Performance Of Asynchronous Linear Iterations With Random Delays

Adrian C. Moga and Michel Dubois

CENG Technical Report 95-06

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213)740-4475

March 1995

Performance of Asynchronous Linear Iterations with Random Delays *

Adrian C. Moga and Michel Dubois

Department of Electrical Engineering-Systems
University of Southern California
Los Angeles, CA 90089-2562
E-mail: {moga,dubois}@paris.usc.edu

March 28, 1995

Abstract

Long time praised for the elimination of synchronization overhead, thus achieving smaller average time per iteration, asynchronous methods have been suspected of consistently increasing the number of iterations until convergence, leading to a speedup trade-off. In this paper we investigate the speedup potential of asynchronous iterative algorithms over their synchronous counterparts by exclusively comparing convergence rates.

We define the asynchronous speedup, which depends on the number of iterations in the synchronous and asynchronous implementations and is independent of the execution time of each iteration. Employing both simulation and analytical approaches, we found several cases and conditions for high asynchronous speedups in our extensive analysis of linear iterations of size two. However, average asynchronous speedups computed over the whole set of iteration matrices which satisfy the convergence criterion reveal that the domain of matrices with high asynchronous speedups is not sizable and that asynchronous iterations perform no better than synchronous ones over the entire convergence space.

Keywords: parallel asynchronous algorithms, linear iterative methods, multiprocessor systems, performance evaluation, simulation.

*This research has been funded by the National Science Foundation Grant No. CCR-9222734.

1 Introduction

Finding a solution x^* to the generic fixed point problem

$$x = f(x), \qquad x \in \mathcal{R}^n$$

is a common way to solve many problems, such as systems of equations or optimizations. While, in some cases, it is possible to provide a direct solution, the large and growing sizes of applications and the availability of parallel and distributed high-performance computing environments make iterative methods more attractive. These methods compute successive approximations of the solution by executing the basic iterative step

$$x_{k+1} = f(x_k) \tag{1}$$

with x_0 being an initial estimation of the solution x^*. Provided conditions of convergence are met, the iterative process stops when the iteration vector x_k is found to be suitably *close* to the solution according to some criterion.

The computation described by (1) is ideally suited for parallel execution by breaking up the set of indexes $1, 2, ..., n$ into disjoint subsets and performing the independent computations of the corresponding components of the iteration vector x_{k+1} on different processors simultaneously. *Synchronous* methods strictly alternate the tasks of computing and communicating the results, and reproduce the exact same sequence of iterations as in the sequential implementation since a new iteration is started by a processor only when all the results from other processors have been received. *Asynchronous* methods [1] decouple computation from communication and allow a new cycle to start even with obsolete information. Asynchronous iterations eliminate synchronization overhead, thus achieving smaller average time per iteration. However, in some cases such as monotone mappings [3], they have been shown to increase the number of iterations until convergence, leading to a performance trade-off.

Several papers have attempted to address these performance aspects. Qualitative results for generic classes of mappings, such as monotone or contractions [3, 4] exist. They are established analytically on very general, deterministic models of the asynchronous mode of operation. These results tell us when asynchronous iterations converge faster, but not how much faster. In other cases, by making additional assumptions on the properties of the iteration mapping, such as a coupling strength for contractions, a crude estimate of the asynchronous convergence rate is computed.

A large number of papers report on observed speedups of iterative algorithms through asynchronous execution on actual multiprocessors. Whereas such experimental results are interesting because of their realism, they are hard to interpret and are specific to the machine and to the way the software was written. Additionally, they can only offer a partial picture of the performance of the asynchronous method, as they, typically, report on a few test cases only. Whether these cases can be extrapolated and used to obtain general conclusions is unclear.

In several papers, some significant asynchronous speedups have been reported [5, 11]. The authors of [5] focus on linear iterative methods, a common particular case where the mapping f is of type

$$f(x) = Ax + b$$

A being an $n \times n$ iteration matrix and b a vector of size n. Linear iterations arise in the solving of large, sparse linear systems of equations. High speedups are observed for cases where the synchronous iteration oscillates around the fixed point. The effect of asynchrony is to damp these oscillations, thereby resulting in much faster convergence. In [5], the authors indicate that this effect only appears if the time required to pass messages is not much greater than the time required for each processor to perform one iteration, and the communication/computation ratio is about one. This intuition is then verified experimentally by actual execution on a multiprocessor for a set of iteration matrices with negative

dominant eigenvalue. A consistent reduction (of up to a factor of 4.3) in the number of iterations is shown for the asynchronous iteration. Additionally, for iteration matrices with positive dominant eigenvalue, asynchronous iterations consistently required more iterations to converge.

In this paper we use simulations and analytical models to investigate cases where asynchronous execution leads to a sizable decrease in the number of iterations required for convergence. The basic cause of this decrease is the randomness injected in the asynchronous iteration by the unpredictable timings of component updates. To shed light on the effect of randomness on the performance of asynchronous iterations we have performed an exhaustive analysis of linear iterations when $n = 2$. The entire domain of 2×2 iteration matrices for which convergence conditions are met is sampled to yield over a quarter million matrices. Iterations corresponding to each of these matrices are compared. In addition to identifying properties of iteration matrices with high asynchronous performance, we present the results of a detailed evaluation of the effects of asynchronism on the convergence rate of linear iterations under various conditions of asynchronism. When comparing two iterative methods we use the asynchronous speedup. The asynchronous speedup is the ratio of the numbers of iterations for the synchronous and asynchronous implementations. Therefore the communication and synchronization overheads of the synchronous iteration does not affect the speedup.

We expose the characteristics of asynchronous iterations in three different simulation models. In the first model, the communication time is negligible and the source of asynchronism is due to random iteration times. Large asynchronous speedups are observed for very small domains of matrices having a dominant eigenvalue close to -1. Two effects contribute to this speed improvement: a Gauss-Seidel effect and pure hazard. However, when averaging performance over the whole set of matrices, there is no advantage for asynchronous iterations. Similar conclusions are reached for two other simulation models with constant iteration times but with random update propagation times. In addition, these models show

that larger average communication delays produce worse asynchronous performance.

Following the simulations, we develop an analytical model of asynchronous linear iterations with stochastic delays to evaluate the asynchronous convergence rate. This model involves the spectral properties of special matrices derived from the original iteration matrices. The observations derived from this analysis corroborate the simulation results. We find that asynchronous iterations do not significantly outperform synchronous iterations on the average, even when the asynchronous execution conditions are optimized for each iteration matrix. As observed in our simulations, significant asynchronous improvements require a large spectral radius and strong error oscillation.

Finally, we show that, in the case of linear systems of equations, the root causes of the high speedups obtained by asynchronism can be related to well-known convergence acceleration techniques for linear iterations.

The rest of the paper is structured as follows. Section 2 contains the background material, including an overview of the convergence criteria for synchronous and asynchronous linear iterations, of the effects of oscillations on synchronous convergence and their damping by asynchronism, as well as of the performance metrics used in the evaluations. In Section 3 we describe three simulation models of asynchronous iterations and report on the evaluation of their performance relative to the synchronous method. Next, in Section 4, an analysis of the convergence rate for linear iterations with stochastic delays is performed. Considerations on the acceleration of linear iterations are presented in Section 5. Finally, Section 6 contains our conclusions.

