the time of the beginning of the kth algorithm phase on processor i.

We assume independence of the RVs but do not require identical distributions for all processors i. We first assume that, given i, the RVs $\{\alpha^i(k)\}$ are independent and identically distributed (i.i.d.) for all k. Independence of RVs means that the computational time $\alpha^i(k)$ for processor i to perform one update does not exhibit a dependence on update-times $\{\alpha^j(k), j \neq i\}$ at other processors. (This assumption is violated, e.g., when workloads of two or more processors are correlated, due for instance to other parallel applications sharing the resources. Relaxing this independence assumption is a topic for future development.) Similarly for given i and j, the RVs $\{n_{i \to j}(k)\}$ are i.i.d. for each k; but it is not required that the distribution of $n_{i \to j}(k)$ be the same as that of $n_{h \to l}(k)$ for $i \neq h$ or $j \neq l$.

Although these are strong assumptions, independence and identical distribution of RVs are widely used in computational models (cf. [30, 1]). The experimental results presented in Section 6 illuminate some aspects of validity and limitations and Section 7 introduces new research directions to loosen the assumptions. We also assume that network times are bounded, that messages are sent at exactly the same time during a broadcast, and that message-sends are free in terms of CPU cycles on the sending processor. The last two simplifying assumptions could be removed by the use of additional RVs at the cost of more complicated modeling.

Recall that $T^i(k) \in \mathbb{R}$ denotes the time of the beginning of the kth phase of the algorithm on processor i. Taking arbitrary processor 1 as reference, we define the *wavefront* $X(k)$ as

$$\begin{aligned} X(k) &= (X_1(k), X_2(k), ..., X_p(k)) \\ &= (0, T^2(k) - T^1(k), ..., T^p(k) - T^1(k)) \in \mathbb{R}^p. \end{aligned} \tag{6}$$

$X(k)$ describes the shape of the line joining the starting times of each processor in the kth phase. $X(k)$ is represented on Fig. 1 as a thick line, and it is shown in Appendix A that for each processor i,

$$\begin{aligned} X_i(k+1) = &\max_{j \in \{1, ..., p\}} [X_j(k) + \alpha^j(k) + n_{j \to i}(k)] \\ &- \max_{j \in \{1, ..., p\}} [X_j(k) + \alpha^j(k) + n_{j \to 1}(k)]. \end{aligned} \tag{7}$$

The wavefront is the key to the stochastic model because, based on the assumption of i.i.d. RVs, Eq. (7) defines a Markov process; specifically, the wavefront $X(k)$ is a time-homogeneous Markov chain. It is shown in Appendix B that, with a few technical assumptions, the Markov chain is finite-state. Eq. (7) can be used to compute the transition probability matrix $\mathbf{P}$ of the Markov chain by using the discrete distributions of the different RVs. Examples of the transition matrix $\mathbf{P}$ can be found in [11].

Since the finite-state chain is recurrent, it has a stationary distribution which describes the behavior of the chain in the long-run (cf. [16, 2, 9]). We refer to the probabilities in this distribution as the π-values; π_s (>0) for state s is the long-run, relative-frequency of occurrence of state s in realizations of the chain.

4. PERFORMANCE CHARACTERIZATION

The wavefront Markov chain is exploited to obtain performance results for the parallel iterative algorithm. The goal is to obtain information on the average execution time. The execution time can be computed as the ratio of the number of iterations to perform over the number of iterations performed per time unit. The number of iterations required can be approximated using the asymptotic convergence rate of the algorithm. This is the topic of the next section.

4.1. Asymptotic Rate of Convergence

The challenge here is to refine the lower bound on the asymptotic rate of convergence in [3] by modified estimates that take into account randomness at the application-level. One can compute three estimates (see Apendix D) respectively called *worst*-, *average*-, and *best-cases* and denoted respectively as $\underline{\mathscr{R}}$, $\hat{\mathscr{R}}$, and $\overline{\mathscr{R}}$. The usefulness of the average- and best-case estimates is yet to be demonstrated; however, the worst-case estimate is a straightforward improvement over the lower bound in [3] as (i) it is higher than Baudet's and (ii) it is still a lower bound on the asymptotic convergence rate.

Let $\mathscr{R}^*$ denote one of the three estimates for this rate. Then user who wants the initial error on the solution vector to be divided by a factor of 10^{ω} may approximate the number of iterations needed as $\omega/\mathscr{R}^*$ assuming that ω is reasonably large. The value of ω chosen by the user is the *convergence criterion*: the larger ω, the smaller the final error. Of course, the choice of $\mathscr{R}^*$ is crucial, and we expect that the three estimates will provide information about this choice.

4.2. Execution Speed

We can use $\mathscr{R}^*$ to estimate the speed of the execution in terms of number of iterations performed per time unit. Let $\Phi(k)$ denote the duration in seconds of the kth algorithm phase on processor 1. Let $N(k)$ denote the number of iterations performed during that phase. Both $\Phi(k)$ and $N(k)$ are RVs and their long-run probability distributions can be approximated by making use of the wavefront π-values. Indeed, the probability distributions of RVs of interest in the model are all conditioned on the wavefront state. The π-values are used to replace those conditional probabilities by unconditional probabilities, and a simple convolution is used to obtain the long-run estimate.

The speed of the execution is entirely described by sums of the random vector $\binom{N(k)}{\Phi(k)}$ for successive values of k. This vector can be used in different ways, as described in the next section.

4.3. Performance Characterization Levels

Level 1. This level provides an estimate of the execution time mean value. Using the Strong law of large numbers [16], it is possible to compute the asymptotic algorithm speed as the ratio of the limiting expected values of $N(k)$ and $\Phi(k)$. Denoting this ratio as $\mathscr{S}$, one obtains Θ_1, an asymptotic estimate for the execution time expected value:

$$\Theta_1 = \frac{\omega}{\mathscr{S}\mathscr{R}^*}.$$

Level 2. This level provides an estimate of the execution time standard deviation. The derivation of the estimate is detailed in Appendix C.

