# ASYNCHRONOUS PARALLEL SUCCESSIVE OVERRELAXATION FOR THE SYMMETRIC LINEAR COMPLEMENTARITY PROBLEM

by

R. De Leone & O. L. Mangasarian

Computer Sciences Technical Report #755

February 1988

# Asynchronous Parallel Successive Overrelaxation for the Symmetric Linear Complementarity Problem[1)]

**R. De Leone & O. L. Mangasarian**

Computer Sciences Department
University of Wisconsin
Madison, Wisconsin 53706

Technical Report #755

February 1988

**Abstract.** Convergence is established for asynchronous parallel successive overrelaxation (SOR) algorithms for the symmetric linear complementarity problem. For the case of a strictly diagonally dominant matrix convergence is achieved for a relaxation factor interval of $(0, 2]$ with line search, and $(0, 1]$ without line search. Computational tests on the Sequent Symmetry S81 multiprocessor give speedup efficiency in the 43%-91% range for the cases for which convergence is established. The tests also show superiority of the asynchronous SOR algorithms over their synchronous counterparts.

**Key Words:** Linear complementarity, parallel algorithms, asynchronous algorithm, successive overrelaxation

**Abbreviated Title:** Asynchronous Parallel SOR for LCP

---

[1)] This material is based on research supported by National Science Foundation Grants DCR-8420963 and DCR-8521228 and Air Force Office of Scientific Research Grant AFOSR-86-0172.

## 1. Introduction

Successive overrelaxation algorithms have been proposed for solving the symmetric linear complementarity problem and linear programming problems [5, 6, 3]. Although successive overrelaxation is intrinsically a serial algorithm, parallel versions were proposed in [7, 8, 3] for the solution of these problems. Inherent in these parallel SOR versions, originally developed for loosely-coupled processors [4], was the synchronization of the various parallel processors. Thus typically each processor was assigned a fixed portion of the problem data, and once a processor completed its computation it awaited all other processors to finish their computations as well, whereupon all results were shared among them all and the process was repeated. By contrast in the asynchronous SOR proposed here the problem data may be assigned initially to each processor (static asynchronous SOR) or dynamically selected by each processor (dynamic asynchronous SOR) and, just as importantly, the results of each processor are broadcast instantly to all other processors as soon as they are obtained. This is easily done with tightly coupled multiprocessors such as the Sequent Symmetry S81 machine [10] that is employed here. Obviously this immediate sharing of newly computed results as well as the balanced allocation of data among processors (either by a dynamic SOR or a balanced static SOR) makes much more efficient use of the parallelism of a multiprocessor by reducing idle time of each processor and by exploiting instantaneously newly computed results. We describe now the essential ideas of the proposed asynchronous SOR algorithms.

Our concern here is the n-dimensional symmetric linear complementarity problem (2.1) which under mild practical assumptions is equivalent to the problem (2.2) of minimizing a quadratic function on the nonnegative orthant [5]. The significance of this problem stems in part from the fact that linear programs can be reduced to such a problem [6]. The asynchronous SOR approach consists of letting each of the $k$ processors solve simultaneously a linear complementarity problem in one variable, with each processor broadcasting its newly computed optimal variable value as soon as it is computed and proceeding to the solution of another one-dimensional linear complementarity problem without waiting for any of the other processors to finish but using the latest values for all variables. This is continued until all $n$ variables are

updated one or more times, at which time all processors are synchronized and the process is repeated.

We briefly summarize now the results of the paper. In Section 2 we give the convergence results for the various asynchronous parallel SOR algorithms that are defined in that section. The basic Asynchronous SOR Algorithm 2.1 contains an arbitrary permutation matrix which can be adjusted to result in a synchronous algorithm, or either a dynamic asynchronous algorithm (problem data automatically distributed among processors according to their relative speeds and data density) or a static asynchronous algorithm (problem data distributed a priori in any desired fashion but typically to achieve load-balancing). Theorem 2.6 establishes convergence of the ASOR Algorithm 2.1 under a relaxation factor $\omega$ interval which is an attenuated (0, 2) interval. If the matrix $M$ of the quadratic objective function is strictly diagonally dominant, then the relaxation factor $\omega$ is (0, 1] (Theorem 2.7). Another attenuated (0, 2) interval for $\omega$ is given in Theorem 2.8 for a modified synchronous parallel SOR algorithm. In Algorithm 2.9, a line search is added to Algorithm 2.1 and Theorem 2.10 gives a larger convergence interval for $\omega$ than that of Theorem 2.6 for Algorithm 2.1 without linear search. When $M$ is strictly diagonally dominant, the convergence interval is (0, 2] for the line-search Algorithm 2.9 as shown by Corollary 2.11.

Section 3 contains numerical test results carried on the Sequent Symmetry S81 multiprocessor [10]. These results indicate that asynchronous SOR algorithms are superior to synchronous ones.

We briefly describe our notation now. For a vector $z$ in the n-dimensional real space $R^n$, $z_+$ will denote the vector with components $(z_+)_i = \max\{z_i, 0\}$, $i = 0, 1, \ldots, n$. The scalar product of two vectors $x$ and $y$ in $R^n$ will be denoted by $xy$. The 2-norm $(zz)^{\frac{1}{2}}$ of a vector $z$ in $R^n$, will be denoted by $\|z\|$. $R^n_+$ will denote the nonnegative orthant or the set of points in $R^n$ with nonnegative components, while $R^{m\times n}$ will denote the set of $m \times n$ real matrices. For $A \in R^{m\times n}$, $A^T$ will denote the transpose, $A_i$ will denote the ith row, $A_{ij}$ the element in row $i$ and column $j$. Throughout this work $P$ and $P^i$ will denote a permutation matrix in $R^{n\times n}$ defined as a permutation of the rows of identity matrix $I \in R^{n\times n}$. It follows that $P^{-1} = P^T$ and $P^T$ is a permutation of the columns of the identity matrix $I$. For $M \in R^{n\times n}$, $PM$

is a permutation of the rows of $M$, $MP^T$ is a permutation of the columns of $M$ and $PMP^T$ is the same permutation of the rows and columns of $M$.

## 2. Asynchronous Parallel Successive Overrelaxation Algorithms

Our concern here is the symmetric linear complementarity problem

$$Mz + q \geq 0,\ z \geq 0,\ z(Mz+q) = 0 \tag{2.1}$$

where $M$ is a given real symmetric matrix in $R^{n\times n}$ and $q$ is a given vector in $R^n$. Every local and global solution of the quadratic program

$$\min_{x\geq 0} f(z) := \min_{z\geq 0} \frac{1}{2} zMz + qz \tag{2.2}$$

satisfies (2.1), and when $M$ is positive semidefinite every solution of (2.1) is a global solution of (2.2).