2 Background

2.1 Deterministic Convergence Conditions

In the following, we briefly review some of the main convergence conditions for the linear iteration

$$x = Ax + b$$

where $x, b \in \mathcal{R}^n$ and A is a $n \times n$ matrix with real elements. There exists an unique x^* satisfying the above relation *iff* $I - A$ is not singular. To find x^*, iterative methods rely on successive approximations, repeatedly updating an iteration vector. When this process is convergent, the iteration vector becomes a suitable approximation of x^* after some number of iterations. Depending on the rules used for updating the components of the iteration vector, there are two major classes of iterative methods, having different convergence properties: synchronous and asynchronous methods[4, 6, 8].

2.1.1 Synchronous Iterations

Synchronous linear iterations can be mathematically described by

$$x_{k+1} = Ax_k + b$$

where the integer k indexes different iterations, thus reflecting the number of updates of all components of the iteration vector. In order to start the $k+1$ iteration, all the computations and updates in step k must be first completed. Convergence is achieved when $\lim_{k\to\infty} x_k = x^*$. The necessary and sufficient condition for convergence is

$$\rho(A) < 1$$

where $\rho(A)$ is the spectral radius of matrix A, defined as

$$\rho(A) = \max_i(|\lambda_i|)$$

where λ_i is an eigenvalue of A.

Sometimes, as a sufficient condition, the maximum norm is used instead of the spectral radius for convenience.

$$\|A\|_\infty \stackrel{\text{def}}{=} \max_i(\sum_j |a_{ij}|) < 1$$

The speed at which x_k approaches x^*, called the *convergence rate* of the synchronous iteration, is governed by the spectral radius $\rho(A)$. This is justified by the following result for the error vector ε_k:

$$\|\varepsilon_k\| \stackrel{\text{def}}{=} \|x_k - x^*\| = \|A\varepsilon_{k-1}\| = \cdots = \|A^k\varepsilon_0\| \leq \|A^k\|\|\varepsilon_0\| \tag{2}$$

In particular, when the norm used for the error vector is euclidian $\|\cdot\|_2$ and when A is normal, the corresponding matrix norm is spectral and $\|A^k\| = \rho(A^k) = \rho^k(A)$, showing that the error vector asymptotically approaches 0 at the same rate as $\rho^k(A)$.

2.1.2 Asynchronous Iterations

Under deterministic assumptions, a sufficient condition for convergence is

$$\rho(|A|) < 1$$

where $\|A\|$ is the matrix with elements $\|a_{ij}\|$. Again, a simpler sufficient verification of the convergence is provided by $\|A\|_\infty < 1$.

Convergence conditions results for models with stochastic delays will be given in Section 4.

2.2 Oscillations and Damping Effects

Error oscillations in synchronous iterations and their damping by asynchronism were first reported by Bull and Freeman in [5]. In some cases, the damping of oscillations translates into significantly faster convergence for asynchronous iterations.

In [5], the cosine of two consecutive error vectors

$$C(k) = \frac{\varepsilon'_{k+1}\varepsilon_k}{\|\varepsilon_{k+1}\|_2\|\varepsilon_k\|_2}$$

is used as a measure of the degree of oscillation in the series of error vectors for the synchronous iteration. It is shown that when the dominant eigenvalue [1] λ_1 of the iteration matrix A is positive, there are no oscillations ($C(k) \to 0$), but strong oscillations occur when it is negative ($C(k) \to -1$). In both cases, the execution of the synchronous iteration is equivalent to applying the power method to A and ε_k asymptotically coaxes on the direction of $\mathcal{X}_1$, the eigenvector associated with λ_1. When λ_1 is positive, the error vectors keep the same direction as $\mathcal{X}_1$ whereas, when it is negative, they constantly switch direction from one iteration to the next, as illustrated in Figure 1. A certain degree of oscillation is shown to be present in other cases, such as when λ_1 is complex ($\lambda_2 = \lambda_1^*$ and $\|\lambda_1\| > \|\lambda_3\|$). In these cases, the degree of oscillation is variable in time, but can be periodic. Another interesting situation is when λ_1 is real and $\lambda_1 = -\lambda_2$. In these cases, the asymptotic amplitude of oscillations strongly depends on the initial initial condition.

2.3 Performance Metrics

In our evaluations, the primary indicator of performance of an iterative scheme is the number of iterations until convergence. The definition of the number of iterations must be applicable to both synchronous and asynchronous schemes.

[1] eigenvalues are indexed by decreasing size of their module.

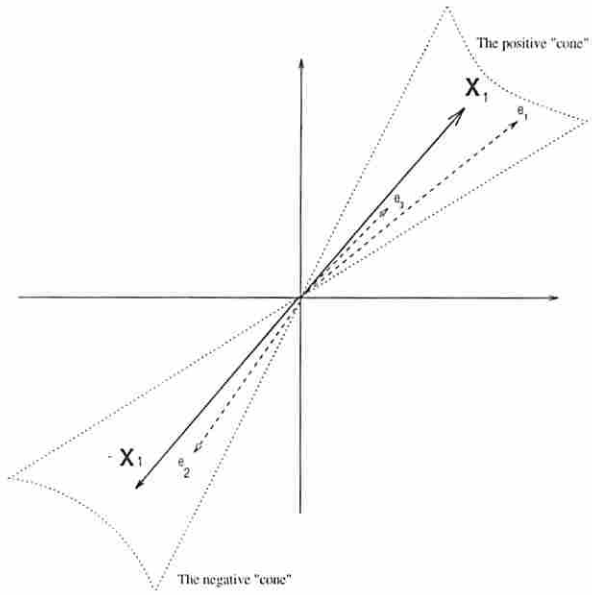

Figure 1: Error vector oscillation when $\lambda_1 < 0$.

2.3.1 Number of Iterations and Asynchronous Speedup

Given a linear iteration

$$x = Ax + b \tag{3}$$

an initial condition $x_0 \in \mathcal{R}^n$ and a termination condition $\mathcal{E}$, the number of iterations until convergence denoted $N_{it}(A, b, x_0, \mathcal{E})$ is defined as the total number of iterate component updates needed to satisfy the termination condition $\mathcal{E}$, divided by n, the size of the iterate vector. This definition applies to scalar components, but it could easily be easily generalized to block-iterative methods.

To compare two iterative schemes by their number of iterations until convergence, the initial condition and the quality of the solution x must be identical for both schemes. This implies independence of the termination condition $\mathcal{E}$ and of the computational process. The most accurate measure of quality for a solution estimate x is the distance from the actual solution x^*. Typically, the distance is euclidian, $\|\cdot\|_2$, but other norms could be used, such

as block $\|\cdot\|_1$ or maximum $\|\cdot\|_\infty$.

2.3.1.1 Synchronous Case

For synchronous iterative methods, the evolution of the iteration vector x from x_0 to the final value is unique and N_{it} is identical in every execution. N_{it} can therefore be found in a single simulation run.

N_{it} can also be estimated analytically for the Jacobi synchronous method [8]. Defining the *average rate of convergence* as

$$R_k(A) \stackrel{\text{def}}{=} -k^{-1}\log(\|A^k\|)$$

we have from relation (2) that

$$\log(\frac{\|\varepsilon_k\|}{\|\varepsilon_0\|}) \leq -kR_k(A)$$

and thus

$$k \approx -(\log\zeta)/R_k(A)$$

ζ being the error reduction factor. In a practical situation, $R_k(A)$ is unknown for a given k. However, for large values of k, the definition of the *asymptotic rate of convergence* as

$$R_\infty(A) \stackrel{\text{def}}{=} \lim_{k\to\infty} R_k(A) = -\log\lim_{k\to\infty}(\|A^k\|)^{1/k} = -\log\rho(A)$$

leads to an analytical approximation for the number of iterations:

$$k \approx -(\log\zeta)/R_\infty(A) = (\log\zeta)/(\log\rho(A)) \tag{4}$$

Therefore, the number of iterations for two synchronous schemes with the same initial

condition and the same quality of solution is asymptotically proportional to the inverse of the logarithm of the spectral radius.

