NUMERICAL LINEAR ALGEBRA WITH APPLICATIONS
Numer. Linear Algebra Appl., **5,** 11–31 (1998)

A Parallel Multisplitting Solution of the Least Squares Problem

R. A. Renaut*

Department of Mathematics, Arizona State University, Tempe, AZ 85287-1804, USA

The linear least squares problem, $\min_x \|Ax-b\|_2$, is solved by applying a multisplitting(MS) strategy in which the system matrix is decomposed by columns into p blocks. The b and x vectors are partitioned consistently with the matrix decomposition. The global least squares problem is then replaced by a sequence of local least squares problems which can be solved in parallel by MS. In MS the solutions to the local problems are recombined using weighting matrices to pick out the appropriate components of each subproblem solution. A new two-stage algorithm which optimizes the global update each iteration is also given. For this algorithm the updates are obtained by finding the optimal update with respect to the weights of the recombination. For the least squares problem presented, the global update optimization can also be formulated as a least squares problem of dimension p. Theoretical results are presented which prove the convergence of the iterations. Numerical results which detail the iteration behavior relative to subproblem size, convergence criteria and recombination techniques are given. The two-stage MS strategy is shown to be effective for near-separable problems.

KEY WORDS least squares; *QR* factorization; iterative solvers; parallel algorithms; multisplitting

1. Introduction

We consider the solution of the overdetermined system of linear equations

$$Ax = b \tag{1.1}$$

where A is an $m \times n$ ($m \geq n$) real matrix of rank n and x and b are vectors of length n. Direct solutions of this system can be obtained by a least squares algorithm for

$$\min_{x \in \mathcal{R}^n} ||Ax - b||_2 \tag{1.2}$$

* Correspondence to R. A. Renaut, Department of Mathematics, Arizona State University, Tempe, AZ 85287-1804, USA.

CCC 1070–5325/98/010011– 21 $17.50

Received 25 June 1995
Revised 1 May 1997

via the QR factorization of A [9]. Typical methods for computing the QR decomposition use Householder transformations, Givens transformations, or the Gram–Schmidt process. One possible approach to the parallelization of these least squares algorithms therefore involves the determination of parallel algorithms for orthogonal transformations [4].

Another straightforward method symmetrizes (1.1) by forming the normal equations

$$A^{\mathrm{T}}Ax = A^{\mathrm{T}}b \tag{1.3}$$

This system can be solved directly using Gaussian elimination or iteratively using any standard method such as conjugate gradients or a Krylov-subspace algorithm [1]. Parallelization of these algorithms can proceed at the matrix operation level, in which appropriate data mapping allows for efficient realizations of matrix–vector update operations [15].

Each of the parallelization strategies mentioned above has the advantage that all convergence characteristics of the serial algorithm are maintained because the serial algorithm itself is not modified. On the other hand, considered separately, each detail of the algorithm may pose conflicting demands for efficient parallelism. Also, the approach is potentially very time intensive because of lack of portability across architectures.

The alternative approach considered here is to develop new algorithms which have great potential for parallelism, are essentially architecture-independent algorithms and use as much serial expertise as possible. The multisplitting (MS) philosophy introduced by O'Leary and White [14] for the solution of regular systems of equations, meets both goals and could be applied at the system level for the solution of the normal equations (1.3). A direct implementation, however, requires the formation of the operator $A^{\mathrm{T}}A$, which is not desirable. Instead we propose least squares MS algorithms for the solution of (1.2). Specifically, MS is an iterative technique which uses domain partitioning to replace a large-scale problem by a set of smaller subproblems, each of which can be solved independently in parallel. The success of MS relies on an appropriate recombination strategy of the subproblem solutions to give the global solution.

This paper presents three least squares (LS) algorithms based on the MS approach;

(i) an LS algorithm with a standard MS approach to solution recombination,
(ii) an iterative refinement implementation of the LS algorithm,
(iii) a two-stage MSLS algorithm which solves a second LS problem to determine the optimal weights at the recombination phase.

Theoretical properties of these algorithms are determined and an estimate of their relative parallel computational costs, ignoring communication, are presented. A performance evaluation via numerical implementation is also provided.

The format of the paper is as follows. In Section 2 we review the linear MS algorithm introduced in [14]. The new algorithms designed for the least squares problem are also presented and estimates of their computational costs are provided. A theoretical analysis of the convergence properties of the algorithms is detailed in Section 3. Results of some numerical tests are reported in Section 4. Finally, conclusions and suggestions for future directions of the research are discussed in Section 5.

2. Description of multisplitting

2.1. Multisplitting for $Ax = b$

Iterative methods based on a single splitting, $A = M - N$, are well known [16]. Multisplittings generalize the splitting to take advantage of the computational capabilities of parallel computers. A multisplitting of A is defined as follows:

Definition 2.1. *Linear multisplitting (LMS). Given a matrix $A \in \mathcal{R}^{n\times n}$ and a collection of matrices $M^{(j)}, N^{(j)}, E^{(j)} \in \mathcal{R}^{n\times n}$, $j = 1 : p$, satisfying*

(i) $A = M^{(j)} - N^{(j)}$ *for each* j, $j = 1 : p$,
(ii) $M^{(j)}$ *is regular,* $j = 1 : p$,
(iii) $E^{(j)}$ *is a non-negative diagonal matrix,* $j = 1 : p$ *and* $\sum_{j=1}^{p} E^{(j)} = I$.

Then the collection of triples $(M^{(j)}, N^{(j)}, E^{(j)})$, $j = 1 : p$ is called a multisplitting of A and the LMS method is defined by the iteration:

$$x^{k+1} = \sum_{j=1}^{p} E^{(j)}(M^{(j)})^{-1}(N^{(j)}x^k + b), \quad k = 1, \ldots \tag{2.1}$$

The advantage of this method is that at each iteration there are p independent problems of the kind

$$M^{(j)}y_j^k = N^{(j)}x^k + b, \quad j = 1 : p \tag{2.2}$$

where y_j^k represents the solution to the local problem. The work for each equation in (2.2) is assigned to one (or a set of) processor(s) and communication is required only to produce the update given in (2.1). In general, some (most) of the diagonal elements in $E^{(j)}$ are zero and therefore the corresponding components of y_j^k need not be calculated. If the y_j^k are disjoint, $j = 1 : p$, the method corresponds to block Jacobi and is called non-overlapping. Then, the diagonal matrices $E^{(j)}$ have only zero and one entries. For overlapped subdomains the elements in $E^{(j)}$ need not be just zeros and ones but Frommer and Pohl [7] showed that the benefit of overlap is the inclusion of extra variables in the minimization for the local variables, and that the updated values on the overlapped portion of the domain should not be utilized, i.e., the weights are still zeros or ones.