It is easy to verify directly (or see [5]) that $z$ is a solution of (2.1) if and only if it satisfies

$$z = \big(z - \omega E(Mz+q)\big)_+ \tag{2.3}$$

for some $\omega > 0$ and some positive diagonal matrix $E$. This simple relationship can be used as the basis of our asynchronous successive overrelaxation algorithm. In fact, if equation (2.3) is interpreted component-wise, then our asynchronous SOR algorithm can be described as each of the $k$ processors solving simultaneously a component of (2.3) for a new value of the $z$ component, broadcasting the new value of the $z$ component when finished and moving on to the next unmodified component of $z$. We state now this algorithm more precisely.

**2.1 Asynchronous SOR Algorithm (ASOR)** Let $y^0 \in R^n_+$. For $i = 0, 1, 2, \ldots$ let

$$y^{i+1} = \Big(y^i - \omega E^i\big(N^i y^i + p^i + K^i(y^{i+1} - y^i)\big)\Big)_+ \tag{2.4}$$

where $P^i$ is an arbitrary permutation matrix in $R^{n\times n}$,

$$y^i = P^i z^i,\ N^i = P^i M P^{i^T},\ p^i = P^i q \tag{2.5}$$

$\omega > 0$, $\{E^i\}$ is a bounded sequence of positive diagonal matrices such that $E^i \geq \alpha I$ for some $\alpha > 0$, $K^i$ is an $n \times n$ matrix defined by

$$K_{rs}^i := \begin{cases} 0 & \text{if } y_r^{i+1} \text{ is computed before} \\ & \text{or while } y_s^{i+1} \text{ is computed} \\ N_{rs}^i & \text{if } y_r^{i+1} \text{ is computed after} \\ & y_s^{i+1} \text{ is computed} \end{cases} \tag{2.6}$$

$r,\ s = 1, 2, \ldots, n.$

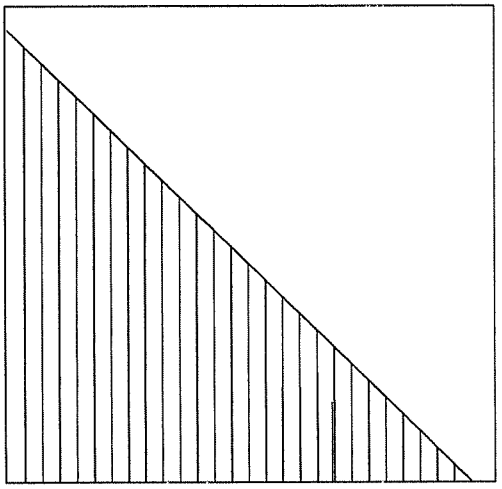
**2.2 Remark** The arbitrary permutation matrix $P^i$ in the ASOR Algorithm 2.1 may arise as follows during a typical asynchronous run on a machine with $k$ parallel processors. During iteration $i+1$ the computation of the first $k$ components $z_1^{i+1}, \ldots, z_k^{i+1}$ of $z^{i+1}$ is initially assigned to each of the $k$ processors. From then on whenever a processor finishes computing $z_j^{i+1}$, it proceeds to compute the component $z_r^{i+1}$ with lowest subscript $r$ which has not been computed yet during the current iteration $i+1$. The permutation $P^i$ is determined by ordering the components of $z^{i+1}$ computed by processor 1 in ascending component order, followed by each of the processors 2 through $k$.

**2.3 Remark** We note here that the asynchronous SOR iteration (2.4) subsumes a number of serial and synchronous parallel SOR algorithms by setting $P^i = I$ and setting $K^i$ as follows:

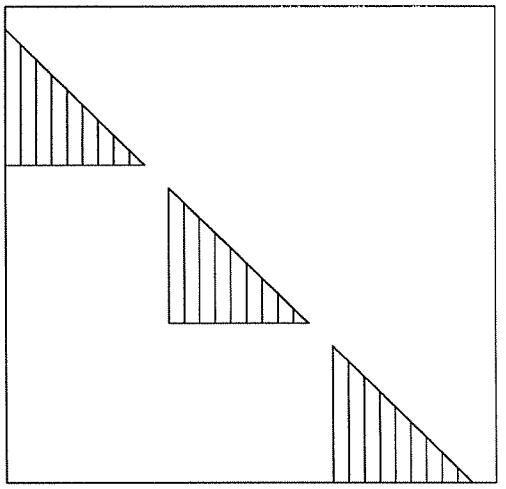
(a) When $K^i = L$, the strictly lower triangular part of $M$ (Figure 1a), we obtain the serial SOR algorithm of [5].

(b) When we set $K^i$ equal to the $k$ diagonal blocks of the strictly lower triangular parts of the $k$ block diagonals of $M$ (Figure lb) we obtain the synchronous parallel SOR of [7]. Convergence of this algorithm without line search was given in [7, Eqn (15)] for $\omega \in (0, \bar{\omega})$, where $\bar{\omega} \in (1, 2)$ for a strictly diagonally dominant matrix $M$. With line search convergence of the synchronous parallel SOR was given in [8] for $\omega \in (0, 2)$.

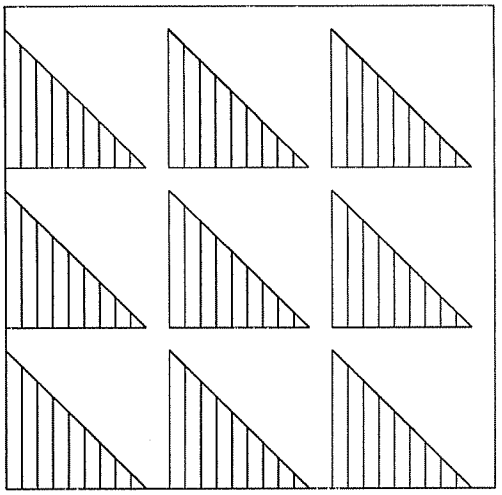
(c) When $K^i$ is made up of $k^2$ strictly lower triangular checkerboard blocks (Figure lc) we have a modified synchronous parallel SOR algorithm where the newly computed values of each of the $k$ components of the vector $z$ are broadcast to all $k$ processors before each processor updates the next component.