5. SIMULATION

Simulation results are summarized here for a gradient algorithm for a multipolynomial cost function with 30 variables (see also [11, Section 5.1]). The distributed environment consist of three processors with the different update-time distributions depicted in Fig. 2, obtained by sampling the distributions of actual workstations. The three processors are interconnected by a network that delivers constant, nonrandom performance, i.e., given i and j, $n_{i \to j}(k)$ is constant for all k and, by convention in Definition 3.1(iv), is 0 for $i=j$. Three implementations are simulated: *synchronous* with $A_i = 1$ and $B_i = 0$ for all processors i; *asynchronous* (Async. 1) with $A_i = B_i = 1$ for all processors i; and *more asynchronous* (Async. 2) with $A_i = 1$ and $B_i = 2$ for all processors i. The asumptions set forth above are non violated in these simulations.

The simulations provide information about accuracy and sensitivity of the two levels for the environment described. First, we observe that the new estimates of asymptotic convergence rate are improvements on estimates in [3]. Table 1 shows the relative errors between the different estimates and the observed convergence rate for different implementations in the simulated environment. The gaps between the four estimates increase with asynchronicity. It is to be noted that no estimate exactly predicts the observed convergence rate. A primary reason is that the estimates depend only on the spectral radius of the matrix associated to the contracting operator, but not on the actual shape of that operator. Therefore, the same estimates will be computed for different operators that happen to have the same contracting matrix.

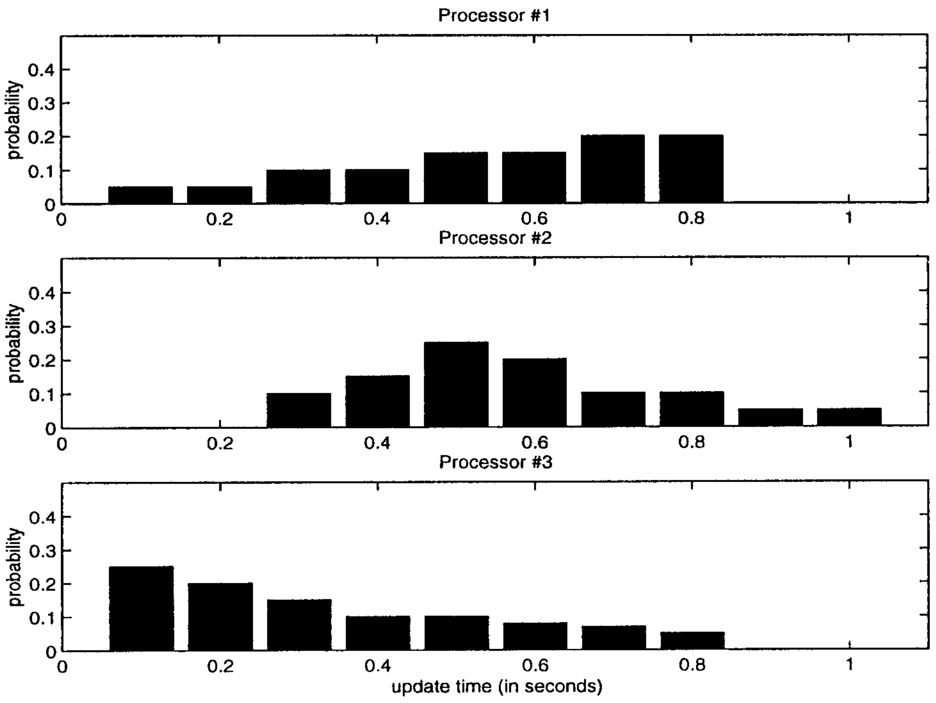

FIG. 2. Update time distributions for the three processors.

The simulations also show that level 2 characterization is rather sensitive. Indeed, as it is based on a binomial Gaussian approximation, its accuracy as an aproximation generally increases with a larger number of samples. It is shown in [11] that the error between level 2 characterization and the observed standard deviation decreases for increasing valules of ω (recall that larger ω implies more iterations performed).

Figure 3 shows the simulation versus the characterization for an asynchronous implementation in the simulated heterogeneous distributed environment. On each graph, the empirical distribution of the execution time is shown as a bar diagram labeled "simulation." The empirical mean is shown as a vertical solid line and the empirical standard deviation is represented as a horizontal line segment inside bold vertical lines on each side of the empirical mean. Level 1 characterization is shown as a dashed vertical line. Level 2 characterization is shown as two horizontal dashed line segments in bold vertical lines on each side of level 1. We show four characterizations—Baudet [3], $\underline{\mathscr{R}}$, $\hat{\mathscr{R}}$, $\overline{\mathscr{R}}$—each corresponding to a different

TABLE 1

Simulation: Convergence Rate Errors

Impl.	$\underline{\mathscr{R}}$	$\hat{\mathscr{R}}$	$\overline{\mathscr{R}}$	$\mathscr{R}_{\text{Baudet}}$
Sync.	7.69%	7.69%	7.69%	7.69%
Async. 1	31.96%	17.53%	36.08%	54.64%
Async. 2	9.55%	17.86%	57.14%	69.05%