Before we continue to develop the MS approach for the solution of the least squares problem it is useful to realize that MS can be seen as a domain decomposition algorithm, in which the update given by (2.2) only provides an updated solution for a portion of the domain. Specifically, the variable domain $x \in \mathcal{R}^n$ is partitioned according to $x = (x_1, x_2, \ldots, x_p)^{\mathrm{T}}$, where each subdomain has $x_i \in \mathcal{R}^{n_i}$ and $\sum_{i=1}^{p} n_i = n$, without overlap of subdomains. The solution y_j^{k+1} is then the solution with updated values only on the j^{th} portion of the partition. The same idea is applied to define a least squares multisplitting (LSMS) approach.

2.2. Linear least squares

For (1.2) the matrix A is partitioned into blocks of columns consistently with the decomposition of x into blocks as $A = (A_1, A_2, \ldots, A_p)$, where each $A_i \in \mathcal{R}^{m\times n_i}$. This is not unlike the row projection methods as described, for example, in [3], except that there the

decomposition of A is into blocks of rows. The two approaches are not equivalent. With the column decomposition $Ax = \sum_{i=1}^{p} A_i x_i$ (1.2) can be replaced by the subproblems

$$\min_{y \in \mathcal{R}^{n_i}} ||A_i y - b_i(x)||_2, \qquad 1 \leq i \leq p$$

where $b_i(x) = b - \sum_{j \neq i} A_j x_j = b - Ax + A_i x_i$. Clearly each of these subproblems is also a linear least squares problem, amenable to solution by QR factorization of submatrix A_i. Equivalently, denote the solution at iteration k by $x^k = (x_1^k, x_2^k, \ldots, x_p^k)$, then the solution at iteration $k+1$ is found from the solution of the local subproblems

$$\min_{y_i^{k+1} \in \mathcal{R}^{n_i}} ||A_i y_i^{k+1} - b_i(x^k)||, \qquad 1 \leq i \leq p \tag{2.3}$$

according to

$$x^{k+1} = \sum_{i=1}^{p} \alpha_i^{k+1} \bar{x}_i^{k+1} \tag{2.4}$$

The updated local solution to the global problem is given by

$$\bar{x}_i^{k+1} = (x_1^k, x_2^k, \ldots, x_{i-1}^k, y_i^{k+1}, x_{i+1}^k, \ldots, x_p^k) \tag{2.5}$$

where the non-negative weights satisfy $\sum_{i=1}^{p} \alpha_i^{k+1} = 1$ and the solutions of the subproblems (2.3) are denoted by y_i^{k+1}. This update equation is still valid for overlapped domains with the one–zero weighting scheme, except that the notation has to be modified in (2.3) to indicate that (2.3) is solved with respect to a larger block and that y_i^{k+1} in (2.5) is the update restricted to the local domain.

For block i of (2.4)

$$\begin{aligned} x_i^{k+1} &= (\alpha_i^{k+1} \bar{x}_i^{k+1})_i + \left(\sum_{\substack{j=1 \\ j \neq i}}^{p} \alpha_j^{k+1} \bar{x}_j^{k+1} \right)_i \\ &= \alpha_i^{k+1} y_i^{k+1} + \sum_{\substack{j=1 \\ j \neq i}}^{p} \alpha_j^{k+1} x_i^k \\ &= \alpha_i^{k+1} y_i^{k+1} + (1 - \alpha_i^{k+1}) x_i^k \end{aligned} \tag{2.6}$$

This is completely local and can be rewritten as

$$\begin{aligned} x_i^{k+1} &= x_i^k + \alpha_i^{k+1} (y_i^{k+1} - x_i^k) \\ &= x_i^k + \alpha_i^{k+1} \delta_i^{k+1} \end{aligned} \tag{2.7}$$

where now $\alpha_i^{k+1} \delta_i^{k+1}$ is the step taken on partition i. The update of $b_i(x^k)$ in (2.3) can then be expressed as:

$$b_i(x^{k+1}) = b_i(x^k) - \sum_{\substack{j=1 \\ j\neq i}}^{p} \alpha_j^{k+1} A_j \delta_j^{k+1} \tag{2.8}$$
$$= b_i(x^k) - \sum_{\substack{j=1 \\ j\neq i}}^{p} \alpha_j^{k+1} B_j^{k+1}$$

where $B_j^{k+1} = A_j \delta_j^{k+1}$. The overlapped update of $b_i(x^{k+1})$ follows similarly but does require communication. The basic LSMS algorithm using p processes follows:

Algorithm 2.1. *LSMS*

For all processes i, $1 \leq i \leq p$,
calculate $Q_i R_i = A_i$,
initialize $y_i^0 = x_i^0$, $k = 0$, $\alpha_i^0 = 0$, $\alpha_i^k = \alpha = \frac{1}{p}$, $b_i(x^0) = b$.
While not converged,
  $k = k + 1$
  calculate $A_i \delta_i = B_i$ *! Matrix vector update,*
  communicate B_i *to all processors ! Global communication,*
  update b_i *via (2.9).*
  Find y_i *to solve (2.3) ! Solve local least squares,*
  $\delta_i = y_i - x_i$
  $x_i = x_i + \alpha \delta_i$ *! Update* x_i,
  test for convergence locally,
  communicate convergence result to all processors,
end while.
End

This algorithm is highly parallel and completely load-balanced when the problem size is the same for each process. For the overlapped case, modifications of the algorithm are necessary, but because of the zero–one weightings used, these modifications are minor.

Testing for convergence can be carried out either locally or globally. In the former case it is only necessary to share a logical variable with all the other processes. Otherwise the vector x must be accumulated and a global check performed. Observe that this algorithm is presented as a slave-only model. A master–slave model requires only minor modification.

From (2.9) and defining the residual $r = b - Ax$

$$r(x^{k+1}) = r(x^k) - \sum_{j=1}^{p} \alpha_j^{k+1} A_j \delta_j^{k+1} \tag{2.9}$$

it is easy to see that iterative refinement requires only a minor modification of Algorithm 2.1. Specifically, after the first iteration, the update of b_i by (2.9) can be replaced by the update of r from (2.9) and in the update (2.7) we use $\delta_i^{k+1} = y_i^{k+1}$. The communication and computation costs are unchanged, but the local least squares problem (2.9) is replaced by

$$\min_{y_i^{k+1} \in \mathcal{R}^{n_i}} ||A_i y_i^{k+1} - r(x^k)||_2$$

for which the right-hand side is now the same for all subproblems. This algorithm is the MS analog of the iterative refinement procedure for least squares introduced by Golub [8]. In light of the investigation by Higham [13], and to give a fair comparison between methods, we have chosen to implement the iterative refinement (LSMSIR) using single precision residuals. Golub and Wilkinson [10] revealed, however, that the procedure is satisfactory only when the true residual vector is sufficiently small. We might expect, therefore, that LSMSIR will not offer improvement compared with LSMS. This is confirmed by the numerical experiments presented in Section 4.