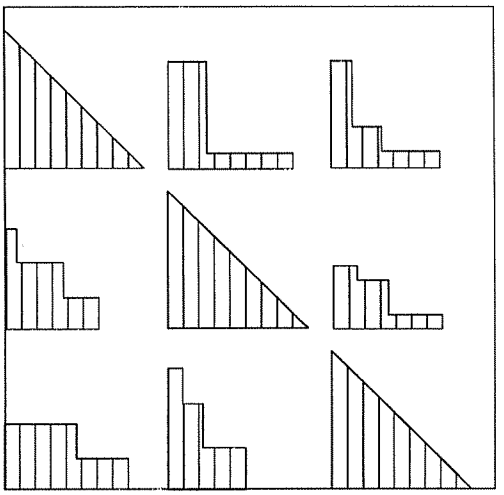
(1a) Serial SOR [5]



(1b) Synchronous Parallel SOR [7]



(1c) Modified Synchronous Parallel SOR (2.4)



(1d) Static & Dynamic Asynchronous Parallel SOR: SASOR & DASOR (2.4)

Figure 1: The substitution operator $K^i$ of the SOR Algorithm 2.1 with 3 processors. Shaded areas represent nonzero elements of the matrix $K^i$.

**2.4 Remark** When Algorithm 2.1 operates in an asynchronous fashion with $k$ parallel processors, the permutation $P^i$ may be either fixed or varied resulting in two classes of asynchronous algorithms as follows:

(a) **Static Asynchronous SOR (SASOR):** The permutation matrix $P^i$ is fixed in a predetermined fashion in order to balance the computing load among the parallel processors.

(b) **Dynamic Asynchronous SOR (DASOR):** The permutation matrix $P^i$ is determined adaptively as described in Remark 2.2 above. In either case, the substitution matrix $K^i$ is made up of $k^2$ blocks with only the $k$ diagonal blocks being strictly lower triangular, and the remaining blocks reflecting the propagation of the new computed values among the processors (Figure 1d).

**2.5 Remark** Since the matrix $K^i$ can be considered as a substitution operator in (2.4), substituting components of $y^{i+1}$ for corresponding components of $y^i$ in the term $N^i y^i$ in (2.4), consequently the "closer" is $K^i$ to $N^i - \left(\omega E^i\right)^{-1}$, the better is the iteration (2.4). For instance if $K^i = N^i - \left(\omega E^i\right)^{-1}$, then indeed it follows (Lemma 2.1 [5]) that $y^{i+1}$ solves the complementarity problem (2.1). Hence it seems reasonable to expect better results from the $K^i$ of Figures 1c and 1d than that of Figure 1b.

In order to establish convergence of the ASOR Algorithm 2.1 we shall invoke Theorem 2.1 of [5]. The only requirement of this theorem is that for some $\gamma > 0$ the following condition be satisfied

$$y\Big(\left(\omega E^i\right)^{-1} + K^i - \frac{N^i}{2}\Big)y \geq \gamma \left\|y\right\|^2 \forall i,\ \forall y \in R^n \tag{2.7}$$

We will now impose conditions on $\omega$ that will ensure the satisfaction of (2.7) and consequently convergence of the ASOR Algorithm.

**2.6 ASOR Convergence Theorem** Let $M$ have a positive diagonal $D$, let $\left(E^i\right)^{-1} = D^i$ where $D^i$ is the diagonal of $N^i$, let $K^i$ be as defined in (2.6) and let

$$0 < \omega < \min_{1 \leq r \leq n} \frac{2}{1 + \sum\limits_{s \neq r} \left|M_{rs}\right| / D_{rr}} \tag{2.8}$$

Then each accumulation point of the sequence $\left\{P^{i^T} y^i\right\}$ solves the linear complementarity problem (2.1) where $\left\{y^i\right\}$ is the sequence generated by the ASOR Algorithm 2.1.

**Proof** By Theorem 2.1 of [5] all we need to show is that (2.7) holds. Let $L^i$, $U^i$, $D^i$ $(L, U, D)$ be the strictly lower triangular, upper triangular and diagonal parts of $N^i$ $(M)$ respectively. There exist $\bar{L}^i \subset L^i$, $\bar{U}^i \subset U^i$, $\hat{L}^i \subset L$ and $\hat{U}^i \subset U$ such that for any $y$ in $R^n$

$$\begin{aligned} y\Big(\big(\omega E^i\big)^{-1} + K^i - \frac{N^i}{2}\Big)y &= \frac{1}{2}y\Big(\big(2\omega^{-1}-1\big)D^i + K^i + K^{i^T} - \big(L^i + U^i\big)\Big)y \\ &= \frac{1}{2}y\Big(\big(2\omega^{-1}-1\big)D^i - \big(\bar{L}^i + \bar{U}^i\big)\Big)y \\ &\quad \text{(Since } K^i_{rs}\,K^i_{sr} = 0 \text{ by (2.6) and} \\ &\quad\; K^i_{rs} = N^i_{rs} \text{ when } K^i_{rs} \neq 0\text{)} \\ &= \frac{1}{2}z\Big(\big(2\omega^{-1}-1\big)D - \big(\hat{L}^i + \hat{U}^i\big)\Big)z,\ z = P^{i^T}y \\ &\geq \gamma\|z\|^2 \text{ for some } \gamma > 0 \text{ (By (2.8))} \\ &= \gamma\|y\|^2. \quad \blacksquare \end{aligned}$$

If $M$ is (strictly) diagonally dominant, that is $D_{ii}(>) \geq \sum_{j\neq i} |M_{ij}|$, for $i = 1, \ldots, n$, then the inequality $(0 < \omega \leq 1)$ $0 < \omega < 1$ implies (2.8) and hence we have the following corollary.

**2.7 ASOR Convergence Theorem for (Strictly) Diagonally Dominant M** Theorem 2.6 holds with $(0 < \omega \leq 1)$ $0 < \omega < 1$ for the case when $M$ is (strictly) diagonally dominant.

