Stochastic Performance Prediction for Iterative Algorithms in Distributed Environments.

Henri Casanova * Michael G. Thomason* Jack J. Dongarra* †

May 25, 1998

*Department of Computer Science, University of Tennessee, TN 37996, USA
†Mathematical Science Section, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA

Running Head:

Performance Prediction for Iterative Algorithms

Corresponding Author:

Prof. Michael G. Thomason

107 Ayres Hall

Dept. of Computer Science, University of Tennessee

Knoxville, TN 37996-1301

Abstract:

The parallelization of certain iterative algorithms is a crucial issue for the efficient solution of large numerical problems. Several theoretical results concerning the sufficient conditions for, and speed of, convergence for parallel iterative algorithms are available. However, there is a gap between those results and what is relevant to the user at the application level. The performance characterization presented in this paper addresses different stochastic characteristics of the algorithmic execution time such as mean values and standard deviations. It is shown how this approach can fill the aforementioned gap thanks to stochastic models based on random variables which characterize the distributed environment used to run the algorithm. We present simulation results as well as experimental results over different time periods. The results of this research provide information about the impact of of distributed environment and implementation style on execution time characteristics.

Keywords:

Iterative algorithms, parallel computing, distributed environments, performance prediction, stochastic modeling, Markov chains.

1 Introduction

Iterative algorithms are widely used in different areas of science and engineering, e.g., medical imaging [21], or network flow problems that occur in electrical networks, communication networks and financial models [4]. A broad class of iterative algorithms aims at finding a fixed point of a given operator. Many well-know numerical methods use such an algorithm with linear or non-linear operators. For problems with large dimension and/or extensive numerical computation for each component at each iteration (e.g. Gradient approximations or Hessian computations for non-linear operators), it is natural to consider a parallel implementation of the iterative algorithm. In this research, we are primarily interested in distributed-memory environments such as clusters of processors on a network.

The parallel implementations that have been investigated in the past usually fall into two categories: *synchronous* and *asynchronous*.

1.1 Motivation

Analyzing the behavior and thus the performance of a parallelized iterative algorithm running in a distributed environment is not an easy task. The environment imposes its own constraints on the execution. In a case when the users do not have access to a dedicated system, the workloads of the processors and the network contention depend on the load of the system generated by other users in addition to the iterative algorithm execution itself. Therefore, the processor workloads are usually disparate and vary over time according to different patterns. Furthermore, the amount of computation performed by a processor to update a component of the solution vector may not be known a priori: in non-linear cases, it may depend on the *shape* of the operator around the current solution vector. In some cases, convergence results indirectly support a quantitative assessment of the parallel algorithm convergence rate. However, almost all these results are purely theoretical and do not take into account the nature of the distributed environment itself. The only commonly available results are lower bounds on the algorithm theoretical rate of convergence. It is difficult for the user to relate this measure of convergence to the actual performance he can expect for his implementation on his distributed environment in terms of execution time for instance.

Due to non-determinism (randomness) both in communication and in computation, stochastic methods appear to be a natural way to move towards more complex and relevant models . These new 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.

1.2 Background

Few attempts at using stochastic models to analyze the performance and behavior of parallel iterative algorithms appear in the literature. Indeed, stochastic approaches seem to lead to complicated models which present difficulties in obtaining useful performance characterizations. Most difficulties come from trying to properly model the distributed environment in connection with the algorithm. Hence, most stochastic approaches in the past used very stringent assumptions and are therefore limited in their domain of application. If some reasonable assumptions could lead to tractable models, then those models should provide insight into the performance analysis of parallel iterative algorithms in given distributed environments.

For instance, [7] (Section 6.3.5) provides comparisons between the synchronous and the asynchronous case, but the model is entirely non-random and therefore difficult to apply for non-deterministic *real-world* systems. In [29], it is shown that asynchronous algorithms have a "good" communication complexity as compared to synchronous ones, but here again, it is difficult to use these results to obtain quantitative measures of the actual performance of the algorithm in a given distributed environment.

A unique reference that proposes a real stochastic approach is Üresin and Dubois [32]. They give an analysis with expected values based on [17]. This work is of interest for shared-memory homogeneous implementations and an *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. By contrast, this paper focuses on distributed-memory systems, removes several of these restrictions and assumes *static scheduling* (which simulations in [32] imply to be superior) to develop a Markov chain model.

2 Parallel Iterative Algorithms

As stated earlier, we focus on iterative algorithms in which an operator is applied to a set of data (usually a vector) repetitively until some convergence criteria are met. If the set of data constructed by the algorithm is a sequence of vectors $\{x(t)\}$ in $\mathbb{R}^m$, then the algorithm can be written as

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

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 some specific problems or finding operators that provide the highest convergence speeds (cf. [23]). In this research, we consider only the general case without consideration of the specifics of the numerical method. The computation, in our case updating the components of the solution vector, has to be distributed among a collection of processors. We have already stated that we would consider only *static scheduling*, meaning that each processor updates one *piece* of $x(t)$, that is a preassigned, fixed subset of the components of this vector.

2.1 Synchronism vs. Asynchronism

Synchronous implementations of iterative algorithms are generalizations of sequential implementations. This makes them attractive as their convergence properties are thus well known. Often, it is rather straightforward to convert a sequential implementation of a given algorithm into a synchronous parallel implementation. Let us 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 piece 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 pieces 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:

$$\forall i, t \quad x_i(t+1) = Op_i(x_1(t), ..., x_m(t)). \tag{2}$$

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 [21]. However, with improvement in network technologies, this becomes less and less a concern if the implementation runs on a local area network. Second, and more importantly, the possible lack of synchronization among the processors [26, 18, 8, 14] can lead to dramatic performance losses because relatively fast processors may be idle waiting for the slower processors. Sources of such lack of synchronization have already been identified in Section 1.1 and become particularly noticeable when using a non-dedicated cluster of workstations to run the iterative algorithm. This phenomenon is a clear motivation for studying asynchronous implementations.

