PARALLEL GRADIENT PROJECTION SUCCESSIVE OVERRELAXATION FOR SYMMETRIC LINEAR COMPLEMENTARITY PROBLEMS AND LINEAR PROGRAMS

by

O. L. Mangasarian & R. De Leone

Computer Sciences Technical Report #659

August 1986

Parallel Gradient Projection Successive Overrelaxation for Sylmmetric Linear Complementarity Problems and Linear Programs

O. L. Mangasarian
&
R. De Leone

Computer Sciences Department
University of Wisconsin, Madison, Wisconsin 53706

Revised August 1987

Abstract. A gradient projection successive overrelaxation (GP-SOR) algorithm is proposed for the solution of symmetric linear complementarity problems and linear programs. A key distinguishing feature of this algorithm is that when appropriately parallelized, the relaxation factor interval (0, 2) is not reduced. In a previously proposed parallel SOR scheme, the substantially reduced relaxation interval mandated by the coupling terms of the problem often led to slow convergence. The proposed parallel algorithm solves a general linear program by finding its least 2-norm solution. Efficiency of the algorithm is in the 50 to 100 percent range as demonstrated by computational results on the CRYSTAL token-ring multicomputer and the Sequent Balance 21000 multiprocessor.

Key Words: Parallel algorithms, SOR gradient projection, linear programming, linear complementarity problem.

Abbreviated Title: Parallel solution of LCP's and LP's.

[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 Grants AFOSR-86-0172 and AFOSR-86-0255.

1. Introduction

In [9] we proposed a parallel successive overrelaxation (SOR) method for the solution of the fundamental symmetric linear complementarity problem

$$Mz + q \geq 0,\ z \geq 0,\ z(Mz + q) = 0 \tag{1.1}$$

where M is a $k \times k$ symmetric matrix and q is a vector in the k-dimensional real space R^k. The significance of this problem arises to a great extent from the fact that the least 2-norm solution of a linear program can be reduced to such a problem [6, 7, 8]. In order to achieve convergence in the parallel SOR algorithm of [9], the relaxation factor ω had to be reduced to an interval considerably smaller than (0, 2), depending on the size of the off-diagonal terms of the matrix M relative to its diagonal elements. This small ω interval often led to very small steps and slow convergence. In the present work we propose a gradient projection-successive overrelaxation algorithm which overcomes the problem of a small relaxation factor. We summarize this approach as follows. We consider the associated quadratic minimization problem on the nonnegative orthant

$$\min_{z \geq 0} f(z) := \min_{z \geq 0} \frac{1}{2} zMz + qz \tag{1.2}$$

and note that the symmetric linear complementarity condition (1.1) is a necessary optimality condition for (1.2). Our basic method consists of modifying the gradient of $f(z)$ by an SOR step so as to use the latest information when computing each component of the gradient, relaxing the modified gradient by a factor ω, projecting the relaxed modified gradient on the nonnegative orthant and finally taking a feasible step in the direction of the projection which minimizes $f(z)$. A key difference between the present approach and that of [9] is that the projected point in [9] was a point that decreased the objective function $f(z)$, whereas in the present approach the projected point merely constitutes a descent direction for $f(z)$ and in fact may not be a point with a decreased $f(z)$. Although the direction generated by the SOR algorithm of [9] is also a descent direction, it is too restrictive a direction which may impose an extremely small bound on the relaxation factor ω that may lead to a very slow algorithm when parallelized. By contrast when the present algorithm is parallelized, the relaxation factor interval remains (0, 2) for the case when M is positive semidefinite with a positive diagonal. This is an important case that includes the least 2-norm linear programming problem. We outline the contents of the paper now.

In Section 2 we begin with Theorem 2.1 which gives an optimality condition for an SOR-relaxed projected gradient $p(z)$ defined by (2.2) for a nonsymmetric linear complementarity problem. Theorem 2.2 shows that $p(z)$ exists and is unique and continuous under appropriate assumptions. The continuity of $p(z)$ depends on a recently established Lipschitz continuity result for linear complementarity problems with P-matrices [10]. In Section 3 we describe a serial version of a gradient projection successive overrelaxation (GP-SOR) algorithm for the solution of (1.1) and establish its convergence in Theorem 3.2. In Corollary 3.3 we show that when M is positive semidefinite the relaxation factor interval remains (0, 2), the same as in classical SOR for the solution of linear equations [12]. In Section 4 we give a parallel version of our GP-SOR algorithm by dividing the matrix M of (1.1) into r contiguous horizontal blocks and applying our GP-SOR algorithm to the principal submatrix of each horizontal block simultaneously. This generates a descent direction for the overall problem which can be used to generate the next iterate. A key property of this approach given in Corollary 4.4 is that when applied to a positive semidefinite M, the relaxation ω for each subproblem can take any value in (0, 2). In Section 5 of the paper we reduce the standard linear programming problem to a symmetric linear complementarity problem via a least 2-norm solution [6] and solve it by the parallel GP-SOR algorithm proposed in Section 4.

Computational experiments were performed on the CRYSTAL [1] multicomputer and the Sequent Balance 21000 multiprocessor. Encouraging results were obtained on both machines. The results show that for each problem size the number of machines that solve the problem in least time increases with density (Figures 1 and 5). We also obtain a closely fitting empirical formula (5.5) for the time-per-iteration on CRYSTAL for various densities (Figure 2). These results again indicate that the number of processors giving the smallest time per iteration increases with density. Efficiency for r processors was measured by the ratio of $(1/r)th$ the running time on one processor to the running time on r processors. This gave a surprising efficiency of over 100% in certain instances, such as shown in Figure 4. This can be explained because even though the same computer program was used on 1 processor and r processors, the underlying algorithm was not identical in both instances. When the program was revised so that the identical algorithm was used on 1 processor and r processors, the efficiencies were all below 100%. This is explained in more detail in

Section 5.

We briefly describe our notation now. For a vector x in the n-dimensional real space R^n, x_+ will denote the vector in R^n with components $(x_+)_i = \max\{x_i, 0\}$, $i = 1, ..., n$. The scalar product of two vectors x and y in R^n will be simply denoted by xy. For $1 \leq p \leq \infty$, the p-norm $\left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}$ of a vector in R^n will be denoted by $\|x\|_p$. For the 2-norm the subscript 2 will be dropped. 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 all $m \times n$ 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, and for $I \subset \{1, ..., m\}$, $J \subset \{1, ..., n\}$, A_I will denote the submatrix of A with rows A_i, $i \in I$, while A_{IJ} will denote the submatrix of A with elements A_{ij}, $i \in I$, $j \in J$. Similarly for $x \in R^n$ and $I_\ell \subset \{1, ..., n\}$, x_{I_ℓ} will denote x_i, $i \in I_\ell$. The set $\{I_1, I_2, ..., I_K\}$ is said to be a consecutive partition of $\{1, ..., n\}$ if it is a partition of $\{1, ..., n\}$ such that $i < j$ for $i \in I_\ell$, $j \in I_{\ell+1}$ and $\ell = 1, ..., K-1$. A matrix in $R^{k\times k}$ is said to be a $P-$matrix if it has positive principal minors. Here and throughout the symbols $:=$ and $=:$ denote definition of the term on the left and right sides of each symbol respectively. For $z \in R^k$ and $M \in R^{k\times k}$, $\|z\|^2_M := zMz$.