We note that for the case of the Modified Synchronous Parallel SOR, where $K^i$ is constant and is given by Figure 1c and $P^i = I$, the term

$$\frac{1}{2}y\Big(\big(2\omega^{-1}-1\big)D^i + K^i + K^{i^T} - \big(L^i + U^i\big)\Big)y$$

in the proof of Theorem 2.6 is equal to

$$\frac{1}{2}\sum_{j=1}^{n} z_j\Big(\big(2\omega^{-1}-1\big)D_{jj}z_j - \sum_{r=1}^{k-1} M_{j(j+r\frac{n}{k})_*}z_r\Big)$$

where $(m)_*$ defines $m$ mod $n$, and for simplicity we have assumed that $n = \ell k$ for some integer $\ell$. Hence in this case, the condition (2.7) is satisfied if we require the strict diagonal dominance condition

$$\big(2\omega^{-1}-1\big)D_{jj} > \sum_{r=1}^{k-1} \big|M_{j(j+r\frac{n}{k})_*}\big| \quad j = 1, \ldots, n$$

or equivalently

$$0 < \omega < \min_{1 \le j \le n} \frac{2}{1 + \sum_{r=1}^{k-1} \left| M_{j(j+r\frac{n}{k})*} \right| / D_{jj}} \tag{2.9}$$

We thus have the following convergence theorem.

**2.8 Convergence Theorem for the Modified Synchronous Parallel SOR** Let $M$ have a positive diagonal $D$, let $E^i = D^{-1}$, let $K^i$ be defined by

$$K^i_{rs} := \begin{cases} M_{rs} & \text{for elements of } K^i \text{ in the strictly} \\ & \text{lower triangular parts of the } k^2, \\ & \text{square sub-blocks of } K^i \text{ (Figure 1c),} \\ 0 & \text{otherwise,} \end{cases} \tag{2.10}$$

let $k \ge 2$and let $\omega$ satisfy (2.9). Then each accumulation point of the sequence $\{y^i\}$ of (2.4) solves the linear complementarity problem (2.1). If $M$ is (strictly) diagonally dominant then (2.9) can be replaced by $(0 < \omega \le 1)\ 0 < \omega < 1$.

By adding a line search to the ASOR Algorithm we can improve on the choice of the relaxation factor $\omega$ by making its interval (0, 2] instead of (0, 1] as in Theorem 2.7 for strictly diagonally dominant $M$. We first state the algorithm.

**2.9 Line-Search Asynchronous SOR Algorithm** Same as the ASOR Algorithm 2.1 except that (2.4) is replaced by

$$t^i = \left( y^i - \omega E^i \left( N^i y^i + p^i + K^i (t^i - y^i) \right) \right)_+ \tag{2.4'}$$

and $y^{i+1}$ is defined by

$$\begin{cases} y^{i+1} = y^i + \lambda^i (t^i - y^i) \\ f\left(y^i + \lambda^i (t^i - y^i)\right) = \min\limits_{\lambda} \left\{ f\left(y^i + \lambda (t^i - y^i)\right) \Big| y^i + \lambda (t^i - y^i) \ge 0 \right\} \end{cases} \tag{2.11}$$

We establish now convergence of this algorithm.

**2.10 Convergence Theorem Algorithm 2.9** Let $M$ have a positive diagonal $D$, let $\left(E^i\right)^{-1} = D^i$ where $D^i$ is the diagonal of $N^i$, let $K^i$ be as defined in (2.6) and let

$$0 < \omega < 2 \min_{1 \le r \le n} \frac{D_{rr}}{\sum\limits_{s \ne r} |M_{rs}|} \tag{2.12}$$

Then each accumulation point of the sequence $\{P^{i^T} y^i\}$ solves the linear complementarity problem (2.1) where $\{y^i\}$ is the sequence generated by Algorithm 2.9.

**Proof** By Theorem 3.2 of [8] all we need to show is that

$$y\Big((\omega E^i)^{-1} + K^i\Big)y \geq \gamma\|y\|^2 \,\forall i,\ \forall y \in R^n \tag{2.13}$$

Let $N^i = L^i + D^i + U^i$ where $L^i$ is strictly lower triangular and $U^i$ strictly upper triangular. Then for any $y$ in $R^n$

$$\begin{aligned}
y\big((\omega E^i)^{-1} + K^i\big)y &= y\Big(\omega^{-1}D^i + \frac{K^i + K^{i^T}}{2}\Big)y \\
&= y\Big(\omega^{-1}D^i + \frac{1}{2}\big(\bar{L}^i + \bar{U}^i\big)\Big)y \ \text{ (By (2.6) for some } \bar{L}^i \subset L^i,\ \bar{U}^i \subset U^i\text{)} \\
&= z\Big(\omega^{-1}D^i + \frac{1}{2}\big(\hat{L}^i + \hat{U}^i\big)\Big)z, \ \ \big(z = P^{i^T}y, \ \text{for some } \hat{L}^i \subset L,\ \hat{U}^i \subset U\big) \\
&\geq \gamma\|z\|^2 \ \text{ for some } \gamma > 0 \ \text{ (By (2.12))} \\
&= \gamma\|y\|^2. \ \blacksquare
\end{aligned}$$

**2.11 Convergence Corollary for Algorithm 2.9** If $M$ is (strictly) diagonally dominant, then Theorem 2.10 holds for $(0 < \omega \leq 2)\ 0 < \omega < 2$.

## 3. Computational Results

Computational testing was carried out on the Sequent Symmetry S81 multiprocessor [10] with 14 tightly coupled 32-bit 80386 Intel microprocessors that share a 40-Megabyte physical memory (256-Megabyte virtual) and a single copy of DYNIX, an enhanced version of the UNIX operating system. Each 80386 processor is accompanied by an 80387 floating point unit, a 64-Kilobyte cache memory and a Weitek 1167 floating point accelerator chip.

All test problems were randomly generated with strictly diagonally dominant matrices with a prescribed density $d$ of nonzero elements. To produce a matrix with unevenly dense rows, each row with index $24(i-1)+1$, $i = 1, 2, \ldots$ was a fully dense row while the other rows had a prescribed density. The solution $z$ was chosen such that 50% of the components were zero and the other 50% uniformly distributed in the interval [0, 1]. The vector $q$ was then chosen such that the linear complementarity conditions (2.1) were satisfied.

Tables 1, 2 and 3, and Figures 2, 3 and 4 show results for each of four algorithms with and without line search for values of $\omega$ of 0.5, 0.9 and 1.8. All computations were carried out until the linear complementarity conditions (2.1) were satisfied to a tolerance of $10^{-8}$. The four parallel algorithms tested were the following special cases of Algorithm 2.1:

(a) Synchronous SOR (SSOR): $P^i = I$, $K^i$ as shown in Figure 1b [3].

(b) Static Asynchronous SOR (SASOR): $P^i = I$, $K^i$ as shown in Figure 1d with $k$ horizontal blocks of equal height.