The study of asynchronous implementations started as early as 1969 [11] 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 assigned to each processor (static scheduling). By contrast with the synchronous implementation, there is no all-to-all communication phase to synchronize the processors. Instead, a processor is “free” to perform another update possibly using **out-of-**

date data for the pieces of the other processors, or not to perform any update at all. In addition, a processor can decide at any time to send its most up-to-date piece to some of the other processors. A good formal description of the asynchronous iteration is given in [2] and is inspired by the definition of chaotic relaxations in [11]. The definition we give here is very similar:

$$\forall i, t = 1, 2, ... \quad x_i(t) = \begin{cases} x_i(t-1) & \text{if } i \notin J_t \\ \\ Op_i(x_1(s_1(t)), ..., x_m(s_m(t))) & \text{if } i \in J_t, \end{cases} \tag{3}$$

where J_t is a subset of $\{1, .., m\}$, and $s_i(t)$ is an integer for all i, and $t = 1, 2, ...$ Baudet in [2] proposes the three additional conditions:

Condition 2.1 *Conditions for asynchronous iterations:*

(i) $s_i(t) \leq t$ *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 make use of *past* components. Condition **(ii)** states that the same value for a component cannot be used indefinitely when computing updates. Condition **(iii)** says that a processor does not abandon a component for ever. 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 to this amount, the implementation is referred to as a *totally asynchronous* implementation in [7]. Otherwise, the implementation is said *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 were performance bottlenecks for a synchronous implementation. However, the convergence of the algorithm is no longer implied by the same conditions as for the sequential case and its convergence rate must be reexamined.

Work devoted to proving and analyzing the convergence of asynchronous parallel iterative algorithms includes [7, 6, 11, 20, 19, 2, 5, 3, 28, 13, 30, 31]. Some of the earliest focused on specific iterative algorithms or on specific implementations. A sufficient condition for convergence for linear operators is available in [11], only for partially asynchronous implementations. In [20, 19], this sufficient condition is generalized to the case of certain non-linear operators, still in a partially asynchronous setting. A recent and general theorem in [7] sufficient condition for convergence of asynchronous iterative algorithms based on a sequence of subsets of $\mathbb{R}^m$ (a "box condition"). However, applications given in [7] are all contractions or pseudo-contractions with respect to a weighted maximum norm, a "Lipschitz-like" condition, as it is difficult to fully exploit the generality of the theorem. 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 [2]. 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 a contracting operator on a subset* D *of* $\mathbb{R}^m$ *if there exists a nonnegative* $m \times m$ *matrix* A *such that*

$$\forall x, y \in D \quad |Op(x) - Op(y)| \leq A|x - y|, \quad \textit{component-wise}$$

and $\rho(A) \leq 1$ *where* $\rho(A)$ *denotes the spectral radius of* A.

Furthermore, Baudet provides a lower bound on the convergence rate of the algorithm defined traditionally as:

$$\mathcal{R} \stackrel{\Delta}{=} \liminf_{t \to \infty} [(-\log \|x(t) - \xi\|)/t]. \tag{4}$$

where $\|.\|$ denotes a norm of $\mathbb{R}^n$ and ξ 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 $1/\mathcal{R}$ 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 Equ. (3) for all i. Without any additional assumptions, Baudet states that:

$$\mathcal{R} \geq -[\liminf_{t \to \infty} (k_t/t)] \log \rho(A). \tag{5}$$

where $\{k_t\}$ is a sequence of integers defined in [2]. Due to space constraints, we will just say that this sequence is increasing and the more asynchronous the implementation, the less rapidly it increases.

The main limitation of those convergence results is that, even if they were more in touch with what the end-user needs to make design decisions, they still provide only lower bounds on the rate of convergence. Experiments in [2] indicate that the bounds are very conservative, and the possibility for stochastic approaches is mentioned. In the next section, we present stochastic models to extend the convergence results.

3 Application-level Models

In this work, we are interested in providing the *end-user* with useful insight into the performance of different implementation strategies for parallel iterative algorithms in some user-defined distributed environment. Some of the low-level elements of the computer system must be ignored or at least approximated to perform an application-level analysis.

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 and are therefore supporting a *heterogeneous* environment. 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 for diverse computer networks, from a Massively Parallel System (MPP) 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 computation is modeled by an RV that describes the time that node spends to perform one update of its piece 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. An important element here is that these discrete probability distributions may be empirically estimated or analytically specified. They are assumed stationary during the run of the iterative algorithm.

3.2 Modeling the Algorithm

In order to make our stochastic models tractable, we segment the algorithm in *phases*. Figure 1 depicts one phase. Each phase is composed of two *sub-phases*. During the first sub-phase, called the α *sub-phase*, each processor performs successive updates on its piece of the solution vector. If a processor performs more than one update during the α sub-phase, then it begins to use out-of-date data for the components of the solution vector that it is not updating. At the end of the α sub-phase, each processor broadcasts its piece of the current solution vector to all the other processors. Just after this broadcast, starts the β *sub-phase*. During the β sub-phase each processor is waiting to receive $p-1$ messages from the other processors. Each processor also has the possibility to perform additional updates on its piece of the solution vector, using more out-of-date data. A processor finishes its β sub-phase when it has received all the $p-1$ messages and it then moves onto the next algorithm phase. For each processor, the user can chose the number of updates to be performed during that processor's α sub-phase and the maximum number of updates to be performed during the β sub-phase.

Note that the model can also describe a synchronous implementation, that is, for each processor, one single update during its α sub-phase and no update during its β sub-phase. In the next section, we give a few definitions, make one assumption, and exhibit a Markov chain (cf. [15]) of interest.

3.3 Underlying Markov chain

Let us define the following constants and RVs:

Definition 3.1

- **(i)** $A_i > 0$ *denotes the number of updates performed by processor* i *during the* α *sub-phase of each algorithm phase (implementation dependent).*

- **(ii)** B_i *denotes the* **maximum** *number of updates that processor* i *is allowed to perform during the* β *sub-phase of each algorithm phase (implementation dependent).*

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

- **(iv)** $n_{i \to j}(k) \in \mathbb{R}$ *is the duration in seconds of the message transfer from processor* i *to processor* j *during the* k^{th} *algorithm phase (RV). 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) $T^i(k) \in \mathbb{R}$ *denotes the time of the beginning of the* k^{th} *algorithm phase on processor* i *(RV).*

We assume that for each given i, the RV $\alpha^i(k)$ are independent and identically distributed (i.i.d) for each k. Similarly, for each given i and j, the RV $n_{i \to j}(k)$ are i.i.d for each k. This assumption is fundamental and has been popular in the past for the purpose of generating tractable models [32, 1]. The experimental results presented in Section 6 will illuminate its validity and limitations. In Section 7 we propose ideas to relax the i.i.d assumption, but we leave them for future work. We also make minor assumptions, such as bounded network times, messages being sent at exactly the same time during a broadcast, and free message sends in terms of CPU cycles on the sending processor.