Equation (2.7) suggests that convergence might be improved by use of a global update using a line search procedure. In particular, in [18] it was suggested that one choice would be to set $\alpha_i^{k+1} = \alpha_i = 1$ which amounts to the update $x_i^{k+1} = y_i^{k+1}$. Another improvement suggested employs a one dimensional line search dependent on $\alpha = \alpha_i^{k+1}$, or a p-dimensional minimization of the residual over the parameters $\{\alpha_i^{k+1}, 1 \leq i \leq p\}$. In the latter case this can be formulated as a least squares minimization

$$\min_{\alpha \in \mathcal{R}^p} ||D\alpha - r(x^k)||_2 \tag{2.10}$$

where $D \in \mathcal{R}^{m \times p}$ has columns $d_j^{k+1} = A_j \delta_j^{k+1}$. For $p << n$ this represents non-parallel overhead of a least squares solve, requiring the formation of the *QR* factorization of D, but it has the potential to improve the speed of convergence of the iteration.

Algorithm 2.2. *Optimal recombination ORLSMS*

For all processors i, $1 \leq i \leq p$
calculate $Q_i R_i = A_i$,
initialize x_i, $k = 1$,
calculate $A_i x_i = B_i$,
form b_i *via (2.9) and* $r = b_i - B_i$,
find y_i *to solve (2.3)*
$\delta_i = y_i - x_i$.
While not converged
    calculate $A_i \delta_i = d_i$
    communicate d_i *to all processors*
    calculate $Q^D R^D = D$ *and solve (2.10) for* α
    update $x_i = x_i + \alpha_i \delta_i$
    test convergence
    update $B_i = B_i + \alpha_i d_i$
    communicate B_i *to all processors*
    form b_i *via (2.9) and* $r = b_i - B_i$
    find y_i *to solve (2.3)*
    $\delta_i = y_i - x_i$,
end while.
End

Note that in this version of the algorithm we have employed a redundant update in which every process solves the outer least squares problem for α. This does have the advantage

that each process can keep a record of the global update, provided that the initial guess is known to each process. The per iteration communication cost is now two global vector exchanges. Hence, the per iteration communication costs are twice those without the optimal recombination. Observe, also, the *OR* algorithm can be modified to act on the *IR* algorithm. Again a master–slave mode of the algorithm follows in a straightforward manner.

2.3. Computation performance analysis

We have already remarked on the communication costs of the algorithms presented in the previous section. Moreover, our intent here is to evaluate the potential parallelism in these MS algorithms, without consideration of communication bandwidths, cache size or other architecture-dependent factors. Therefore, we do have to define a measure of parallel efficiency. To do this we estimate to the highest order the computation costs associated with each algorithm as compared with a direct solve by the least squares solution of the whole system. We make an assumption that the *QR* factorization is calculated by Householder transformations, which are a little cheaper than the Givens rotations.

Serial cost of the *QR* solution of (1.2) is given by

$$C_S = 2n^2(m - n/3) + mn + n^2$$

where the first term is for the determination of R, the second for the update of b and the final for the back substitution to give x. The per process cost for Algorithm 2.1 is

$$C_P = 2n_i^2(m - n_i/3) + K(2mn_i + n_i^2)$$

where K is the total number of iterations required to achieve a specified convergence criterion. The first term represents the formation of the *QR* factorization of A_i. Costs to first order in m or n are ignored. When *IR* is incorporated, the costs are unchanged, provided the residual is calculated to the same precision as the remainder of the operations.

Algorithm 2.2 does have a basic iteration cost that is greater because of the *QR* factorization of D. Hence in this case

$$C_{POR} = 2n_i^2(m - n_i/3) + K(2mn_i + n_i^2 + 2p^2(m - p/3) + mp + p^2)$$

The percentage parallel efficiency achieved is given by

$$E = \frac{C_S \times 100}{C_P \times p} \tag{2.11}$$

where p is the number of processes used in the calculation. Moreover, for the *OR* algorithms C_P is replaced by C_{POR}. Measurements of these efficiencies are given in our presentation of the numerical results. Note that when overlap is introduced into the systems, the formulae are still valid but with n_i replaced by $n_i = n/p + o_i$ where o_i determines the amount of the overlap.

3. Convergence

3.1. *Convergence of the linear LSMS algorithm*

In order to investigate the convergence of Algorithm 2.1 we need to determine the linear iterative scheme satisfied by the global update x^{k+1}. We shall assume that the system matrix A in (1.1) has full rank so that the solution to (1.2) exists and is unique.

Because the matrix A is of full column rank, so are the submatrices A_i. Therefore the solution of the least squares problem (2.3) exists, is unique and is given by

$$y_i = (A_i^{\mathrm{T}} A_i)^{-1} A_i^{\mathrm{T}} b - \sum_{\substack{j=1 \\ j \neq i}}^{p} (A_i^{\mathrm{T}} A_i)^{-1} A_i^{\mathrm{T}} A_j x_j^k$$

Hence the ith component of the global update (2.4) can be written

$$x_i^{k+1} = \sum_{j=1}^{p} C_{ij} x_j^k + \alpha_i (A_i^{\mathrm{T}} A_i)^{-1} A_i^{\mathrm{T}} b$$

where

$$C_{ij} = \begin{cases} (1-\alpha_i) I_{n_i} & i = j \\ -\alpha_i (A_i^{\mathrm{T}} A_i)^{-1} A_i^{\mathrm{T}} A_j & i \neq j \end{cases} \tag{3.1}$$

Here I_{n_i} is the identity matrix of order n_i, the matrices C_{ij} are the ij blocks of a matrix C, with block size $n_i \times n_j$, and $C \in \mathcal{R}^{n \times n}$, consistent with $x \in \mathcal{R}^n$. Equivalently, (2.4) becomes

$$x^{k+1} = C x^k + \tilde{b} \tag{3.2}$$

where $\tilde{b}_i = \alpha_i (A_i^{\mathrm{T}} A_i)^{-1} A_i^{\mathrm{T}} b$. Moreover, $C = C(\alpha)$, so that convergence depends on the parameter α.

It is easily seen that iterative refinement for Algorithm 2.1 leads exactly to (3.2). The convergence behavior of both algorithms is thus, the same. They differ only in implementation and, consequently, effects of finite precision arithmetic.

