Generalized subspace correction methods for parallel solution of linear systems

GENERALIZED SUBSPACE CORRECTION METHODS FOR PARALLEL SOLUTION OF LINEAR SYSTEMS*

PETTER KOLM[1], PETER ARBENZ[2] and WALTER GANDER[2]

[1] *Center for Computational Mathematics and Mechanics, Royal Institute of Technology, S-100 44 Stockholm, Sweden. email: kolm@nada.kth.se*

[2] *Institut für Wissenschaftliches Rechnen, Eidgenössische Technische Hochschule, CH-8092 Zürich, Switzerland. email: {arbenz, gander}@inf.ethz.ch*

Abstract.

Iterative methods for the solution of linear systems on parallel computer architectures are presented. Two fundamentally different iteration schemes evolve from the theory of subspace correction: the *generalized parallel subspace correction* (*PSC) and the *generalized successive subspace correction* (*SSC). The natural parallelism of the *PSC is used to construct several overlapping block stationary iterative methods. Convergence is proved for a class of these algorithms. Numerical experiments on a 96 node Intel Paragon XP/S5+ show that these methods have potential as scalable preconditioners on distributed memory systems.

AMS subject classification: 65F10, 65Y05

Key words: Linear systems, projection methods, preconditioning, parallel computing.

1 Introduction.

One of the most fundamental problems in scientific computing is to solve systems of linear equations. Often these systems arise from the discretization of differential equations by finite difference, finite volume or finite element methods. Such systems usually have a very sparse structure. Iterative methods exploiting this sparse structure have proven to be very effective on conventional computers for a wide area of applications. Due to the rapid development and increasing popularity of parallel computers it has become important to design iterative methods specialized for these new architectures.

A general framework for iterative methods for linear equations is *subspace correction* first introduced by Xu [38]. Subspace correction methods are induced by a decomposition of the underlying vector space in subspaces. An iterative process is defined by beginning with an approximation to the solution of the linear system and then computing corrections to this approximation on each subspace.

*Parts of this article are based upon the first author's master thesis [18] conducted at Institut für Wissenschaftliches Rechnen, Eidgenössische Technische Hochschule, Zürich, Switzerland.

The subspace corrections are then combined and used to improve the original approximation. Xu established two different categories of algorithms, the *parallel subspace correction* (PSC) and the *successive subspace correction* (SSC). His main goal was to set the framework for a general convergence theory of a class of iterative methods for symmetric positive definite (SPD) linear equations. Unfortunately, in the more general case, where the system matrix A is indefinite, nonsymmetric or both, many of Xu's ideas can not be used.

In this work we present and analyze two different types of methods, the *generalized parallel subspace correction* (*PSC) and the *generalized successive subspace correction* (*SSC), for the solution of nonsingular linear equations. Both methods are extensions of the respective algorithms, introduced by Xu [38] for symmetric positive definite problems. These algorithms differ in, that *PSC exhibits a natural parallelism, whereas *SSC has a truly sequential structure. In addition to Xu, we introduce weighting to the subspace corrections in the *PSC methods. In fact, this weighting process can improve the convergence rate of the methods significantly. Further, some connections to already known iterative algorithms are discussed. Under specific but still rather general conditions, we prove the convergence of *SSC and *PSC. Some aspects of the implementation of these algorithms on distributed memory machines and their complexity are discussed. Numerical experiments conducted on the Intel Paragon XP/S5+ with 96 compute nodes at ETH Zürich show that the *PSC methods are well suited for distributed memory systems.

The outline of the article is as follows. In Section 2 we present a general setting for the subspace correction methods and study two realizations, the row and column oriented *PSC and *SSC methods. We prove the convergence for these two methods in Section 3. Further, in Section 4 we discuss the parallel implementation and provide simple complexity estimates. Section 5 is devoted to some numerical experiments on the Intel Paragon XP/S5+ reflecting the properties of the *PSC methods. Finally, related work is discussed in Section 6.

2 Subspace Correction Methods.

In the first part of this section we will set a general framework and introduce extensions of the PSC and SSC algorithms introduced by Xu [38], the *PSC and the *SSC algorithms. Thereby, we will see that the *PSC methods have a structure similar to the Jacobi iteration and therefore explore natural parallelism. The *SSC methods, however, will be seen to be truly sequential and their connection to Gauss-Seidel will be apparent. As opposed to Xu we only assume the invertability of the linear system and do not rely on the fact that for the SPD case the system matrix induces an inner product. We remark that many other iterative methods not discussed here in detail like classical splittings, multigrid and domain decomposition methods can be incorporated into the framework. For multigrid in the SPD case, see Xu [38]. In the second part of this section we consider some realizations. A connection to the *multisplitting algorithm* is established in Section 2.3.

2.1 General Framework.

For a finite-dimensional vector space $\mathcal{V}$ we consider the linear equation

$$Ax = b\,, \quad A \in L(\mathcal{V})\,, \tag{2.1}$$

with invertible A. A general approach of defining iterative methods is the following. Let x^k be an intermediate approximate solution of (2.1) and define the corresponding residual as $r^k := b - Ax^k$. With $e^k := x - x^k$, denoting the error, and with use of equation (2.1) we get the *correction equation*

$$Ae^k = r^k. \tag{2.2}$$

Thus, we have $x = x^k + e^k$ by solving (2.2) for the error. However, since (2.2) is as difficult to solve as (2.1), we instead compute an approximation $\hat{e}^k := Rr^k$, with $R \approx A^{-1}$ in some sense. Summarizing, we have the following single step iterative method for solving (2.1).

ALGORITHM 2.1. *Basic iterative method*

choose initial guess x^0, $k = 0$
repeat
  $r^k = b - Ax^k$
  $\hat{e}^k = Rr^k$
  $x^{k+1} = x^k + \hat{e}^k$
  $k = k + 1$
until *convergence*

Clearly, this algorithm can be written as

$$x^{k+1} = Qx^k + d \tag{2.3}$$

and with the *residual iteration*

$$r^{k+1} = Sr^k\,, \tag{2.4}$$

where $Q = I - RA$, $d = Rb$ and $S = I - AR$. If Q and R do neither depend on the iteration index k, nor on the iterates x^k, Algorithm 2.1 is called a *linear stationary method of first degree.*

EXAMPLE 2.1. Let $A = D - L - U$ where D denotes the diagonal of A, L the strictly lower and U the strictly upper triangular matrices of A. Then $R = D^{-1}$, $R = (D - L)^{-1}$ and $R = \alpha I$ in Algorithm 2.1 correspond to the Jacobi, the Gauss-Seidel and the Richardson methods respectively.

Obviously, the properties of the basic iterative method rely on the selection of R. We will focus our attention on choices especially suited for parallel computer architectures. Recall that for any linear map X the *Moore-Penrose generalized inverse* [8] is the unique linear map $X^\dagger$ such that the following four identities

$$XX^\dagger X = X\,, \quad X^\dagger XX^\dagger = X^\dagger\,, \quad (XX^\dagger)^T = XX^\dagger\,, \quad (X^\dagger X)^T = X^\dagger X \tag{2.5}$$

are satisfied. In the following we will write $X^{i,j}$ for a generalized inverse satisfying the i-th and j-th conditions in (2.5). In particular note that when X has maximal rank then $X^{1,4} = X^T(XX^T)^{-1}$ and $X^{1,3} = (X^TX)^{-1}X^T$ are the minimal norm and the least squares inverses respectively.

Now, let $(G_i)_{i\in\mathcal{I}}$, $G_i \in L(\mathcal{V})$, $\mathcal{I} \subset \mathbb{N}$, be a finite family of linear maps. By applying these maps to (2.1) we get

$$G_iAx = G_ib\,, \quad i \in \mathcal{I}\,, \tag{2.6}$$

which can be viewed as (2.1) restricted to the subspace $\mathcal{R}(G_i)$, the range of G_i. For an approximate solution, x^k, of the linear equation (2.1), we denote by e_i^k the error of the i-th equation of (2.6). Then we can write the corresponding correction equation

$$G_iAe_i^k = G_ir^k\,, \tag{2.7}$$

and

$$e_i^k = (G_iA)^\dagger G_ir^k\,,$$

where $r^k := b - Ax^k$. For each i, $x^k + e_i^k$ gives a new approximate solution of (2.1). By weighting these approximations we can combine them to get the next iterate x^{k+1}.

DEFINITION 2.1. A family $(E_i)_{i\in\mathcal{I}}$, $\mathcal{I} \subset \mathbb{N}$, of maps $E_i \in L(\mathcal{V})$, is a *consistent weighting* if

(i) $E_i \geq 0$, for all i,

(ii) E_i is self-adjoint, for all i and

(iii) $\sum_{i\in\mathcal{I}} E_i = I$, where I is the identity map.

We will call (iii) the *consistency condition*. With the weighting maps we have

$$x^{k+1} = x^k + \sum_{i\in\mathcal{I}} E_ie_i^k\,, \tag{2.8}$$

for the next iterate. Now we can formulate the *generalized parallel subspace correction* (*PSC) algorithm.

ALGORITHM 2.2. **PSC*

choose linear maps $(G_i)_{i\in\mathcal{I}}$ *and consistent weighting* $(E_i)_{i\in\mathcal{I}}$
choose initial guess x^0, $k = 0$
repeat
  for all i **do in parallel**
    $e_i^k = (G_iA)^\dagger G_ir^k$
      $= (G_iA)^\dagger(G_ib - G_iAx^k)$
  end
  $x^{k+1} = x^k + \sum_{i\in\mathcal{I}} E_ie_i^k$
  $k = k + 1$
until *convergence*

One way of improving the convergence behavior of the *PSC algorithm is to calculate each correction at a time and thereby using the most recent approximation of the solution in a Gauss-Seidel type fashion. Then of course, the natural parallelism in the *PSC algorithm is lost. Assuming $\mathcal{I} = \{1, \ldots, m\}$, we state the *generalized successive subspace correction* (*SSC) algorithm.