We define the *wavefront*, $X(k)$ as

$$X(k) = (0, T^2(k) - T^1(k), .., T^p(k) - T^1(k)) \in \mathbb{R}^p, \tag{6}$$

which describes the shape of the line joining the entry points of each processor in the k^{th} phase, taking arbitrarily processor 1 as reference. It is represented on figure 1 as a thick line, and one can show that (See Appendix A):

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

Thanks to our i.i.d assumption, equation 7 is a Markov equation, and the wavefront appears as a time-homogeneous Markov chain. It is shown in Appendix B that, with a few technical assumptions, the Markov chain is also finite-state. Equation 7 can be used to compute the transition probability matrix of the Markov chain. Examples of computations of the Markov chain transition matrix and of its stationary distribution can be found in [10].

4 Performance Characterization

The wavefront Markov chain is exploited to obtain performance results for the parallel iterative algorithm. The goal is to obtain as much information as possible on the execution time. The execution time can be computed as the ratio of the number of iterations to perform over the number of iterations performed by time unit. The number of iterations to perform can be approximated with the asymptotic convergence rate of the algorithm which is the object of the next section.

4.1 Asymptotic Rate of Convergence

The challenge here is to extend Baudet's work in [2] and replace his lower bound on the asymptotic rate of convergence by modified estimates that take into account the randomness in the algorithm implementation. One can compute three estimates (see Appendix D) respectively called *worst-*, average-, and *best-*case estimates (and denoted by $\underline{\mathcal{R}}$, $\widehat{\mathcal{R}}$, and $\overline{\mathcal{R}}$) At this point, the meaning of the average- and best-case estimates is unclear. The worst-case estimate however is a clear improvement over Baudet's lower bound as (i) it is higher than Baudet's and (ii) we guarantee that it is still a lower bound on the asymptotic convergence rate.

Let $\mathcal{R}^*$ denote one of the three estimates for this rate. Then, if the user wants the initial error on the solution vector to be divided by a factor of 10^{ω}, we approximate the necessary number of iterations to be performed by $\omega/\mathcal{R}^*$ assuming that ω is reasonably large. Of course, the choice of $\mathcal{R}^*$ is crucial, and we expect that our three estimates will provide good guidance for this choice.

4.2 Implementation speed

We can use $\mathcal{R}^*$ to estimate the speed of the implementation in terms of number of iterations performed per time unit. Let $\Phi(k)$ denote the duration in seconds of the k^{th} 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 probability distributions can be approximated by making use of the wavefront π-values. The speed of the implementation is therefore entirely described by sums of the random vector $\binom{N(k)}{\Phi(k)}$. This vector can be used in different ways, as described in the next section.

4.3 Performance Characterization Levels

Level 1 : Using the Strong Law of Large Numbers [15], it is possible to compute the asymptotic algorithm speed as the ratio of the expected values of $N(k)$ and $\Phi(k)$. Calling $\mathcal{S}$ this ratio, one obtains θ_1, an asymptotic estimate for the execution time expected value:

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

Level 2 : It is possible to compute an asymptotic estimate of the standard deviation of the execution time distribution (See Appendix C).

<u>Level 3</u> : The third level, based on Large Deviation Theory [9, 27, 33], will be presented in a subsequent paper.

5 Simulation

In order to validate our approach, we performed a series of simulations. The results are related in Section 5.1 of [10] and are excellent as none of our basic assumptions is violated. We simulated a Gradient algorithm for a real multi-polynomial cost function with thirty variables. The simulated distributed environment consisted of three processors with different arbitrary (non standard) workload distributions (depicted in Figure 2), interconnected by a network that delivers constant performance (see [10], p.143, for details).

The simulation provides relevant information about the accuracy and sensitivity of our different characterization levels. First, we observe that our new asymptotic convergence rate estimates provide much better results than Baudet's estimate. Table D shows the relative errors between the different estimates and the observed convergence rate for different implementations in our simulated distributed environment, for which our average-case estimate provides a better guess for the convergence rate than Baudet's estimate. The gaps between the four estimates increases 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 convergence rate estimates will be computed for different operators that happen to have the same contracting matrix.

The simulation also shows that level 2 characterization is rather sensitive. Indeed, as it is based on a binomial Gaussian approximation, it requires a large number of *samples* to be accurate. It was shown in [10] that the error between level 2 characterization and the observe standard deviation decreases for increasing values of ω, the user's convergence criterion.

Figure 3 shows the simulation versus our characterization for an asynchronous implementation in our simulated heterogeneous distributed environment. On each graph, the empirical distribution of the execution time is shown as a bar diagram. The empirical mean is shown as a vertical solid line and the empirical standard deviation is represented as a horizontal line segment 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 on each side of level 1. We show four characterizations, each corresponding to a different asymptotic convergence rate estimate. One observes the errors on level 1 characterizations quantified in the third line of Table D. The error on level 2 is only about 20% if the average-case estimate is chosen, and this error decreases when ω increases. The error is less than 1% for the synchronous implementation which is not shown here.

6 Experiment

This section presents experimental results [10] with a Gradient algorithm for a real multi-polynomial cost function with thirty variables 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 our first asynchronous implementation (*mildly* asynchronous: at most one update performed in β sub-phase for each processor). This corresponds to 862 consecutive observations for each implementation. The measurements for our second asynchronous implementation (*fairly* asynchronous: at most two update performed in β sub-phase for each processor) are not shown because they would be difficult to distinguish from the ones of the first asynchronous implementation on that time scale.

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

It seems that, in this environment, a good choice is to use an asynchronous implementation 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 research works include examples for which asynchronous implementations outperform synchronous ones [2, 4, 21].

A fundamental observation on Figure 4 is that the execution time is *bursty*. In fact, the distributed environment, and therefore the algorithm, behaves very differently at different times in our time period, for the system is in use for a variety purposes during the experimental runs. In order to illustrate these different behaviors, Figures 5(a), (b) and (c) show three close-ups of the execution times for each implementation during three short sub-periods about two hour long. Without going into details, one can immediately see that the distributed environment behaved according to distinct *modes* during the week.