Theorem 3.1. *The iterative scheme defined by (3.2) with $\alpha_i = \alpha = 1$ is a block Jacobi iterative scheme for the solution of the normal equations (1.3).*

Proof
Set $\alpha = 1$ in (3.1). Then it is clear that the equivalent form of (3.2) is

$$M x^{k+1} = N x^k + \bar{b} \tag{3.3}$$

where M is a diagonal matrix with entries $A_i^{\mathrm{T}} A_i$, N is given by

$$N_{ij} = \begin{cases} 0 & i = j \\ -A_i^{\mathrm{T}} A_j & i \neq j \end{cases}$$

and $\bar{b}_i = A_i^{\mathrm{T}} b$. Therefore (3.2) solves the equation

$$(M - N) x = \bar{b}$$

and from (3.1), $M - N = A^{\mathrm{T}}A$ and $\bar{b} = A^{\mathrm{T}}b$. ■

Corollary 3.1. *The iterative scheme defined by (3.2) for fixed α, $0 < \alpha < 1$, is a relaxed block Jacobi scheme for the solution of the normal equations (1.3).*

The condition for the convergence of (3.2) when $\alpha = 1$ is now immediate because the system matrix $A^{\mathrm{T}}A$ is symmetric and positive definite (SPD), (see Corollary 5.47 in Chapter 7 of [2]).

Theorem 3.2. *The iteration defined by (3.2) with $\alpha_i = \alpha = 1$ converges for any initial vector x^0 if and only if $M + N$ is positive definite.*

Moreover, the Gauss–Seidel implementation of Algorithm 2.1 necessarily converges because for successive-over-relaxation (SOR) convergence is given by Corollary 5.48 in Chapter 7 of [2].

Theorem 3.3. *The block SOR method converges for all $0 < \alpha < 2$.*

It is now helpful to introduce the notation $\mu = \rho(M^{-1}N)$, the spectral radius of $M^{-1}N$, $\tilde{\mu} \in \sigma(M^{-1}N)$, an element in the spectrum of $M^{-1}N$ and μ^* the smallest eigenvalue of $M^{-1}N$.

Lemma 3.1. *All the eigenvalues $\tilde{\mu}$ of $M^{-1}N$ satisfy $\tilde{\mu} < 1$.*

Proof
The system matrix defined by the normal equations is SPD. M is also SPD and the iteration with $\alpha = 1$ is symmetrizable, i.e., there exists a matrix W, $\det W \neq 0$, such that $W(I - M^{-1}N)W^{-1}$ is SPD [11]. In this case a choice for W is $W = M^{1/2}$. Therefore, by Theorem 2.2.1 in [11] the eigenvalues of $M^{-1}N$ are real and satisfy $\tilde{\mu} < 1$. ■

Theorem 3.4. *The relaxed iteration converges for any sufficiently small positive α satisfying*

$$0 < \alpha < \frac{2}{1 - \mu^*}$$

Proof
The result follows from the observation that the iteration matrix of the relaxed block Jacobi iteration is given by

$$H = (1 - \alpha)I + \alpha M^{-1}N$$

Therefore if $\lambda \in \sigma(H)$, we have $\lambda = 1 - \alpha(1 - \tilde{\mu})$, and $\rho(H) < 1$ if and only if $0 < \alpha < \frac{2}{1-\mu^*}$. ■

By Theorem 3.2 $\rho(M^{-1}N) < 1$ when $M + N$ is positive definite and therefore we can conclude $\mu^* > -1$ in the above to give:

Corollary 3.2. *The relaxed block Jacobi iteration converges if $M + N$ is positive definite and $0 < \alpha < 1$.*

Remark 1
The row projection methods [3] use block algorithms to solve the set of equations

$$\begin{cases} AA^{\mathrm{T}}y = & b \\ x = & A^{\mathrm{T}}y \end{cases}$$

with and without preconditioning. On the other hand, the column decomposition introduced here solves the normal equations.

3.2. Convergence of the ORLSMS algorithm

Unlike Algorithm 2.1 the convergence of Algorithm 2.2 cannot be investigated by the determination of a linear iteration for x. Rather, Algorithm 2.2 needs to be seen as a procedure for the minimization of the non-linear function $f(x) = ||Ax - b||_2^2$. As such Algorithm 2.2 can then be interpreted as a modification of the parallel variable distribution (PVD) algorithm for non-linear functions, $f(x)$, introduced by Ferris and Mangasarian [6].

In the PVD, minimization of f occurs in two stages, a parallel stage and a synchronization stage. The former corresponds to the determination of the parallel solution of the local problem (2.3) but with the additional local update of the non-local variables by a scaling of the search direction for those variables. These search directions are a set of vectors d^k for iterates x^k, usually given by $d^k = -\frac{\nabla f(x^k)}{||\nabla f(x^k)||}$. A version for which these search directions are taken to be zero is denoted by PVD0. In the synchronization stage, x^{k+1} is updated via the minimization of f, but now with respect to the weightings for the linear combination of x^k with the local solutions $\bar{x}_i^{k+1}$. The minimization is constrained by the requirement that the update x^{k+1} is a strictly convex combination of x^k and the $\bar{x}_i^{k+1}$. This ensures $f(x^{k+1}) < f(x^k)$ and hence a decrease in the objective function each iteration. Convergence of the PVD algorithm for $f \in LC_K^1(\mathcal{R}^n)$ is given by Theorem 2.1 in [6]. Here the set $LC_K^1(\mathcal{R}^n)$ is the set of functions with Lipschitz continuous first partial derivatives on $\mathcal{R}^n$ with Lipschitz constants K.

Theorem 3.5. *For a bounded sequence* $\{d^k\}$*, either the sequence* $\{x^{k+1}\}$ *terminates at a stationary point* $x^{\bar{k}}$*, i.e., a point at which* $\nabla f(x) = 0$*, or each of its accumulation points is stationary and* $\lim_{k\to\infty} \nabla f(x^k) = 0$.

The proof of this theorem employs not only the requirement that f has a Lipschitz continuous gradient, that the sequence $\{d^k\}$ is bounded, but also that in the synchronization step $f(x^{k+1}) \leq \frac{1}{p}\sum_{l=1}^{p} f(\bar{x}_l^{k+1})$, because of the convex update of x^k.

In Algorithm 2.2 the search directions are zero, and thus, ORLSMS is actually a version of PVD0. Furthermore, for $f(x) = ||Ax - b||_2^2$ we have

$$||\nabla f(y) - \nabla f(x)||_2^2 = ||2A^{\mathrm{T}}A(y - x)||_2^2$$

and $f \in LC_K^1(\mathcal{R}^n)$ with $K = 2\rho(A^{\mathrm{T}}A)$. To prove convergence for ORLSMS it therefore only remains to check that the modification of the synchronization stage employed in Algorithm 2.2 satisfies the assumption $f(x^{k+1}) \leq \frac{1}{p}\sum_{l=1}^{p} f(\bar{x}_l^{k+1})$ used in the proof of Theorem 3.5 But at synchronization in Algorithm 2.2, x is updated by (2.4), for which $r(\bar{x}_l^{k+1}) = f(\bar{x}_l^{k+1}) < r(x^k)$. Therefore, this condition is necessarily satisfied, otherwise

$\frac{1}{p}\sum_{l=1}^{p} f(\bar{x}_l^{k+1}) < \sum_{l=1}^{p} \alpha_l^{k+1} f(\bar{x}_l^{k+1})$ and the minimum is not found, contradicting the minimization of the outer stage. Thus, Theorem 3.5 applies for the ORLSMS algorithm.