2.3.1.2 Asynchronous Case

In asynchronous methods, the trajectory of the iteration vector and the number of iterations N_{it} are no longer uniquely defined by the initial condition and by the convergence criterion. The sequence of component updates (or trajectory of the iterate vector) as well as the number of iterations depends on the execution environment.

In this case, we track the *expected* trajectory of the iterate vector. In simulations, the expected trajectory is estimated by generating multiple random trajectories based on some stochastic distribution. The number of iterations is taken as the average of the number of iterations in each trajectory. Therefore, in the asynchronous case, N_{it} will be an average over multiple trajectories.

Based on the expected trajectory, analytical estimates of the average number of iterations can also be obtained by a formula similar to (4), where matrix A is replaced by an augmented matrix taking into account the distribution of possible trajectories, as we will see in Section 4.

2.3.1.3 Asynchronous Speedup

To compare the performance of synchronous and asynchronous methods, we will use the *asynchronous speedup*. The asynchronous speedup is the ratio between the numbers of synchronous and asynchronous iterations needed to meet the convergence condition:

$$\mathcal{S}_P = \frac{N_{it}^{sync}(A)}{N_{it}^{async}(A)} \tag{5}$$

Note that in this definition, $N_{it}^{async}(A)$ is the expected number of asynchronous iterations over all possible iterate trajectories or an estimate thereof.

2.3.2 Averages over Sets of Iteration Matrices

Given a finite set $\mathcal{A}$ of matrices, we define the *average number of iterations until convergence over* $\mathcal{A}$ as

$$\bar{N}_{it}(\mathcal{A}) = \sum_{A \in \mathcal{A}} N_{it}(A) W(A) \tag{6}$$

$W(A)$ is a weight associated with matrix A, such that $\sum_{A \in \mathcal{A}} W(A) = 1$. In this definition, it is assumed that $\mathcal{A} \subseteq \mathcal{C}$, where $\mathcal{C}$ is the set of all iteration matrices that guarantee convergence.

This definition provides an accurate measure for comparing two iterative methods, convergent over domains $\mathcal{C}_1$, respectively $\mathcal{C}_2$, in the presence of a known workload distribution modeled by $W(A)$ over a set $\mathcal{A} \subseteq \mathcal{C}_1 \cap \mathcal{C}_2$. This workload distribution is affected by various characteristics of the application such as system sparsity and the method of solution including acceleration techniques aimed at reducing the spectral radius of the iteration matrix.

Needless to say, finding the exact value of $\bar{N}_{it}(\mathcal{A})$ when $\mathcal{A}$ is a continuum is analytically intractable and therefore the space of matrices $\mathcal{A}$ must be discretized.

The average asynchronous speedup over a set of iteration matrices is also given by:

$$\mathcal{S}_1 = \sum_{A \in \mathcal{A}} \frac{N_{it}^{sync}(A)}{N_{it}^{async}(A)} W(A) \tag{7}$$

The set of matrices $\mathcal{A}$ in this paper was derived by discretizing the space of 2×2 matrices of real numbers which yield convergence for both synchronous and asynchronous iterations. Let $\mathcal{C}$ be the set of 2-by-2 matrices A for which

$$\|A\|_\infty = \max_i (\sum_j |a_{ij}|) < 1$$

Thus, convergence is guaranteed for both the synchronous and asynchronous iterations with iteration matrix in $\mathcal{C}$. $\mathcal{C}$ is discretized into $\mathcal{C}_\sharp$ by selecting the elements found at the intersection with a four-dimensional non-uniform grid.

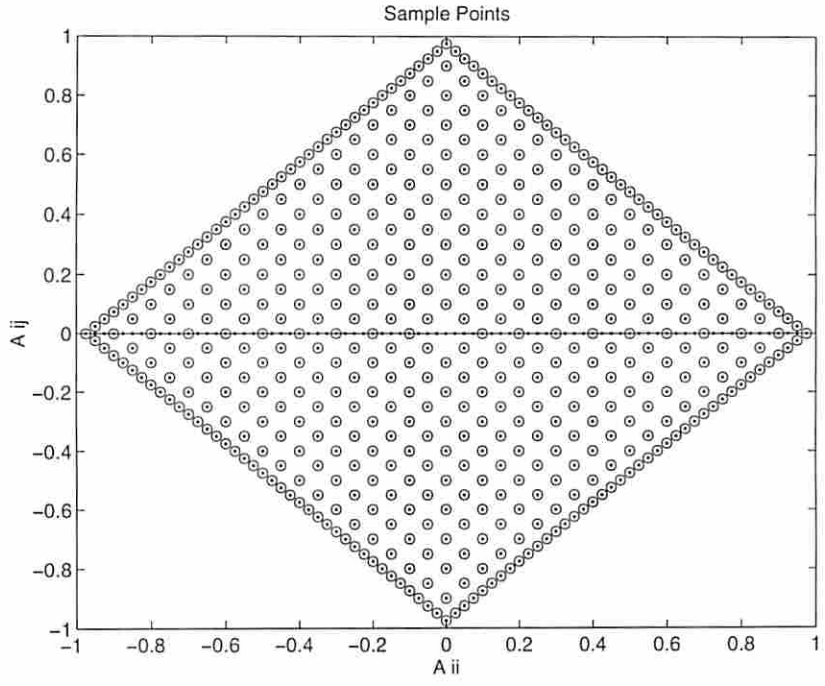

Figure 2: The set of values taken by (a_{00}, a_{01}) or (a_{10}, a_{11}).

A two dimensional section through $\mathcal{C}$ obtained by fixing the coefficients of a row of matrix A is displayed in Figure 2. This section contains 516 points, corresponding to all the possible values for the coefficient of the other row. The total number of elements in $\mathcal{C}_\sharp$ is 516x516=266,256. As can be observed, we have selected more sample points towards the edges of the domain because the behavior of asynchronous algorithms changes much faster in these areas and finer grain sampling provides better estimations. Of course, the weights $W(A)$ in the definitions of the average speedups take into account the non-uniform sampling of the matrix space.

3 Simulation Models

In this section we report on the results of extensive simulations of the synchronous and asynchronous executions of a linear iterative process with two variables on a multiprocessor

system. Given $A = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix}$, the linear iteration

$$X_{k+1} = \begin{pmatrix} x_0^{k+1} \\ x_1^{k+1} \end{pmatrix} = AX_k = \begin{pmatrix} a_{00}x_0^k + a_{01}x_1^k \\ a_{10}x_0^k + a_{11}x_1^k \end{pmatrix} \tag{8}$$

is assigned for execution on a 2-processor system, so that each processor is in charge of one iteration variable.

We have simulated three different models for the asynchronous execution of this iterative process. In the following subsections we describe these models, together with our results and interpretations. Throughout these experiments, the synchronous iteration is of Jacobi type and the number of synchronous iterations is the same for all three models given the initial condition, the iteration matrix and the convergence condition.

3.1 First Simulation Model

Our first model is characterized by instantaneous communications and random execution times in each iteration.

3.1.1 Asynchronous Computation Model

Under this model, each processor repeatedly executes a basic cycle which consists of fetching the current value of both variables, of computing the value of its assigned component and of updating the component. The update is immediately visible at the other processor, since variables are shared and store latency is neglected. Real time is kept for each processor P_i and is incremented after every cycle k by the corresponding amount of time spent in the computation, $t_{it}^{P_i}(k)$. A processor is free to start a new cycle immediately after finishing the current one without waiting on the other processor.