2. GP-SOR Optimality Conditions for the LCP

An important role in obtaining our GP-SOR algorithms will be played by a gradient projection optimality condition that has been modified by an additional SOR term. We motivate this optimality condition as follows. It is well known [5] that the LCP (1.1) is equivalent to finding a $z \in R^k$ such that

$$z = \big(z - \omega(Mz + q)\big)_+ \quad \text{for some} \quad \omega > 0 \tag{2.1}$$

When M is symmetric this condition can be considered as a "gradient projection" condition, that is, the projection on R^k_+ of z perturbed by any negative multiple of $Mz + q$ is z itself. The equivalence between (2.1) and (1.1) is of course valid for any $M \in R^{k\times k}$, symmetric or not [5, Lemma 2.1]. Hence for the rest of this section we will not assume that M is symmetric. We will be interested in perturbing the "gradient" $Mz + q$ to $Mz + q + K(y - z)$ where K is some matrix in $R^{k\times k}$ and y is some vector in R^k. The point of this perturbation is to enable us to replace the data in each computed component of $Mz + q$ by the "latest" information as must be done in an SOR iteration. We therefore wish to replace (2.1) by the following more general GP-SOR optimality condition. For a point $z \in R^n$ define a "relaxed projected modified gradient" relative to the linear complementarity problem (1.1) as any $p(z)$ in R^k such that

$$p(z) = \Big(z - \omega E\big(Mz + q + K(p(z) - z)\big)\Big)_+ \tag{2.2}$$

where ω is some positive number, E is a positive diagonal matrix in $R^{k\times k}$ and K is some matrix in $R^{k\times k}$. We note immediately that (2.2) is equivalent to the following linear complementarity problem in $p(z)$

$$\begin{aligned} w(z) &= \big((\omega E)^{-1} + K\big)p(z) + \big((M - K) - (\omega E)^{-1}\big)z + q \geq 0 \\ p(z) &\geq 0,\ p(z)w(z) = 0 \end{aligned} \tag{2.3}$$

This equivalence is a direct consequence of the following trivially verifiable equivalence for a pair of real numbers a and b

$$a = b_+ \iff a - b \geq 0,\ a \geq 0,\ a(a - b) = 0 \tag{2.4}$$

We also note that (2.3) can also be obtained as a splitting $\big((\omega E)^{-1}+K\big)+\big(M-K-(\omega E)^{-1}\big)$ of the matrix M in the sense of [3]. However our algorithmic approach is fundamentally

different from both that of [3] and [5] where $p(z)$ is taken as the next iterate to succeed z, whereas in our approach $p(z) - z$ is merely taken as a direction along which a stepsize procedure is performed. This is a basic difference which allows us to have a much better relaxation factor in the parallel version of the algorithm. We also note that (2.3) need not be solvable in general for $p(z)$. We shall however impose sufficient conditions (such as $(\omega E)^{-1} + K$ is positive definite) to ensure that it is. We begin with the following characterization of the linear complementarity problem (1.1).

2.1 Theorem (GP-SOR optimality condition) Let M, K, $E \in R^{k\times k}$, $q \in R^k$, $\omega > 0$, and let E be a positive diagonal matrix.

(a) If z solves the LCP (1.1) and $(\omega E)^{-1} + K$ is positive definite then $p(z) = z$ is the unique solution of (2.2).

(b) If $p(z)$ solves (2.2) and $p(z) = z$ then z solves the LCP (1.1).

Proof (a) Let z solve the LCP (1.1) and let $(\omega E)^{-1} + K$ be positive definite. Then (2.3) or equivalently (2.2) has a unique solution $p(z)$ [11]. Since z solves the LCP (1.1), it follows that $p(z) = z$ solves (2.3) or equivalently (2.2). Hence z is the unique solution of (2.2).

(b) If $p(z)$ solves (2.2) or equivalently (2.3) and $p(z) = z$, it follows from (2.3) that z solves that the LCP (1.1). ∎

We now give sufficient conditions that will ensure that $p(z)$ of (2.2) exists and is a continuous function of z. This is an interesting result but is not needed in the convergence proof of the GP-SOR algorithms.

2.2 Theorem (Uniqueness and continuity of the SOR projected gradient $p(z)$). Let $\omega > 0$, and let E be a positive diagonal matrix and let K be a strictly lower or upper triangular matrix, or let $(\omega E)^{-1} + K$ be a $P-$matrix. Then for z in R^k there exists a unique continuous function $p(z)$ satisfying (2.2).

Proof For the case when K is strictly lower (upper) triangular, $p(z)$ can be computed componentwise $p_j(z)$, $j = 1, ..., k\,(j = k, ..., 1)$ as a unique piecewise linear, and hence continuous, function of z. For the case when $(\omega E)^{-1} + K$ is a $P-$matrix, $p(z)$ exists and is unique [11], and by Theorem 3.3 [10] is Lipschitz continuous and hence continuous in z. ∎

Note that the cases of $K = 0$ and $(\omega E)^{-1} + K$ positive definite are included as special cases of the above theorem.

3. Serial GP-SOR Algorithm for the LCP

From this point onward we specialize to the case of symmetric M. Hence the LCP (1.1) can be considered as a necessary optimality condition for the problem (1.2). With this connection in mind we specify our algorithm.

3.1 GP-SOR Algorithm for LCP Let $z^0 \geq 0$

Direction Generation Define the direction

$$d^i := p(z^i) - z^i \tag{3.1}$$

such that

$$p(z^i) = \Big(z^i - \omega E^i\Big(Mz^i + q + K^i\big(p(z^i) - z^i\big)\Big)\Big)_+ \tag{3.2}$$

where $\{E^i\}$, $\{(E^i)^{-1}\}\{K^i\}$ are bounded sequences of matrices in $R^{k\times k}$, with each E^i being a positive diagonal matrix, $\omega > 0$ and such that for some $\gamma > 0$

$$z\big((\omega E^i)^{-1} + K^i\big)z \geq \gamma\|z\|^2,\ \forall i,\ \forall z \in R^k \tag{3.3}$$

Stop if $d^i = 0$, else continue.

Stepsize Generation $z^{i+1} = z^i + \lambda^i d^i$

where

$$f(z^i + \lambda^i d^i) = \min_{\lambda}\{f(z^i + \lambda d^i)\big|z^i + \lambda d^i \geq 0\} \tag{3.4}$$

Note that because f is quadratic (3.4) is essentially equivalent to solving a linear equation in one variable. Remark also that if $E^i = I$, $K^i = 0$, $\lambda^i = 1$, and $\omega \in \big(0, 2/\rho(M)\big)$, where $\rho(M)$ is the spectral radius of M, then the above algorithm becomes the classical Levitin-Poljak gradient projection algorithm [2], which for our method (3.1) can be considered a projected Jacobi method [5]. This method in general is not as effective as a projected SOR method. If on the other hand we let $\lambda^i = 1$ and ω, E^i and K^i satisfy the assumptions of Algorithm 2.1 [5] then we have a projected SOR scheme which when parallelized [9] may lead to a very small relaxation factor ω that may cause slow convergence. The parallel version of Algorithm 3.1 on the other hand which we will describe

in the next section, will have $\omega \in (0, 2)$ when M is positive semidefinite with a positive diagnonal. This is the key difference from [9] which critically improves the convergence of the parallel scheme.