(c) Dynamic Asynchronous SOR (DASOR): $P^i$ as described in Remark 2.2, $K^i$ as shown in Figure 1d with $k$ horizontal blocks of unequal height, the heights being determined adaptively by the relative processing speeds of the parallel processors.

(d) Dynamic Asynchronous SOR10 (DASOR10): Same as (c) above except that 10 sweeps through the components $z_1, \ldots, z_n$ are carried out by the $k$ parallel processors before they are synchronized.

It is interesting to note that no two runs of DASOR or DASOR10 were identical because of the dynamic way that variables are allocated to processors.

Each of the above four algorithms was run with and without a line search step. Tables 1, 2 and 3 give for each algorithm the number of iterations, the machine time in seconds and the speedup efficiency of the parallel algorithm measured by

$$\text{Efficiency} = \frac{\text{Solution time using 1 processor}}{k \cdot (\text{Solution time using } k \text{ processors})} \tag{3.1}$$

**3.1 Remark** We make the following observations regarding the numerical results given in Tables 1, 2 and 3:

(i) For the cases for which convergence has been established (columns denoted by $C$) efficiency is in the 43%-91% range, and for SASOR and DASOR in this category it is over 60%.

(ii) The fastest solution time of 3.2 seconds was obtained with SASOR without line search with 13 processors and $\omega = 0.9$ with a speedup of 8.6 and speedup efficiency of 66% over the corresponding serial SOR.

(iii) For the cases **without** line search, speedup efficiency improves in going from SSOR to SASOR to DASOR and to DASOR10 (with the minor deviation of the case of $\omega = 0.9$ for DASOR and DASOR10 with 13 processors). For the cases with line search a similar trend prevails with the exception of DASOR10 with $\omega = 1.8$.

(iv) Some speedup efficiencies, DASOR10 with line search and $\omega = 0.9$, exceeded 100%. The base case for computing these efficiences consisted of a serial SOR which contained a line search after **each** step. When the line search in this base case was made only at each tenth step to bring it in conformity with DASOR10, where a line search is made only after ten sweeps, the efficiencies were reduced to below 100%.

Figure 2, 3 and 4 give the computing times with $\omega = 0.9$ for the four parallel algorithms: SSOR, SASOR, DASOR and DASOR10, with and without line search, as well as the "linear speedup" time of $T(1)/k$ where $T(1)$ is the solution time with one processor and $k$ is the number of processors. Figure 2 depicts the results for the four SOR algorithms without line search and indicates the following descending computing times the four algorithms:

$$SSOR \geq SASOR \geq DASOR \geq DASOR10 \tag{3.2}$$

Figure 3 depicts the results with $\omega = 0.9$ for SSOR, SASOR and DASOR with line search as well as the linear speedup time $T(1)/k$. A similar trend to (3.2) is depicted. Figure 4 depicts the results with $\omega = 0.9$ for DASOR10 as well as $T(1)^*/k$ where the single processor time $T(1)^*$ is obtained with a line search after each tenth step only in order to conform to the DASOR10 line search procedure which occurs after ten sweeps. The figure shows that DASOR10 attains 2/3 or more of the linear speedup time.

We mention in closing that in other experiments in which the rows of the matrix $M$ contained a greater number of fully dense rows, there was a more pronounced improvement in going from SASOR from DASOR than that shown in Tables 1, 2 and 3 and Figures 2, 3 and 4.

We conclude by pointing out two interesting future tasks. One is the extension of the asynchronous algorithms presented here to the nonsymmetric linear complementarity problem in the spirit of the serial iterative algorithms of [1, 9]. The other is establishing convergence proof of the multisweep DASOR10 algorithm by using ideas of finite asynchronization delay [2].

## References


1. B. H. Ahn: "Solution of nonsymmetric linear complementarity problems by iterative methods", Journal of Optimization Theory and Applications 33, 1981, 175-185.

2. G. M. Baudet: "Asynchronous iterative methods for multiprocessors", Journal of the Association for Computing Machinery 25, 1978, 226-244.

3. R. De Leone & O. L. Mangasarian: "Serial and parallel solution of large scale linear programs by augmented Lagrangian successive overrelaxation", UW Computer Sciences Technical Report #701, July 1987, to appear in Proceedings of Workshop of Advanced Computation Techniques, Parallel Processing and Optimization, Karlsruhe, February 23-24, 1987, Springer-Verlag 1988.

4. D. DeWitt, R. Finkel & M. Solomon: "The CRYSTAL multicomputer: Design and implementation experience", IEEE Transactions on Software Engineering SE-13, 1987, 953-966.

5. O. L. Mangasarian: "Solution of symmetric linear complementarity problem by iterative methods", Journal of Optimization Theory and Applications 22, 1977, 465-485.

6. O. L. Mangasarian: "Sparsity-preserving SOR algorithms for separable quadratic and linear programming", Computer and Operations Research 11, 1984, 105-112.

7. O. L. Mangasarian & R. De Leone: "Parallel successive overrelaxation methods for symmetric linear complementarity problems and linear programs", Journal of Optimization Theory and Applications 54, 1987, 437-446.

8. O. L. Mangasarian & R. De Leone: "Parallel gradient projection successive overrelaxation for symmetric linear complementarity problems and linear programs", UW Computer Sciences Technical Report #659, August 1986, to appear in Annals of Operations Research 1988.

9. J.-S. Pang: "On the convergence of a basic iterative method for the implicit complementarity problem", Journal of Optimization Theory and Applications 37, 1982, 149-162.