The computation time per iteration is generated according to

$$t_{it}^{P_i}(k) = t_{const} + t_{rand}^{P_i}(k)$$

where t_{rand} is a random fluctuation. It is generated independently for each processor from a uniform distribution in the interval $[0, fluct_{max}]$.

Processors repeat the execution of the basic cycle until the detection of the termination condition:

$$\|x_k\|_2 = \sqrt{x_0^2 + x_1^2} < \varepsilon = 10^{-4}$$

at which time, the total number of iterations is recorded. In the synchronous case, the iteration is executed only once. In the asynchronous case, the simulation is repeated 100 times with different sequences of fluctuation components, and the numbers of asynchronous iterations are averaged over the 100 simulation runs.

3.1.2 Observations

Even for two iterate components, the direct visualization of the speedups is not possible, since the display space has five dimensions. We can observe three-dimensional sections of this space by fixing two coefficients of the iteration matrix A.

In Figure 3 the asynchronous speedup is displayed for all matrices with $a_{10} = 0.95$ and $a_{11} = -0.025$. Although the asynchronous speedup is slightly less than 1 for most of the points in the plot, it soars to tens and hundreds (the grid for matrix discretization is not fine enough to show these extreme speedups), generating a thin peak which culminates around $a_{00} = -0.85, a_{01} = 0.125$. Towards the edge of the convergence domain, as $\rho(A) \nearrow 1$, the value of the asynchronous speedup becomes unbounded.

We now examine more closely why the asynchronous iteration converges much faster

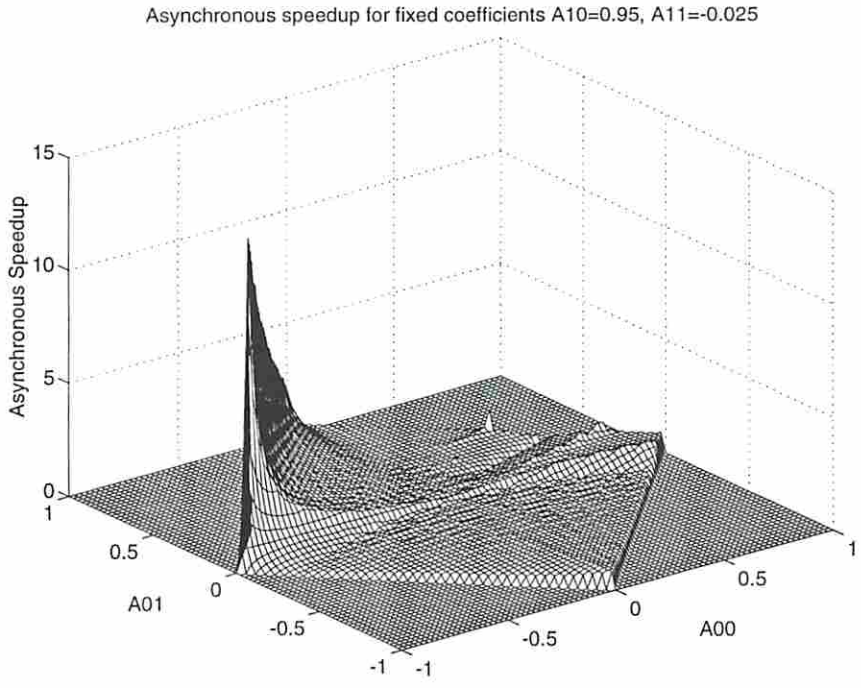

Figure 3: 3-D visualization of the asynchronous speedup for $t_{const} \gg fluct_{max}$.

than the synchronous iteration for the particular case of matrix $A = \begin{pmatrix} -0.83 & 0.16 \\ 0.974 & -0.025 \end{pmatrix}$ and $fluct_{max} = 2$. This case exhibits an asynchronous speedup of about 60 and the matrix is just a bit off the sectioning grid used to display the speedup in Figure 3.

The trajectories followed by the iteration vector for synchronous and asynchronous executions are displayed in Figure 4 and the sequence of values for the asynchronous iteration is listed in Table 1.

The trajectory of the Jacobi synchronous iteration (solid line) exhibits very slow convergence in an oscillatory regime. Each transition in this trajectory corresponds to one iteration. The spectral radius $\rho(A)$ is 0.9913, which explains the slow convergence, and the dominant eigenvalue for A is -0.9913, causing the oscillations as explained in Section 2.2.

The trajectory of an asynchronous iteration (dash-dotted line) shows the next position of the iterate vector whenever anyone of its two components is updated. Therefore each transi-

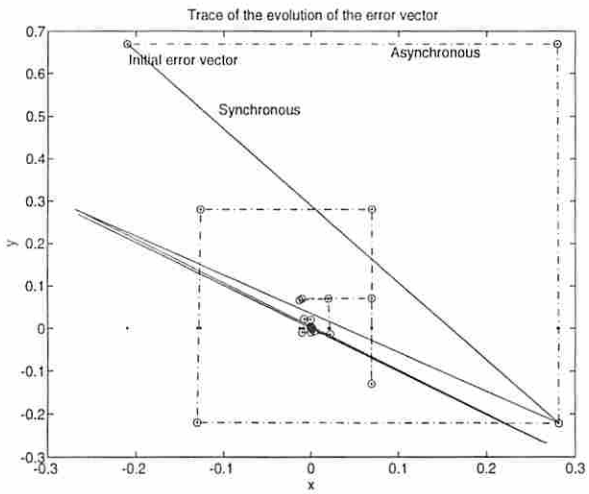

Figure 4: Trajectories of the iteration vector.

tion in this trajectory is equivalent to one half of a synchronous iteration. The asynchronous trajectory is a closing spiral and the computation more or less follows a Gauss-Seidel model in which updates of x_1 follow updates of x_0, as can also be seen from Table 1. In this regime, which holds for the first six transitions, the iteration is described by:

$$x_0^{k+1} = a_{00}x_0^k + a_{01}x_1^{k-1}$$

$$x_1^{k+1} = a_{10}x_0^k + a_{11}x_1^k$$

We can re-write these relations by augmenting the iterate vector:

$$\begin{pmatrix} x_0^{k+1} \\ x_1^{k+1} \\ x_1^k \end{pmatrix} = \begin{pmatrix} a_{00} & 0 & a_{01} \\ a_{10} & a_{11} & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_0^k \\ x_1^k \\ x_1^{k-1} \end{pmatrix}$$

The spectral radius for the iteration matrix in this last relation is 0.6663, considerably smaller than $\rho(A)$, which explains the faster convergence. The asynchronous speedup in this phase

Updated	x_0	x_1
	-0.21000	0.670000
x_0	0.281500	0.670000
x_1	0.281500	-0.221290
x_0	-0.126445	-0.221290
x_1	-0.126445	0.279713
x_0	0.069543	0.279713
x_1	0.069543	-0.130150
x_1	0.069543	0.070989
x_0	-0.012967	0.070989
x_1	-0.012967	0.065960
x_0	0.022120	0.065960
x_1	0.022120	-0.014278
x_0	-0.007806	-0.014278
x_1	-0.007806	0.021902
x_0	0.004195	0.021902
x_1	0.004195	-0.008151
x_0	0.000023	-0.008151

Table 1: Sequence of asynchronous updates of the iterate vector.

can be computed as $\log(0.6663)/\log(0.9913) = 46.46$. However, the observed asynchronous speedup for the overall iterative process is higher. Table 1 shows that the seventh step breaks the strict alternation of computations and recomputes x_1. As clearly visible in Figure 4, this break in the pattern of updates becomes the starting point of a new spiral closing in much closer to the solution. Had the initial Gauss-Seidel evolution been strictly followed, several more transitions would have been needed to achieve what this "change of step" did by pure hazard. Several such occurrences can further cut down on the total number of iterations, leading to even higher speedups. However, hazard is not always so generous. Sometimes, the steady Gauss-Seidel evolution is slowed down by the occurrence of such changes in the sequence of computations. Over a large number of trials, it all evens out such that the speedup is largely attributable to the Gauss-Seidel effect.