We state now and prove a fundamental convergence result for Algorithm 3.1 which will serve as a vehicle for our parallel scheme.

3.2 Theorem (Serial GP-SOR convergence) Let M be symmetric. Either the sequence $\{z^i\}$ generated by Algorithm 3.1 terminates at a solution of the LCP (1.1) or each accumulation point of $\{z^i\}$ solves the LCP (1.1).

Proof The sequence $\{z^i\}$ terminates only if for some i, $p(z^i) = z^i$, in which case by Theorem 2.1, z^i solves the LCP (1.1). Suppose now $\{z^i\}$ does not terminate and that $\bar{z}$ is an accumulation point of $\{z^i\}$. Let $p^i := p(z^i)$. We then have for $i = 0, 1, \ldots$ that

$$\begin{aligned}
-\bigtriangledown f(z^i)d^i &= -\bigtriangledown f(z^i)(p^i - z^i) \\
&= \left[z^i - \omega E^i\left(Mz^i + q + K^i(p^i - z^i)\right) - p^i\right](\omega E^i)^{-1}\left[p^i - z^i\right] \\
&\quad + \left\|p^i - z^i\right\|^2_{(\omega E^i)^{-1}+K^i} \\
&\geq \left\|p^i - z^i\right\|^2_{(\omega E^i)^{-1}+K^i} \qquad \text{(By (3.2) and Lemma 2.2 [5])} \\
&\geq \gamma\left\|p^i - z^i\right\|^2 \qquad \text{(By (3.3))}.
\end{aligned}$$

Hence

$$-\bigtriangledown f(z^i)d^i \geq \gamma\left\|p^i - z^i\right\|^2 \geq 0, \; i = 0, 1, \ldots \tag{3.5}$$

Now let $\{z^{i_j}\}$ be a subsequence converging to the accumulation point $\bar{z}$. We claim that $\{p^{i_j}\}$ is bounded. For if it were not bounded it would follow from (3.2) that

$$\lim_{j\to\infty} \frac{p(z^{i_j})}{\left\|p(z^{i_j})\right\|} = \lim_{j\to\infty}\left(-\omega E^{i_j}K^{i_j}\frac{p(z^{i_j})}{\left\|p(z^{i_j})\right\|}\right)_+$$

and hence for some accumulation point $(\bar{p}, \bar{E}, \bar{K})$ we would have

$$\bar{p} = (-\omega\bar{E}\bar{K}\bar{p})_+, \;\; \bar{p} \neq 0. \tag{3.6}$$

This however is equivalent by (2.4) to $0 \neq \bar{p} \geq 0$, $\bar{p}\left((\omega\bar{E})^{-1}+\bar{K}\right)\bar{p} = 0$, $\left((\omega\bar{E})^{-1}+\bar{K}\right)\bar{p} \geq 0$ which contradicts (3.3). Consequently without loss of generality, let $\{z^{i_j}, p^{i_j}, E^{i_j}, K^{i_j}\} \to (\bar{z}, \bar{p}, \bar{E}, \bar{K})$ and hence $\{d^{i_j}\} \to \bar{d} = \bar{p} - \bar{z}$. Now, since $p^{i_j} \geq 0$ it follows that

$$z^{i_j} + \lambda d^{i_j} = (1 - \lambda)z^{i_j} + \lambda p^{i_j} \geq 0 \quad \text{for} \quad 0 \leq \lambda \leq 1$$

Consequently

$$f(z^{i_j} + \lambda d^{i_j}) \geq f(z^{i_j+1}) \geq f(z^{i_{j+1}}) \quad \text{for} \quad 0 \leq \lambda \leq 1$$

Letting $j \to \infty$ we have that

$$f(\bar{z} + \lambda \bar{d}) \geq f(\bar{z}) \quad \text{for} \quad 0 \leq \lambda \leq 1$$

Hence $\bigtriangledown f(\bar{z})\bar{d} \geq 0$. Combining this with (3.5) gives

$$0 \geq - \bigtriangledown f(\bar{z})\bar{d} = \lim_{j \to \infty} - \bigtriangledown f(z^{i_j}) d^{i_j} \geq \lim_{j \to \infty} \gamma \left\| p^{i_j} - z^{i_j} \right\|^2 = \gamma \left\| \bar{p} - \bar{z} \right\|^2 \geq 0$$

Hence

$$\bar{p} = \bar{z}$$

But since

$$p^{i_j} = \left(z^{i_j} - \omega E^{i_j} \left(M z^{i_j} + q + K^{i_j} (p^{i_j} - z^{i_j}) \right) \right)_+$$

it follows that

$$\begin{aligned} \bar{z} = \bar{p} &= \left(\bar{z} - \omega \bar{E} \left(M \bar{z} + q + \bar{K} (\bar{p} - \bar{z}) \right) \right)_+ \\ &= \left(\bar{z} - \omega \bar{E} (M \bar{z} + q) \right)_+ \end{aligned}$$

By the positivity of $\omega \bar{E}$ and (2.4) we have that $\bar{z}$ solves the LCP (1.1). ∎

3.3 Corollary (GP-SOR special cases) Condition (3.3) of Algorithm 3.1 holds under <u>either</u> of the following two assumptions:

(i) $K^i = L$ or U, $E^i = E$ and $0 < \omega < \min\limits_{j=1,\ldots,k} 2 \Big/ \left(E_{jj} \sum\limits_{\substack{\ell=1 \\ \ell \neq j}}^{k} |M_{j\ell}| \right)$

(ii) $K^i = L$ or U, $E^i = D^{-1} > 0$, M is positive semidefinite and $0 < \omega < 2$

where

$$L + D + U := M, \tag{3.7}$$

L is the strictly lower triangular part of M, U is the strictly upper triangular part of M and D is the diagonal of M.

Proof

(i) $z\big((\omega E^i)^{-1} + K^i\big)z = z\big((\omega E)^{-1} + \frac{L+U}{2}\big)z \geq \gamma \left\| z \right\|^2$ where the last inequality follows from the strict diagonal dominance of $(\omega E)^{-1} + \frac{L+U}{2}$ and positivity of E.