ALGORITHM 2.3. **SSC*

choose linear maps $(G_i)_{i\in\mathcal{I}}$
choose initial guess x^0, $k = 0$
repeat
  $y^1 = x^k$
  for $i = 1, \ldots, m$ **do**
    $e_i = (G_i A)^\dagger G_i (b - Ay^i)$
    $y^{i+1} = y^i + e_i$
  end
  $x^{k+1} = y^{m+1}$
  $k = k + 1$
until *convergence*

The *PSC and *SSC algorithms generalize the PSC and SSC algorithms, introduced by Xu [38], to general invertible linear maps and by considering the family $(G_i)_{i\in\mathcal{I}}$ instead of just orthogonal projections. Further, *PSC also incorporates consistent weighting. Obviously, the *algorithmic structures* of the *PSC and *SSC algorithms are similar to the Jacobi and the Gauss-Seidel iterations respectively.

EXAMPLE 2.2. For $A = (a_{ij})$ the Jacobi iteration can be written as

$$x^{k+1} = x^k + D^{-1} r^k \,,$$

or in components

$$x_i^{k+1} = x_i^k + a_{ii}^{-1} r_i^k \,, \quad i = 1, \ldots, n.$$

We see that each component of x^{k+1} can be computed independently. Thus Jacobi has a natural parallel structure.

EXAMPLE 2.3. The Gauss-Seidel iteration has the form

$$x^{k+1} = D^{-1}(Lx^{k+1} + Ux^k + b) \,,$$

or in components

$$\begin{aligned} x_i^{k+1} &= \frac{1}{a_{ii}}\Big(b_i - \sum_{j<i} a_{ij} x_j^{k+1} - \sum_{j>i} a_{ij} x_j^k\Big) \\ &= x_i^k + \frac{1}{a_{ii}}\Big(b_i - \sum_{j<i} a_{ij} x_j^{k+1} - \sum_{j\geq i} a_{ij} x_j^k\Big) \,, \quad i = 1, \ldots, n. \end{aligned} \tag{2.9}$$

Denote by e_i the i-th canonical basis vector in $\mathbb{R}^n$. From (2.9) we observe that the mapping $x^k \mapsto x^{k+1}$ can be described as

$$
\begin{aligned}
y^1 &= x^k\,, \\
y^{i+1} &= y^i + \frac{1}{a_{ii}}\Big(b_i - \sum_j a_{ij}\, y^i_j\Big)\cdot e_i\,, \quad i = 1,\ldots,n \qquad (2.10)\\
\text{and} & \\
x^{k+1} &= y^{n+1}\,.
\end{aligned}
$$

Hence, the Gauss-Seidel iteration has a truly sequential updating pattern for its components. It is therefore less suitable for parallel implementation.

The following propositions show that both *PSC and *SSC are linear stationary methods of first degree.

PROPOSITION 2.1. *The *PSC algorithm can be written in the forms of the iterations (2.3) and (2.4) with* $R = \sum_{i\in\mathcal{I}} E_i(G_iA)^\dagger G_i$.

PROOF. Since

$$
\begin{aligned}
x^{k+1} &= x^k + \sum_{i\in\mathcal{I}} E_i e_i^k \\
&= x^k + \sum_{i\in\mathcal{I}} E_i(G_iA)^\dagger G_i r^k \\
&= x^k + \sum_{i\in\mathcal{I}} E_i(G_iA)^\dagger G_i(b - Ax^k) \\
&= \Big(I - \sum_{i\in\mathcal{I}} E_i(G_iA)^\dagger G_iA\Big)x^k + \sum_{i\in\mathcal{I}} E_i(G_iA)^\dagger G_ib\,,
\end{aligned}
$$

the proposition easily follows. □

PROPOSITION 2.2. *The *SSC algorithm can be written in the forms of the iterations (2.3) and (2.4) with*

$$
R = R_m + \sum_{j=1}^{m-1} Q_m\cdots Q_{j+1}R_j\,,
$$

$$
Q = Q_m\cdots Q_1 \quad \textit{and} \quad S = S_m\cdots S_1\,,
$$

where $R_i := (G_iA)^\dagger G_i$, $Q_i = I - R_iA$ *and* $S_i = I - AR_i$.

PROOF. We prove the proposition by induction over m. The case $m = 1$ is trivial. Now assume the proposition is true for $m = p$. Then for $m = p+1$ we have

$$
\begin{aligned}
x^{k+1} = y^{p+1} &= y^p + (G_pA)^\dagger G_p(b - Ay^p) \\
&= Q_py^p + R_pb \\
&= Q_pQ_{p-1}\cdots Q_1y^1 + R_pb + Q_p\Big(R_{p-1} + \sum_{j=1}^{p-2} Q_{p-1}\cdots Q_{j+1}R_j\Big)b
\end{aligned}
$$

$$= \quad Q_p \cdots Q_1 x^k + \Big(R_p + \sum_{j=1}^{p-1} Q_p \cdots Q_{j+1} R_j \Big) b$$

and we also easily conclude that $r^{k+1} = S_p \cdots S_1 r^k$, which proves the first part. For the second part consider

$$\begin{aligned} RA &= R_m A + \sum_{j=1}^{m-1} Q_m \cdots Q_{j+1} R_j A \\ &= I - Q_m + \sum_{j=1}^{m-1} Q_m \cdots Q_{j+1} (I - Q_j) \\ &= I - \sum_{j=1}^{m} Q_m \cdots Q_j + \sum_{j=1}^{m-1} Q_m \cdots Q_{j+1} \\ &= I - Q_m \cdots Q_1 \end{aligned}$$

and in the same way $AR = I - S$. □

2.2 Realizations.

We will now study in detail some realizations of the *PSC and *SSC methods. Since our concerns are with solving linear equations on parallel computer architectures, we will focus on *PSC-type algorithms that exhibit a natural parallelism. For completeness, however, we will also present the *SSC versions. We remark that for linear systems with certain structure, typically banded or block tridiagonal matrices, parallelism can be exploited also for the *SSC methods. The difference of the various versions will be in the selection of the family $(G_i)_{i \in \mathcal{I}}$. By choosing it appropriately, we get row and column oriented block methods.

2.2.1 Row and Column Block Methods.

Let $\mathcal{V} = \mathbb{R}^n$ with the canonical basis $\{e_1, \ldots, e_n\}$ and consider the linear equation (2.1) with invertible $A \in L(\mathbb{R}^n)$. Further, we denote by $\mathcal{J} = \{1, \ldots, n\}$ the set of indices, and consider the family of not necessarily disjoint subsets $(\mathcal{J}^i)_{i \in \mathcal{I}}$ such that

$$\bigcup_{i \in \mathcal{I}} \mathcal{J}^i = \mathcal{J} \, .$$

For each subset, $\mathcal{J}^i = \{j_1^i, \ldots, j_{l_i}^i\}$, the matrix

$$V_i = (e_{j_1^i}, \ldots, e_{j_{l_i}^i}) \in \mathbb{R}^{n \times l_i}$$

induces the subspace $\mathcal{V}_i = \mathcal{R}(V_i)$ and the orthogonal projection

$$P_{\mathcal{V}_i} : \mathcal{V} \longrightarrow \mathcal{V}_i$$

given by $P_{\mathcal{V}_i} = V_i V_i^{1,3} = V_i V_i^T$, such that

$$\mathcal{V} = \sum_{i \in \mathcal{I}} \mathcal{V}_i \ . \tag{2.11}$$

To formalize we consider the following definitions.

DEFINITION 2.2. Decomposition, Induced Projections

(i) A finite family of subsets $(\mathcal{V}_i)_{i \in \mathcal{I}}$, $\mathcal{I} \subset \mathbb{N}$, is a *decomposition* of the vector space $\mathcal{V}$ if for all $v \in \mathcal{V}$ there exists $v_i \in \mathcal{V}_i$ with $v = \sum_{i \in \mathcal{I}} v_i$. Normally, we will write $\mathcal{V} = \sum_{i \in \mathcal{I}} \mathcal{V}_i$, $\mathcal{V}_i \subset \mathcal{V}$.

(ii) A decomposition, $(\mathcal{V}_i)_{i \in \mathcal{I}}$, induces a family $(P_{\mathcal{V}_i})_{i \in \mathcal{I}}$ of orthogonal projections such that for all i, $P_{\mathcal{V}_i} : \mathcal{V} \longrightarrow \mathcal{V}_i$ with $P_{\mathcal{V}_i}^2 = P_{\mathcal{V}_i}$ and $P_{\mathcal{V}_i}^T = P_{\mathcal{V}_i}$.

DEFINITION 2.3. Overlapping, Block and Regular Decomposition, Overlap

(i) A decomposition is *overlapping* if $\mathcal{J}^i \cap \mathcal{J}^j \neq \emptyset$ for some $i \neq j$.

(ii) A decomposition is a *block decomposition* if for all subsets $\mathcal{J}^i$ and $j, l \in \mathcal{J}^i$ with $j < l$ and for all k such that $j < k < l$, it follows $k \in \mathcal{J}^i$.

(iii) An overlapping block decomposition is called *regular* if only subsequent subsets in the family $(\mathcal{J}^i)_{i \in \mathcal{I}}$ intersect, i.e. if $\mathcal{J}^i \cap \mathcal{J}^j = \emptyset$ for $|i - j| > 1$.

(iv) A regular block decomposition has *overlap* ϕ if $\phi = |\mathcal{J}^i \cap \mathcal{J}^j|$ for $|i-j| = 1$.

In the following we will assume all decompositions to be *regular block decompositions* and for simplicity consider $\mathcal{I} = \{1, \ldots, m\}$. First, we choose the family $(G_i)_{i \in \mathcal{I}}$ equal to $(P_{\mathcal{V}_i})_{i \in \mathcal{I}}$, $P_{\mathcal{V}_i} = V_i V_i^T$. Thus, the correction equations (2.7) become

$$P_{\mathcal{V}_i} A e_i^k = P_{\mathcal{V}_i} r^k \ , \tag{2.12}$$

which for all i, are underdetermined systems. Writing $A_i := V_i^T A \in \mathbb{R}^{l_i \times n}$ and $r_i^k := V_i^T r^k \in \mathbb{R}^{l_i}$, these systems have the *minimal norm* solutions

$$\begin{aligned} e_i^k &= (P_{\mathcal{V}_i} A)^{1,4} P_{\mathcal{V}_i} r^k \\ &= A^T P_{\mathcal{V}_i} (P_{\mathcal{V}_i} A A^T P_{\mathcal{V}_i})^{-1} P_{\mathcal{V}_i} r^k \\ &= A^T V_i (V_i^T A A^T V_i)^{-1} V_i^T r^k \\ &= A_i^T (A_i A_i^T)^{-1} r_i^k \\ &= A_i^T (A_i A_i^T)^{-1} (b_i - A_i x^k) \ . \end{aligned}$$

Since the A_i's are row blocks of A the resulting *PSC and *SSC algorithms are called *row oriented *PSC and *SSC*, RPSC and RSSC respectively.