Furthermore, the function $f(x) = ||Ax - b||_2^2$ is strongly convex,

$$f(y) - f(x) - \nabla f(x)(y - x) \geq \frac{K}{2}||y - x||_2^2, \forall x, y \in \mathcal{R}^n$$

where $K = 2\rho(A^{\mathrm{T}}A)$. Therefore, the linear convergence result of Ferris and Mangasarian [6] also applies.

Theorem 3.6. *The sequence $\{x^k\}$ defined by the algorithm ORLSMS converges linearly to the unique solution x_{LS} of (1.2) at the linear root rate*

$$||x^k - x|| \leq \left(\frac{f(x^0) - f(x_{\mathrm{LS}})}{\rho(A^{\mathrm{T}}A)}\right)^{1/2}\left(1 - \frac{1}{p}\left(\frac{K}{K_1}\right)^2\right)^{k/2}$$

where K_1 is the Lipschitz constant for $\nabla_l f(x_l)$.

On the contrary, however, when we seek to apply Theorem 3.5 to the LSMS algorithm, we do not have an update at the synchronization stage for which it is necessary that $f(x^{k+1}) \leq f(x^k)$. In particular, this reduction in the objective function is just the reduction in the residual function and we see, not unexpectedly, that when we determine the requirement for this decrease we obtain exactly the restriction on α as given by Theorem 3.4 In order to force convergence, the implementation used actually updates x, either as given by (2.7) or, when this update does not lead to a decrease in the objective function, the update to x is taken as the local solution x_i which leads to the minimum of f for that iteration. Therefore, the convergence theory for the PVD is useful in this case for determining the weakness of the relaxed splitting and immediately suggests the modification required to force convergence.

4. Numerical results

Here, numerical results of tests of the algorithms in Section 2 are presented for three examples. Further results can be found in [12] and [17]. The first example comes from a table-flatness problem and generates a structured matrix. This is one of the structured matrices used by Duff and Reid in [5], and for the case we chose generates a matrix of size 840×484 with 2 518 non-zero entries, which is reasonably conditioned, condition number ≈ 105. Results presented for this test case are referred to as results for measured data. To indicate how the algorithms perform for sparse matrices with arbitrary structure no attempt was made to fit the splitting to the structure of the problem by reordering the unknowns. For dense matrices, we used two matrices of size 600×500 with random entries generated according to a normal probability distribution and a uniform probability distribution, respectively. The condition numbers of these matrices were approximately 20 and 398, respectively. The results presented for these matrices are referred to as results for normal and uniform data, respectively. Note, all matrices used in the evaluation were reasonably well conditioned.

Table 1. Comparison of four methods, for tolerance 10^{TE}, $TE = -3$ and $TE = -5$. Measured data

Algorithm			LSMS		LSMSIR		ORLSMS		ORLSMSIR	
P	O	TE	K_R	K_E	K_R	K_E	K_R	K_E	K_R	K_E
2	0	−3	12	49	12	50	6	23	6	23
	0	−5	31	86	31	89	14	41	14	43
3	0	−3	15	70	15	70	6	23	6	23
	0	−5	45	91	45	169	15	43	15	43
4	0	−3	19	89	19	89	6	23	6	23
	0	−5	57	146	57	N	15	43	15	43
5	0	−3	22	106	22	106	6	23	6	23
	0	−5	69	206	69	406	15	45	15	43
6	0	−3	25	123	25	123	6	21	6	21
	0	−5	80	188	80	252	15	41	15	43
7	0	−3	28	137	28	137	6	23	6	23
	0	−5	90	280	90	281	15	43	15	43
8	0	−3	30	84	30	84	6	232	6	230
	0	−5	75	N	75	N	15	N	14	N

Table 2. Comparison of four methods, for tolerance 10^{TE}, $TE = -3$ and $TE = -5$. Normal data

Algorithm			LSMS		LSMSIR		ORLSMS		ORLSMSIR	
P	O	TE	K_R	K_E	K_R	K_E	K_R	K_E	K_R	K_E
2	0	−3	47	172	47	172	35	129	35	129
		−5	220	489	220	599	126	343	127	355
3	0	−3	58	243	58	243	43	183	43	183
		−5	386	884	384	1 072	198	564	196	561
4	0	−3	69	287	69	287	47	213	47	213
		−5	519	1 272	525	N	233	682	235	679
5	0	−3	75	316	75	316	50	230	50	230
		−5	668	1 703	665	2 036	262	760	261	756
6	0	−3	80	339	80	339	51	245	51	245
		−5	786	2 012	789	2 427	284	794	285	816
7	0	−3	85	343	85	343	52	249	52	249
		−5	877	N	890	N	290	812	293	848
8	0	−3	90	358	90	358	53	255	53	255
		−5	992	N	992	N	300	878	298	884

A representative selection of the results is given in Tables 1–7. The notation is as follows:

P—number of processors
O—overlap between domains
TE—10^{TE} is the tolerance
K_R—number of iterations to convergence 10^{TE} in l_2 norm of the relative error
K_E—number of iterations to convergence 10^{TE} in l_2 norm of the relative residual
N—convergence was not achieved to this tolerance after 2 500 iterations

Tables 1–3 present a comparison of the four algorithms without overlap at tolerances 10^{-3} and 10^{-5}. Tables 4–7 show how the algorithms perform when overlap is incorporated. All calculations are in single precision.