Figure 7 shows the execution times for the parallel iterative algorithm throughout a 24 hour time period at the end of the week. The distributed environment exhibited a fairly stable behavior, leading to relatively smoother observations. We expect our stochastic models to yield better results for the 24-hour time period than for the entire week as burstiness indicates violation of our i.i.d assumption. In the following sections, we present and comment on some of the results for both time periods.

6.2 Applying the Model

6.2.1 The One Week Time Period

The workload distributions of the three processors where sampled throughout the week and are shown in figure 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 is the most striking as it is smaller than the observed standard deviation by a factor of 50! In fact, level 2 is very sensitive to the violation of the i.i.d assumption as a Gaussian approximation is involved.

The workload distributions were sampled and are shown on Figure 10. 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 our average-case estimate (see Section 4.1), then the error on the mean prediction is only 4% (as opposed to 116% with Baudet's estimate). The observations made on Figure 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 24 hours and should lead to improvements, especially for the level 2 characterization as the i.i.d assumption should be less violated.

6.2.2 The 24 Hours Time Period

Figure 11, for a synchronous implementation, demonstrates that the level 2 characterization becomes much more accurate for this shorter time period. One can easily observe that its error is about 27% whereas it was a factor of 50 for the whole week. The same kind of improvements were observed for all implementations. Furthermore, the level 1 characterization is more informative than for the one week time period. Indeed, the use of our models states clearly that an asynchronous implementation will on average outperform a synchronous one by about 40 seconds which perfectly agrees with Figure 7.

7 Conclusion and Future Work

Parallelizing iterative algorithms for the solution of large or complex optimization problems is a crucial issue. Indeed, the amount of computation required to solve such problems can be prohibitive for a sequential implementation, especially in non-linear cases. We examined different popular parallelization strategies as well as the corresponding theoretical results available in the literature and came to the conclusion that there is a gap between those results and what the end-user needs to know about the execution of his parallel iterative algorithm. This work showed that this gap can be filled using stochastic models that take into account the distributed environment used to run the algorithm. Such models have been developed and used to obtain a variety of performance characterizations that are directly meaningful to the end-user.

An additional level of performance characterization based on Large Deviation Theory will be covered in a future paper. Additional research may also extend the current models to Markov-modulated random processes to model bursty behaviors in more details [22, 24, 12, 25, 16].

APPENDIX

A Wavefront Equation

Let $\beta^i(k) \in \mathbb{R}$ be the the duration in seconds of the β sub-phase of the k^{th} algorithm phase on processor i. $\beta^i(k)$ is a RV and it can be computed as follows. The β sub-phase of the k^{th} algorithm phase on processor i clearly starts at time $\beta^i_{start}(k) = T^i(k) + \alpha^i(k)$. 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 $\beta^j_{start}(k) + n_{j\to i}(k)$. Therefore,

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

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

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

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

By definition 3.1(**v**) and Equ. 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

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

one obtains Equ. 7. ∎

B Wavefront State-Space

Lemma B.1 $\exists M, \forall k \geq 1 \quad \|X(k)\|_\infty \leq M.$

Proof. Let us consider processor 1 and processor $i \neq 1$ during the k^{th} algorithm phase. The times at which 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 $\max_{h\in\{1,..,p\}}(|n_{h\to 1}(k) - n_{h\to i}(k)|)$. The communication times are assumed to be bounded above and below as:

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

On can then write that:

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

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

The wavefront vector is now in a closed ball of $\mathbb{R}^p$. If one assumes that the RVs $\alpha^i(k)$, $n_{i\to j}(k)$, 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 and 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 depends on the distributions of $\alpha^i(k)$ and $n_{i\to j}(k)$ (see [10]).

C Level 2 Characterization

One can make a binomial Gaussian approximation for 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

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

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:

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

This result is available in [15] for instance.

D Asymptotic Convergence Rate Estimates

To compute estimates of the algorithm asymptotic rate of convergence, we need to extend Equ. 5. In [2], the sequence $\{t_k\}$ is defined as:

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

where $\{a_k\}$ and $\{b_k\}$ 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 Equ. 5 is then defined as:

$$k_t \stackrel{\Delta}{=} \sup\{k \in \mathbb{N} | a_0 + b_0 + ... + a_{k-1} + b_{k-1} \leq t\}.$$

In our setting, one has:

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

where N(k) denotes the number of iterations performed during the k^{th} algorithm phase. One can then compute:

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

The long-term probability distribution of the RV $N(k)$ can be approximated using the wave-front π-values, leading to the probability distribution of k_t for each t. It is then possible to compute three estimates for the asymptotic rate of convergence by replacing k_t in equation 5 by its minimal observable value, its expectation, or its maximal observable value. A formal proof of the convergence of the limit in equation 5 for each estimate is left for future work. The existence of a finite limit has been witnessed in every simulation and experiment.

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*, pages 61–73, May 1993.

[2] G. Baudet. Asynchronous Iterative Methods for Multiprocessors. *Journal of the Association for Computing Machinery*, 25:226–244, April 1978.

[3] D. El Baz. M-functions and parallel asynchronous algorithms. *SIAM Journal of Numerical Analysis*, 27:136–140, 1990.

[4] D. El Baz, P. Spiteri, J.C. Miellou, and D. Gazen. Asynchronous Iterative Algorithms with Flexible Communication for Nonlinear Network Flow Problems. *Journal of Parallel and distributed Computing*, 38:1–15, 1996.

[5] D. P. Bertsekas. Distributed asynchronous computation of fixed point. *Math. Programming*, 27:107–120, 1983.

[6] D. P. Bertsekas and J. N. Tsitsiklis. Convergence rate and termination of asynchronous iterative algorithms. In *Proceedings of the Int. Conf. on Supercomputing*, pages 461–470, 1989.

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

[8] 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 (ICPP)*, volume III, pages 153–160, 1989.

[9] James A. Bucklew. *Large Deviation Techniques in Decision, Simulation, and Estimation.* Wiley-Interscience, New York, NY, 1990.

[10] H. Casanova. *Stochastic Models for Performance Anlyses of Iterative Algorithms in Distributed Environments.* PhD thesis, Dept. of Computer Science, University of Tennessee, Knoxville, TN, 1998.

[11] D. Chazan and W. Miranker. Chaotic relaxation. *Linear Algebra and Applications*, 2:199–222, 1969.