(ii) $z\big((\omega E^i)^{-1} + K^i\big)z = z\big(\omega^{-1} D + \frac{L+U}{2}\big)z = \frac{1}{2} z\big((2\omega^{-1} - 1)D + M\big)z \geq \gamma \left\| z \right\|^2$ where the last inequalilty follows from the positive semidefiniteness of M, the positivity of D and $0 < \omega < 2$. ∎

4. Parallel GP-SOR Algorithm for the LCP

By appropriately choosing the matrix K^i of the GP-SOR Algorithm 3.1, the algorithm can be split into a parallel algorithm that can be run simultaneously on a number of processors. To see this consider the principal step (3.2) of the GP-SOR Algorithm 3.1. In this step K^i can be considered as a substitution operator which replaces the old data z^i by the new data $p(z^i)$. Thus in the serial GP-SOR procedure $K^i = L$ where L is the strictly lower triangular part of the whole matrix M. In this case $p_j(z^i)$ replaces z^i_j during the computation of $p_\ell(z^i)$ for <u>all</u> $\ell > j$. This algorithm is sequential and cannot be split among simultaneous processors. If however we choose K^i to be a block diagonal matrix consisting of the strictly lower triangular part of the principal submatrix of <u>each</u> horizonal partition of the matrix M, then Algorithm 3.1 can be performed simultaneously on a number of processors equal to the number of submatrices in the horizontal partition of M. Such a choice for K^i was first proposed in [9] for a different SOR algorithm which did not involve a stepsize generation procedure. This is a key difference. In the present parallel algorithm the relaxation factor ω depends only on the local matrices E^i and K^i of each block but not on the remaining global parts of the matrix M. This can be easily seen from the main condition (3.3) of Algorithm 3.1 in which only E^i and K^i appear but not M. As we shall see in Theorem 4.3 below, this allows ω to range over the interval (0, 2) when M is positive semidefinite. In contrast in Algorithm 3.1 of [9] ω was restricted to a considerably smaller interval which depended critically on the coupling terms of the matrix M. This was a consequence of the appearance of M in the key convergence condition [9, equation (5)] in addition to E^i and K^i. To specify our parallel algorithm, partition the matrix M into r contiguous horizontal blocks as follows:

$$M =: \begin{bmatrix} M_{I_1} \\ M_{I_2} \\ \vdots \\ M_{I_r} \end{bmatrix} \tag{4.1}$$

where the blocks M_{I_j} correspond to the variables z_{I_j} and $\{I_1, I_2, ..., I_r\}$ is a consecutive partition of $\{1, 2, ..., k\}$. Now partition M_{I_j} as follows

$$M_{I_j} =: \begin{bmatrix} M_{I_j I_j} & M_{I_j \bar{I}_j} \end{bmatrix} \tag{4.2}$$

where $\bar{I}_j$ is the complement of I_j in $\{1, 2, ..., k\}$. Thus $M_{I_j I_j}$ is a principal square submatrix of M with elements M_{st}, $s \in I_j$ and $t \in I_j$. We further partition $M_{I_j I_j}$ into the sum

of its strictly lower triangular part $L_{I_jI_j}$, its diagonal part $D_{I_jI_j}$ and its strictly upper triangular part $U_{I_jI_j}$ as follows

$$M_{I_jI_j} =: L_{I_jI_j} + D_{I_jI_j} + U_{I_jI_j} \tag{4.3}$$

Now let K^i of Algorithm 2.1 be defined by a block diagonal matrix as follows

$$K^i = K := \begin{bmatrix} L_{I_1I_1} & & & \\ & L_{I_2I_2} & & \\ & & \ddots & \\ & & & L_{I_rI_r} \end{bmatrix} \tag{4.4}$$

where each $L_{I_jI_j}$ is the strictly lower triangular part of $M_{I_jI_j}$. Algorithm 3.1 can now be performed for each row block I_j, $j = 1, ..., r$, simultaneously, that is in parallel. More specifically we have the following.

4.1 Parallel GP-SOR Algorithm for the LCP Let $\{I_1, I_2, ..., I_r\}$ be a consecutive partition of $\{1, 2, ...k\}$, let E be a positive diagonal matrix in $R^{k\times k}$ and let $z^0 \geq 0$. For $i = 0, 1, 2, ...,$ do the following:

Direction Generation Define the direction

$$d^i := p(z^i) - z^i := \begin{pmatrix} p_{I_1}(z^i) - z^i_{I_1} \\ \vdots \\ p_{I_r}(z^i) - z^i_{I_r} \end{pmatrix} \tag{4.5}$$

such that $p(z^i)$ satisfies

$$p_{I_j}(z^i) = \left(z^i_{I_j} - \omega E_{I_jI_j}\left(M_{I_j}z^i + q_{I_j} + L_{I_jI_j}(p_{I_j}(z^i) - z^i_{I_j})\right)\right)_+, \ j = 1, ..., r \tag{4.6}$$

where $\omega > 0$ is chosen such that for some $\gamma > 0$

$$z_{I_j}\left((\omega E_{I_jI_j})^{-1} + L_{I_jI_j}\right)z_{I_j} \geq \gamma \left\| z_{I_j} \right\|^2, \ \forall z_{I_j}, \ j = 1, ..., r \tag{4.7}$$

Stop if $d^i = 0$, else continue.

Stepsize Generation $z^{i+1} = z^i + \lambda^i d^i$

where

$$f(z^i + \lambda^i d^i) = \min_{\lambda} \{ f(z^i + \lambda d^i) \big| z^i + \lambda d^i \geq 0 \} \tag{4.8}$$

4.2 Remark The principal part of this algorithm consists of the direction generation part (4.6), which can be performed in parallel on r processors. Once this is done the stepsize generation (4.8) is performed and the new value z^{i+1} is shared between the r processors.

By applying Theorem 3.2 to Algorithm 4.1 with the identifications $E^i = E$ and $K^i = K$ as defined by (4.4) we have the following convergence result.

4.3 Theorem (Convergence of the Parallel GP-SOR Algorithm) Either the sequence $\{z^i\}$ generated by the Parallel GP-SOR Algorithm 4.1 terminates at a solution of the LCP (1.1) or each of its accumulation points solves the LCP (1.1).

Two important special cases of the convergence Algorithm 4.1 are worth noting explicitly and are given in the following.

4.4 Corollary (Parallel GP-SOR special cases) Condition 4.7 of Algorithm 4.1 holds under either of the following two assumptions:

$$0 < \omega < \min_{j=1,\ldots,r} \min_{i \in I_j} \frac{2}{E_{ii} \sum_{\substack{\ell \in I_j \\ \ell \neq i}} |M_{i\ell}|} \tag{4.9}$$

$$0 < \omega < 2, \; E = D^{-1} \quad \text{and } M \text{ is positive semidefinite} \tag{4.10}$$

Proof If (4.9) holds then

$$z_{I_j}\big((\omega E_{I_j I_j})^{-1} + L_{I_j I_j}\big) z_{I_j} = z_{I_j}\big((\omega E_{I_j I_j})^{-1} + (L_{I_j I_j} + U_{I_j I_j})/2\big) z_{I_j} \geq \gamma \big\| z_{I_j} \big\|^2$$

where the last inequality follows from the strict diagonal dominance of $(\omega E_{I_j})^{-1} + (L_{I_j I_j} + U_{I_j I_j})/2$ induced by (4.9). Hence (4.9) implies (4.7) and Theorem 4.3 applies.