Table 3. Comparison of four methods, for tolerance 10^{TE}, $TE = -3$ and $TE = -5$. Uniform data

Algorithm			LSMS		LSMSIR		ORLSMS		ORLSMSIR	
P	O	TE	K_R	K_E	K_R	K_E	K_R	K_E	K_R	K_E
2	0	−3	42	158	42	158	30	109	30	109
		−5	218	1 948	217	2 158	130	N	130	N
3	0	−3	59	204	59	204	42	159	42	163
		−5	349	N	351	N	255	N	300	N
4	0	−3	70	247	70	247	51	201	51	205
		−5	481	N	479	N	312	N	317	N
5	0	−3	75	248	75	248	53	336	53	368
		−5	586	N	587	N	625	N	639	N
6	0	−3	86	268	86	268	62	391	62	405
		−5	689	N	690	N	527	N	568	N
7	0	−3	98	280	98	280	70	437	70	315
		−5	745	N	739	N	521	N	560	N
8	0	−3	99	294	99	294	72	361	72	333
		−5	889	N	898	N	599	N	610	N

Table 4. Effect of overlap on convergence for ORLSMS and ORLSMSIR and error convergence 10^{-4}. Measured data

		Splitting						
Algorithm	O	2	3	4	5	6	7	8
ORLSMS	0	31	33	33	33	33	33	2 193
	10	31	33	33	33	31	33	310
	20	31	33	33	33	31	31	321
	30	33	33	33	33	33	31	31
ORLSMSIR	0	31	33	33	33	33	33	2 180
	10	31	33	33	33	31	33	308
	20	31	33	33	33	31	31	323
	30	33	33	33	33	33	31	31

Table 5. Effect of overlap on convergence for ORLSMS and ORLSMSIR and residual convergence 10^{-4}. Measured data

		Splitting						
Algorithm	O	2	3	4	5	6	7	8
ORLSMS	0	10	10	10	10	10	10	9
	10	10	10	10	10	10	10	19
	20	10	10	10	10	10	10	20
	30	10	10	10	10	10	10	10
ORLSMSIR	0	10	10	10	10	10	10	9
	10	10	10	10	10	10	10	19
	20	10	10	10	10	10	10	20
	30	10	10	10	10	10	10	10

Table 6. Effect of overlap on convergence for ORLSMS and ORLSMSIR and error convergence 10^{-4}. Uniform data

		Splitting						
Algorithm	O	2	3	4	5	6	7	8
ORLSMS	0	N	717	920	N	1 599	1 396	1 475
	10	104	273	369	470	428	504	561
	20	117	246	304	334	339	461	490
	30	136	219	260	315	369	400	407
ORLSMSIR	0	N	N	N	2 248	2 159	1 896	1 857
	10	97	279	342	491	432	498	575
	20	122	250	312	324	339	448	477
	30	124	202	260	311	365	386	396

Table 7. Effect of overlap on convergence for ORLSMS and ORLSMSIR and residual convergence 10^{-4}. Uniform data

		Splitting						
Algorithm	O	2	3	4	5	6	7	8
ORLSMS	0	72	105	144	162	191	201	227
	10	55	107	96	109	112	121	135
	20	66	64	97	103	106	109	121
	30	56	82	91	104	104	110	112
ORLSMSIR	0	72	105	145	162	192	201	227
	10	57	107	96	109	112	121	136
	20	66	80	97	100	106	109	121
	30	56	83	91	104	103	111	110

In summary the numerical results show:

(i) Convergence for a minimum residual solution is achieved more quickly than for a minimal error solution.
(ii) Iterative refinement does not improve the convergence rates, for either Algorithm 2.1 or 2.2, confirming the observation of Golub and Wilkinson [10].
(iii) Algorithm 2.2 has generally much better convergence properties for a minimum residual solution than Algorithm 2.1 because of the optimal recombination of the local solutions at each iteration.
(iv) Overlap can improve the rate of convergence. The amount of overlap to use is problem dependent. For a dense matrix the cost of the subproblem solution increases with overlap so that at some point more overlap is no longer beneficial. For separable or near separable problems the ideal overlap is often immediately clear. In other cases graph theoretic techniques may be needed to determine optimal groupings of variables.
(v) Overlap does not always reduce the number of outer iterations to convergence of the *OR* algorithms. A decrease in the objective function is guaranteed each iteration but the minimization with respect to the weights, α_i, may lead to different subspaces being weighted differently than in the non-overlapped case. Hence, faster convergence is not guaranteed, i.e., rate of convergence is dependent on the vectors α^k.

Figures 1–5 illustrate the results of Tables 1–7, using the estimate of percentage parallel efficiency given by (2.11). In Figures 1–3 the line types are O, $+$, X and $*$ for the algorithms LSMS, LSMSIR, ORLSMS and ORLSMSIR, respectively. In Figures 4 and 5 the line types $*$, O, X and $+$ indicate overlap 0, 10, 20 and 30, respectively. Efficiency for the random matrices is less than for the structured cases. Also, because overlap is more costly, the gain in rate of convergence is not recognized in terms of parallel efficiency when overlap is large. Efficiencies greater than 100 indicate speed-up of the split algorithm as compared with a straightforward direct *QR* solve. The introduction of *OR* is effective at improving parallel efficiency.

5. Conclusions

The algorithms presented in this paper provide a viable parallel strategy for the solution of the linear least squares problem. In particular, a two-level approach to minimization in which subproblem solutions are obtained independently, but then combined to give an optimal global update, is very successful. The method has been demonstrated to work not only for a sparse test example but also for dense random matrices. We conclude that the approach is of particular value for:

(i) Problems that are inherently near separable. In these cases the optimal local solutions quickly converge to the global optimum and the cost of each subproblem solve is relatively cheap. The method is then viable.
(ii) Large dense problems which are too memory intensive to be solved on a single processor machine. Although, in this case, parallel efficiency is very low, the algorithm provides an effective solution technique.

Furthermore, these algorithms have the additional advantages, compared with direct parallelization of the serial algorithm, of simplicity, portability and flexibility.