Analogously, by choosing the family $(G_i)_{i \in \mathcal{I}}$ equal to $(P_{\mathcal{V}_i} A^T)_{i \in \mathcal{I}}$, column oriented versions result. Here, the correction equations (2.7) take the form

$$P_{\mathcal{V}_i} A^T A e_i^k = P_{\mathcal{V}_i} A^T r^k \ ,$$

which have the the *least squares* solutions

$$\begin{aligned} e_i^k &= P_{\mathcal{V}_i}(AP_{\mathcal{V}_i})^{1,3}r^k \\ &= P_{\mathcal{V}_i}(P_{\mathcal{V}_i}A^TAP_{\mathcal{V}_i})^{-1}P_{\mathcal{V}_i}A^Tr^k \\ &= V_i(V_i^TA^TAV_i)^{-1}V_i^TA^Tr^k \\ &= V_i(A^{i^T}A^i)^{-1}A^{i^T}r^k \\ &= V_i(A^{i^T}A^i)^{-1}A^{i^T}(b-Ax^k)\,, \end{aligned}$$

where $A^i := AV_i$ is a column block of matrix A. We will call these versions the *column oriented *PSC and *SSC*, CPSC and CSSC. For a summary, the algorithms are given in combined form in Algorithms 2.4 and 2.5.

In practice, it is important to choose the decomposition such that the work is equally distributed on the processors of the parallel computer, thus giving a well balanced work load. Often a uniform block decomposition with equally large subsets and with the same overlap give good results. If the system matrix has a special structure as is often the case when it represents the discrete counterpart of a differential equation this information can be used to reduce work and obtain better parallel performance. It is important, however, to keep in mind that different decompositions may show different convergence rates. One convenient choice of the consistent weighting is to only use diagonal matrices. Then, the j-th component of the e_i^k's has only to be calculated if the j-th diagonal element of E_i is nonzero. This makes the weighting process computationally attractive.

2.2.2 A Connection between RPSC and CPSC.

We want to establish a connection between the RPSC and CPSC methods. Denote by $Q_R(A)$ and $Q_C(A)$ the iteration matrices of the RPSC and CPSC methods applied to the linear system (2.1). Referring to Proposition 2.1, they can be written as

$$Q_R(A) = I - \sum_{i\in\mathcal{I}} E_i(P_{\mathcal{V}_i}A)^{1,4}P_{\mathcal{V}_i}A$$

and

$$Q_C(A) = I - \sum_{i\in\mathcal{I}} E_iP_{\mathcal{V}_i}(AP_{\mathcal{V}_i})^{1,3}A\,.$$

With the properties of the generalized inverse, we obtain

$$\begin{aligned} \Big(I - \sum_{i\in\mathcal{I}} E_i(P_{\mathcal{V}_i}A)^{1,4}P_{\mathcal{V}_i}A\Big)^T &= I - \sum_{i\in\mathcal{I}} (P_{\mathcal{V}_i}A)^{1,4}P_{\mathcal{V}_i}AE_i \\ &= I - \sum_{i\in\mathcal{I}} (A^TP_{\mathcal{V}_i})(A^TP_{\mathcal{V}_i})^{1,3}E_i \\ &= A^T\Big(I - \sum_{i\in\mathcal{I}} P_{\mathcal{V}_i}(A^TP_{\mathcal{V}_i})^{1,3}E_iA^T\Big)A^{-T}\,, \end{aligned}$$

ALGORITHM 2.4. *RPSC/CPSC*

choose a regular block decomposition and consistent weighting $(E_i)_{i\in\mathcal{I}}$
choose initial guess x^0, $k = 0$

repeat

(RPSC *)*	*(* CPSC *)*
for $i = 1, \ldots, m$ **do in parallel**	$r = b - Ax^k$
$r_i = b_i - A_i x^k$	**for** $i = 1, \ldots, m$ **do in parallel**
$e_i = A_i^T (A_i A_i^T)^{-1} r_i$	$e_i = V_i ({A^i}^T A^i)^{-1} {A^i}^T r$
end	**end**

$x^{k+1} = x^k + \sum_{i\in\mathcal{I}} E_i e_i$
$k = k + 1$

until *convergence*

ALGORITHM 2.5. *RSSC/CSSC*

choose a regular block decomposition
choose initial guess x^0, $k = 0$

repeat

$y^1 = x^k$

(RSSC *)*	*(* CSSC *)*
for $i = 1, \ldots, m$ **do**	**for** $i = 1, \ldots, m$ **do**
$r_i = b_i - A_i y^i$	$r^i = b - Ay^i$
$e_i = A_i^T (A_i A_i^T)^{-1} r_i$	$e_i = V_i ({A^i}^T A^i)^{-1} {A^i}^T r^i$
$y^{i+1} = y^i + e_i$	$y^{i+1} = y^i + e_i$
end	**end**

$x^{k+1} = y^{m+1}$
$k = k + 1$

until *convergence*

and therefore $Q_R(A)^T$ is similar to the column oriented iteration matrix with *pre-weighting* on $A^T x = b$,

$$Q_C^{pre}(A^T) = I - \sum_{i \in \mathcal{I}} P_{\mathcal{V}_i} (A^T P_{\mathcal{V}_i})^{1,3} E_i A^T \ .$$

Analogously, one can show that $Q_C(A)^T$ is similar to the row oriented iteration matrix with *pre-weighting* on $A^T x = b$. In particular, we have $\rho(Q_R(A)) = \rho(Q_C^{pre}(A^T))$ and $\rho(Q_C(A)) = \rho(Q_R^{pre}(A^T))$ where $\rho(\cdot)$ denotes the spectral radius. It it therefore, in terms of convergence rates of both methods, enough to consider one of the algorithms.

REMARK 2.1. Formally, one could also consider a combination of pre- and post-weighting by introducing

$$Q_R^\lambda(A) = I - \sum_{i \in \mathcal{I}} E_i^\lambda (P_{\mathcal{V}_i} A)^{1,4} P_{\mathcal{V}_i} E_i^{1-\lambda} A$$

and

$$Q_C^\lambda(A) = I - \sum_{i \in \mathcal{I}} E_i^\lambda P_{\mathcal{V}_i} (A P_{\mathcal{V}_i})^{1,3} E_i^{1-\lambda} A$$

where $\lambda \in [0,1]$. Thus, if $\lambda = 1$ then we have the *post-weighting* case as above. For $\lambda = 0$ or $\lambda = \frac{1}{2}$ we have a *pre-weighting* or *symmetric* scheme. We will not explore these properties any further in this work, but rather concentrate on the post-weighting case throughout.

2.3 The Multisplitting Method.

The multisplitting method has found much interest after its introduction by O'Leary and White [22]. Many theoretical properties of its behavior and convergence have been established, see for example [36], [37], [12] and [13]. We will not go into details about these results but rather describe the algorithm and show its connection with the *PSC methods.

Consider the linear equation (2.1) with invertible $A \in L(\mathbb{R}^n)$.

DEFINITION 2.4. The finite family $(M_i, N_i, E_i)_{i \in \mathcal{I}}$, $\mathcal{I} \subset \mathbb{N}$, is a multisplitting of A if

(i) $A = M_i - N_i$, with M_i invertible, for all $i \in \mathcal{I}$ and

(ii) $(E_i)_{i \in \mathcal{I}}$ is a consistent weighting, for E_i diagonal.

For each i we thus have an iteration scheme in the form

$$x^{k+1} = M_i^{-1} N_i x^k + M_i^{-1} b \ . \tag{2.13}$$

All these iteration schemes can be computed independently of each other and then be combined using the weighting matrices E_i. Altogether, we have the multisplitting method (MS).

ALGORITHM 2.6. *MS*

choose multisplitting (M_i, N_i, E_i)
choose initial guess x^0, $k = 0$
repeat
  for all i **do in parallel**
    $y^i = M_i^{-1} N_i x^k + M_i^{-1} b$
  end
  $x^{k+1} = \sum_{i \in \mathcal{I}} E_i y^i$
  $k = k + 1$
until *convergence*

Clearly, the j-th component of y^i does only have to be computed if the j-th diagonal element of E_i is nonzero. As in the *PSC algorithms, the E_i's are to be chosen such that the work is equally distributed among the different processors. Along the lines of Definition 2.3 we can introduce overlapping, block and regular multisplittings. In the following, we will focus our attention to a special *regular block multisplitting*. Let $(\mathcal{K}^i)_{i \in \mathcal{I}}$ be a family of consecutive overlapping subsets of indices. Define the *regular block Jacobi multisplitting* of $A = (a_{kl})$ as

$$(M_i)_{kl} = \begin{cases} a_{kl} & \text{if } k = l \text{ or } k, l \in \mathcal{K}^i \\ 0 & \text{otherwise.} \end{cases} \tag{2.14}$$

The k-th diagonal element of E_i is chosen to be zero if $k \notin \mathcal{K}^i$ and non-negative otherwise, such that (ii) of Definition 2.4 is satisfied. If the overlap is chosen to be zero the above multisplitting becomes the *block-Jacobi* iteration. Thus, with the following proposition, the multisplitting algorithm is related to the *PSC methods.