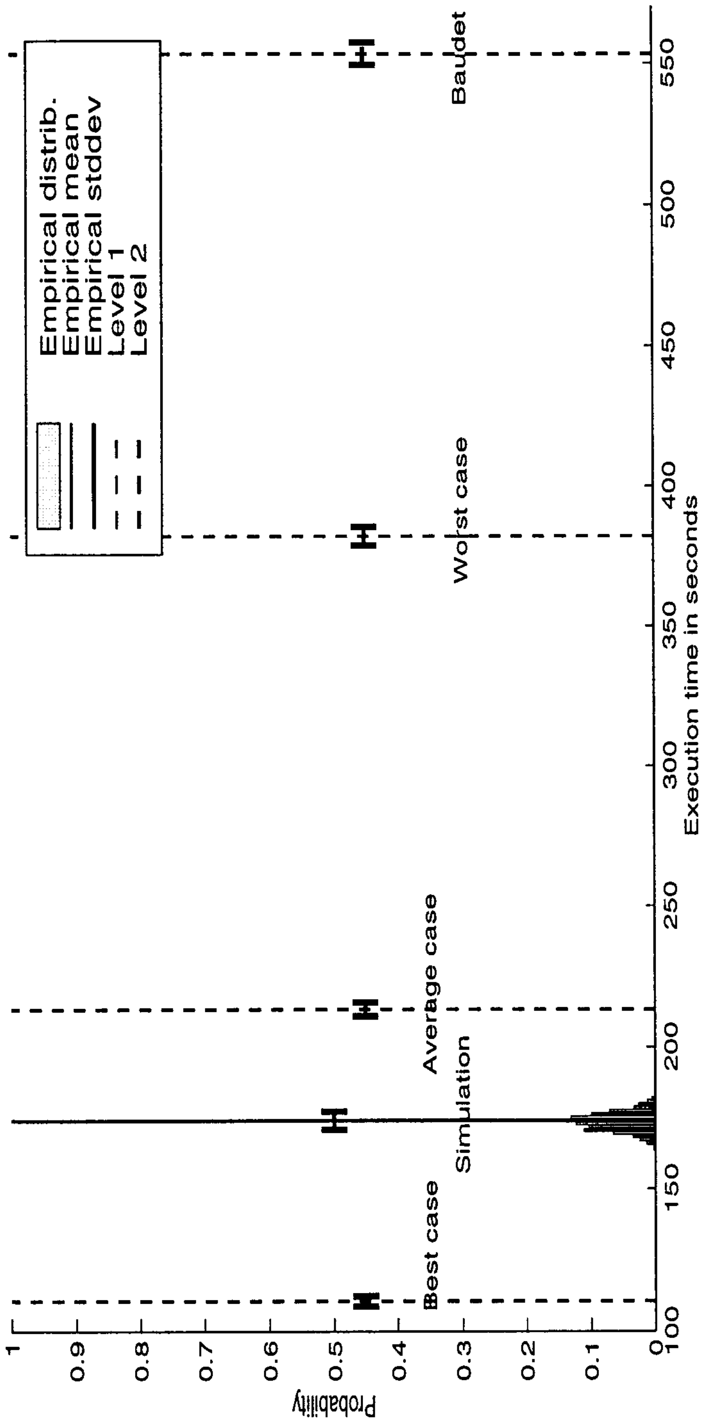

FIG. 3. Simulation vs characterizations for an asynchronous implementation.

asymptotic convergence rate estimate. Note that the average-case estimate is a dramatic improvement over the estimate in [3] for the execution time distribution (22% vs 214% error).

6. EXPERIMENT

This section presents experimental results [11] with a gradient algorithm for a real multipolynomial cost function with 30 variables. The algorithm runs in parallel on three Sun Sparc Ultra 1 workstations interconnected by a standard 10 Mbps Ethernet. Those workstations are being used by students for course-work, as well as for personal research. Measurements were obtained throughout one week (Nov. 17–24, 1997).

6.1. Preliminary Remarks

Figure 4 shows the execution times observed throughout the whole week for the synchronous implementation and for the first asynchronous implementation (*mildly* asynchronous: at most one update performed in β subphase for each processor). This corresponds to 862 observations for each implementation. The measurements for the second asynchronous implementation (*fairly* asynchronous: at most two update performed in β subphase for each processor) are not shown on Fig. 4 because they would be difficult to distinguish from the ones of the first asynchronous implementation on that time scale. The three different implementations were run in a round-robin fashion, each run using the three workstations and taking about 2 min.

The first observation is that the asynchronous implementations are generally more efficient than the synchronous one. The first asynchronous implementation is up to 150s faster than the synchronous implementation, and 30s faster on average. On average the second implementaion is faster than the first one by about 1.9s. But in only 15% of the observations is the absolute difference between the two implementations more than 10s.

It seems that, in this environment, a good choice is an asynchronous implementations as opposed to a synchronous one. However, a *mild* asynchronicity is sufficient to obtain improvement over a synchronous implementation. This can be explained both by the nature of the distributed environment and by the nature of the iterative algorithm. Several other references also include examples for which asynchronous implementations outperform synchronous ones [3, 5, 22].

A fundamental observation on Fig. 4 is that the execution time is *bursty*. In fact, the distributed environment, and therefore the algorithm, behaves very differently at different times in the time period for the system is in use for a variety purposes during the experimental runs. In order to illustrate these different behaviors, Fig. 5a, b, and c show three close-ups of the execution times for each implementation during three short subperiods about two hours long. The distributed environment apears to exhibit distinct modes during the week, which violates our i.i.d. assumption.