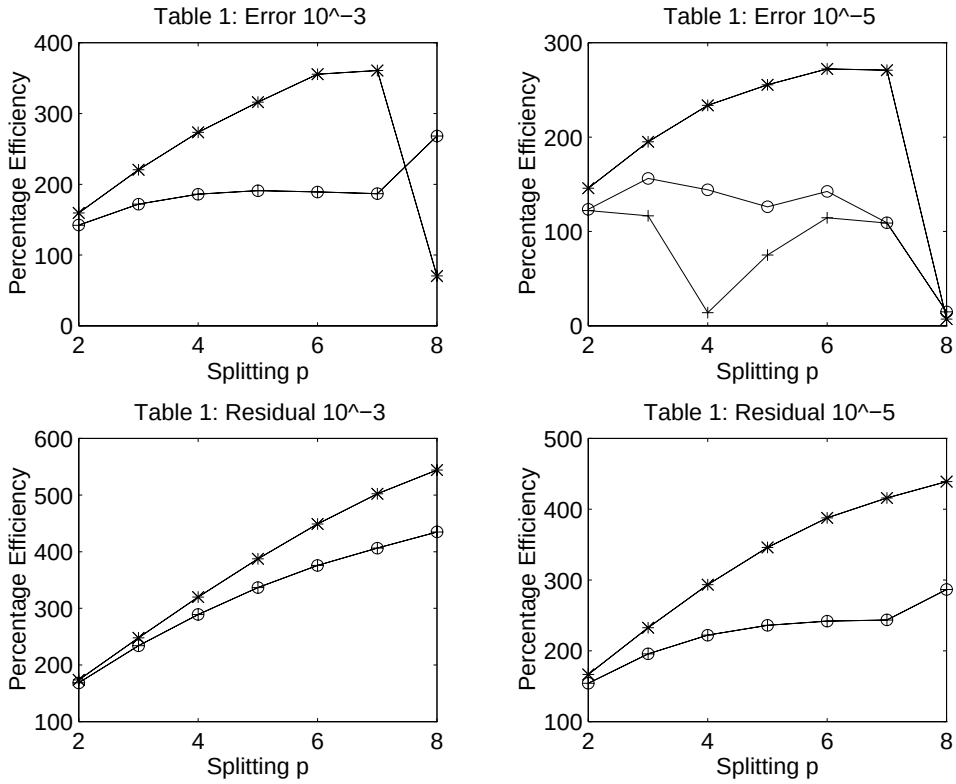

Figure 1. Comparison of parallel efficiency of algorithms for data from Table 1

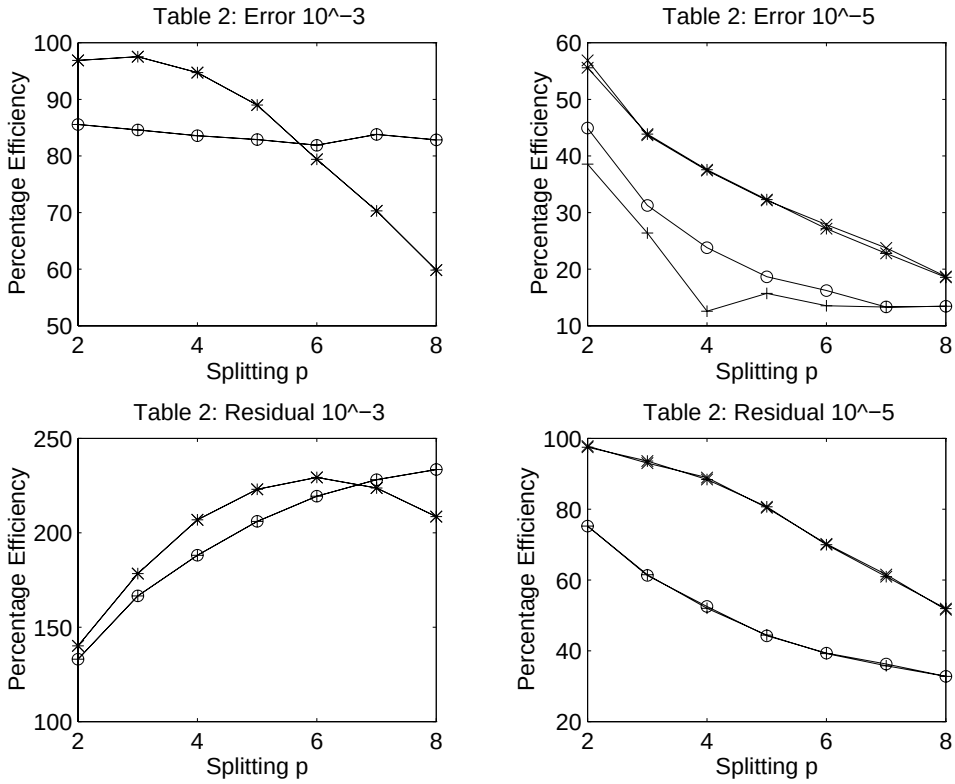

Figure 2. Comparison of parallel efficiency of algorithms for data from Table 2

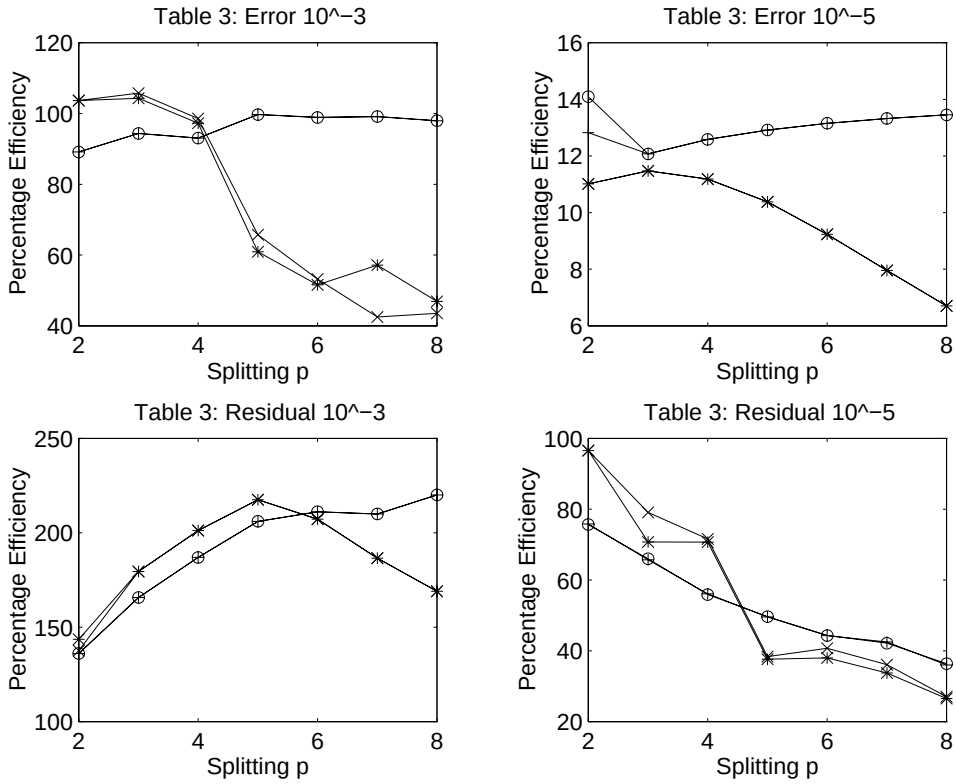

Figure 3. Comparison of parallel efficiency of algorithms for data from Table 3

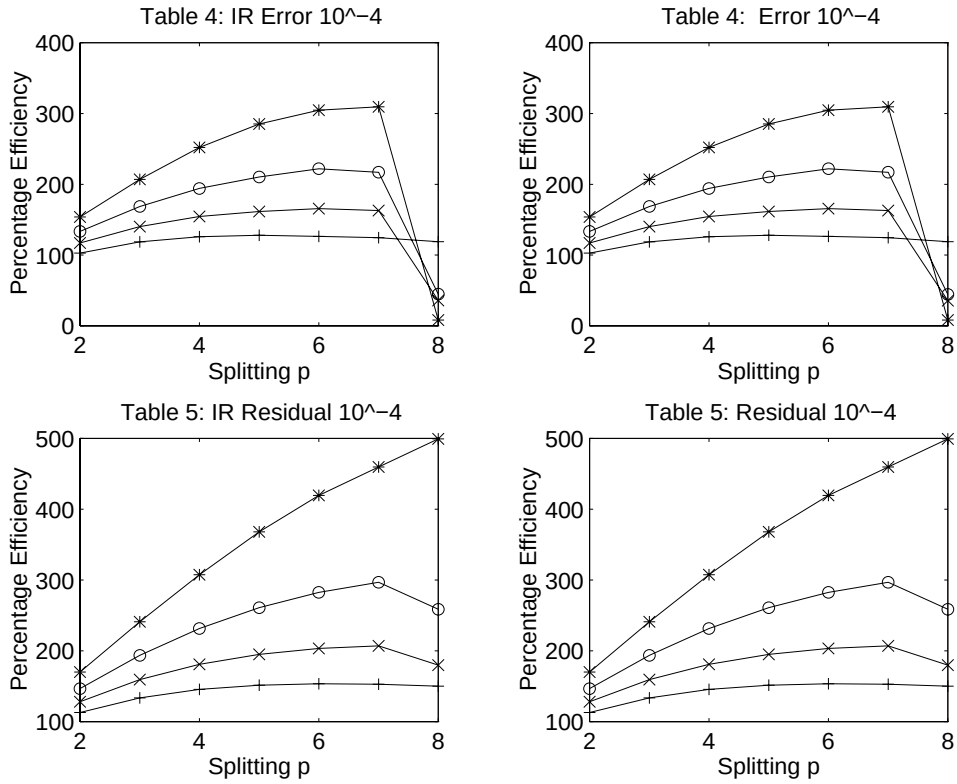

Figure 4. Comparison of parallel efficiency for overlap 0, 10, 20, 30 for data from Tables 4 and 5

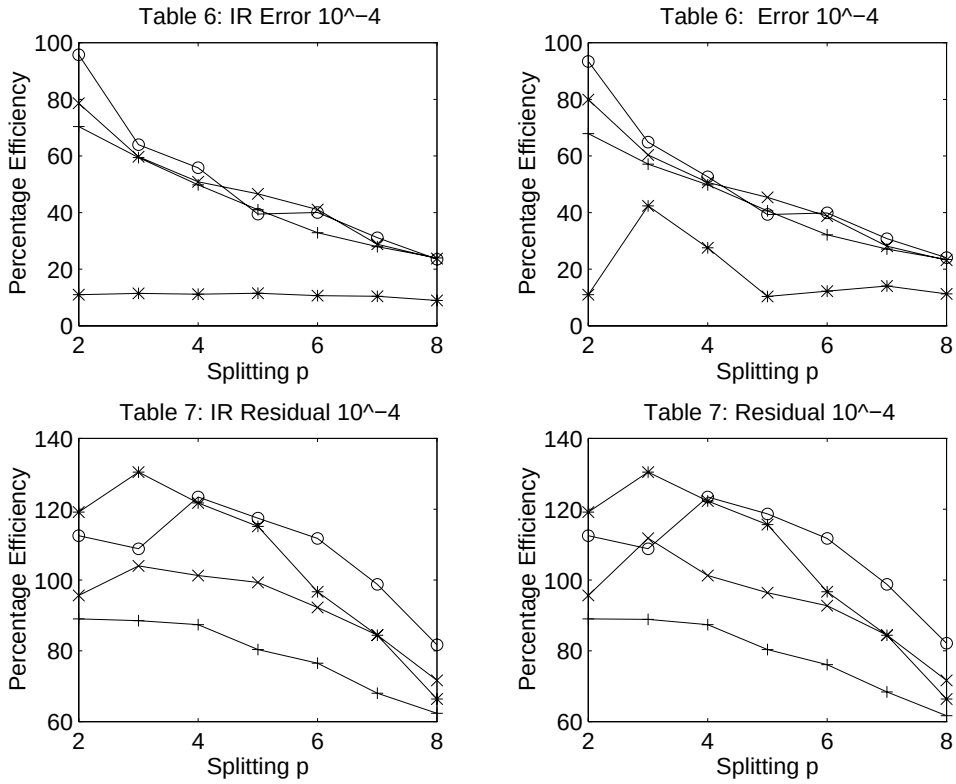

Figure 5. Comparison of parallel efficiency for overlap 0, 10, 20, 30 for data from Tables 6 and 7

REFERENCES

1. R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H. van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, Philadelphia, 1994.
2. A. Berman and R. J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences*. Classics in Applied Mathematics, SIAM, Philadelphia, 1994.
3. R. Bramley and A. Sameh. Row projection methods for large nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 13, 168–193, 1992.
4. E. Chu and A. George. QR factorization of a dense matrix on a hypercube multiprocessor. *SIAM J. Sci. Stat. Comput.*, 11(5), 990–1028, 1990.
5. I. S. Duff and J. K. Reid. A comparison of some methods for the solution of sparse overdetermined systems of linear equations. *J. Inst. Maths. Applics.*, 17, 267–280, 1976.
6. M. C. Ferris and O. L. Mangasarian. Parallel variable distribution. *SIAM J. Optimization*, 4(4), 815–832, 1994.