PROPOSITION 2.3. *If the the overlap is zero, i.e.* $\phi = 0$, *then*

(i) *RPSC applied to (2.1) is equivalent to block-Jacobi on the system*

$$\begin{aligned} AA^T y &= b \\ x &= A^T y\,, \end{aligned}$$

(ii) *CPSC applied to (2.1) is equivalent to block-Jacobi on the system*

$$A^T A x = A^T b\,.$$

PROOF. The proof follows from some algebraic manipulations. Assume $\mathcal{I} = \{1, \ldots, m\}$.
(i) The RPSC iteration without overlap

$$x^{k+1} = \Big(I - \sum_{i \in \mathcal{I}} A_i^T (A_i A_i^T)^{-1} A_i\Big) x^k + \sum_{i \in \mathcal{I}} A_i^T (A_i A_i^T)^{-1} b_i$$

can be written as

$$x^{k+1} = (I - A^T D^{-1} A)x^k + A^T D^{-1} b$$

with $A^T = [A_1^T, \ldots, A_m^T]$ and the block diagonal matrix $D = (D_{ii}) = (A_i A_i^T)$. Substituting $x = A^T y$ we get

$$\begin{aligned} A^T y^{k+1} &= (A^T - A^T D^{-1} A A^T) y^k + A^T D^{-1} b \\ &= A^T((I - D^{-1} A A^T) y^k + D^{-1} b) \ . \end{aligned}$$

(ii) For $x^T = (x_1^T, \ldots, x_m^T)$ and $D_{ii} = A^{iT} A^i$, we can write the CPSC method without overlap as

$$\begin{aligned} x_i^{k+1} &= x_i^k + (A^{iT} A^i)^{-1} A^{iT} (b - Ax^k) \\ &= x_i^k + D_{ii}^{-1} A^{iT} (b - Ax^k) \ , \end{aligned}$$

which completes the proof. □

3 Convergence and Consistency.

In this section we will prove the convergence and the complete consistency of the row and column *SSC and *PSC methods.

*3.1 *SSC.*

By Proposition 2.2 the *SSC algorithms are linear stationary methods of first degree in the form $x^{k+1} = Qx^k + Rb$, $r^{k+1} = Sr^k$. To prove convergence it is therefore sufficient to show $\|Q\| < 1$ or $\|S\| < 1$ in some norm, see for example [35, p. 13]. Then complete consistency immediately follows from convergence, see for example [39, p. 67]. Throughout this section we denote by Q_{RSSC} the iteration matrix and by S_{CSSC} the residual iteration matrix resulting from the RSSC and the CSSC methods applied to the linear system (2.1), respectively.

Theorem 3.1. *The RSSC method is convergent and completely consistent, i.e.* $\|Q_{RSSC}\| < 1$.

With Proposition 2.2, it is easily seen that for the CSSC method $S_i = I - AP_{\mathcal{V}_i}(AP_{\mathcal{V}_i})^\dagger$, such that $S_{CSSC} = S_m \cdots S_1$. Since $AP_{\mathcal{V}_i}(AP_{\mathcal{V}_i})^\dagger = (P_{\mathcal{V}_i} A^T)^\dagger P_{\mathcal{V}_i} A^T$, we have $S_{CSSC}(A) = Q_{RSSC}(A^T)$ and therefore the convergence of the CSSC method follows from Theorem 3.1. In particular, we have the following theorem.

Theorem 3.2. *The CSSC method is convergent and completely consistent, i.e.* $\|S_{CSSC}\| < 1$.

We are left with

Proof of Theorem 3.1. We write $Q = Q_{RSSC}$. Recall that with the decomposition, $(\mathcal{V}_i)_{i \in \mathcal{I}}$, we have $Q = Q_m \cdots Q_1$ with $Q_i = I - (P_{\mathcal{V}_i} A)^\dagger P_{\mathcal{V}_i} A$. Further, note that

$$\|Q\| \leq \|Q_m\| \cdots \|Q_1\| = 1 \ , \tag{3.1}$$

since Q is a product of projections. Suppose $\|Q\| = 1$ then there exists an x such that

$$\|Qx\| = \|x\| \,. \tag{3.2}$$

We will show that $x \in \mathcal{N}(A) = \{0\}$, where $\mathcal{N}(A)$ is the null space of A. Assume the converse, i.e. $x \notin \mathcal{N}(A)$. Then there is a smallest index $k \in \{1, \ldots, m\}$ such that $Q_k x \neq x$. Thus with

$$\|Q_k \cdots Q_1 x\| = \|Q_k x\| < \|x\|$$

and equation (3.1) we see

$$\begin{aligned} \|Qx\| &\leq \|Q_m \cdots Q_{k+1}\| \, \|Q_k \cdots Q_1 x\| \\ &= \|Q_m \cdots Q_{k+1}\| \, \|Q_k x\| \\ &< \|x\| \,, \end{aligned}$$

which is a contradiction to (3.2). We conclude x is the zero vector and thus $\|Q\| < 1$. □

*3.2 *PSC.*

We will prove the convergence of the RPSC algorithm by studying an equivalent RSSC algorithm on an extended linear system. Throughout this section we will consider the decomposition $(\mathcal{V}_i)_{i \in \mathcal{I}}$ with $\mathcal{I} = \{1, \ldots, m\}$ and the weighting $E_i := \frac{1}{m} I$ for all i.

In the RPSC method we solve the linear system

$$Ax = b \,, \tag{3.3}$$

by restricting it to the subspaces $\mathcal{V}_i$, where we compute the corrections to the subspace equations

$$A_i x = b_i \,, \quad i = 1, \ldots, m \tag{3.4}$$

for all i independently with $A_i \in \mathbb{R}^{l_i \times n}$, $x \in \mathbb{R}^n$ and $b_i \in \mathbb{R}^{l_i}$. If $(e_i^k)_{i \in \mathcal{I}}$ are the subspace corrections to the k-th approximation, x^k, then the $(k+1)$-th approximation takes the form

$$x^{k+1} = x^k + \frac{1}{m} \sum_{i=1}^{m} e_i^k \,.$$

In the RPSC method we do not take into account, that in general the subspace solutions are actually coupled. That is, if $z_1, \ldots, z_m$ are solutions of the subspace equations (3.4), then $z_1, \ldots, z_m$ do satisfy the linear equation (3.3) if and only if $z_1 = \ldots = z_m$. Summarizing, we have the following lemma.

LEMMA 3.3. *The linear equation (3.3) is equivalent to the subspace equations (3.4) together with the constraints*

$$\begin{array}{ccccc} z_1 & - & z_2 & = & 0 \\ z_2 & - & z_3 & = & 0 \\ & \vdots & & \vdots & \\ z_{m-1} & - & z_m & = & 0 \,. \end{array} \tag{3.5}$$

With $x^k + e_i^k$ being the solution to the i-th subsystem (3.4), denote by $\hat{e}_i^k$ the correction to $x^k + e_i^k$ obtained from the minimal norm solution of the linear system (3.5). By substituting $x^k + e_i^k + \hat{e}_i^k$ into (3.5) we have

$$\begin{array}{rcl} \hat{e}_1^k - \hat{e}_2^k &=& e_2^k - e_1^k \\ \hat{e}_2^k - \hat{e}_3^k &=& e_3^k - e_2^k \\ \vdots & \vdots & \vdots \\ \hat{e}_{m-1}^k - \hat{e}_m^k &=& e_m^k - e_{m-1}^k \,, \end{array} \tag{3.6}$$

which we write as

$$C \cdot \begin{bmatrix} \hat{e}_1^k \\ \hat{e}_2^k \\ \vdots \\ \hat{e}_m^k \end{bmatrix} = \begin{bmatrix} e_2^k - e_1^k \\ e_3^k - e_2^k \\ \vdots \\ e_m^k - e_{m-1}^k \end{bmatrix} , \tag{3.7}$$

with $C \in \mathbb{R}^{(m-1)n \times mn}$. For $e_1^k, \ldots, e_m^k$ given, we have the following lemma.

LEMMA 3.4. *The minimal norm solution of (3.7), that is the solution* $(\hat{e}_i^k)_{i \in \mathcal{I}}$ *that minimizes* $\sum_{i=1}^m \|\hat{e}_i^k\|_2$, *is given by*

$$\hat{e}_i^k = \frac{1}{m}\Big(\sum_{j=1}^m e_j^k\Big) - e_i^k \;,$$

for all $i \in \mathcal{I}$.

PROOF. First we notice from (3.6) that

$$\hat{e}_i^k = e_j^k - e_i^k + \hat{e}_j^k \,,$$

for all $i \neq j$. Then for some i we have

$$\sum_{j=1}^m \|\hat{e}_j^k\|_2 = m\, \hat{e}_i^{k\,T} \hat{e}_i^k + \sum_{\substack{j=1 \\ j \neq i}}^m \Big((e_i^k - e_j^k)^T (e_i^k - e_j^k) + 2\, \hat{e}_i^{k\,T} (e_i^k - e_j^k) \Big) =: F(\hat{e}_1^k, \ldots, \hat{e}_m^k) \,.$$

By taking partial derivatives and minimizing F with respect to $\hat{e}_i^k$, we get after some easy algebraic manipulations

$$\hat{e}_i^k = \frac{1}{m} \sum_{\substack{j=1 \\ j \neq i}}^m \Big(e_j^k - e_i^k \Big) = \frac{1}{m}\Big(\sum_{j=1}^m e_j^k\Big) - e_i^k \;.$$

This concludes the proof. ∎

We will write $C^T(CC^T)^{-1}((e_2^k - e_1^k)^T, \ldots, (e_m^k - e_{m-1}^k)^T)^T$ for the minimal norm solution of (3.7). With LEMMA 3.4, we have

$$x^k + e_i^k + \hat{e}_i^k = x^k + \frac{1}{m} \sum_{j=1}^m e_j^k \,,$$

for all $i \in \mathcal{I}$. In particular, we have shown that RPSC is equivalent to a RSSC scheme applied to the extended system defined by the equations (3.4) and (3.5), where the mapping $x^k \mapsto x^{k+1}$ is described by

$$
\begin{aligned}
&y^1 = x^k \\
&e_i^k = A_i^T (A_i A_i^T)^{-1} (b_i - A_i y^1)\,, \quad i = 1, \ldots, m \\
&y^2 = x^k + e_1^k \\
&(\hat{e}_1^{k\,T}, \ldots, \hat{e}_m^{k\,T})^T = C^T (CC^T)^{-1} ((e_2^k - e_1^k)^T, \ldots, (e_m^k - e_{m-1}^k)^T)^T \\
&y^3 = y^2 + \hat{e}_1^k \\
&x^{k+1} = y^3\,.
\end{aligned}
$$

With THEOREM 3.1 the convergence and the complete consistency of RPSC follows.