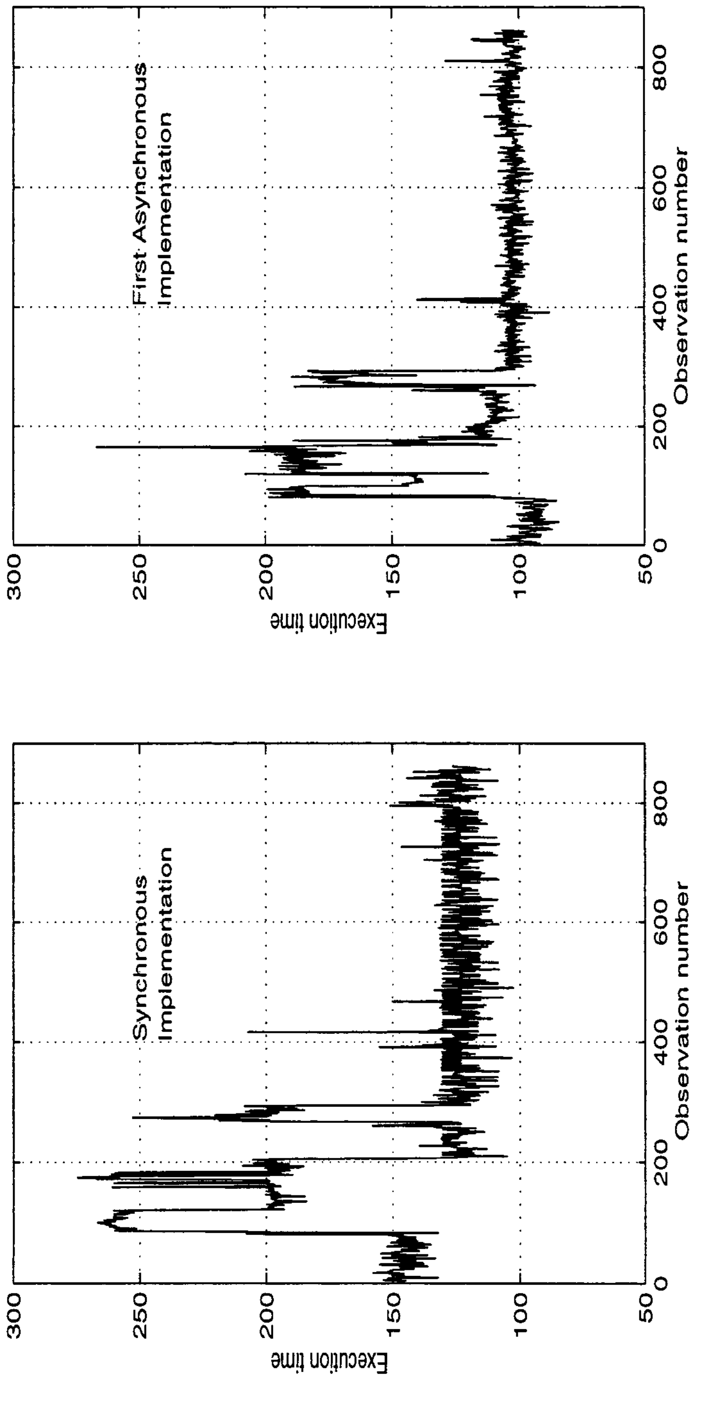

FIG. 4. Execution time measurements over a week.

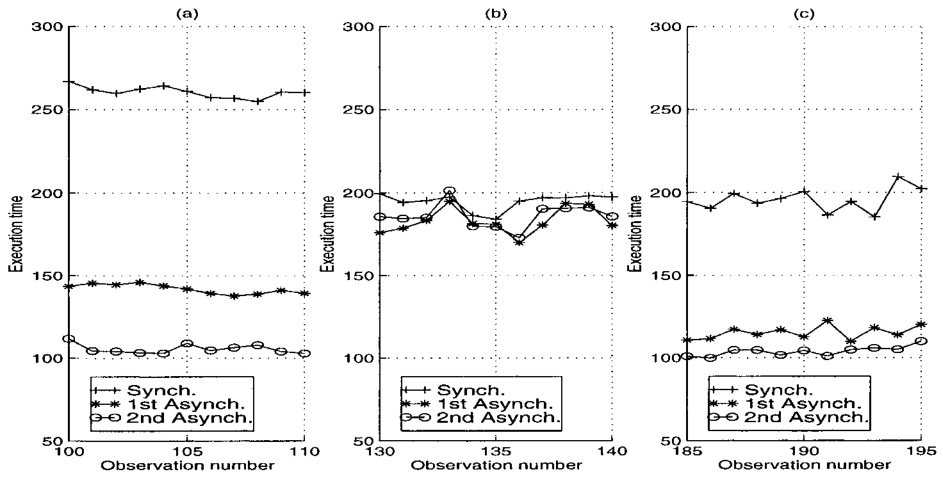

FIG. 5. Different experimental behaviors throughout one week.

Figure 7 shows the execution times for the parallel iterative algorithm throughout a 24-h time period at the end of the week. The distributed environment exhibited a fairly stable behavior, leading to relatively smoother observations. We expect the stochastic models to yield better results for the 24-h time period than for the entire week as burstiness indicates violation of the stationarity assumption. We do not

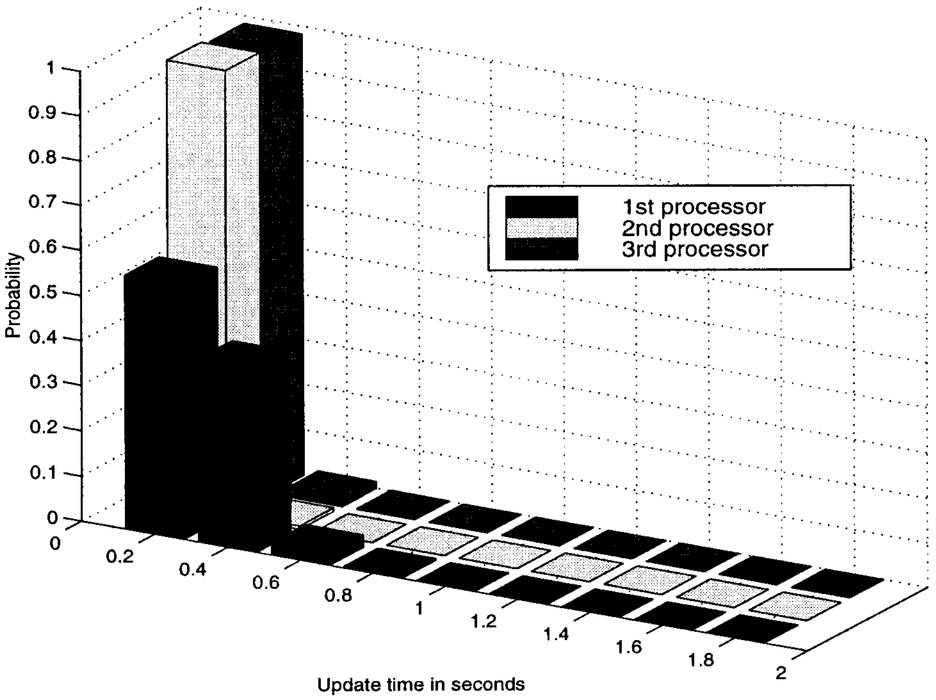

FIG. 6. Experimental update time distributions for the three processors.