The reverse effect is also observed and, in some cases, the average number of asynchronous

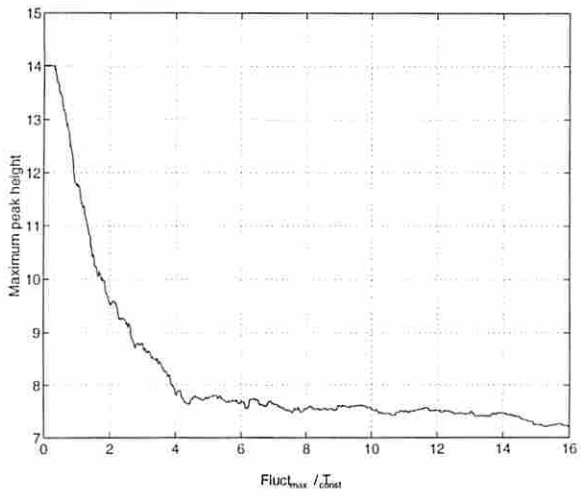

Figure 5: Maximum speedup on the sectioning grid of Figure 2 as a function of $fluct_{max}$.

iterations is increased. For example, when $A = \begin{pmatrix} -0.83 & -0.16 \\ 0.974 & -0.025 \end{pmatrix}$, $\rho(A) = 0.506$ and the synchronous execution converges very fast, but the spectral radius of the iteration matrix corresponding to the approximate Gauss-Seidel asynchronous execution is 0.9923, inducing slow convergence.

The height of the peak in Figure 3 depends on the value of $fluct_{max}$ relative to t_{const}. The peak value is largest for very small values of the fluctuations, in "near synchronous" conditions. As the fluctuation grows, the peak value of the asynchronous speedup decreases, eventually stabilizing when $fluct_{max} \gg t_{const}$, as shown in Figure 5. It appears that small fluctuations favor a stricter alternation of the computation of the two variables in a Gauss-Seidel sequence whereas, for larger fluctuations, some benefits of this effect are lost because of frequent changes in the order of evaluation.

Figure 6 plots the average asynchronous speedup, as defined by (7), for different magnitudes of the fluctuation. It shows that, for a randomly selected matrix, the number of iterations until convergence is, on the average, slightly less in the synchronous than in the asynchronous iteration. The speedup is also not very sensitive to the magnitude of the fluc-

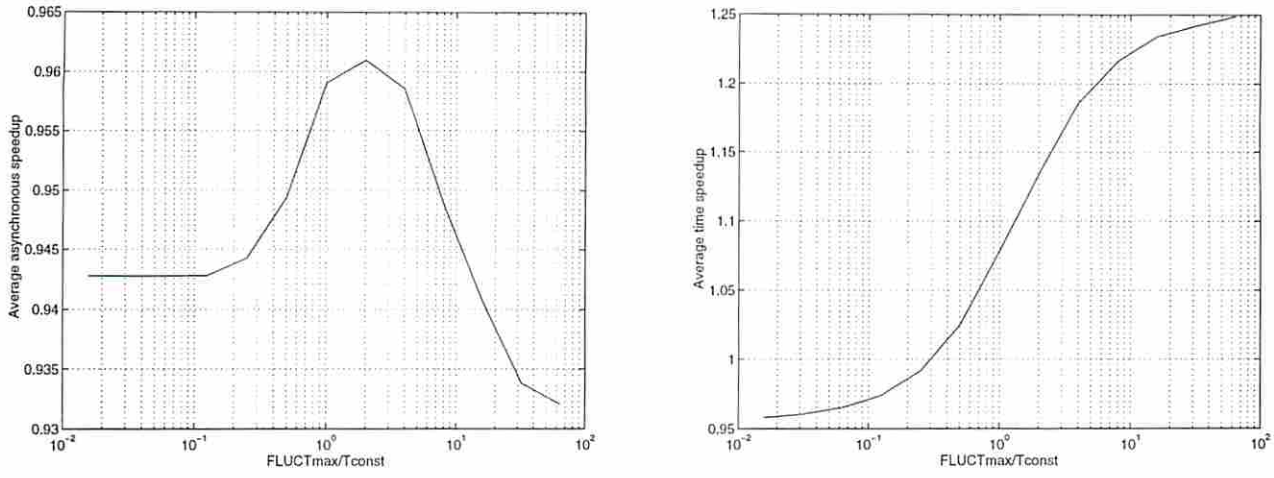

Figure 6: Average asynchronous speedups and time speedups as a function of $fluct_{max}$.

tuation. As observed before, the speedup peaks and dips tend to cancel each other when averaged over all matrices so that the average speedup does not change much for any size of the fluctuation.

Although we are not concerned in this study with the effects of synchronization on the asynchronous speedup, we include the time speedup in Figure 6 for reference. As expected, larger fluctuations impose a higher penalty on the synchronous execution leading to larger time speedups.

Overall, the synchronous iteration has better average asynchronous speedup. However, in some extreme cases it is very slow due to oscillations. One possible idea to avoid being trapped in a very slow converging process would be to start the iteration process synchronously and, if slow convergence is detected after a number of steps, to switch to asynchronous execution until convergence by skipping the synchronization points.

3.2 Second Simulation Model

Our first model restricted the order in which prior iterate values were used in current computations. Namely they were used in the same order as they were produced. By contrast, in this second model, a processor uses the most recent value of its assigned iterate component, but the values of the other iterate are selected randomly from the values produced by the other processor in a time window defined by a maximum delay. It is therefore possible to first use x_0^k and later on x_0^{k-1} in the computations of x_1, for example.

In another departure from the first model, delays are now bounded and the iterative process can be categorized as partially asynchronous.

3.2.1 Asynchronous Computation Model

In this model of asynchronous iterations, the computation times for every iterate component are constant and are each equal to one time unit. A new value is computed as per relation (8) with the modification that the value of x_i assigned to the other processor is randomly selected from a FIFO buffer B_i of size S_i holding the values of x_i produced in previous iterations. At the end of iteration k, the new value of x_i^k is stored into the buffer and, if $k \geq S_i$, $x_i^{k-S_i}$ is discarded. Initially, the buffers are filled with the values of the starting point. The size of the buffer B_i is related to the maximum delay that a computed component x_i can experience until its use: $D_i^{max} = S_i - 1$. The expected delay in this model given the uniform distribution for the random selection policy is $D_i^{max}/2$. We also tried a geometric distribution for the selection of the buffer entry and the results were similar.

The results, which we report next, have been collected with the same set of iteration matrices, the same initial condition and the same termination condition as in the first model. As before, the number of iterations for the asynchronous iteration is an average over 100 simulation runs.

3.2.2 Observations

Slices through the matrix space reveal the same speedup peaks as in our first model. The peaks are very similar in height and location in the matrix space. We do not repeat these observations here, but we concentrate on the average asynchronous speedup.