THEOREM 3.5. *For the consistent weighting $E_i = \frac{1}{m} I$, $i \in \{1, \ldots, m\}$, the RPSC method is convergent and completely consistent.*

The convergence and the complete consistency of CPSC follows from THEOREM 3.5 by noting

$$
\begin{aligned}
Q_{RPSC} &= I - \frac{1}{m} \sum_{i=1}^{m} (P_{\mathcal{V}_i} A)^{1,4} P_{\mathcal{V}_i} A \\
&= I - \frac{1}{m} \sum_{i=1}^{m} A^T P_{\mathcal{V}_i} (A^T P_{\mathcal{V}_i})^{1,3} \\
&= A^T \Big(I - \frac{1}{m} \sum_{i=1}^{m} P_{\mathcal{V}_i} (A^T P_{\mathcal{V}_i})^{1,3} A^T \Big) A^{-T} \\
&= A^T \cdot Q_{CPSC} \cdot A^{-T}\,,
\end{aligned}
$$

such that $\rho(Q_{RPSC}) = \rho(Q_{CPSC})$.

THEOREM 3.6. *For the consistent weighting $E_i = \frac{1}{m} I$, $i \in \{1, \ldots, m\}$, the CPSC method is convergent and completely consistent.*

4 Implementation of Subspace Correction Methods.

In this section we will focus on the implementation of the *PSC and *SSC methods on distributed memory architectures. Due to the computational similarity of the different methods, we will restrict our discussion to the RPSC algorithm. However, the considerations discussed also apply for the other cases.

4.1 Solution of the correction equations.

The RPSC rely on accurate calculation of orthogonal projections by solving the correction equations, see (2.7) and (2.12), on each subspace in the chosen subspace decomposition. The major computational step in the RPSC algorithm thus resembles the solution of systems of underdetermined equations (2.7) of the form

$$
Cy = d\,, \tag{4.1}
$$

in minimal norm sense, where $C \in \mathbb{R}^{l\times n}$, $l \ll n$, is sparse with maximal rank and $d \in \mathbb{R}^l$. This can be accomplished by both direct and iterative methods. Iterative methods suffer from the fact that the work done by solving the minimal norm problem once can not be used in solving it again, resulting in an enormous computational overhead. Direct methods, were a matrix decomposition is computed, can be used repeatedly for successive right hand sides. As long as enough storage is available direct methods are advantageous for the *PSC methods and in the following we will focus only on them.

Probably, the most straight forward approach is to explicitly form the *normal equations* and solve

$$CC^T z = d\,, \tag{4.2}$$

for z by standard Cholesky decomposition [15]. However, since the condition number of CC^T is the square of the condition number of C, $\kappa(CC^T) = \kappa^2(C)$, and a loss of information can occur when computing CC^T, this method should only be used for well-conditioned problems. This approach was used by Bramley and Sameh [7] as well as Benzi et al. [4] for the so-called Cimmino method, see Section 6.

Another approach is the *augmented system method* where the indefinite linear system

$$\begin{pmatrix} I & C^T \\ C & 0 \end{pmatrix}\begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} 0 \\ d \end{pmatrix}$$

is solved by means of LDL^T decomposition. However, since $z = -(CC^T)^{-1}d$ and $y = C^T(CC^T)^{-1}d$, the normal equations are implicitly formed. The same stability problems as by the normal equations can therefore be expected for less well-conditioned problems. By using pivoting and introducing a scaling parameter, Björck [5] has shown how one can get an acceptable-error stable method. However, for sparse problems the used pivoting strategy will be a compromise between stability and preserving sparsity whereas accuracy can be achieved by iterative refinement. Arioli et al. [19] have reported their experience with this modified approach in connection with the Cimmino algorithm.

The problem of stability that arises above can be solved by using successive *orthogonal transformations* on C directly, to obtain its LQ decomposition. These methods obtain a factorization of the form

$$C = \begin{pmatrix} L & 0 \end{pmatrix}\begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix} = LQ_1\,, \tag{4.3}$$

where $L \in \mathbb{R}^{l\times l}$ is lower triangular and $(Q_1^T\ Q_2^T) \in \mathbb{R}^{n\times n}$ is orthogonal. This is essentially the QR factorization of C^T. The solution computed first by doing a forward substitution of

$$Lz = d\,, \tag{4.4}$$

for z, and then multiplying

$$y = Q_1^T z\,, \tag{4.5}$$

is the minimal norm solution of (4.1) since

$$C^{1,4} = C^T(CC^T)^{-1} = Q_1^T L^T (LQ_1 Q_1^T L^T)^{-1} = Q_1^T L^{-1} .$$

Due to their flexibility, we propose the usage of Givens rotations for an efficient computation of the decomposition (4.3) for sparse matrices. The drawbacks of the LQ decomposition over the other direct methods are higher computational and storage costs. Since the factorization is computed only once in the beginning and an improved stability is achieved the trade-off for a more expensive method is justified. For efficiency the multiply (4.5) is computed by using the representation of Q_1^T in Givens rotations, $Q_1^T = G_k^T \cdots G_1^T$. This also saves storage. To minimize the time for computing the decomposition (4.3) and doing the multiply (4.5), we want to keep the number of Givens rotations as low as possible. Thereto, the computational complexity of the forward substitution (4.4) depends on the sparsity structure of L. Different row and column orderings have been proposed in the literature, see for example [14], [23] and [10], to reorder the original matrix to a matrix that better fulfills these purposes. Note that while both row and column orderings have an impact on the overall computations, only the row ordering influences the final sparsity pattern in L. This is due to the fact that L equals, except from possible sign differences of columns, the Cholesky factor of CC^T, and the structure of the Cholesky factor is invariant under column permutations of C.

Our implementation of the LQ decomposition is based upon an implementation of the QR decomposition by Robey and Sulsky [28]. First, an a priori minimum degree row ordering is computed from the structure of CC^T to minimize the non-zeros in L. Then, during the actual factoring process the *variable pair strategy* proposed by Robey and Sulsky [28] is used to determine the column ordering. It attempts minimizing local intermediate fill-in and thus also the number of Givens rotations used, resulting in reduced computational time.

4.2 A Distributed Memory Implementation.

For the description of our distributed memory implementation of the RPSC algorithm we make use of a simple message-passing notation using the following blocking send and receive primitives:

send(<*data*>, <*destination*>)
recv(<*data*>)

send(buf, j) issued in the node program on processor i sends the contents of buffer buf to processor j and resumes program execution as soon as the message has left the application space. **recv**(buf) blocks the node application until a message arrives in the memory buffer. These two routines are similar to the blocking send and receive functions provided by Intel's NX message passing interface [27]. For simplicity we assume that p processors are available and the system matrix, A, is split into p overlapping parts, $(A_i)_{i=1,\ldots,p}$, according to a chosen regular block decomposition. The vector x representing the approximate solution is distributed in an overlapped fashion such that the components of

```
(* Global set-up *)
choose regular block decomposition (J^i)_{i=1,...,p}
choose consistent weighting (E_i)_{i=1,...,p}
choose initial guess x
(* On i-th processor *)
A_i = P_{V_i} A, b_i = P_{V_i} b
m = min{ j | e_j ∉ N(A_i)}, M = max{ j | e_j ∉ N(A_i)}
x_loc = (0, ..., 0, x_m, ..., x_M, 0, ..., 0)^T
LQ ← A_i                                   (* LQ decomposition *)
repeat
    r = b_i − A_i x_loc
    e = L^{-1} r                           (* forward solve *)
    e = Q^T e                              (* multiply *)
    e = Ee                                 (* weighting *)
    for j ∈ TO_i do
        buf = marshal(e, i, j)
        send(buf, j)
    end
    x_loc = add(x_loc, e)                  (* update x_loc *)
    RECV = {}
    while RECV ≠ FROM_i do
        recv(buf)
        (e, j) = unmarshal(buf)
        x_loc = add(x_loc, e)              (* update x_loc *)
        RECV = RECV ∪ {j}
    end
until convergence
```

Figure 4.1: A distributed memory implementation of the RPSC algorithm.

each local part, x_{loc}, correspond to the non-zero columns of the local part of the system matrix. With TO_i and $FROM_i$, we denote sets containing the numbers of the processors where data is sent or from where data is supposed to come. The marshaling function, $buf = \mathbf{marshal}(e, i, j)$, extracts those components of the correction vector, e, on processor i that are significant to x_{loc} on processor j. The extracted components are put into buf together with the processor tag i. The unmarshaling function, $(e, j) = \mathbf{unmarshal}(buf)$, recovers the correction vector, e, and its origin j. In $\mathbf{add}(x_{loc}, e)$ the local approximate solution is corrected with e. The above functions are implemented for an appropriate data structure using sparse storage techniques for efficiency and to save memory.

In Figure 4.1 we present our simplified distributed memory implementation of the RPSC algorithm. It is straightforward to generalize this implementation to more general decompositions. The bottle neck of the routine is the transmission of the local correction vectors and the update of the local parts of the approx-

imate solution. During this phase we need to communicate one floating point vector per floating point vector operation. Thus, the efficiency depends strongly on the performance of the interconnection network at hand.

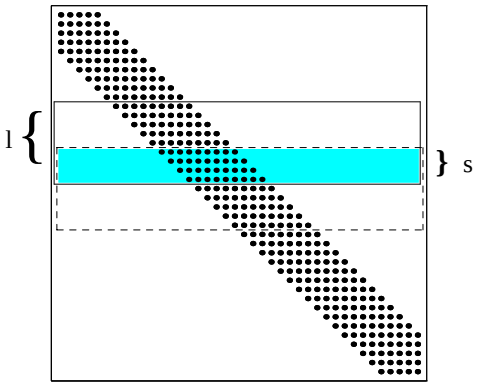


Figure 4.2: Two overlapping blocks of width l and regular overlap s.