When (4.10) holds we have that

$$z_{I_j}\big((\omega E_{I_j I_j})^{-1} + L_{I_j I_j}\big) z_{I_j} = \frac{1}{2} z_{I_j}\big((2\omega^{-1} - 1) D_{I_j I_j} + M_{I_j I_j}\big) z_{I_j} \geq \gamma \big\| z_{I_j} \big\|^2$$

where the last inequality follows from the positive semidefiniteness of $M_{I_j I_j}$ (which is a consequence of the positive semidefiniteness of M), the positivity of $D_{I_j I_j}$ and $0 < \omega < 2$. Hence (4.10) implies (4.7) and again Theorem 4.3 applies. ∎

5. Parallel Solution of Linear Programs

As in previous works [6, 7, 8, 9], the linear programming problem can be converted to a symmetric linear complementarity problem by solving for the least 2-norm solution of the linear program. More specifically if we consider the linear program

$$\min \quad cx \quad \text{s.t.} \quad Ax \geq b, \; x \geq 0 \tag{5.1}$$

where $c \in R^n$, $b \in R^m$ and $A \in R^{m \times n}$, then its unique least 2-norm solution $\bar{x}$ is a solution of the quadratic program

$$\min \quad cx + \frac{\varepsilon}{2} xx \quad \text{s.t.} \quad Ax \geq b, \; x \geq 0 \tag{5.2}$$

for any $\varepsilon \in (0, \bar{\varepsilon}]$ for some $\bar{\varepsilon} > 0$ [6]. The quadratic dual [4] of (5.2) leads to the following quadratic minimization on the nonnegative orthant R^{m+n}_+

$$\min_{(u,v) \geq 0} \theta(u, v) := \min_{(u,v) \geq 0} \frac{1}{2} \left\| A^T u + v - c \right\|_2^2 - \varepsilon b u \tag{5.3}$$

Any solution $\big(u(\varepsilon), v(\varepsilon)\big)$ of (5.3) with $\varepsilon \in (0, \bar{\varepsilon}]$ leads to the unique least 2-norm solution $\bar{x}$ of (5.1) through the relationship

$$\bar{x} = \frac{1}{\varepsilon} \big(A^T u(\varepsilon) + v(\varepsilon) - c \big), \; \forall \varepsilon \in (0, \bar{\varepsilon}] \tag{5.4}$$

Problem (5.3) is exactly of the form (1.2) and hence can be solved by the parallel GP-SOR procedures of the previous section.

We describe now some of our numerical experiments on the Parallel GP-SOR Algorithm 4.1 for the solution of the linear program (5.1) by converting it to the quadratic minimization problem (5.3) for a sufficiently small but fixed ε. The linear programs were randomly generated with various densities d of the constraint matrix A, with the values of the nonzero elements of the matrix distributed uniformly in the interval [-100, 100]. A density coefficient $d \in [0, 1]$ indicates that on the average dn of the elements of each row of A are nonzero and are randomly distributed throughout the row. Thus a d close to zero means lots of zeros in A while a d close to 1 means very few zeros in A. The vectors c and b were chosen so that the linear program (5.1) had a pre-specified solution which was not necessarily unique. The stopping criterion for Algorithm 4.1 was that the norm of

the difference between successive iterates was less than a certain tolerance, usually 10^{-6}. With a fixed $\varepsilon = 1.5$ and an $\omega \in [1.2, 1.8]$ the parallel algorithm led to a solution of the linear program x which was primal feasible to within a 10^{-6} relative tolerance, that is $\|(Ax - b)_+\|_\infty / \|(-b)_+\|_\infty \leq 10^{-6}$ and an objective function value accurate to 3 significant places. Since the main purpose of the numerical experiments was comparison of the effectiveness of the parallel algorithm with different number of machines and densities, the issue of having very accurate solutions was not of great importance.

Algorithm 4.1 was implemented both on the CRYSTAL multicomputer [1] and the Sequent Balance 21000. CRYSTAL consists of 20 Vax 11/750's each with 2 megabytes of memory, connected by an 80 megabit/second token ring. The Sequent Balance 21000 is a multiprocessor that incorporates eight NS32032 processors running at 10MHz, each with a floating point unit, memory management and an 8-kbyte cache sharing a global memory via a 32-bit wide pipelined bus. The machine has 8-Mbytes of physical memory. The operating system DYNIX, is a version of Berkeley 4.2 bsd unix. The computational results for the CRYSTAL machine are depicted in Figures 1 to 4, while those for the Balance 21000 are shown in Figures 5 and 6.

Figure 1 shows the total computing time for four different densities d: 5, 10, 20 and 40 percent for a linear program with 100 constraints and 400 variables. For the low density of 5% the use of parallel computing is not beneficial because the relatively slow communication time between machines on the loosely coupled CRYSTAL multicomputer dominates the individual computation time on each machine. However for a density bigger than 5% we observe that as we increase the number of parallel machines, the total computing time decreases to a minimum for that specific density. We also note that the optimum number of machines that solves the problem in minimum time increases with density. Thus for d = 5, 10, 20 and 40 percent, the optimal number of machines is 1, 2, 3 and 4 respectively. Figure 2 shows the effect of parallelism on the time per iteration for Algorithm 4.2. The solid lines depict computationally observed times per iteration for each of the four different cases. We note that the number of machines that gives the least time per iteration again increases with density as expected, because the fraction of communication time relative to computing time per iteration decreases with increasing density. The dashed lines in Figure 2 show predicted time per iteration as a function of number of machines and is based on

the following formula derived from a simple operation count

$$T = \left[\alpha\left(\frac{7m}{r} + (4 + \frac{5 + 4dm}{r})n\right)\right] + [\beta n(r-1)] \tag{5.5}$$

where r is the number of machines employed. The first square bracket represents computing time per iteration for each machine including the line search, while the second square bracket represents the communication time between r machines. The constants α and β are determined by an a posteriori least square fit to the observed time per iteration. The closeness of the fit indicates that formula (5.5) is a reasonable approximation for time per iteration. Figure 3 shows total time and time per iteration for the case of 1000 constraints and 4000 variables with a density of 1%. Three machines is the optimal number which minimizes total time, while 4 machines minimize the time per iteration. This shows that for larger problems parallelism is useful in cutting computing time even for very sparse problems such as this one.

We now make some observations regarding the efficiency $E(r)$ of our algorithm which we define as follows:

$$E(r) := \frac{T(1)}{rT(r)} \tag{5.6}$$

where $T(r)$ is the total time for solving a given problem with r parallel machines. If we let $\bar{r}$ denote the optimal number of machines which solve the problem in least total time we observe from Figure 1 that $E(\bar{r})$ ranges between 60% and 44% for densities in the 10% to 40% range. However from Figure 3 we observe that $E(3)$=137% for the problem with 1000 constraints, 4000 variables and a density of 1%. This is a very encouraging result which indicates that for large sparse problems efficiency may be quite high. However we need to give an explanation why the efficiency exceeds 100%. The explanation is that our Parallel GP-SOR 4.1 changes with the number of machines used because the matrix K^i defined by (4.4) changes with the number of blocks into which M is divided. Thus strictly speaking we are not comparing the exact same algorithm when we evaluate the ratio $T(1)/rT(r)$ of (5.6). Nevertheless the expression is a valid measure of efficiency in the sense of comparing the anticipated reduced time of $T(1)/r$ to the observed time $T(r)$ for an algorithm with a variable K^i that depends on the partition of M. If the matrix K^i is held fixed for $i = 1$ and $i = r$ then we obtain efficiencies for exactly the same algorithm. These efficiencies are

shown in Figure 4, where an r-block algorithm on one machine is defined as the GP-SOR Algorithm 4.1 with the matrix K^i given by (4.4). We then have precisely the identical algorithm running on one machine and r machines and the efficiency will indeed be less than 100%. This efficiency is denoted by $E(1,r)$ and is defined by

$$E(1,r) := \frac{T(1,r)}{rT(r)} \tag{5.7}$$

where $T(1,r)$ is the time to solve the problem on one machine using the r-block algorithm.