[12] N. Duffield, J. Lewis, N. O'Connel, R. Russell, and F. Toomey. Entropy of ATM Traffic Streams: A Tool for Estimating QoS Parameters. *IEEE Journal on Selected Areas in Communications*, 13(6):981–989, August 1995.

[13] A. Frommer. On asynchronous iterations in partially ordered spaces. *Numerical Funct. Anal. Optimization*, 12(3 & 4):315–325, 1991.

[14] A. Greenbaum. Synchronization costs on multiprocessors. *Parallel Computing*, 10:3–14, 1989.

[15] S. Karlin and H. Taylor. *A First Course in Stochastic Processes.* Academic Press, New York, NY, second edition, 1975.

[16] K. Kawahara, Y. Oie, M. Murata, and H. Miyahara. Performance Analysis of Reactive Congestion Control for ATM Networks. *IEEE Journal on Selected Areas in Communications*, 13(4):651–661, May 1995.

[17] C. P. Kruskal and A. Weiss. Allocating independent subtasks on parallel processors. In *Proceedings of the International Conference on Parallel Processing (ICPP)*, pages 236–240, 1984.

[18] B. Lubachevsky and D. Mitra. Chaotic Asynchronous Algorithm for Computing the Fixed Point of a Nonnegative Matrix of Unit Spectral Radius. *Journal of the ACM*, 33(1):130–150, January 1986.

[19] J.C. Miellou. Algorithmes de relaxation à retards. *Revue d'Automatique, Informatique et Recherche Opérationnelle*, 9:55–82, 1970.

[20] J.C. Miellou. Itérations chaotiques à retards. *Comptes Rendus de l'Acad, Sci. Paris*, 278:957–960, 1974.

[21] S. P. Olesen. *Parallel Computation for Positron Emission Emission Tomography with Reduced Processor Communications*. PhD thesis, Dept. of Computer Science, University of Tennessee, Knoxville, TN, 1996.

[22] R. Onvural. *Asynchronous Transter Mode Networks, Performance Issues*. Artech House, Inc., Norwood, MA, second edition, 1995.

[23] J. M. Ortega and W. C. Rheinboldt. *Iterative solution of nonlinear equations in several variables*. Academic Press, New York, 1970.

[24] P. Pruthi and A. Erramilli. Heavy-Tailed ON/OFF Source Behavior and Self-Similar Traffic. *Unknown IEEE Journal*, 2:445–450, 1995.

[25] J-F. Ren, J. Mark, and J. Wong. End-to-End Performance in ATM Networks. *Unknown IEEE Journal*, pages 996–1002, 1994.

[26] J. T. Robinson. Some Analysis Techniques for Asynchronous Multiprocessor Algorithms. *IEEE Transactions on Software Engineering*, SE-5(1):24–31, January 1979.

[27] Adam Shwartz and Alan Weiss. *Large Deviation for Performance Analysis*. Chapman & Hall, London, UK, 1995.

[28] E. Tarazi. Some convergence results for asynchronous algorithms. *Numerical Mathematics*, 39:325–340, 1982.

[29] J. N. Tsitsiklis and G.D. Stamoulis. On the average communication complexity of asynchronous distributed algorithms. Technical Report LIDS-P-1986, Laboratory for Information and Decision Systems, 1990.

[30] A. Üresin and M. Dubois. Sufficient conditions for the convergence of asynchronous distributed algorithms. *Parallel Computing*, 10:83–92, November 1989.

[31] A. Üresin and M. Dubois. Parallel asynchronous algorithms for discrete data. *Journal of the ACM*, 37(3):588–606, 1990.

[32] A. Üresin and M. Dubois. Effects of Asynchronism on the Convergence Rate of Iterative Algorithms. *Journal of Parallel and Distributed Computing*, 34:66–81, 1996.

[33] Alan Weiss. An Introduction to Large Deviations for Communication Network. *IEEE Journal on Selected Areas in Communications*, 13(6):938–952, 1995.

Table 1: Convergence rate errors.

Impl.	$\underline{\mathcal{R}}$	$\widehat{\mathcal{R}}$	$\overline{\mathcal{R}}$	$\mathcal{R}_{Baudet}$
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%