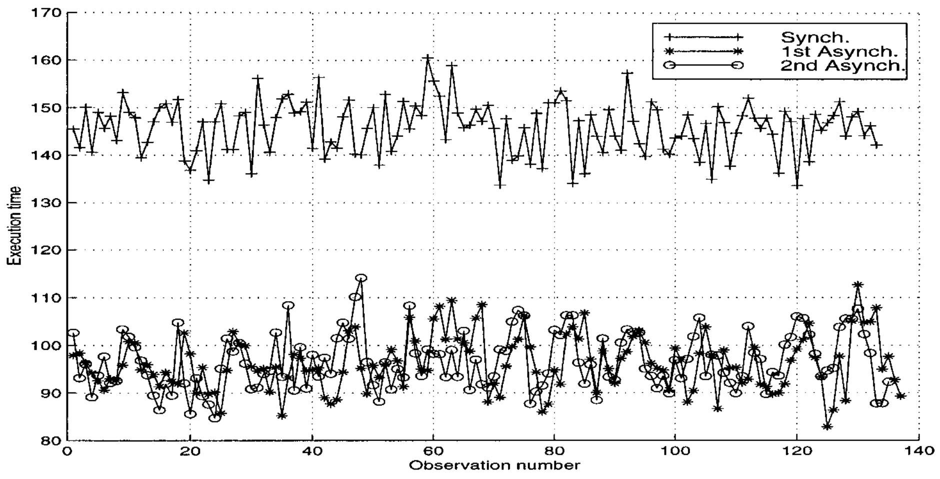

FIG. 7. Measurements during 24 h for the three implementations.

claim that any 24-h time period would lead to better result (as the execution time may be bursty on many time scales). We merely chose to highlight a subset of the time line that exhibited close-to-stationary behavior to evaluate how the model would perform in a more stable environment. In the following sections, we present and comment on some of the results for both time periods.

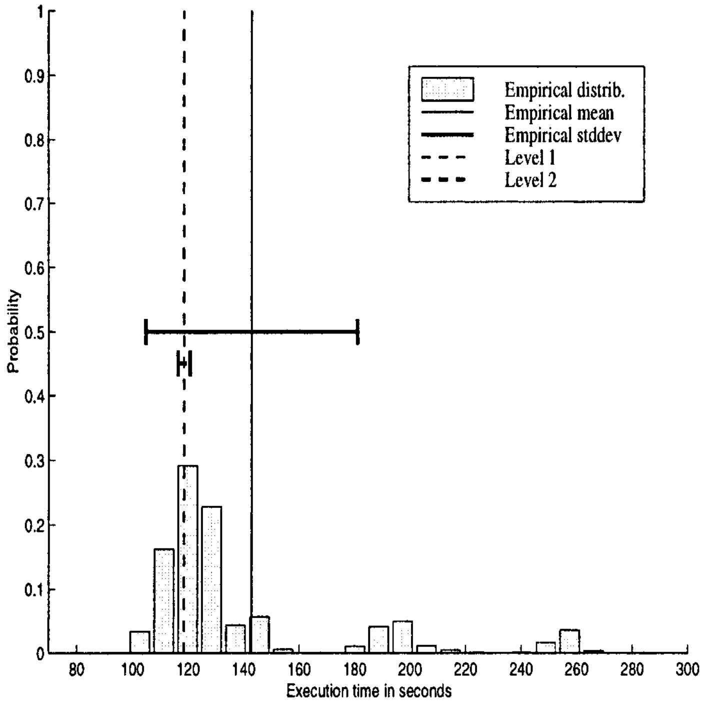

FIG. 8. Experiment vs characterization for the synchronous implementation.

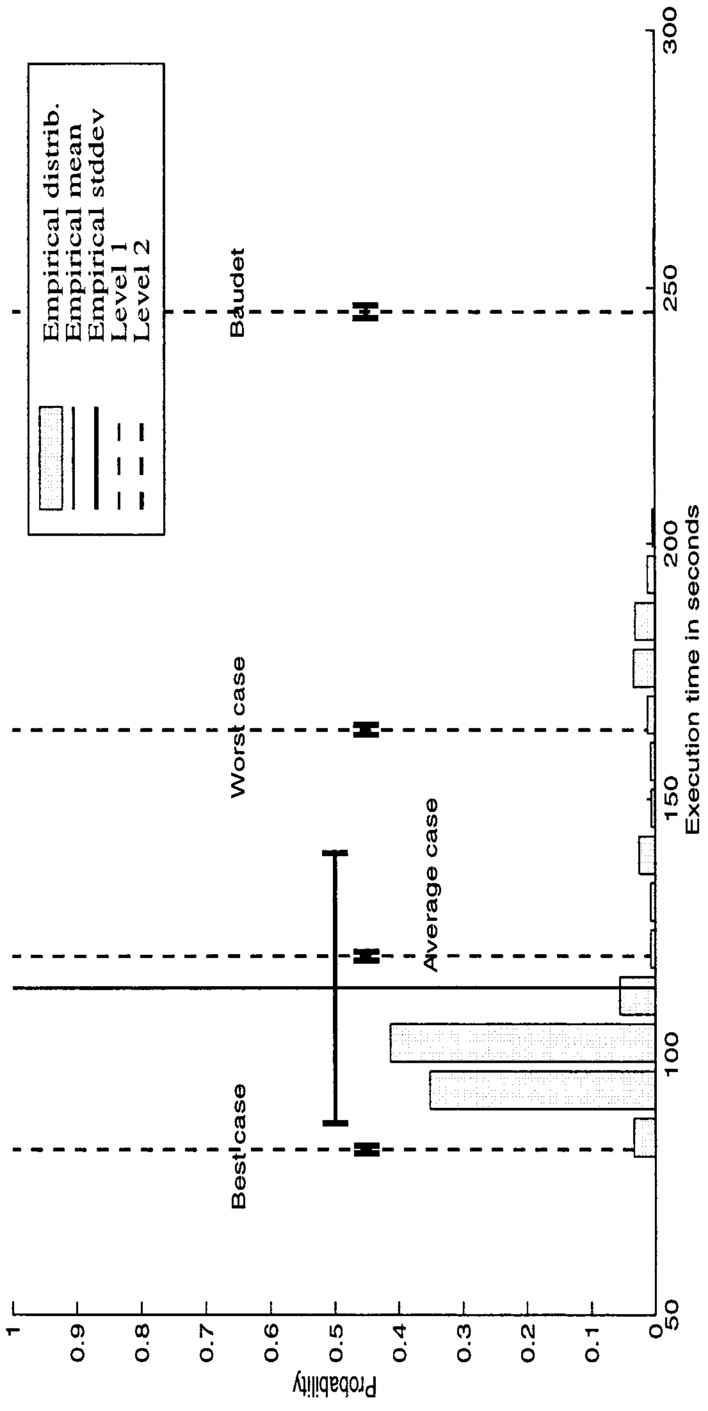

FIG. 9. Experiment vs characterization for the first asynchronous implementation.