Similar experiments were also performed on the much more closely coupled Balance 21000 multiprocessor. The key distinguishing features of these experiments are that we can handle larger problems (up to 10,000 constraints and 40,000 variables) and the efficiencies are higher on the Balance 21000 then those obtained on CRYSTAL. The Balance 21000 results are shown in Figures 5 and 6.

Acknowledgement

We are grateful to the Argonne National Laboratory Advanced Computer Research Facility for permission to use their Sequent Balance 21000 for some of our numerical experimentation. We are also indebted to the referees for valuable comments that have improved the paper.

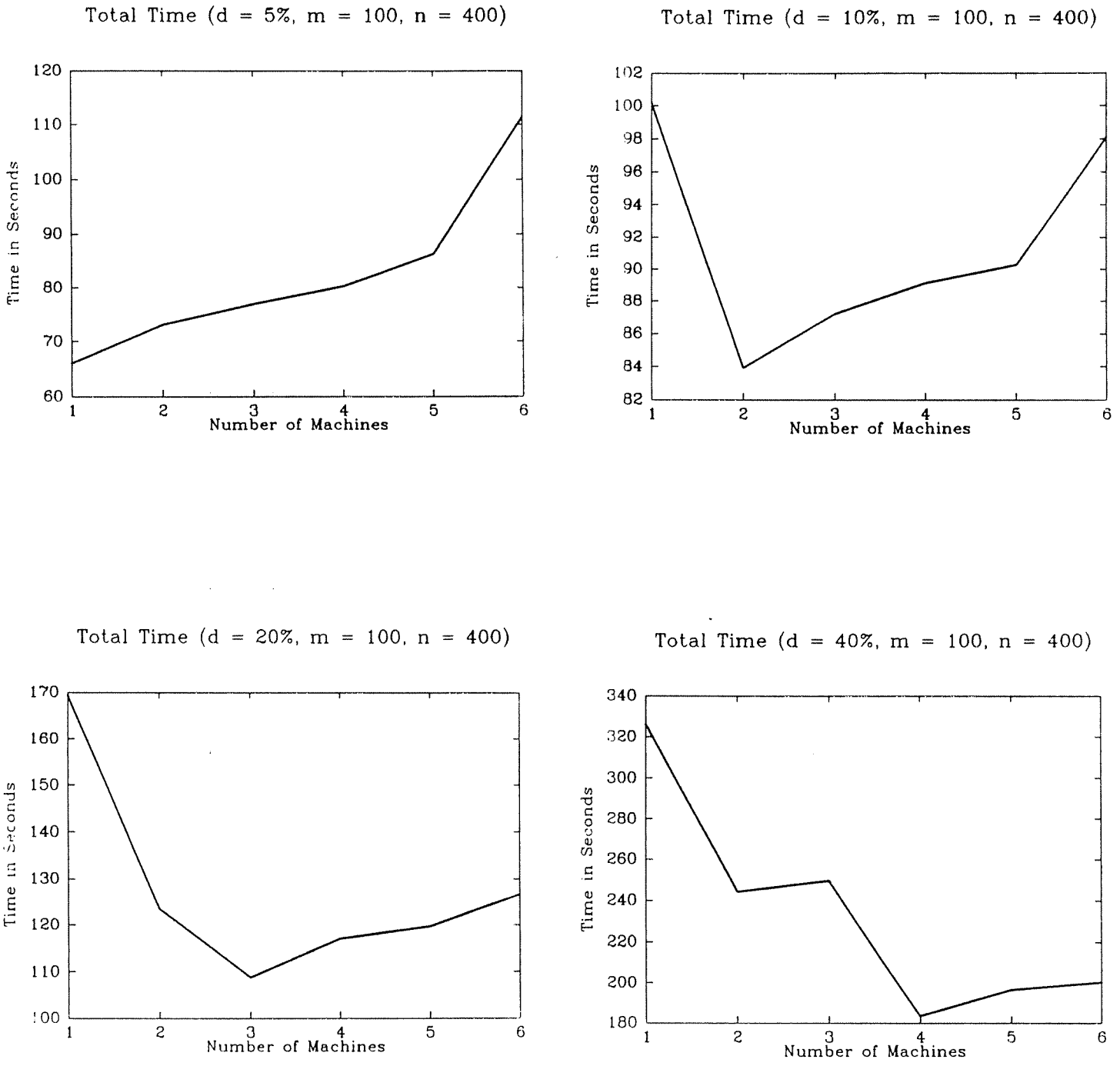

Fig. 1. CRYSTAL: Total time for parallel GP-SOR to solve linear program versus number of machines for various densities d. (Average of 5 randomly generated cases with 100 constraints and 400 variables.)