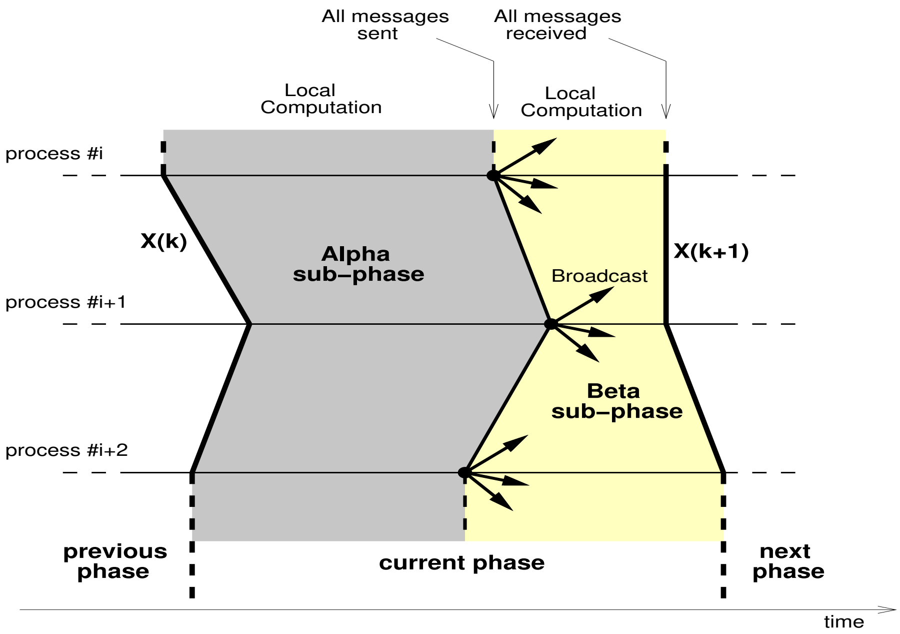

Figure 1: Decomposition of the algorithm in phases

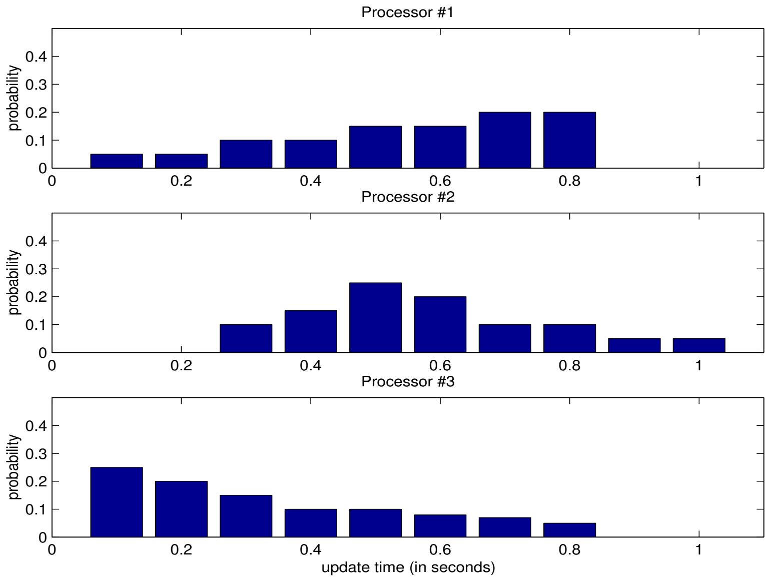

Figure 2: Update time distributions for the three processors

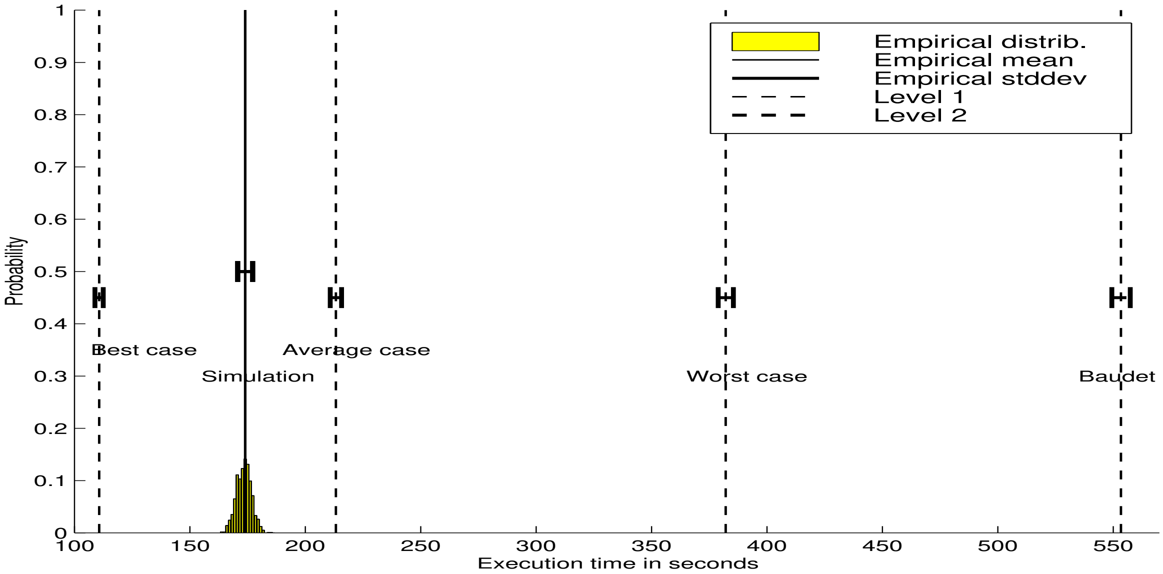

Figure 3: Simulation vs. characterizations for an asynchronous implementation

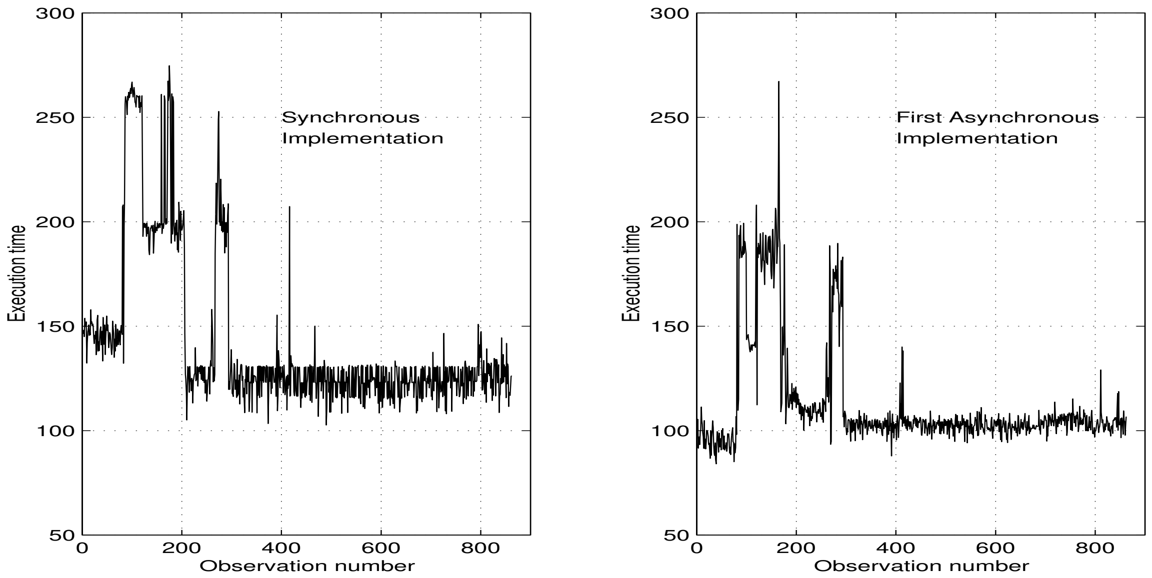

Figure 4: Execution time measurements over a week

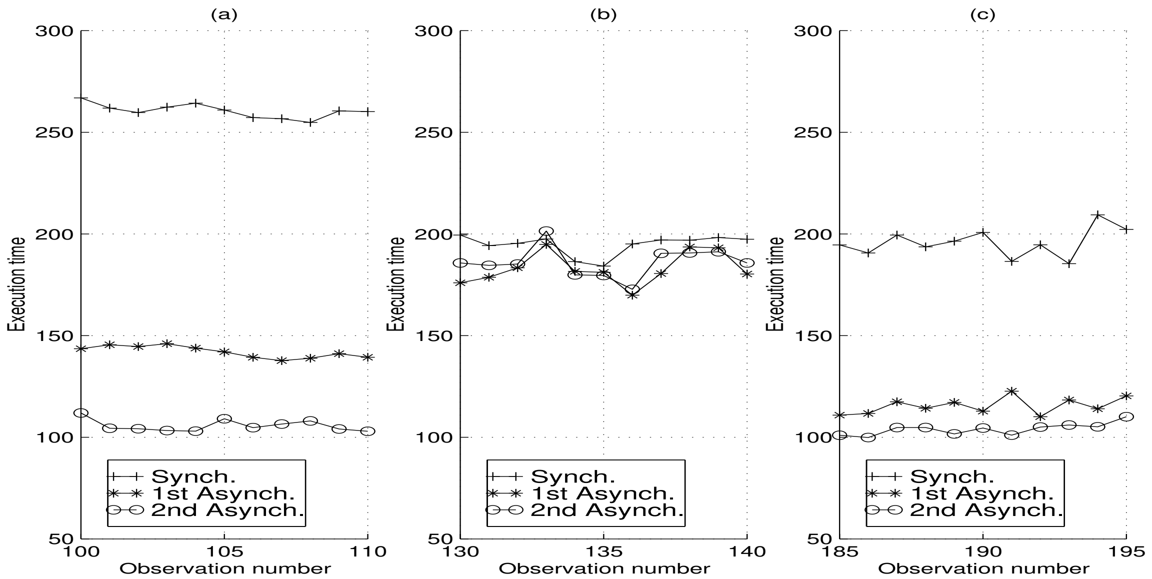

Figure 5: Different experimental behaviors throughout one week

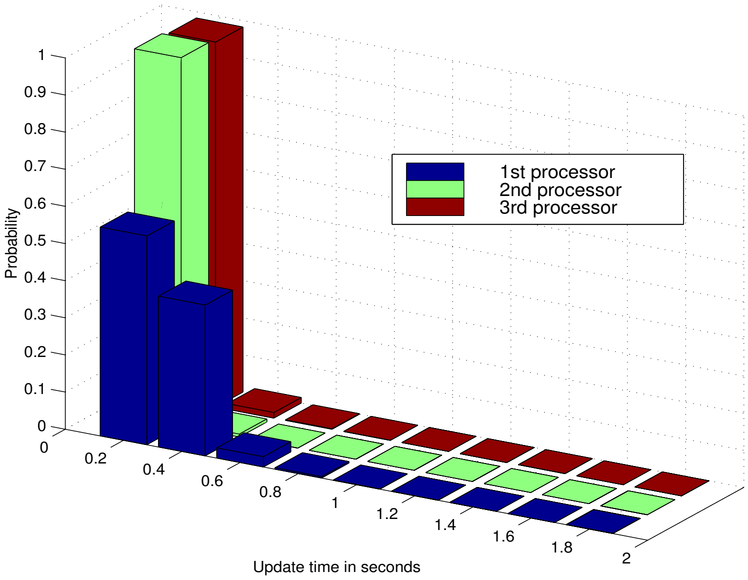

Figure 6: Simulated update time distributions for the three processors

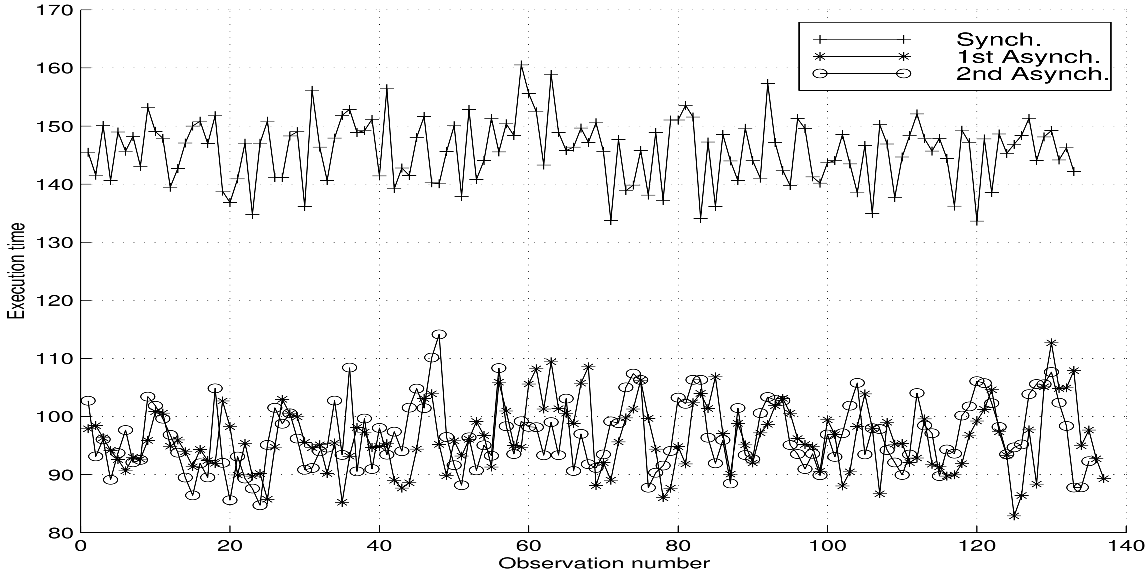

Figure 7: Measurements during 24 hours for the three implementations

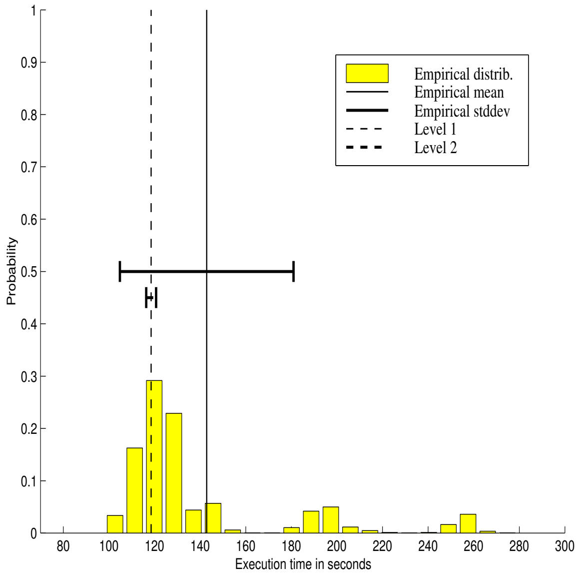

Figure 8: Experiment vs. Characterization for the synchronous implementation