6.2. Applying the Model

6.2.1. The One Week Time Period

The workload distributions of the three processors were sampled throughout the week (roughly every 20 s) and are shown in Fig. 6. Figure 8 shows the results for the synchronous implementation. The empirical distribution is clearly multi-modal as already seen in Section 6.1. The level 1 characterization makes an error of about 17% in predicting the mean of the execution time. The level 2 characterization underestimates the observed standard deviation by a factor of 50. In fact, level 2 is very sensitive to the violation of the stationary assumption as a Gaussian approximation is involved.

Figure 9 shows the results for the mildly asynchronous implementation. Four characterizations are shown, one of each estimate of the asymptotic convergence rate. If one uses the average-case estimate (see Section 4.1), then the error on the mean prediction is only 4% (as opposed to 116% with the estimate in [3]). The observations made on Fig. 8 about level 2 are still valid. The results for the second asynchronous implementation are not shown here as they are fairly similar to the results for the first asynchronous implementation. The next section reduces the time period to 24h and should lead to improvements, especially for the level 2 characterization as the stationary assumption should be less violated.

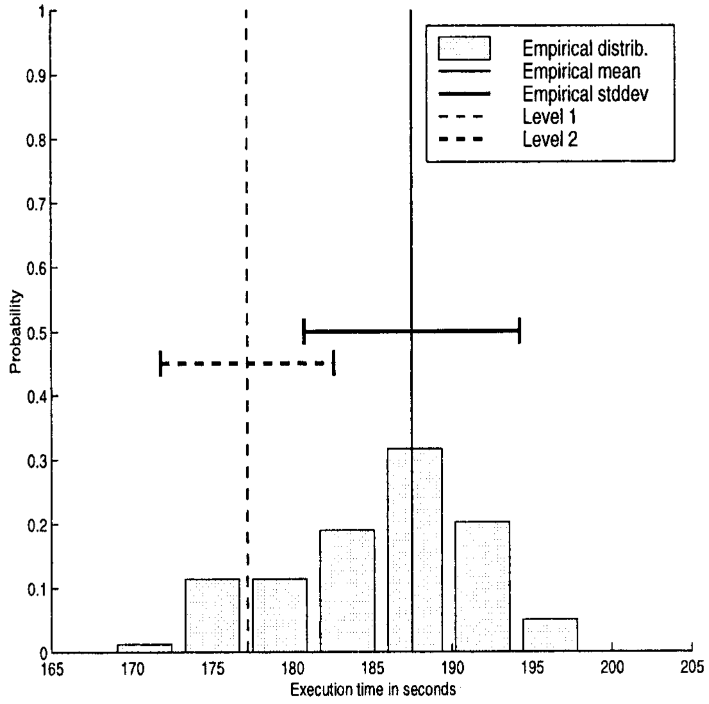

FIG. 10. Experiment vs characterization for the synchronous implementation.

6.2.2. *The 24 Hour Time Period*

Figure 10, for a synchronous implementation, demonstrates that the level 2 characterization becomes much more accurate for this shorter, more nearly stationary time period. Its error is here about 27%, whereas it was a factor of 50 for the whole week. Similar improvements were observed for all implementations. Furthermore, the level 1 characterization is more informative than for the one week time period. Even though the level 1 characterizations do not exactly agree with the observed mean, they are sufficient for purposes of comparisons. In this experiment, the level 1 characterizations computed in [11] imply that an asynchronous implementation will out perform a synchronous one on average by 40 s (which agrees with the experimental results on Fig. 7).

7. CONCLUSION AND FUTURE WORK

Parallelizing iterative algorithms for the solution of complex problems is a crucial issue. The amount of computation required to solve such problems may be prohibitive for a sequential implementation, especially in nonlinear cases. We use synchronous and asynchronous paradigms for parallel iteration and develop stochastic models that take characteristics of multiuser, distributed-memory environments into account. The models are used to obtain performance characterizations that are directly meaningful to the end-user. In practice, a user must provide the probability distributions for network and CPU loads (as finitely specified, discrete distributions obtained through sampling or from tools such as the NWS [31], for instance) and employ the model for comparisons among degrees of asynchronism in implementations in terms of the estimates on algorithmic convergence rates, in particular, for the worst-case.

An additional level of performance characterization based on large deviation theory is developed in [11] and will be reported in the future. Further research may also incorporate Markov-modulated random processes to model bursty behaviors in more detail [23, 13, 17] and may investigate more complex models by introducing dependences among RVs.

APPENDIX A: WAVEFRONT EQUATION

Let $\beta^i(k) \in \mathbb{R}$ be the duration in seconds of the β subphase of the kth algorithm phase on processor i. $\beta^i(k)$ is a RV and it can be computed as follows. The β subphase of the kth algorithm phase on processor i clearly starts at time $\beta^i_{\text{start}}(k) = T^i(k) + \alpha^i(k)$. It ends when the last expected message has been received by processor i. The message expected from processor j is received by processor i at time $\beta^j_{\text{start}}(k) + n_{j\to i}(k)$. Therefore,

$$\begin{aligned}\beta^i(k) &= \max_{j \in \{1, \ldots, p\}} [\beta^j_{\text{start}}(k) + n_{j\to i}(k)] - \beta^i_{\text{start}}(k)\\ &= \max_{j \in \{1, \ldots, p\}} [\beta^j_{\text{start}}(k) + n_{i\to j}(k) - \beta^i_{\text{start}}(k)].\end{aligned}$$

Since $n_{i\to i}(k)=0$, one obtains

$$\beta^i(k)=\max[0, \max_{j\in\{1,\ldots,p\}-\{i\}} (T^j(k)+\alpha^j(k)-T^i(k)-\alpha^i(k)+n_{j\to i}(k))].$$

By Definition 3.1(v) and Eq. (6),

$$X^i(k+1)=X^i(k)+\alpha^i(k)+\beta^i(k)-\alpha^1(k)-\beta^1(k).$$

Replacing $\beta^i(k)$ and $\beta^j(k)$ by their expression and using the fact that

$$\forall x, y\in\mathbb{R}, \qquad \max(0, x-y)+y=\max(x, y),$$

one obtains Eq. (7). ▮

APPENDIX B: WAVEFRONT STATE-SPACE

LEMMA B.1 $\exists M, \forall k\geqslant 1, \|X(k)\|_\infty\leqslant M$.