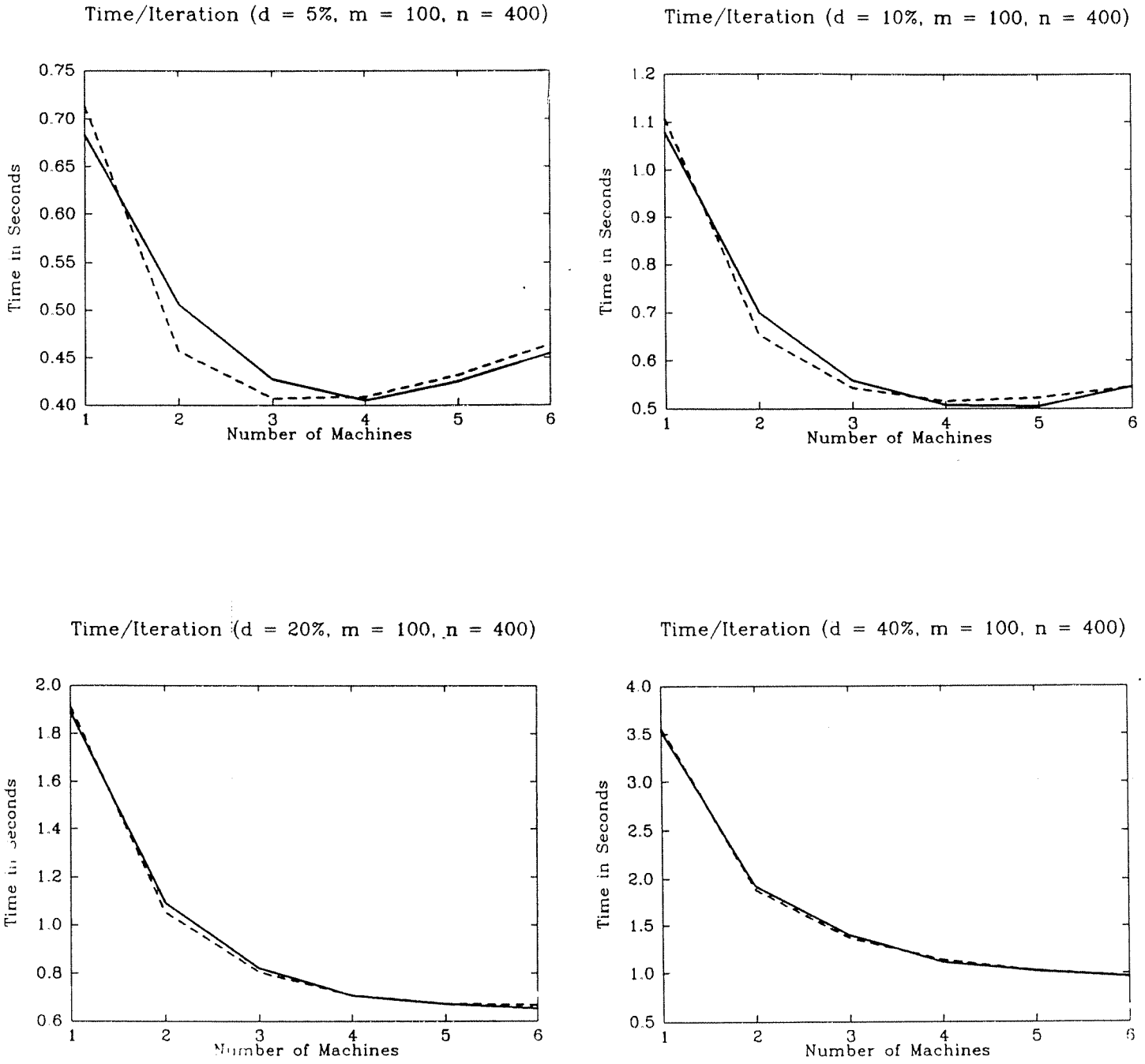

Fig. 2. CRYSTAL: Time per iteration for parallel GP-SOR to solve linear program versus number of machines for various densities d. Solid lines denote observed times while dashed lines represent a fit by formula (5.5). (Average of 5 randomly generated cases with 100 constraints and 400 variables.)

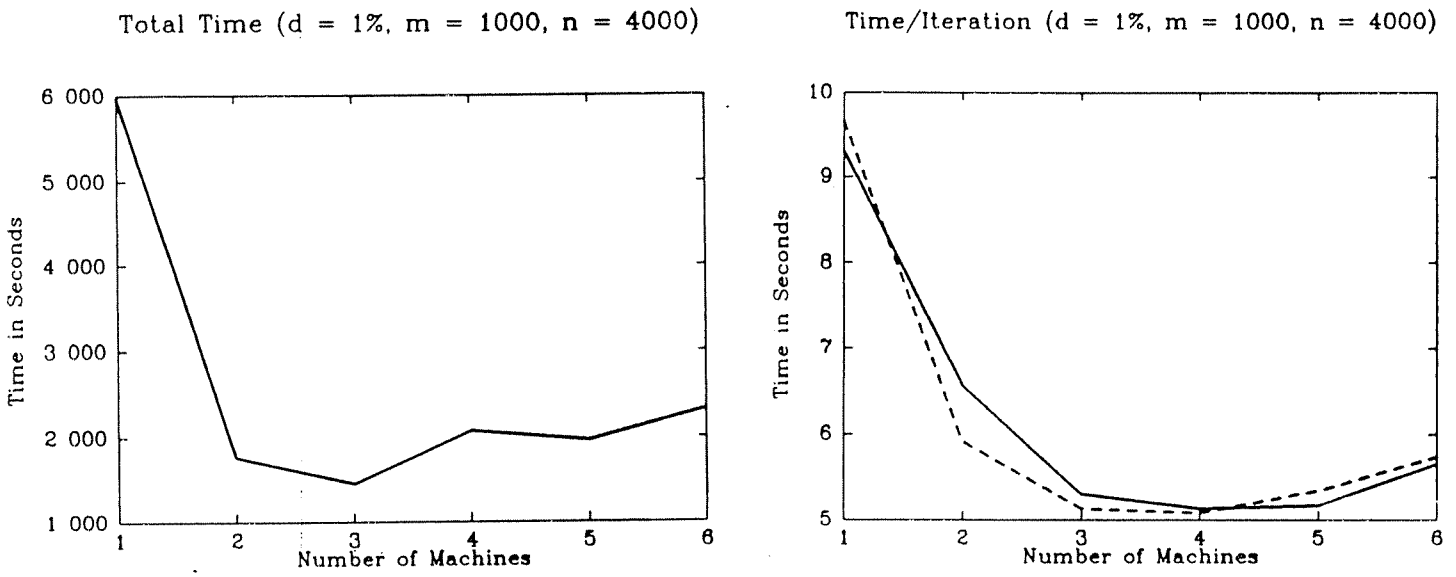

Fig. 3. CRYSTAL: Total time and time per iteration for parallel GP-SOR to solve a linear program versus number of machines with 1% density, 1000 constraints and 4000 variables. Solid line denotes observed time while dashed line represents a fit by Formula (5.5).

	$T(1,r)$	$T(r)$	$E(r)$	$E(1,r)$
No. of Blocks r	Time Sec. r-Block Alg. Single Processor	Time Sec. r-Block Alg. r Processors	$\frac{T(1)}{rT(r)}$	$\frac{T(1,r)}{rT(r)}$
1	5976	5976	——	——
2	2607	1772	169%	74%
3	2705	1459	137%	62%
4	4013	2071	72%	48%
5	3898	1964	61%	40%

Fig. 4. CRYSTAL: Efficiency table for parallel GP-SOR for r machines versus 1 machine for linear program with 1% density, 1000 constraints and 4000 variables.

$E(r)$: Efficiency comparing single-block single-machine algorithm with r-block r-machine algorithm.

$E(1,r)$: Efficiency comparing r-block single-machine algorithm with r-block r-machine algorithm.