10. Sequent Computer Systems, Inc.: "Symmetry Technical Summary", Beaverton, Oregon 97006, 1987.

| Relaxation Factor $\omega = 0.5$; No. of variables $n = 1000$; density $d = 25\%$ | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Without Line Search | | | | | | | | | | | | |
| No. of Proc. | SSOR (C) | | | SASOR (C) | | | DASOR (C) | | | DASOR10 (NC) | | |
| | Iter. | Time Sec. | Eff. | Iter. | Time Sec. | Eff. | Iter. | Time Sec. | Eff. | Iter. | Time Sec. | Eff. |
| 1 | 42 | 75.1 | — | 42 | 75.1 | — | 42 | 75.1 | — | 42 | 75.1 | — |
| 5 | 43 | 27.9 | 54% | 42 | 21.7 | 69% | 42 | 21.4 | 70% | 42 | 19.1 | 79% |
| 10 | 43 | 14.7 | 51% | 42 | 12.4 | 61% | 42 | 11.1 | 68% | 44 | 9.7 | 77% |
| 13 | 43 | 12.0 | 48% | 42 | 8.9 | 65% | 42 | 8.8 | 66% | 44 | 7.8 | 74% |
| With Line Search | | | | | | | | | | | | |
| No. of Proc. | SSOR (C) | | | SASOR (C) | | | DASOR (C) | | | DASOR10 (NC) | | |
| | Iter. | Time Sec. | Eff. | Iter. | Time Sec. | Eff. | Iter. | Time Sec. | Eff. | Iter. | Time Sec. | Eff. |
| 1 | 17 | 100.8 | — | 17 | 100.8 | — | 17 | 100.8 | — | 17 | 100.8 | — |
| | | | | | | | | | | (32 | 74.6)* | |
| 5 | 19 | 28.4 | 71% | 17 | 23.2 | 87% | 17 | 24.0 | 84% | 38 | 20.5 | 98% |
| | | | | | | | | | | | | (73%)+ |
| 10 | 18 | 15.0 | 67% | 18 | 14.1 | 72% | 17 | 12.1 | 83% | 37 | 10.7 | 94% |
| | | | | | | | | | | | | (70%)+ |
| 13 | 18 | 11.2 | 69% | 17 | 9.6 | 81% | 17 | 9.4 | 83% | 38 | 8.5 | 91% |
| | | | | | | | | | | | | (68%)+ |

*Line search performed every tenth set

+Efficiency based on ( )*

C: Convergence established

NC: No convergence established

Table 1: Number of iterations, times in seconds and efficiency versus number of processors for four algorithms: the Synchronous SOR (SSOR), the Static Asynchronous SOR (SASOR), the single sweep Dynamic Asynchronous SOR (DASOR) and the 10 sweep Dynamic Asynchronous SOR (DASOR10).

| Relaxation Factor $\omega = 0.9$; No. of variables $n = 1000$; density $d = 25\%$ | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Without Line Search | | | | | | | | | | | | |
| No. of Proc. | SSOR (C) | | | SASOR (C) | | | DASOR (C) | | | DASOR10 (NC) | | |
| | Iter. | Time Sec. | Eff. | Iter. | Time Sec. | Eff. | Iter. | Time Sec. | Eff. | Iter. | Time Sec. | Eff. |
| 1 | 15 | 27.5 | — | 15 | 27.5 | — | 15 | 27.5 | — | 15 | 27.5 | — |
| 5 | 17 | 11.4 | 48% | 15 | 7.9 | 70% | 15 | 7.9 | 70% | 15 | 7.0 | 79% |
| 10 | 17 | 5.9 | 47% | 15 | 4.5 | 61% | 15 | 4.3 | 64% | 15 | 4.0 | 69% |
| 13 | 17 | 4.9 | 43% | 15 | 3.2 | 66% | 15 | 3.4 | 62% | 18 | 3.3 | 64% |
| With Line Search | | | | | | | | | | | | |
| No. of Proc. | SSOR (C) | | | SASOR (C) | | | DASOR (C) | | | DASOR10 (NC) | | |
| | Iter. | Time Sec. | Eff. | Iter. | Time Sec. | Eff. | Iter. | Time Sec. | Eff. | Iter. | Time Sec. | Eff. |
| 1 | 12 | 75.5 | — | 12 | 75.5 | — | 12 | 75.5 | — | 12 | 75.5 | — |
| | | | | | | | | | | (14 | 34.9)* | |
| 5 | 12 | 18.3 | 83% | 12 | 16.8 | 90% | 12 | 16.6 | 91% | 15 | 9.6 | 157% |
| | | | | | | | | | | | | (73%)+ |
| 10 | 13 | 11.0 | 69% | 12 | 9.5 | 79% | 12 | 8.6 | 88% | 15 | 5.2 | 145% |
| | | | | | | | | | | | | (67%)+ |
| 13 | 13 | 8.3 | 70% | 12 | 6.8 | 85% | 12 | 6.7 | 87% | 16 | 4.1 | 142% |
| | | | | | | | | | | | | (65%)+ |

*Line search performed every tenth set

+Efficiency based on ( )*

C: Convergence established

NC: No convergence established

Table 2: Number of iterations, times in seconds and efficiency versus number of processors for four algorithms: the Synchronous SOR (SSOR), the Static Asynchronous SOR (SASOR), the single sweep Dynamic Asynchronous SOR (DASOR) and the 10 sweep Dynamic Asynchronous SOR (DASOR10).

| Relaxation Factor $\omega = 1.8$; No. of variables $n = 1000$; density $d = 25\%$ | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Without Line Search | | | | | | | | | | | | |
| No. of Proc. | SSOR (NC) | | | SASOR (NC) | | | DASOR (NC) | | | DASOR10 (NC) | | |
| | Iter. | Time Sec. | Eff. | Iter. | Time Sec. | Eff. | Iter. | Time Sec. | Eff. | Iter. | Time Sec. | Eff. |
| 1 | 192 | 337.3 | — | 192 | 337.3 | — | 192 | 337.3 | — | 192 | 337.3 | — |
| 5 | 763 | 491.2 | 14% | 193 | 96.6 | 70% | 193 | 96.6 | 70% | 194 | 82.4 | 82% |
| 10 | Did not converge | | | 192 | 56.0 | 60% | 193 | 49.9 | 68% | 194 | 42.7 | 79% |
| 13 | Did not converge | | | 194 | 40.1 | 65% | 196 | 40.2 | 65% | 196 | 34.6 | 75% |
| With Line Search | | | | | | | | | | | | |
| No. of Proc. | SSOR (C) | | | SASOR (C) | | | DASOR (C) | | | DASOR10 (NC) | | |
| | Iter. | Time Sec. | Eff. | Iter. | Time Sec. | Eff. | Iter. | Time Sec. | Eff. | Iter. | Time Sec. | Eff. |
| 1 | 16 | 95.3 | — | 16 | 95.3 | — | 16 | 95.3 | — | 16 | 95.3 | — |
| | | | | | | | | | | (62 | 140.3)* | |
| 5 | 15 | 22.8 | 84% | 16 | 22.1 | 86% | 16 | 21.9 | 87% | 189 | 97.4 | 20% |
| | | | | | | | | | | | | (29%)+ |
| 10 | 15 | 12.5 | 76% | 17 | 14.0 | 68% | 16 | 11.4 | 84% | 191 | 50.1 | 19% |
| | | | | | | | | | | | | (28%)+ |
| 13 | 18 | 11.2 | 65% | 16 | 9.1 | 81% | 16 | 9.0 | 81% | 192 | 39.7 | 18% |
| | | | | | | | | | | | | (28%)+ |

*Line search performed every tenth set

+Efficiency based on ( )*

C: Convergence established

NC: No convergence established

Table 3: Number of iterations, times in seconds and efficiency versus number of processors for four algorithms: the Synchronous SOR (SSOR), the Static Asynchronous SOR (SASOR), the single sweep Dynamic Asynchronous SOR (DASOR) and the 10 sweep Dynamic Asynchronous SOR (DASOR10).