4.3 Complexity.

We will now give complexity estimates for the case where the system matrix is narrow-banded and a band LQ decomposition is used for the underdetermined problems. Complexity estimates for more general patterns are highly dependent on the matrix structure and lie outside the scope of this paper. Our intention is not to give rigorous bounds, but to try to reflect the general behavior of the algorithm. We assume the system matrix, $A \in \mathbb{R}^{n\times n}$, to be narrow-banded with bandwidth r, i.e. $a_{ij} = 0$ for $|i-j| > r$, where $2r+1 \ll n$. Using p processors, the system matrix is decomposed into p overlapping blocks, $(A_i)_{i=1,\dots,p}$, with overlap s, such that $A_1, A_p \in \mathbb{R}^{(n/p+s/2)\times n}$ and $A_i \in \mathbb{R}^{(n/p+s)\times n}$ for $i = 2,\dots,p-1$. For transparency, we define the block width of A_i by

$$l := \begin{cases} \frac{n}{p} + s\,, & 1 < i < p\,, \\ \frac{n}{p} + \frac{s}{2}\,, & i = 1, p\,, \end{cases} \tag{4.6}$$

see Figure 4.2. We will assume $l + 1 > r$, which guarantees that communication only takes place between neighboring processors. By referring back to Figure 4.1 we see that the parallel computational complexity of the RPSC method has the form

$$C_{RPSC}(n, r, p, s) = C_{LQ} + \text{iter} \cdot \left(C_{solve} + C_{comm} + C_{update} \right),$$

where C_{LQ}, C_{solve}, C_{comm} and C_{update} represent the complexities of the LQ decomposition, the forward solve and multiply phase, the communication and the update, all taken on the processor with the highest work load. *iter* stands for

the number of iterations needed until convergence is reached. The update phase includes weighting, calculation of both the new iterate and the new residual. The complexity is measured in flops, i.e. floating point operations. For simplicity an addition, a subtraction, a multiplication or a square root all count as one flop. On the other hand, we have for the serial complexity

$$C^{ser}_{RPSC}(n,r,p,s) \leq p \cdot C_{LQ} + p \cdot \text{iter} \cdot \left(C_{solve} + C_{update} \right),$$

since in this case there is no communication. It can be shown that

$$\begin{aligned} C_{LQ} &= 6\,r\,(2\,r+1)\,(l-2\,r) + 3\,r^2(3\,r+1) + 2\,r\,(2\,r-1)\,(r-1) \text{ flops}, \\ C_{solve} &= (4\,r-1)\,l - 2\,r\,(2\,r-1) + 6\,l\,r \text{ flops} \end{aligned}$$

and

$$C_{update} = 4\,l\,r + 6\,r + 5\,l + 4\,s \text{ flops}.$$

In this analysis we assume that the transfer time of a message between two processors is independent of their physical distance. Let σ be the startup time measured in number of floating point operations and τ the number of floating point operations executed during the transmission of one floating point number. Then the common linear model for the transmission time between two processors for a message of m floating point numbers is given by

$$\sigma + m\,\tau\,.$$

Here the communication complexity takes the form

$$C_{comm}(r,s) = 4\,\sigma + 4\,(r+s)\,\tau \text{ flops},$$

where we have taken into account that when $l+1 > r$, processor i needs only to communicate once with its neighboring processors $i-2, i-1, i+1$ and $i+2$ per iteration.

We define the *speedup* of the RPSC algorithm by

$$S_{RPSC}(n,r,p,s) := \frac{C^{ser}_{RPSC}}{C_{RPSC}}\,.$$

After some simplification we get

$$S_{RPSC}(n,r,p,s) \approx \frac{p}{1 + \frac{p\,C_{comm}}{14\,r\,n}}\,.$$

Thus, the speedup increases with p, but it is bounded,

$$\lim_{p\to\infty} S_{RPSC}(n,r,p,s) = \frac{14\,r\,n}{C_{comm}}\,.$$

However, if the problem size n increases proportional to the number of processors p, we see $\frac{p\,C_{comm}}{14\,r\,n} = const$ and so the speedup is proportional to p.

5 Numerical Experiments.

In this section we report on a number of numerical experiments conducted on the Intel Paragon XP/S5+ with operating system OSF/1, Release 1.3, at ETH Zürich. This distributed memory multicomputer has 96 compute nodes were each node consists of two i860XP RISC processors, the application and the communication processors. Each node has about 22 MB storage available for the application making it impossible to run larger problems on a single or a few processors without swapping data to the slow disk memory. Clearly, disk usage should be avoided for the sake of efficiency. All the computations reported here are done with a C implementation of the RPSC algorithm using message passing primitives from Intel's NX library [27]. The codes are run with IEEE arithmetic and compiled with the compiler switch -O3, making the compiler do basic scalar and pipelining optimizations where possible. We use a slightly modified definition of *speedup* and *efficiency* of *base* p_0

$$S_{p_0}(p) := \frac{T(p_0)\cdot p_0}{T(p)}\,, \quad E_{p_0}(p) := \frac{S_{p_0}(p)}{p}\,,$$

where p_0 is the minimum number of processors such that the program and data fit into main memory. For $p_0 = 1$, this resumes the standard definition.

As model problem we use the elliptic partial differential equation

$$-\Delta u + \gamma\left(x\frac{\partial u}{\partial x} + y\frac{\partial u}{\partial y}\right) + \beta u = g\,, \quad (x,y)\in(0,1)\times(0,1) \tag{5.1}$$

with Dirichlet boundary conditions. The differential equation is discretized over an $n\times n$ grid using centered differences for the first and second order derivatives resulting in a banded linear system of the form $A(\gamma,\beta)u = \hat{g}$, with $N = n^2$ unknowns. The parameters γ and β can be varied to make the problem more or less difficult to solve. By changing γ and β we influence the symmetry and the definiteness of the system. The right hand side of the system is chosen such that the solution is the same random vector for all our experiments. We use a uniform regular block decomposition and take the overlap to be the same for all blocks, so the work of solving each correction equation is approximately the same. We consider the following four types of consistent weighting:

1. $E_i := \frac{1}{m} I\,, \quad i = 1,\dots,m\,;$
2. $E_i(l,k) := \begin{cases} 1\,, & l = k \wedge (l \in \mathcal{J}^i \ \wedge\ l \notin \mathcal{J}^{i-1}\cup\mathcal{J}^{i+1})\,, \\ \frac{1}{2}\,, & l = k \wedge (l \in \mathcal{J}^{i-1}\cap\mathcal{J}^{i} \ \vee\ l \in \mathcal{J}^{i}\cap\mathcal{J}^{i+1})\,, \\ 0\,, & \text{otherwise}, \end{cases} \quad i = 1,\dots,m\,;$
3. $E_1 := \operatorname{diag}(\underbrace{1,\dots,1,\tfrac{\phi}{\phi+1},\dots,\tfrac{1}{\phi+1}}_{\text{components } j_1^1,\dots,j_{l_1}^1},0,\dots,0),$

   $E_i := \operatorname{diag}(0,\dots,0,\underbrace{\tfrac{1}{\phi+1},\dots,\tfrac{\phi}{\phi+1},1,\dots,1,\tfrac{\phi}{\phi+1},\dots,\tfrac{1}{\phi+1}}_{\text{components } j_1^i,\dots,j_{l_i}^i},0,\dots,0)\,,\ i = 2,\dots,m-1,$

   and $E_m := \operatorname{diag}(0,\dots,0,\underbrace{\tfrac{1}{\phi+1},\dots,\tfrac{\phi}{\phi+1},1,\dots,1}_{\text{components } j_1^m,\dots,j_{l_m}^m})\,;$

4. $E_i(l,k) := \begin{cases} 1\,, & l = k \wedge ((i,l) \in \{1\} \times \{j_1^1, \ldots, j_{l_1}^1 - \frac{\phi}{2}\})\,, \\ 1\,, & l = k \wedge ((i,l) \in \{2, \ldots, m-1\} \times \{j_1^i + \frac{\phi}{2}, \ldots, j_{l_i}^i - \frac{\phi}{2}\})\,, \\ 1\,, & l = k \wedge ((i,l) \in \{m\} \times \{j_1^m + \frac{\phi}{2}, \ldots, j_{l_m}^m\})\,, \\ 0\,, & \text{otherwise}\,, \end{cases}$

$i = 1, \ldots, m\,.$

Type 1 is the simplest weighting that satisfies the consistency condition of Definition 2.1 (iii). It weights each vector by $\frac{1}{m}$ and thus the global correction is the mean of the subspace corrections. By type 2 and type 3 the overlapping components are halved or multiplied by weights which are uniformly distributed between zero and one. Furthermore, by type 4 the components corresponding to half of the overlap of the subspace corrections is set equal to zero. We remark that by the weighting of type 2, 3 and 4, only the components of the corrections corresponding to nonzero diagonal elements of E_i's have to be computed.

Table 5.1: Timings of the LQ decomposition and 1 iteration in seconds for RPSC with different types of weighting(w). The problem with $N = 90000$ is decomposed in 90 blocks with an overlap of 20.

	Number of Processors				
w	15	18	30	45	90
none	9.26/0.85	7.64/0.66	4.50/0.43	2.93/0.30	1.38/0.18
1	9.82/0.81	8.15/0.69	4.72/0.44	3.05/0.32	1.41/0.20
2	9.81/0.81	8.15/0.68	4.73/0.43	3.04/0.32	1.41/0.20
3	9.81/0.81	8.15/0.68	4.73/0.43	3.04/0.32	1.41/0.20
4	9.72/0.81	8.15/0.69	4.75/0.44	3.05/0.32	1.41/0.20

Table 5.2: Speedup and efficiency of the iterations for RPSC with different types of weighting(w). The problem with $N = 90000$ is decomposed in 90 blocks with an overlap of 20.