Proof. Let us consider processor 1 and processor $i\neq 1$ during the kth algorithm phase. The times at wich these two processors receive a message from a processor h are apart by $|n_{h\to 1}(k)-n_{h\to i}(k)|$ seconds, since we assume that processor h sends all messages at the exact same time. Therefore, the times at which processors 1 and i receive the last messages they were expecting are apart by at most $\max_{h\in\{1,\ldots,p\}}(|n_{h\to 1}(k)-n_{h\to i}(k)|)$. The communication times are assumed to be bounded above and below as

$$\forall s, d \quad \exists\, \underline{n_{s\to d}}, \overline{n_{s\to d}}, \quad \forall_k, \quad \underline{n_{s\to d}}\leqslant n_{s\to d}(k)\leqslant\overline{n_{s\to d}}.$$

One can then write

$$\begin{aligned}\forall h \quad |n_{h\to 1}(k)-n_{h\to i}(k)| &\leqslant \max(\overline{n_{h\to 1}}-\underline{n_{h\to i}}, \overline{n_{h\to i}}-\underline{n_{h\to 1}})\\ &\leqslant \max(\overline{n_{h\to 1}}, \overline{n_{h\to i}})\\ &\leqslant \max_{j\in\{1,\ldots,p\}}(\overline{n_{h\to j}}).\end{aligned}$$

This implies that the times at which processors 1 and i receive the last message that they were expecting, they are apart by at most $\max_{h,j\in\{1,\ldots,p\}}(\overline{n_{h\to j}})$. But those times are also apart by $X_i(k)$, according to Definition 3.1(v) and Eq. (6). Since $\|X(k+1)\|_\infty\equiv\max_{i\in\{1,\ldots,p\}}|X_i(k+1)|$, the proof is complete. ▮

The preceding establishes that the wavefront vector is in a closed ball of $\mathbb{R}^p$. If one assumes that $\alpha^i(k)$, $n_{i\to j}(k)$, and the components of $X(0)$ are rational (in $\mathbb{Q}$), then for each k, $X(k)$ is in a finite subset of $\mathbb{R}^p$ that does not depend on k. Those assumptions are really purely technical; the data being manipulated is in $\mathbb{Q}$ since it is processed by computers with finite arithmetic. The size of the state-space of the wavefront Markov chain depends on the finitely specified, discrete distributions of

$\alpha^i(k)$ and $n_{i\to j}(k)$ [11]. The π-values for the chain are the unique values in the solution of the linear system $\Pi = \Pi\mathbf{P}$, where $\mathbf{P}$ is the transition probability matrix, $\sum_S \pi_S = 1$, and $\pi_s > 0$ for every state s [16, 2, 9].

APPENDIX C: LEVEL 2 CHARACTERIZATION

One can make a binomial Gaussian approximation of the distribution of the random vector $\binom{N\ (k)}{\Phi\ (k)}$ (with covariance matrix C). The covariance matrix of the sum of those vectors for each algorithm phase until convergence, C', can then be estimated as

$$C' = \frac{\omega}{\mathbb{E}\{N(k)\}\ \mathscr{R}^*}\ C,$$

where $\mathbb{E}\{N(k)\}$ denotes the expected value of $N(k)$ (this expected value does not depend on k). Using C', it is then easy to obtain an estimate of the standard deviation of the execution time. Indeed, if

$$C' = \begin{bmatrix} \sigma_X^2 & \sigma_{XY} \\ \sigma_{XY} & \sigma_Y^2 \end{bmatrix},$$

then the standard deviation estimate is computed as [16]

$$\sigma = \sigma_Y \sqrt{1 - \left(\frac{\sigma_{XY}}{\sigma_X \sigma_Y}\right)^2}.$$

APPENDIX D: ASYMPTOTIC CONVERGENCE RATE ESTIMATES

To compute estimates of the algorithm asymptotic rate of convergence, we must extend Eq. (5). In [3], the sequence $\{t_k\}$ is defined as

$$\begin{aligned} t_0 &= 0 \\ t_k &= t_k + a_k + b_k, \end{aligned}$$

where $\{a_k\}$ and $\{b_k\}$ are defined as

(i) starting with the $(t_k + a_k)$th iteration, no solution vector update makes use of values of components corresponding to iterates with indices smaller than t_k.

(ii) all solution vector components are updated at least once between the $(t_k + a_k)$th and the $(t_k + a_k + b_k)$th iterations.

The sequence $\{k_t\}$ of Eq. (5) is then defined as

$$k_t \triangleq \sup\{k \in \mathbb{N} \mid a_0 + b_0 + \cdots + a_{k-1} + b_{k-1} \leqslant t\}$$

for nonnegative integers $\mathbb{N}$.

In our setting,

$$\forall_k = 0, 1, ..., \quad \begin{cases} a_k = N(k), \\ b_k = 0, \end{cases}$$

where $N(k)$ denotes the number of iterations performed during the kth algorithm phase. One can then compute

$$k_t = \sup \left\{ k \,\middle|\, \sum_{l=0}^{k-1} \max_{i \in \{1, ..., p\}} (A_i + N(l)) \leqslant t \right\},$$

The long-term probability distribution of the RV $N(k)$ can be approximated using the π-values for the wavefront Markov chain, leading to the probablity distribution of k_t or each t. It is then possible to compute three estimates or the asymptotic rate of convergence by replacing k_t in Eq. (5) by its minimal observable value, its expectation, or its maximal observable value. A formal proof of the convergence of the limit in Eq. (5) for each estimate is left for future work. A finite limit has been obtained in all simulations and experiments.

ACKNOWLEDGMENTS

The authors thank the reviewers for their constructive criticism and suggestions.