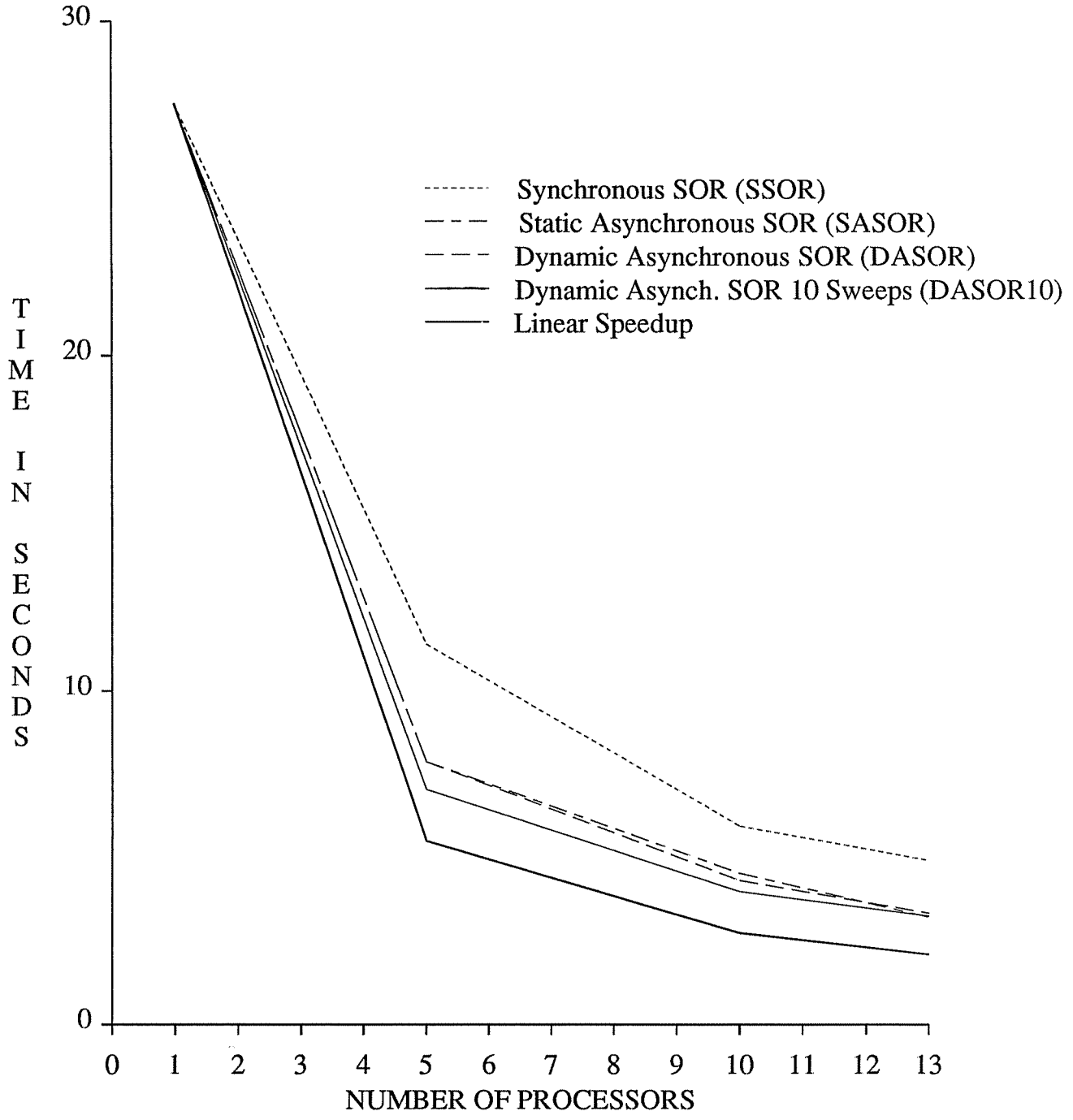



Figure 2: Sequent Symmetry S81 times for four parallel SOR algorithms without line search, $n = 1000$, $d = 25\%$, $\omega = 0.9$.