Figure 7 shows the average asynchronous speedup as a function of buffer sizes S_i. The first observation is that the synchronous approach performs better at any buffer size. This observation confirms the conclusion from the first simulation model in which the average asynchronous speedup was less than one for all values of the fluctuation. The second observation is that the larger the buffer and the average delay are, the slower the asynchronous iteration is. This is similar to the general result for monotone mappings stating that more frequent updates of the variables and smaller delays produce faster convergence [3]. Another observation is that for a buffer size of two and a maximum delay of one, the average asynchronous speedup is almost identical to the value obtained in the previous model for large fluctuations. This can be explained by the fact that, in the first model, at large fluctuations, the delays have a distribution that closely matches the distribution of the delays in the second model with a buffer size of two.

3.3 Third Simulation Model

This model approximates more closely the delays in a multiprocessors were communication times fluctuate and messages can be delivered out of order.

3.3.1 Asynchronous Computation Model

Each processor performs one iteration per time unit, updating its local variable as specified by relation (8). The value of the component assigned to the other processor is the most recently received one. Values can be received out of the order in which they are produced, according to the following model. At the end of iteration k, the newly computed value is

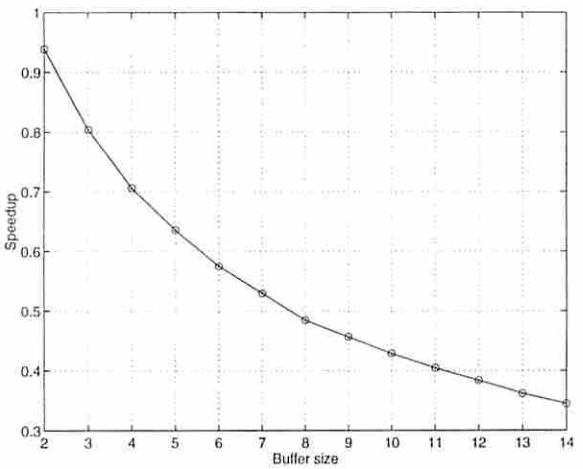

Figure 7: Average asynchronous speedup for the second simulation model.

assigned a reception time $t_{rec}(k) = k + 1 + \Delta$, where Δ is a discrete random variable over $0, 1, ..., \Delta_{max}$ and has a uniform distribution.

This model, just like the one before, does not enforce a FIFO property for the communication of the variables (FIFO communication implies that values must be used in the order in which they were produced.) However, because the most recent value received is always selected, the expected delay for a communicated variable is now reduced with respect to the previous model and better asynchronous speedups can be anticipated.

3.3.2 Observations

Figure 8 shows that the average asynchronous speedup for this model has the same characteristics as for the second simulation model. The asynchronous iterations perform slightly worse on the average than the synchronous ones and the longer the delays are, the slower the convergence is. We compare the two models with constant computation times and variable communication delays in the same figure. The two models lead to identical performance when the maximum delay is one, in which case the expected delay is the same and the variables are always used in the order in which they are produced. For larger maximum

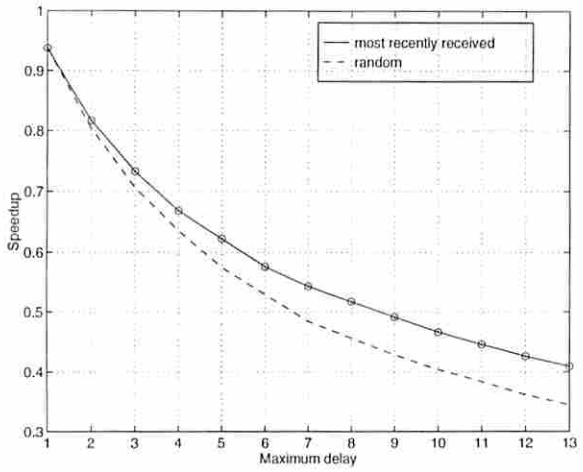

Figure 8: Average asynchronous speedup for the third simulation model

delays, the model in which the most recently received value is selected yields better average speedups.

4 Analytical Model with Stochastic Delays

In [2] a model of asynchronous linear iterations with stochastic delays was developed and sufficient conditions for convergence were derived. The model fits the case of variable load conditions of the communication network and, by including some knowledge about the behavior of the asynchronous environment in the form of probability transition matrices $\tilde{P}_{ji}$, the convergence conditions are relaxed with respect to the deterministic case. The key idea in the convergence analysis is *state augmentation.* A new iteration vector is constructed from the current components and the components of past iteration vectors produced in a time window corresponding to a maximum delay. The value of this maximum delay can in theory be different for each processor. Using the recurrence relations in the augmented space, the authors demonstrate sufficient conditions for convergence in mean and for convergence in the second moment.

In the special and simple case where delays are independent it is shown in [2] that a sufficient conditions for convergence in the mean and for convergence in the second moment is:

$$\rho(\sum_{m=1}^{l} C_m p_m) < 1$$

C_m is a matrix describing a possible recurrence relation in the augmented space. If $B_{j,i}$ is the maximum delay to propagate a message from processor i to processor j, then there is one C_m matrix for each of the $l = \prod_{i,j}(B_{j,i} + 1)$ cases obtained by combining possible delays for the use of x_j in the computation of x_i. The probability for having C_m describe the computations performed at iteration k is p_m, independent of k and constant. Since the C_m matrices, which describe transformations in the larger, augmented space, are larger than the original iteration matrix A and grow with the sum of the maximum delays, the analysis is much more complex than for the synchronous case.

This result for convergence in the mean is sufficient for our performance evaluation purposes. The reason is that, in our analysis, the iteration matrices meet stronger, deterministic convergence conditions and we are only concerned with the rate of convergence. Hence, the asynchronous speedup for the asynchronous model with independent stochastic delays is (see Section 2):

$$\mathcal{S} = \frac{ln(\rho(\sum_{m=1}^{l} C_m p_m))}{ln(\rho(A))}$$

4.1 Model

We consider an asynchronous environment with stochastic delays. The communication delays are sequences of independent, identically distributed random variables. The probability that a value is communicated to processor 1 (respectively processor 2) with a delay of zero or one is p_1 and $1 - p_1$ (respectively p_2 and $1 - p_2$).

Taking the expected values of both sides of relation (8), we obtain the the following

relations for the expected asynchronous trajectory:

$$\bar{x}_0^{n+1} = a_{11}\bar{x}_0^n + a_{12}(p_1\bar{x}_1^n + (1-p_1)\bar{x}_1^{n-1})$$

$$\bar{x}_1^{n+1} = a_{22}\bar{x}_1^n + a_{21}(p_2\bar{x}_0^n + (1-p_2)\bar{x}_0^{n-1})$$

Using the space augmentation technique, the following recurrence relation is obtained:

$$\bar{X}_{n+1} = \begin{pmatrix} \bar{x}_0^{n+1} \\ \bar{x}_0^n \\ \bar{x}_1^{n+1} \\ \bar{x}_1^n \end{pmatrix} = \begin{pmatrix} a_{11} & 0 & p_1 a_{12} & (1-p_1)a_{12} \\ 1 & 0 & 0 & 0 \\ p_2 a_{21} & (1-p_2)a_{21} & a_{22} & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \bar{x}_0^n \\ \bar{x}_0^{n-1} \\ \bar{x}_1^n \\ \bar{x}_1^{n-1} \end{pmatrix} = A_{aug}\bar{X}_n$$

For given p_1 and p_2, the asynchronous speedup is given by

$$\frac{ln(\rho(A_{aug}))}{ln(\rho(A))}$$

4.2 Average Asynchronous Speedup

Therefore, to estimate the asynchronous speedup, we simply have to compute the augmented matrix and its spectral radius. To see the effect of the delay probabilities, we have computed the average asynchronous speedup over all matrices in $\mathcal{C}_\sharp$ and for each possible combination of p_1 and p_2 between 0 and 1, with a resolution of 0.1. The results are plotted in Figure 9. Small amounts of asynchronism can produce slight improvements of up to 10%, over the synchronous execution. As the preference for older iterate values is increased (i.e., p_1 and p_2 are close to zero), the performance quickly degrades by as much as 30%. When $p_1 = p_2 = 0.5$, the expected speedup is 0.9895, matching the results obtained through simulation and given in Section 3.