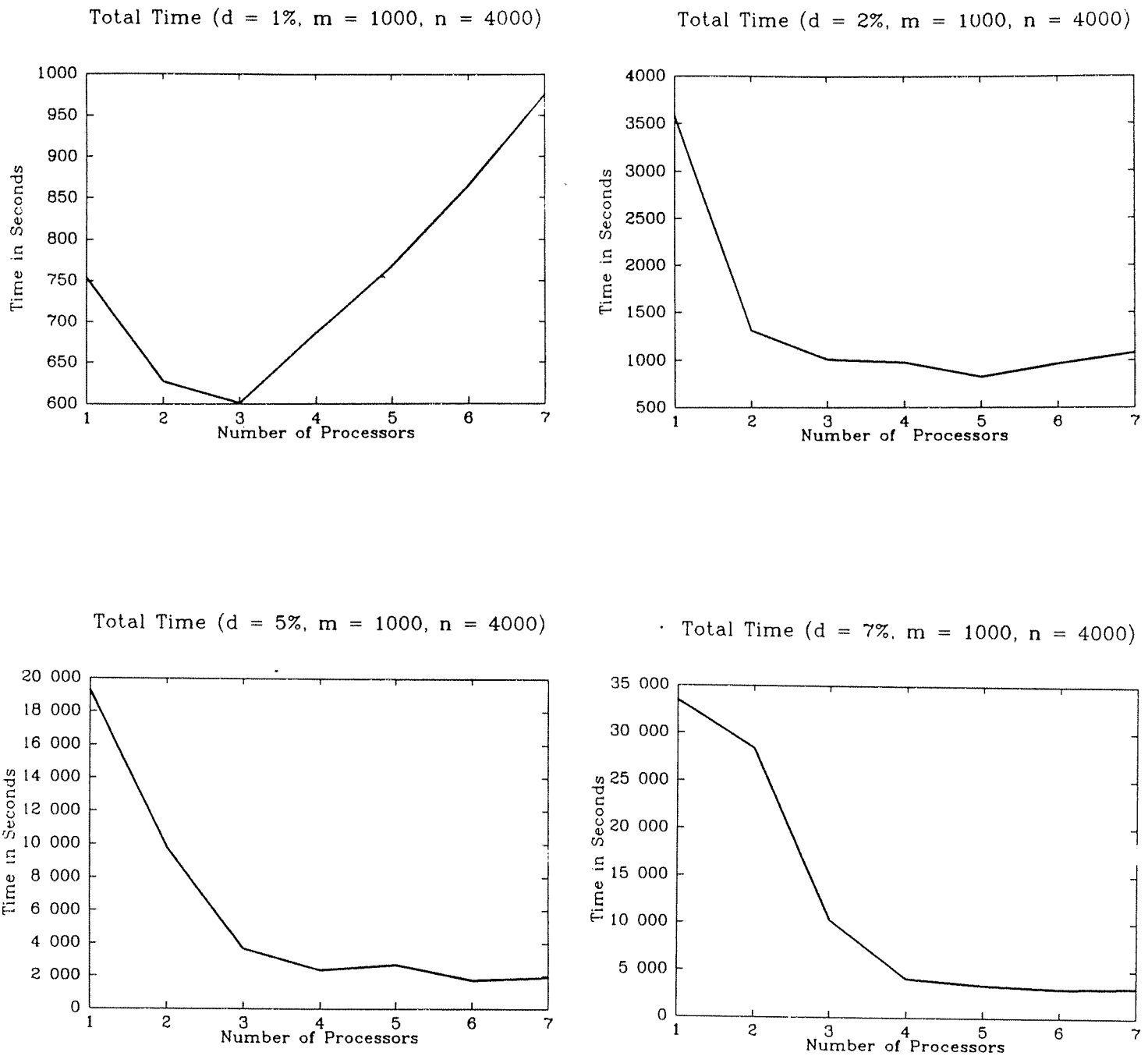

Fig. 5. Balance 21000: Total time for parallel GP-SOR to solve linear program versus number of processors for various densities d. (Average of 5 randomly generated cases with 1000 constraints and 4000 variables.)

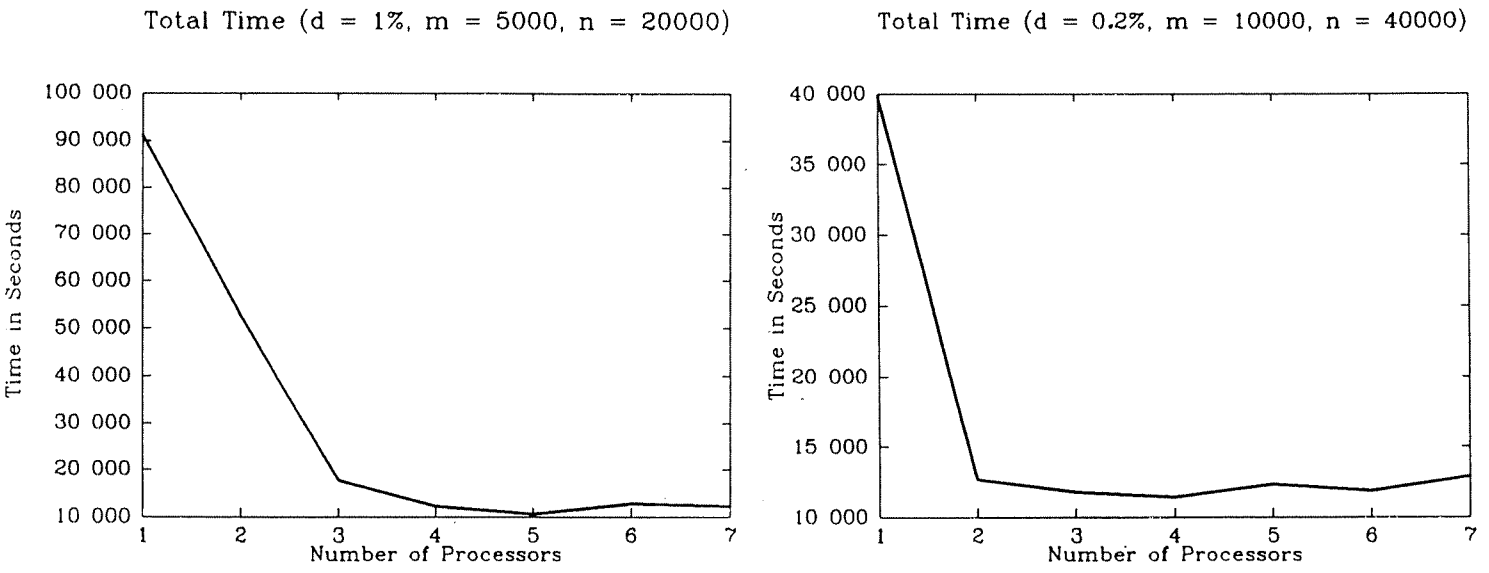

Fig. 6. Balance 21000: Total time for parallel GP-SOR to solve linear program versus number of processors (d = density, m = number of constraints, n = number of variables), for the largest problems solved on the Balance 21000.

References

1. D. DeWitt, R. Finkel & M. Solomon: "The CRYSTAL multicomputer: Design and implementation experience", University of Wisconsin-Madison, Computer Sciences Department Report No. 553, 1984, to appear in IEEE Transactions on Software Engineering 1987.

2. E. S. Levitin & B. T. Poljak: "Constrained minimization methods", USSR Computational Mathematics and Mathematical Physics (English translation) 6, 1966, 1-50.

3. Y. Y. Lin & J. S. Pang: "Iterative methods for large convex quadratic programs: A survey", SIAM Journal on Control and Optimization 25, 1987, 383-411.

4. O. L. Mangasarian: "Nonlinear programming", McGraw-Hill, New York 1969.

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

6. O. L. Mangasarian: "Normal solution of linear programs", Mathematical Programming Study 22, 1984, 206-216.

7. O. L. Mangasarian: "Sparsity-preserving SOR algorithms for separable quadratic and linear programs", Computers and Operations Research 11, 1984, 105-112.

8. O. L. Mangasarian & R. De Leone: "Error bounds for strongly convex programs and (super)linearly convergent iterative schemes for the least 2-norm solution of linear programs", University of Wisconsin-Madison, Computer Sciences Department Report No. 631, February 1986, to appear in Applied Mathematics and Optimization.

9. 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.

10. O. L. Mangasarian & T.-H. Shiau: "Lipschitz continuity of solutions of linear inequalities, programs and complementarity problems", SIAM Journal on Control and Optimization 25, 1987, 583-595.

11. K. G. Murty: "On the number of solutions to the complementarity problem and spanning properties of complementarity cones", Linear Algebra and Its Applications 5, 1972, 65-108.

12. J. W. Ortega: "Numerical analysis a second course", Academic Press, New York 1972.