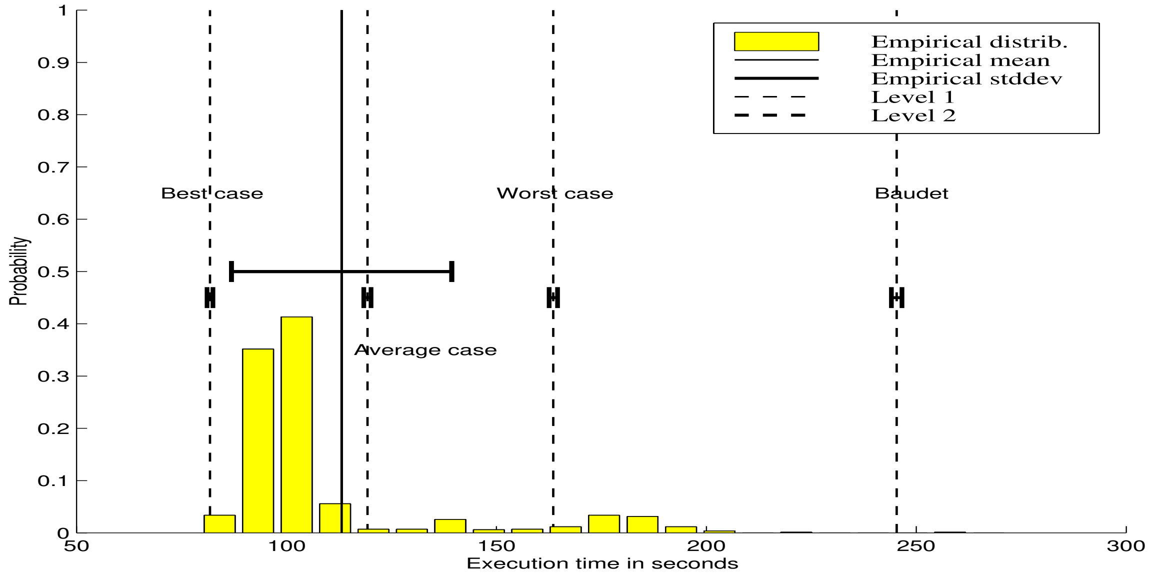

Figure 9: Experiment vs. characterization for the first asynchronous implementation

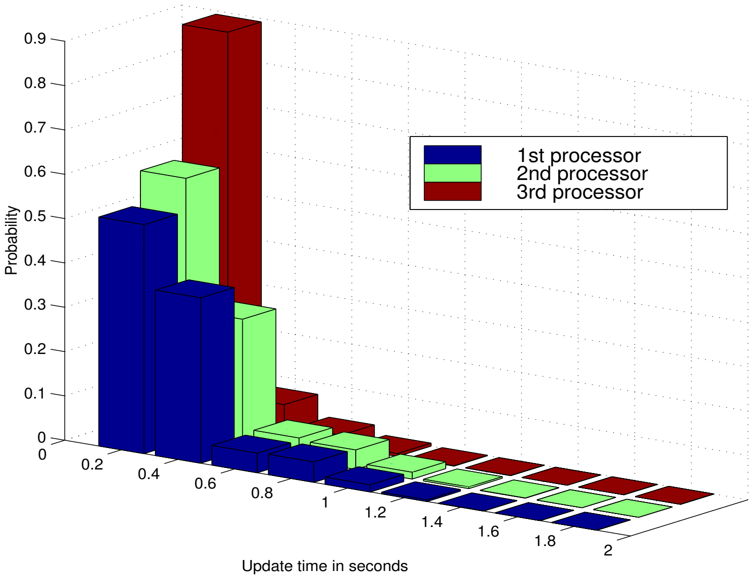

Figure 10: Experimental update time distributions for the three processors

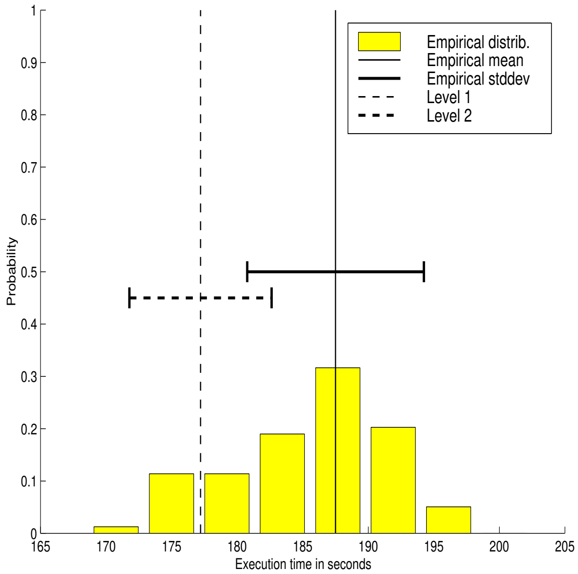

Figure 11: Experiment vs. Characterization for the synchronous implementation

Michael G. Thomason received the BS from Clemson University in 1965, MS from Johns Hopkins University in 1970, and PhD from Duke Univiversity in 1973. He worked for Westinghouse (Baltimore) and currently is Professor of Computer Science at the University of Tennessee, Knoxville. His research interests include pattern/image analysis, parallel/distributed computation, and stochastic models in computer science. He is a member of ACM and senior member of IEEE.

Henri Casanova received the BS from l'Ecole Nationale Supérieure d'Electrotechnique, d'Informatique et d'Hydraulique de Toulouse (ENSEEIHT) in 1993, MS from l'Université Paul Sabatier, Toulouse, in 1994, and PhD from the University of Tennessee, Knoxville, in 1998, and is currently a postdoctoral research associate at the University of Tennessee, Knoxville. His research interests include metacomputing, parallel/distributed computing, performance modeling, and stochastic models.

Jack Dongarra Jack Dongarra received the PhD in Applied Mathematics from the University of New Mexico in 1980, MS in Computer Science from the Illinois Institute of Technology in 1973, and BS in Mathematics from Chicago State University in 1972. Dongarra is a Distinguished Scientist specializing in numerical algorithms in linear algebra at the University of Tennessee'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).