	Number of Processors				
w	15	18	30	45	90
none	15/1.00	19.1/1.06	29.4/0.98	42.0/0.93	68.9/0.77
1	15/1.00	17.8/0.99	28.6/0.95	40.6/0.90	72.9/0.80
2	15/1.00	17.8/0.99	27.9/0.93	37.9/0.84	61.8/0.68
3	15/1.00	17.8/0.99	27.9/0.93	37.9/0.84	61.8/0.68
4	15/1.00	17.7/0.98	27.4/0.91	37.9/0.84	61.8/0.69

First we study the *scalability* of the algorithm when the number of processors vary. To do so we compare timings for the LQ decompositions and iterations for different weighting and overlap. Each iteration step consists of an update of the local residual and local solution vectors, a forward solve, multiplication by

Table 5.3: Timings of the LQ decomposition and 1 iteration in seconds for RPSC with different overlap(ov). The problem with $N = 90000$ is decomposed in 90 blocks and weighting of type 3 is used.

	Number of Processors				
ov	15	18	30	45	90
0	9.25/0.78	7.64/0.66	4.49/0.43	2.93/0.31	1.40/0.19
2	9.26/0.79	7.66/0.66	4.51/0.43	2.95/0.31	1.39/0.19
4	9.32/0.77	7.70/0.66	4.52/0.43	2.96/0.31	1.40/0.19
10	9.57/0.81	7.96/0.68	4.59/0.45	3.00/0.32	1.40/0.20
20	9.80/0.80	8.12/0.68	4.74/0.44	3.05/0.33	1.43/0.20
40	–	8.45/0.70	4.93/0.45	3.16/0.33	1.44/0.21
80	–	9.22/0.71	5.34/0.47	3.39/0.35	1.49/0.22
120	–	9.89/0.73	5.07/0.49	3.63/0.36	1.55/0.23
160	–	10.58/0.76	6.03/0.51	3.85/0.37	1.59/0.25
200	–	11.35/0.78	6.48/0.53	4.06/0.40	1.63/0.26
280	–	–	6.93/0.56	4.26/0.43	1.73/0.29
360	–	–	7.79/0.59	4.88/0.44	1.84/0.31

Givens rotations and the weighting of the correction, cf. Figure 4.1. The costs per iteration is measured by performing ten iterations and taking the average. We use the model problem (5.1) with $\gamma = 96$, $\beta = 0$ and $N = 90000$. In Table 5.1 the costs for the weighting in the RPSC algorithm with 90 blocks and an overlap of 20 is compared with the costs when no weighting is used. The case of no weighting resumes the block Cimmino method, see Section 6. Here the overlap is always zero and thus the LQ decompositions and the iterations are cheaper to compute. The resulting speedups and efficiencies are seen in Table 5.2. Clearly, the costs of the different weighting schemes 1 through 4 hardly differ.

Since even for larger overlaps there are practically no difference in the costs of the different weighting schemes we will now keep the weighting fixed, using type 3, and vary the overlap. In Table 5.3 the costs for different overlaps are shown. Table 5.4 contains the corresponding speedups and efficiencies. Computations where the whole problem did not fit into main memory have been omitted. We see that for our model problem and a fixed number of processors fairly large overlaps are computationally competitive to smaller ones and that by modest overlaps the RPSC method scales well per iteration.

We will now focus on the numerical convergence properties of the RPSC method for different weighting and overlap. Each test is started with the the same random vector and is stopped if convergence is not reached after 5000 steps. As convergence criterion we have chosen $\|u - u^k\|_\infty < TOL$, with u being the exact solution and $TOL = 10^{-4}$, which of course is rather unrealistic but serves well for our experiments. For the first test we set $\gamma = 96$, $\beta = 0$ and $N = 10000$, decomposing the problem in ten blocks on ten processors. The results are shown in Table 5.5. For the second test we consider $\gamma = 96$, $\beta = 0$ and $N = 90000$,

Table 5.4: Speedup and efficiency of the iterations for RPSC with different overlap(ov). The problem with $N = 90000$ is decomposed in 90 blocks and weighting of type 3 is used.

	Number of Processors				
ov	15	18	30	45	90
0	15/1.00	17.7/0.98	27.1/0.90	38.1/0.84	63.2/0.70
2	15/1.00	17.86/0.99	27.5/0.92	38.6/0.86	63.5/0.71
4	15/1.00	17.6/0.98	27.0/0.90	37.8/0.84	62.3/0.69
10	15/1.00	17.7/0.98	27.0/0.90	37.3/0.83	59.6/0.66
20	15/1.00	17.7/0.98	27.1/0.90	37.0/0.82	60.8/0.68
40	–	18/1.00	27.7/0.92	37.8/0.84	60.6/0.67
80	–	18/1.00	27.2/0.91	37.1/0.83	58.8/0.65
120	–	18/1.00	27.0/0.90	36.8/0.82	56.6/0.63
160	–	18/1.00	26.7/0.89	36.5/0.81	55.2/0.61
200	–	18/1.00	26.5/0.88	35.5/0.79	53.5/0.59
280	–	–	30/1.00	38.8/0.86	58.6/0.65
360	–	–	30/1.00	40.1/0.89	56.8/0.63

decomposing the problem in ninety blocks on ninety processors. The results are shown in Table 5.6. Both tests show that the RPSC method with and without overlap is superior to the block Cimmino method that does not converge within 5000 iterations. This is because the weighting 'filters out' less relevant information in the corrections. Especially, weighting of type 2–4 improves the convergence significantly. Note that by increasing the overlap we get faster convergence on the one hand, but increase the computational complexity for the LQ decomposition and each iteration on the other hand. Therefore, the overall computing time decreases until a *break point*, which signals a 'balance' between convergence rate and floating point operations. By the appropriate combination of overlap and weighting time gains around 20–40% can be achieved.

We finish this section by a comparison of the RPSC method and the *single-width separator* approach (SWS). The SWS is a parallel direct method designed for banded matrices, see [16] or [2]. Briefly, the SWS performs in parallel a block LU decomposition of the permuted original system reducing it to a block tridiagonal system of order $(p-1)k$ where k is the block size and p is the number of processors. This reduced system is solved by block cyclic reduction. While cyclic reduction is not perfectly parallelizable, the subsequent back substitution is. Arbenz and Gander [2] assume the system matrix to be diagonally dominant implying that the factorization can be achieved without pivoting. By taking $\beta \geq 2\gamma n$ this is assured for the model problem (5.1). They concluded that the SWS is effective for *very* narrow banded matrices but does not scale so well. The comparison for $\gamma = 96$, $\beta = 19200$ and $N = 10000$ is shown in Figure 5.1 (a), whereas for $\gamma = 96$, $\beta = 57600$ and $N = 90000$ it is presented in Figure 5.1 (b). For both measurements we used RPSC with weighting of type 4, $TOL = 10^{-8}$ and decomposed the problem such that each processor handled one block. For

Table 5.5: Number of iterations and time in seconds needed for convergence of the RPSC method for different weighting(w) and overlap. The problem with $N = 10000$ is decomposed in 10 blocks on 10 processors.

Overlap							
w	0	2	4	10	20	40	80
none	>5000	–	–	–	–	–	–
1	>5000	>5000	>5000	>5000	>5000	>5000	>5000
2	782/61.2	782/61.3	780/61.2	765/60.9	727/59.5	596/51.0	524/47.9
3	782/61.2	782/61.3	780/61.2	767/61.1	728/59.6	612/52.1	548/49.7
4	782/61.2	782/61.3	780/61.2	769/61.2	732/59.8	627/53.1	597/53.3
Overlap							
w	120	160	200	240	280	320	360
1	>5000	>5000	>5000	>5000	>5000	>5000	>5000
2	475/47.7	454/52.9	390/48.8	335/46.4	341/46.9	355/52.5	361/60.5
3	481/48.2	405/48.6	319/42.5	268/40.2	263/39.4	262/43.1	238/47.1
4	578/55.8	460/53.2	278/38.7	245/37.9	253/38.3	262/42.9	218/44.7

this diagonally dominant case RPSC converges rather quickly and due to the large bandwidth in our model problem it is competitive with the SWS method.

Diagonal dominance and high bandwidth favors the iterative solver, in particular if the band is sparsely populated as in this test case. However, for more general cases it sometimes needs many iterations to reach desired convergence. In such situations, the RPSC method will have to be accelerated by some extrapolation techniques or by combining it with some other e.g. CG-type iterative process [18].

Thanks to its good scaling behavior, the RPSC method seems to be a good candidate for a successful parallel preconditioner.

6 Related Work.

If the overlap in the RPSC method is chosen to be zero one obtains various types of the Cimmino projection method. With the system matrix decomposed into single rows we have the Cimmino method as originally proposed by Cimmino, see [9] and [32], in 1939. Later, non-overlapping block versions have been studied by Elfving [11], Ruiz [29], Arioli et al. [19] and Bramley and Sameh [7] in combination with conjugate gradient acceleration. These approaches differs mainly in the computation of the orthogonal projections as discussed in Section 4.1 and in the partitioning of the system matrix by using different subspace decompositions. Zilli [40] calculates the projections iteratively in the block-Cimmino method with the LSQR algorithm [24], which is based upon the Lanczos process, but does not achieve a better degree of accuracy for his test problems than a relative error of about 10^{-3} due to stagnation.

The RSSC method is basically the same as the PSH method (Projektion auf Schnitträume von Hyperebenen), [25], [30] and [31], and both become the Kaczmarz algorithm, introduced 1937 by Kaczmarz [17], for the single row case and

Table 5.6: Number of iterations and time in seconds needed for convergence of the RPSC method for different weighting(w) and overlap. The problem with $N = 90000$ is decomposed in 90 blocks on 90 processors.