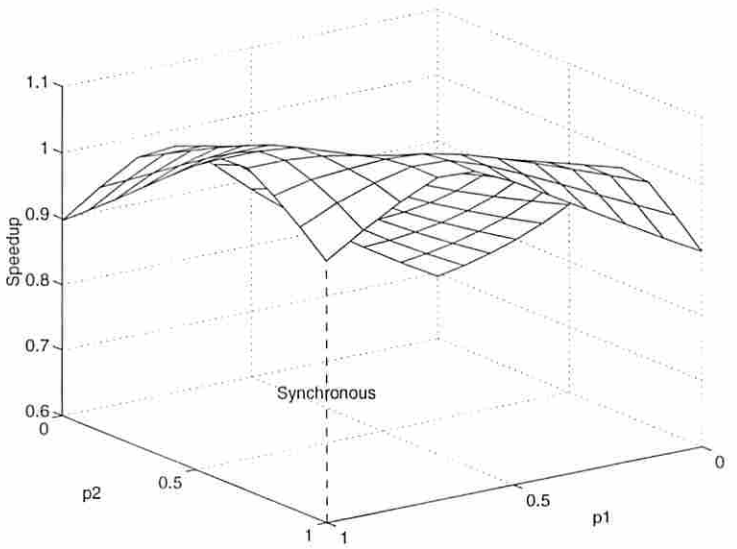

Figure 9: Expected asynchronous speedup as a function of the delay probabilities.

4.3 Average Maximum Asynchronous Speedup

We have observed that for a given iteration matrix the asynchronous speedup depends on the probability distribution of the delays. In particular, for each matrix A, there is a distribution of delays that maximizes the asynchronous speedup. To try to evaluate the best possible average asynchronous speedup achievable by asynchronism, we have chosen the most favorable execution environment for the asynchronous iterations. Optimal probabilities p_1 and p_2 were selected for each iteration matrix A by minimizing the spectral radius $\rho(A_{aug})$. This minimization is approximate since p_1 and p_2 are discretized in [0,1]. This "minimized" spectral radius is used in the computation of the asynchronous speedup. It is possible (in fact, quite frequent) that the optimal "asynchronous" evolution corresponds to the synchronous one ($p_1 = p_2 = 1$), and that the speedup is 1.

In the presentation of these results, we correlate the maximum achievable asynchronous speedup with the spectral radius of the iteration matrix. Figure 10 presents the plot of the maximum asynchronous speedup averaged over all the matrices A with a spectral radius $\rho(A)$

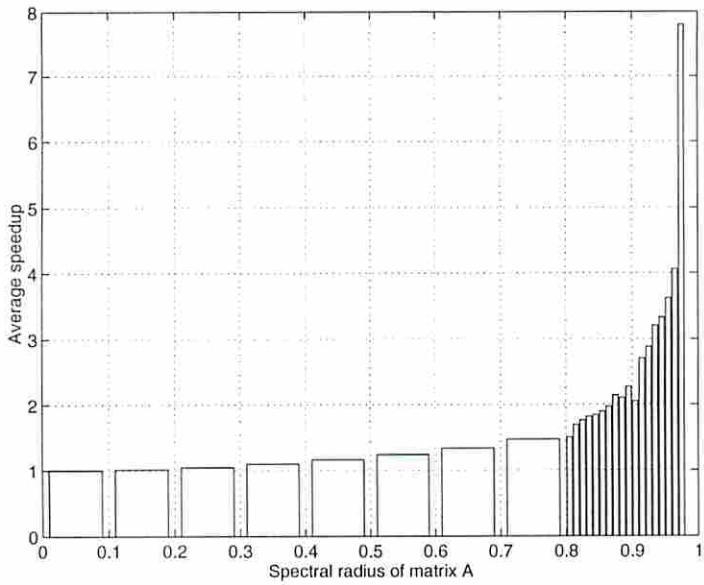

Figure 10: Maximum asynchronous speedups sorted by values of $\rho(A)$.

contained in a certain interval. Our approximation shows that cases of reasonable expected speedup require $\rho(A) \geq 0.9$.

Figure 11 shows a histogram of the maximum speedups sorted by their magnitude. As expected, only a small fraction of matrices yield asynchronous speedups of at least one order of magnitude.

The average maximum asynchronous speedup observed over the entire set of matrices is 1.381. This result indicates that asynchronous execution is not worth considering in the general case, even when the environment is "customized" to favor asynchronism. Unless synchronous convergence is very slow, i.e. the spectral radius of the iteration matrix is close to 1, significant improvements in the convergence rate are not to be expected in asynchronous iterations.

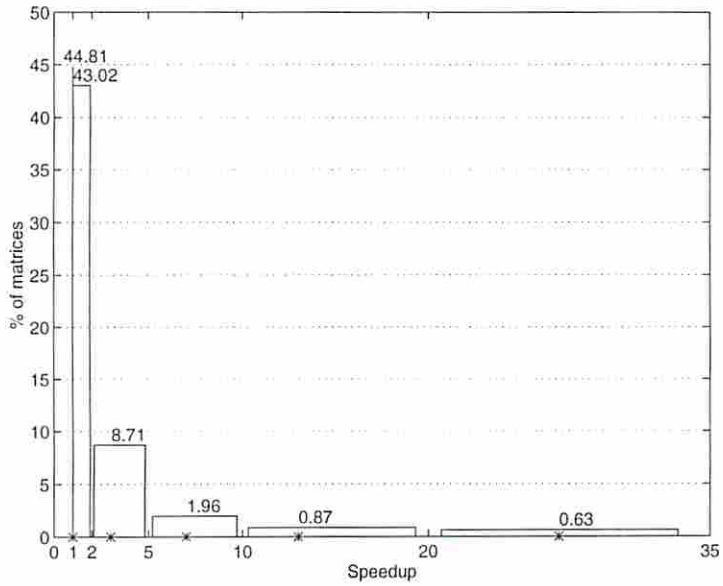

Figure 11: Histogram of the maximum asynchronous speedup sorted by magnitude.

5 Relation with Convergence Acceleration Techniques

There are two classes of approaches to improve the convergence rate of linear iterations: semi-iterative methods, such as polynomial and non-polynomial acceleration, and conjugate gradient methods. Semi-iterative methods attempt to minimize the spectral radius of the iteration matrix whereas conjugate gradient methods transform the problem into a minimization problem.

These methods share a common feature with asynchronous iterations. They compute the current iterate by employing information contained not only in the most recent values of the iteration vector, but in the values obtained in past iterations. However, asynchronous iterations do this in an uncontrolled, random manner. Moreover, they freely inter-mix values from different past iterations, thus using finer "ingredients" than "planned" acceleration techniques, which only use values from the same past iteration. This feature expands the search space for optimizing the speed of convergence and therefore may lead to better con-

vergence rates of the asynchronous iteration over the optimized, synchronous one. Of course, one could certainly devise synchronous acceleration techniques using values from different past iterations, but the computation of each new iterate value would be very complex. In stochastic models of asynchronous iterations, loose restrictions, such as maximum delay bounds and delay probabilities, which are imposed on the construction space, limit this complexity.