REFERENCES

1. V. Adve and M. Vernon, The influence of random delays on parallel execution times, *in* "Proceedings of the 1993 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, May 1993," pp. 61–73.
2. R. Ash, "Information Theory," Dover, Mineola, NY, 1990.
3. G. Baudet, Asynchronous iterative methods for multiprocessors, *J. Assoc. Comput. Mach.* **25** (April 1978), 136–244.
4. D. El Baz, *M*-functions and parallel asynchronous algorithms, *SIAM J. Numer. Anal.* **27** (1990), 136–140.
5. D. El Baz, P. Spiteri, J. C. Miellou, and D. Gazen, Asynchronous Iterative Algorithms with Flexible Communication for Nonlinear Network Flow Problems, *J. Parallel Distrib. Comput.* **38** (1996), 1–15.
6. D. P. Bertsekas, Distributed asynchronous computation of fixed point, *Math. Programm.* **27** (1983), 107–120.
7. D. P. Bertsekas and J. N. Tsitsiklis, Convergence rate and termination of asynchronous iterative algorithms, *in* "Proceedings of the Int. Conf. on Supercomputing, 1989," pp. 461–470.
8. D. P. Bertsekas and J. N. Tsitsiklis, "Parallel and Distributed Computation," Prentice–Hall, Englewood Cliffs, NJ, 1989.
9. A. Bharucha-Reid, "Elements of the Theory of Markov Processes and their Applications," Dover Publications, Mineola, NY, 1997.
10. L. Brochard, J.-P. Prost, and F. Fauire, Synchronization and load unbalance effects of parallel iterative algorithms, *in* "Proceedings of the International Conference on Parallel Processing (ICCP)," Vol. III, 1989, pp. 153–160.

11. H. Casanova, “Stochastic Models for Performance Analyses of Iterative Algorithms in Distributed Environments,” Ph.D. thesis, Dept. of Computer Science, University of Tennesse, Knoxville, TN, 1998. [Available as TR ut-cs-98-386]

12. D. Chazan and W. Miranker, Chaotic relaxation, *Linear Algebra and Applications* **2** (1969), 199–222.

13. N. Duffield, J. Lewis, N. O’Connel, R. Russell, and F. Toomey, Entropy of ATM traffic streams: A tool for estimating QS parameters, *IEEE J. Selected Areas Commun.* **13**(6) (August 1995), 981–989.

14. A. Frommer, On asynchronous iterations in partially ordered spaces, *Numer. Funct. Anal. Optim.* **12**(3 & 4) (1991), 315–325.

15. A. Greenbaum, Synchronization costs on multiprocessors, *Parallel Comput.* **10** (1989), 3–14.

16. S. Karlin and H. Taylor, “A First Course in Stochastic Processes,” 2nd ed., Academic Press, New York, 1975.

17. K. Kawahara, Y. Oie, M. Murata, and H. Miyahara, Performance analysis of reactive congestion control for ATM networks, *IEEE J. Selected Areas. Commun.* **13**, 4 (May 1995), 651–661.

18. C. P. Kruskal and A. Weiss, Allocating independent subtasks on parallel processors, *in* “Proceedings of the Internatinal Conference on Parallel Processing (ICCP), 1984,” pp. 236–240.

19. B. Lubachevsky and D. Mitra, Chaotic asynchronous algorithm for computing the fixed point of a nonnegative matrix of unit spectral radius, *J. of the ACM* **33**(1) (January 1986), 130–150.

20. J. C. Miellou, Algorithmes de relaxation à retards, *Rev. Automat. Inform. Recherche Opér.* **9** (1970), 55–82.

21. J. C. Miellou, Itérations chaotiques à retards, *C.R. Acad. Sci. Paris* **278** (1974), 957–960.

22. S. P. Oleson, J. Gregor, M. G. Thomason, and G. T. Smith, EM-ML PET reconstruction on multiple processors with reduced comunications, *Internat. J. Imaging Systems Technol.* **7**(3) (1996), 215–223.

23. R. Onvural, “Asynchronous Transter Mode Networks, Performance Issues,” Artech House, Norwood, MA, second edition, 1995.

24. J. M. Ortega and W. C. Rheinboldt, “Iterative Solution of Nonlinear Equations in Several Variables,” Academic Press, New York, 1970.

25. J. T. Robinson, Some analysis techniques for asynchronous multiprocessor algorithms, *IEEE Trans. Software Eng.* **SE-5**(1) (January 1979), 24–31.

26. E. Tarazi, Some convergence results for asynchronous algorithms, *Numer. Math.* **39** (1982), 325–340.

27. J. N. Tsitsiklis and G. D. Stamoulis, “On the Average Communication Complexity of Asynchronous Distributed Algorithms,” Technical Report LIDS-P-1986, MIT Laboratory for Information and Decision Systems, 1990.

28. A. Üresin and M. Dubois, Sufficient conditions for the convegence of asynchronous distributed algorithms, *Parallel Comput.* **10** (November 1989), 83–92.

29. A. Üresin and M. Dubois, Parallel asynchronous algorithms for discrete data, *J. of the ACM* **37**(3) (1990), 588–606.

30. A. Üresin and M. Dubois, Effects of asynchronism on the convergence rate of iterative algorithms, *J. of Parallel and Distributed Comput.* **34** (1996), 66–81.

31. R. Wolski, Dynamically forecasting network performance using the network weather service, Technical Report TR-CS96-494, U.C. San Diego, October 1996.

HENRI CASANOVA received the B.S. from L’Ecole Nationale Supérieure d’Electrotechnique, d’informatique et d’Hydraulique de Toulouse (ENSEEIHT) in 1993, the M.S. from l’Université Paul Sabatier, Toulouse, in 1994, and the Ph.D. from the University of Tennessee, Knoxville, in 1998. He is currently a project scientist at the University of California, San Diego. His research interests include metacomputing, parallel/distributed computing, performance modeling, and stochastic models.

MICHAEL G. THOMASON received the B.S. from Clemson University in 1965, the M.S. from Johns Hopkins University in 1970, and the Ph.D. from Duke University in 1973. He worked for Westinghouse (Baltimore) and currently is Professor of Computer Science at the University of Tenessee, Knoxville. His research interests include pattern/image analysis, parallel/distributed computation, and stochastic models in computer science. He is a member of the Association for Computing Machinery (ACM) and a senior member of the Institute of Electrical and Electronica Engineers (IEEE).

JACK DONGARRA received the B.S. in mathematics from Chicago State University in 1972, the M.S. in computer science from Illinois Institute of Technology in 1973, and the Ph.D. in applied mathematics from the University of New Mexico in 1980. Dongarra is a distinguished scientist, specializing in numerical algorithms in linear algebra at the University of Tenessee's Computer Science Department and Oak Ridge National Laboratory's Mathematical Sciences Section. Professional activities include membership in the Society for Industrial and Applied Mathematics and in the Association for Computing Machinery (ACM).