Overlap				
w	0	40	80	120
none	>5000	–	–	–
2	3913/616.9	4371/712.1	3414/560.6	3135/531.0
3	3913/616.9	4352/708.6	3402/558.6	3053/510.0
4	3913/616.9	4339/706.6	3334/546.9	3251/542.0
Overlap				
w	160	200	280	360
2	3085/521.0	2865/490.8	3232/568.6	3002/542.2
3	3026/511.4	2938/503.1	3145/553.3	2820/509.6
4	2955/498.0	3217/549.1	3211/563.2	2738/490.5

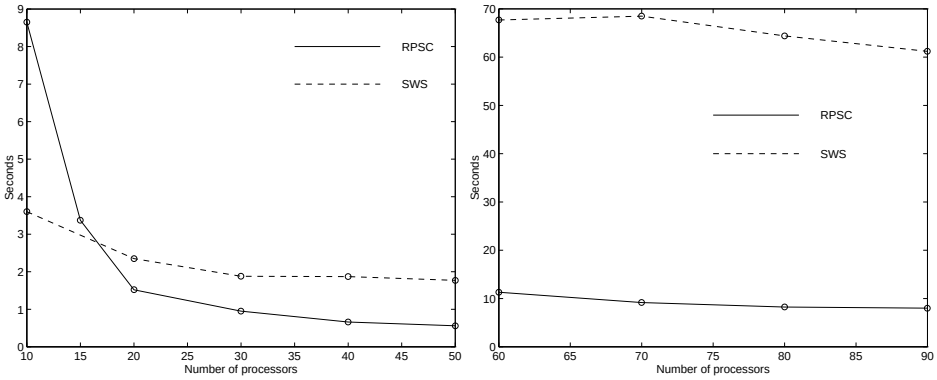
the block Kaczmarz projection method or its symmetrized version, block-SSOR, [11] and [7], for the non-overlapping block case. In [7], Bramley and Sameh also introduce the V-RP method which show computational advantage over the block-Kaczmarz method by using fewer matrix-vector operations. Benzi and Meyer [3] present a direct method that uses a Kaczmarz type scheme, which for some problems in chemical kinetics shows to be superior to standard sparse factorization techniques.

The CSSC algorithm is connected to the SPA method (Spaltenapproximation) introduced by Albrand, [1], and studied by Maess, [20], and by Schott, [30] and [31]. For the single column case these methods become the De la Garza algorithm, [21].

We remark that these types of algorithms can also be extended to inconsistent linear equations, i.e. systems where $b \notin \mathcal{R}(A)$ and $A \in \mathbb{R}^{n\times m}$, for the computation of generalized inverse solutions, see for example [33], [34], [25], [26], [11], [6] as well as [4]. For connections with the multisplitting method we refer to the references already given in Section 2.3.

7 Summary.

The theory of subspace correction introduced by Xu [38] for a unified treatment of a large class of iterative methods for the solution of symmetric positive definite linear systems was extended to the general nonsingular case. This was done by restricting the linear system to a finite set of intersecting subspaces, the union of which covers the original vector space. For solving the arising subsystems for the corrections, either in parallel or in succession, two very general algorithms were proposed, the *PSC and the *SSC methods. In the studied realizations, the underlying vector space was decomposed such that the subsystems became overlapping row or column blocks of the system matrix. Thereby, the usage of different weighting and overlap was incorporated to give good possibil-



(a) (b)

Figure 5.1: Comparison of RPSC and SWS for the model problem. In (a) $\gamma = 96$, $\beta = 19200$ and $N = 10000$, whereas in (b) $\gamma = 96$, $\beta = 57600$ and $N = 90000$.

ities of balancing the work load for parallel computations as well as improving the convergence rate. First, the convergence of the RSSC and CSSC methods was proved. Then, we proved the convergence of RPSC and CPSC, for a weighting corresponding to a component-wise mean of the subspace corrections. In the numerical experiments we focused on the RPSC method due to its natural parallelism. This method was shown to scale well on the Paragon XP/S5+, a distributed memory architecture. The convergence as well as the overall computational time were improved significantly by using weighting in combination with overlap. For diagonally dominant systems the RPSC has good convergence and for our test problem it was shown to be competitive with a direct solver, the single-width separator algorithm.

However, as a stationary method, RPSC in some cases needs too many iterations to achieve convergence. Therefore, it will be important to consider acceleration techniques or to combine the RPSC method with some other iterative process. We believe that these methods are potential preconditioners in parallel environments for standard iterative methods.

REFERENCES

1. H. J. Albrand, *Über die Lösung linearer Gleichungssysteme durch Spaltenapproximation*, Wiss. Z. Univ. Rostock, Math.-naturwiss., 21 (1972), pp. 755–757.

2. P. Arbenz and W. Gander, *A survey of direct parallel algorithms for banded linear systems*, Tech. Rep. 221, ETH Zürich, Computer Science Department, October 1994.

3. M. Benzi and C. D. Meyer, *A direct projection method for sparse linear systems*, SIAM J. Sci. Comput., 16 (1995), pp. 1159–1176.

4. M. Benzi, F. Sgallari, and G. Spaletta, *A parallel block projection method of the Cimmino type for finite Markov chains*, in Computations with Markov Chains: Proceedings of the Second International Workshop on the Numerical Solution of Markov Chains, Raleigh, NC, January 16-18, 1995, W. J. Stewart, ed., Kluwer Academic Publishers, 1995, pp. 65–80.
5. Å. Björck, *Pivoting and stability in the augmented system method*, Tech. Rep. LiTH-MAT-R-1991-30, Linköping University, Department of Mathematics, June 1991. `http://math.liu.se/ftp/reports/augms.dvi`.
6. Å. Björck and T. Elfving, *Accelerated projection methods for computing pseudoinverse solutions of systems of linear equations*, BIT, 19 (1979), pp. 145–163.
7. R. Bramley and A. Sameh, *Row projection methods for large nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 13 (1992), pp. 168–193.
8. S. L. Campbell and C. D. Meyer, *Generalized Inverses of Linear Transformations*, Surveys and Reference Works in Mathematics 4, Pitman, London, 1979.
9. G. Cimmino, *Calcolo approssimato per le soluzioni di sistemi di equazioni lineari*, Ric. Sci. Progr. tecn. econom. naz., 9 (1939), pp. 316–333.
10. I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, Oxford University Press, London, 1986.
11. T. Elfving, *Block-iterative methods for consistent and inconsistent linear equations*, Numer. Math., 35 (1980), pp. 1–12.
12. A. Frommer and G. Mayer, *Convergence of relaxed parallel multisplitting methods*, Lin. Alg. Appl., 119 (1989), pp. 141–152.
13. ———, *On the theory and practice of multisplitting methods in parallel computation*, Computing, 49 (1992), pp. 63–74.
14. A. George and E. Ng, *On row and column ordering in Givens reduction on sparce matrices*, SIAM J. Numer. Anal., 20 (1983), pp. 326–344.
15. G. H. Golub and C. F. van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 2nd ed., 1989.
16. S. Johnsson, *Solving narrow banded systems on ensemble architectures*, ACM Trans. Math. Softw., 11 (1985), pp. 271–288.
17. S. Kaczmarz, *Angenäherte Auflösung von Systemen linearer Gleichungen*, Bull. Internat. Acad. Polon. Sci. Lettres, Ser. A, 1937 (1937), pp. 355–357.
18. P. Kolm, *Parallel iterative methods for linear systems*, Master's thesis, Institute for Scientific Computing, Eidgenössische Technische Hochschule, Zurich, June 1994.
19. M. Arioli, I. Duff, J. Noailles, and D. Ruiz, *A block projection method for sparse matrices*, SIAM J. Sci. Stat. Comput., 13 (1992), pp. 47–70.
20. G. Maess, *Iterative Lösung linearer Gleichungssysteme*, Nova Acta Leopoldina, 52 (1979), pp. 5–79.

21. ——, *Projection methods solving rectangular systems of linear equations*, in Iterative Methods for the Solution of Linear Systems, A. Hadjidimos, ed., Elsevier Science Publishers B.V., 1988, pp. 107–119.

22. D. O'Leary and R. E. White, *Multi-splittings of matrices and parallel solution of linear systems*, SIAM J. Alg. Disc. Meth., 6 (1985), pp. 630–640.

23. G. Ostrouchov, *Symbolic Givens reduction and row-ordering in large sparse least squares problems*, SIAM J. Sci. Stat. Comput., 8 (1987), pp. 248–264.

24. C. C. Paige and M. A. Saunders, *LSQR: An algorithm for sparse linear equations and sparse least squares*, ACM Trans. Math. Softw., 8 (1982), pp. 43–71.

25. W. Peters, *Lösung linearer Gleichungssysteme durch Projektion auf Schnitträume von Hyperebenen und Berechnung einer verallgemeinerten Inversen*, Beiträge Num. Math., 5 (1976), pp. 129–146.

26. ——, *Eigenschaften einer nach dem Projektionsverfahren auf Schnitträume von Hyperebenen berechneten verallgemeinerten Matrixinversen*, Beiträge Num. Math., 6 (1977), pp. 127–132.

27. P. Pierce, *The NX message passing interface*, Parallel Comput., 20 (1994), pp. 463–480.

28. T. H. Robey and D. L. Sulsky, *Row ordering for a sparse QR decomposition*, SIAM J. Matrix Anal. Appl., 15 (1994), pp. 1208–1225.

29. D. F. Ruiz, *Résolution de grands systèmes linéaires creux non symétriques par une méthode itérative par blocs dans un environnement multiprocesseur*, techn. report, CERFACS, Toulouse, 1992.

30. D. Schott, *Endlich erzeugte Projektionsverfahren zur Lösung linearer Gleichungen im Hilbertraum*, Rostocker Mathematisches Kolloquium, 16 (1981), pp. 103–128.

31. ——, *Konvergenzsätze für Verallgemeinerungen von PSH- und SPA-Verfahren*, Math. Nachr., 118 (1984), pp. 89–103.

32. F. Sloboda, *A projection method of the Cimmino type for linear algebraic systems*, Parallel Computing, 17 (1991), pp. 435–442.

33. K. Tanabe, *Projection method for solving a singular system of linear equations and its applications*, Numer. Math., 17 (1971), pp. 203–214.

34. ——, *Characterization of linear stationary iterative processes for solving a singular system of linear equations*, Numer. Math., 22 (1974), pp. 349–359.

35. R. S. Varga, *Matrix Iterative Analysis*, Prentice-Hall Inc., 1962.

36. R. E. White, *Multiplitting with different weighting schemes*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 481–493.

37. ——, *Multisplitting of a symmetric positive definite matrix*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 69–82.

38. J. Xu, *Iterative methods by space decomposition and subspace correction*, SIAM Rev., 34 (1992), pp. 581–613.

39. D. M. Young, *Iterative Solution of Large Linear Systems*, Academic Press, New York, 1971.

40. G. Zilli, *Parallel implementation of a row-projection method for solving sparse linear systems*, Supercomputer, (1993), pp. 33–43.