7. A. Frommer and B. Pohl. A comparison result for multisplittings and waveform relaxation methods. *Numer. Linear Algebra Appl.*, 2, 335–346, 1995.
8. G. H. Golub. Numerical methods for solving least squares problems. *Numer. Math.*, 7, 206–216, 1965.
9. G. H. Golub and C. van Loan. *Matrix Computations*, second edition. John Hopkins Press, Baltimore, 1989.
10. G. H. Golub and J. H. Wilkinson. Note on the iterative refinement of least squares solution. *Numer. Math.*, 9, 139–148, 1966.
11. L. A. Hageman and D. M. Young. *Applied Iterative Methods*. Academic Press, New York, 1981.
12. Q. He. *Parallel multisplittings for nonlinear minimization*. Ph.D. thesis, Arizona State University, 1997. In preparation.
13. N. J. Higham. Iterative refinement enhances the stability of QR decomposition methods for solving linear equations. *BIT*, 31, 447–468, 1991.
14. D. P. O'Leary and R. E. White. Multi-splitting of matrices and parallel solution of linear systems. *SIAM J. Alg. Disc. Meth.*, 6, 630–640, 1985.
15. J. M. Ortega. *Introduction to Parallel and Vector Solutions of Linear Systems*. Plenum Press, New York and London, 1988.
16. J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Academic Press, New York, 1970.
17. R. A. Renaut, Q. He and F.-S. Horng. Parallel multisplitting for minimization, in *Grand Challenges in Computer Simulation*, A. Tentner, editor, pp. 317–322, High Performance Computing 1995. Society for Computer Simulation, 1995.
18. R. A. Renaut and H. D. Mittelmann. Parallel multisplittings for optimization. *J. Parallel Alg. and Appl.*, 7, 17–27, 1995.