We have already seen in Section 4 an example of convergence rate optimization in asynchronous iterations, when selecting the delay probabilities $p_{1,2}$ such that the spectral radius of the iteration matrix in the augmented space is minimized. Generalizing this simple example, we can generate a sequence of augmented space matrices, which are then used adaptively in the iterative process:

$$P = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} p_1 & 1-p_1 \\ p_2 & 1-p_2 \end{pmatrix} \rightarrow \begin{pmatrix} p_1 & q_1 & 1-p_1-q_1 \\ p_2 & q_2 & 1-p_2-q_2 \end{pmatrix} \rightarrow \cdots$$

where P is a stochastic matrix. As P expands, the dimension of the augmented space and of A_{aug} increases. By adding one more set of values from past iterations in the computation, the spectral properties of the augmented matrix can be further refined, further contracting the spectral radius. It is obvious that the expansion operation cannot worsen the spectral properties because, when "expanding" P with a column of zeros, we preserve the spectrum of P. Our experiments with a few cases show that the newly added coefficients to P vanish and that a delay bound of 3 or 4 is sufficient to reap all the possible improvement.

This technique is similar to semi-iterative acceleration methods, albeit more general. Polynomial acceleration attempts to enhance the convergence of the basic iterative method by creating an additional (and faster-converging) iterative sequence y_k, the elements of which are built in the Krylov subspace [9] created by the entire sequence of x-iterates, rather than

using just the most recent x-iterate, as basic methods do.

$$y_k = \sum_{i=0}^{k} \alpha_{k,i} x_i$$

with the restriction

$$\sum_{i=0}^{k} \alpha_{k,i} = 1$$

to guarantee the consistency property. This leads to the creation of a family of polynomials Q_k defined as

$$Q_k(x) = \alpha_{k,0} + \alpha_{k,1} x + \ldots + \alpha_{k,k} x^k$$

such that

$$\varepsilon_k = Q_k(A)\varepsilon_0$$

Acceleration techniques aim at controlling ε_k and most often resort to minimizing the spectral radius of $Q_k(A)$ by employing special polynomials, such as Chebyshev polynomials [7]. These polynomials can be described recursively and recurrence relations between the vectors y_k can be computed, thus eliminating the need to store all the values of x_k. Trying to apply this approach to asynchronous algorithm is much more complicated and we do not have a formal analysis for the general case as yet.

We now show that, under certain probabilistic assumptions about the execution rate of the processors, asynchronous iteration methods lead to convergence rate enhancements similar to extrapolation techniques applicable to synchronous iterations. Assume that the computation events are ordered in time and that, at every t_k corresponding to an event, there is a probability p_i that processor P_i, $i = 0, 1$ updates its component. Naturally, $1 \leq p_0 + p_1 \leq 2$. This model closely match the execution model used in the first simulation experiments when $fluct_{max}$ is very large with respect to t_{const}. (When t_{const} is not negligible, a Markovian model would be more appropriate.) At every t_k, one of the following transformations of the

iteration vectors is applicable:

$$\begin{pmatrix} 1 & 0 \\ a_{10} & a_{11} \end{pmatrix}, \begin{pmatrix} a_{00} & a_{01} \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix}$$

with probabilities $1 - p_0, 1 - p_1$ and $p_0 + p_1 - 1$, respectively. After simple calculations, the expected value of the iteration vector is given by

$$X_{k+1} = ((I - P) + PA)X_k$$

where $P = \begin{pmatrix} p_0 & 0 \\ 0 & p_1 \end{pmatrix}$.

This relation generalizes the relation obtained under extrapolation techniques, which use an optimum scalar λ to minimize the spectral radius of a symmetric and positive definite matrix A through the transformation $(1-\lambda)I+\lambda A$. As a note, the transformation involving P does not preserve the eigenvectors of A, contrary to extrapolation and polynomial acceleration techniques. It is not known, as yet, whether this refinement can bring an added benefit in the quest for a transformation of the original iteration matrix A with a smaller spectral radius.

6 Conclusions

We have performed a systematic investigation of the effects of asynchronism on the convergence rate of linear iterations of size two. The performance metrics used throughout was the asynchronous speedup, the ratio between the number of iterations in the synchronous and asynchronous algorithms. The set of matrices employed in our evaluation was obtained through non-uniform sampling of the space where both synchronous and asynchronous convergence conditions are met. Given our sampling resolution, the set of iteration matrices

contains over a quarter million matrices. Three asynchronous execution models have been simulated. One of the models uses fluctuating computation time per iteration and instantaneous communication. The other two models use fixed computation time and random delays for the communication of the results. A fourth model, using stochastic delays, is investigated analytically.

Our results show that asynchronous linear iterations have the power to produce occasionally a very rapid convergence. This is attributed to an oscillation phenomenon present in the synchronous execution, especially when the iteration matrix has a dominant eigenvalue in the neighborhood of -1. This oscillation is damped in the presence of asynchronism. However, such instances are not very frequent and well-established acceleration techniques can perform similarly. We leave it as an open question whether asynchronous execution can provide significant and systematic means of speedup, beyond what acceleration techniques based on linear combinations currently provide. On the other hand, it is encouraging to note that asynchronous iterations do not perform much worse than synchronous iterations even in the worst case. With the added benefit of synchronization removal, they might fill the gap for the worst case and lead to consistently outperforming of synchronous methods. However, high average gains are not to be expected.

A interesting topic of research would be to search for similar oscillations in non-linear iterations, for which no good acceleration technique exists in general.

References

[1] Gerard M. Baudet. Asynchronous iterative methods for multiprocessors. *Journal of the ACM*, 25(2):226–244, April 1978.

[2] Bassem F. Beidas and George P. Papavassilopoulos. Convergence analysis of asynchronous linear iterations with stochastic delays. *Parallel Computing*, 19:281–302, March 1993.

[3] Dimitri P. Bertsekas and John N. Tsitsiklis. Convergence rate and termination of asynchronous iterative algorithms. In *Proceedings, International Conference on Supercomputing, Crete, Greece*, pages 461–470, June 1989.

[4] Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and distributed computation.* Prentice Hall, 1989.

[5] J.M. Bull and T.L. Freeman. Numerical performance of an asynchronous Jacobi iteration. In L. Bouge et al., editors, *Parallel Processing: CONPAR 92-VAPP V*, pages 361–366. Springer-Verlag, 1992.

[6] G.H. Golub and C.H. Van Loan. *Matrix Computations.* Johns Hopkins University Press, 1989.

[7] W. Hackbusch. *Iterative Solution of Large Sparse Systems of Equations.* Springer-Verlag, 1994.

[8] L.A. Hageman and D.M. Young. *Applied Iterative Methods.* Academic Press, 1981.

[9] O. Nevanlinna. *Convergence of Iterations for Linear Equations.* Birkhäuser Verlag, 1993.

[10] O. Nevanlinna. How fast can iterative methods be? In G. Golub et al., editors, *Recent advances in iterative methods*, pages 135–147. Springer-Verlag, 1994.

[11] J. Nieplocha, T.Z. Mai, and C.C. Carroll. Asynchronous algorithms for solving large systems of linear equations on parallel computers. In W. Joosen and E. Milgrom, editors, *Parallel Computing: From Theory to Sound Practice*, pages 76–79. IOS Press, Amsterdam, Netherlands, 1992.