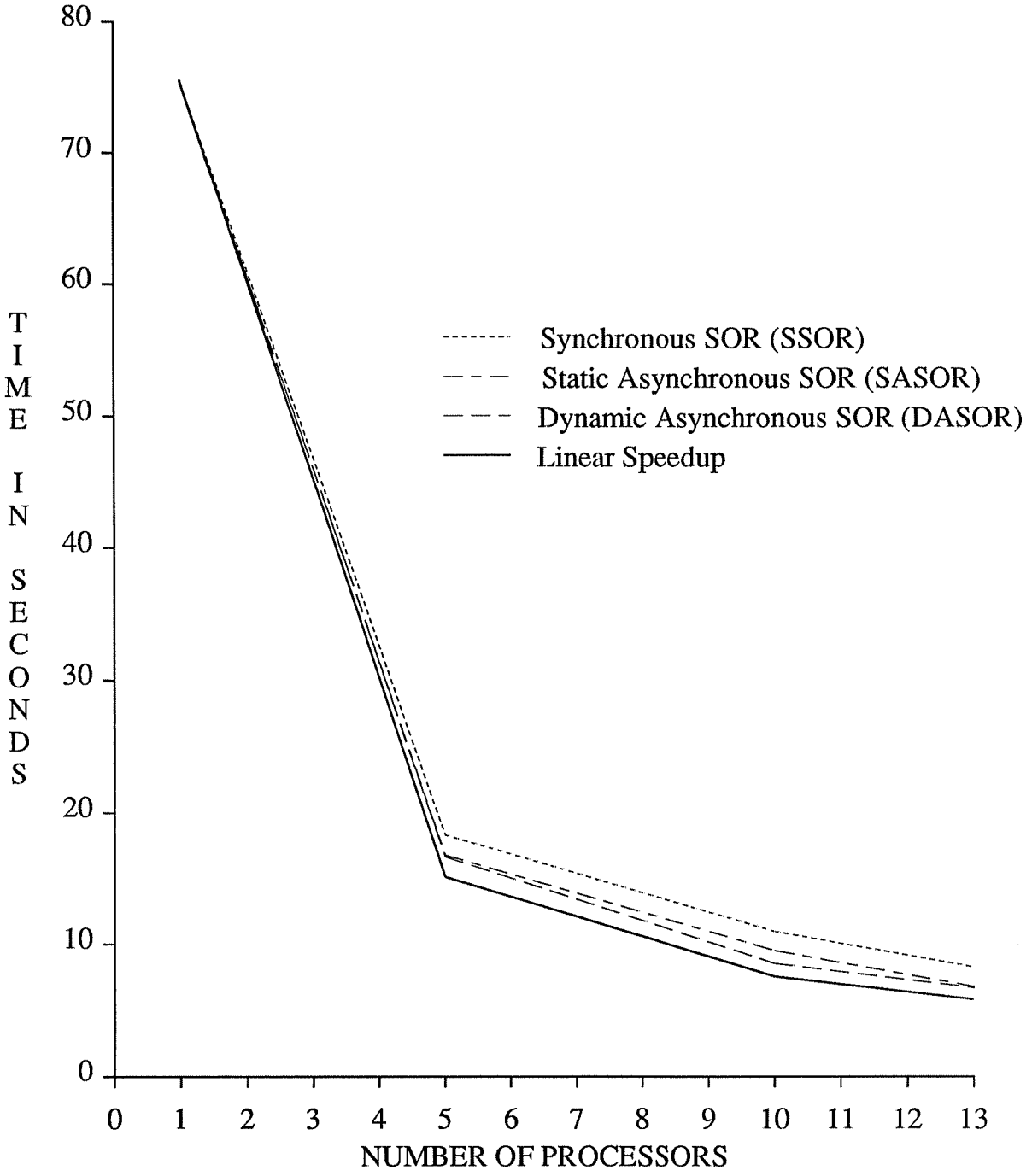



Figure 3: Sequent Symmetry S81 times for three parallel SOR algorithms with line search, $n = 1000$, $d = 25\%$, $\omega = 0.9$.

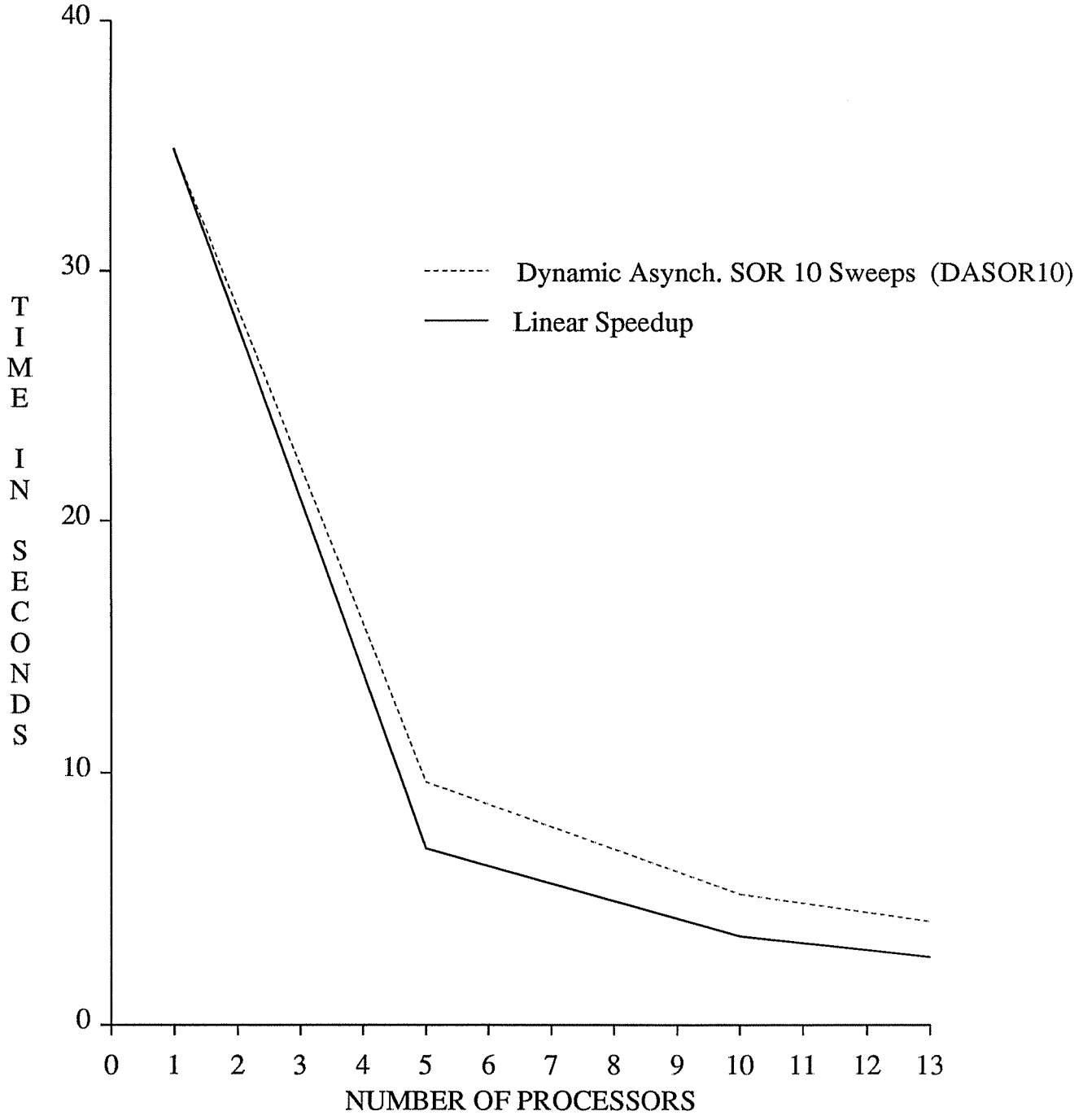



Figure 4: Sequent Symmetry S81 times for DASOR10 algorithm with line search, $n = 1000$, $d = 25\%$, $\omega = 0.9$.