Electronic Transactions on Numerical Analysis.
Volume 5, pp. 48-61, June 1997.

ISSN 1068-9613.

ETNA
Kent State University
etna@mcs.kent.edu

ASYNCHRONOUS WEIGHTED ADDITIVE SCHWARZ METHODS*

ANDREAS FROMMER†, HARTMUT SCHWANDT‡, AND DANIEL B. SZYLD§

Abstract. A class of asynchronous Schwarz methods for the parallel solution of nonsingular linear systems of the form $Ax = f$ is investigated. This class includes, in particular, an asynchronous algebraic Schwarz method as well as asynchronous multisplitting. Theorems are obtained demonstrating convergence for the cases when A^{-1} is nonnegative and when A is an H-matrix. The results shown are for both the situations with or without overlap between the domains in which an underlying mesh is divided, if such a mesh exists. Numerical experiments on systems of up to over ten million variables on up to 256 processors are presented. They illustrate the convergence properties of the method, as well as the fact that when the domains are not all of the same size, the asynchronous method can be up to 50% faster than the corresponding synchronous one.

Key words. Asynchronous methods, monotone matrices, H-matrices, linear system, parallel algorithms, multisplittings, additive Schwartz.

AMS subject classifications. 65F10, 65Y05.

1. Introduction.

Consider the discretization of a linear second order elliptic boundary value problem (b.v.p.) on a domain Ω, and a fixed number of open subdomains $\Omega_l \subset \Omega$, $l = 1, \ldots, L$, such that $\cup\Omega_l = \Omega$. In general, one may assume that contiguous subdomains have a nonempty intersection and, in fact, this intersection may be larger than just the boundary of the subdomains, i.e., there may be some overlap between contiguous subdomains; see Section 4 for specific examples of these subdomains.

The general idea of the Schwarz Alternating Procedure (SAP) is to solve the b.v.p. restricted to each subdomain, using as boundary conditions the function values of the (approximate) solution of neighboring subdomains. This process is repeated and, under certain conditions, the process converges to the solution of the b.v.p. on Ω. In the additive Schwarz approach, which is the one considered in this paper, the approximate solution on Ω is obtained by adding up the solutions on all subdomains. When there is overlap, some weights are introduced to keep the resulting approximation consistent with the original b.v.p. These ideas go back to Schwarz [21], and have been revived in the last decade as the basis for domain decomposition methods, where these procedures are used as a preconditioner for the solution of the discretized nonsingular linear system

$$Ax = f, \;\; A \in \mathbb{R}^{n\times n}, \;\; f \in \mathbb{R}^n \tag{1.1}$$

which are solved with a conjugate gradient or another Krylov-type method [6, 14, 15, 22]. Since there is no need for consistency when a preconditioner is used, one usually takes unweighted sums on the overlaps in these situations.

Weighted additive Schwarz methods can be applied to linear systems of the form (1.1) even if they are not obtained from discretizations of b.v.p., e.g., by considering linear systems $A_m y^m = f_m$, analogous to solutions in a subdomain Ω_m, and taking as an approximation to the solution of (1.1) the vector $x = \sum E_m y^m$, where the weighting matrices are such that $\sum E_m = I$. Such algebraic additive Schwarz algorithms have been studied, e.g., in

* Received January 21, 1997. Accepted for publication June 3, 1997. Communicated by R. S. Varga.
† Fachbereich Mathematik, Bergische Universität GH Wuppertal, D-42097 Wuppertal, Germany, frommer@math.uni-wuppertal.de
‡ Fachbereich Mathematik, Technische Universität Berlin, D-10623 Berlin, Germany, schwandt@math.tu-berlin.de
§ Department of Mathematics, Temple University, Philadelphia, Pennsylvania 19122-2585, USA, szyld@math.temple.edu. Supported in part by National Science Foundation grants INT-9123273 and DMS-9625865

[8, 18, 19, 20]. In the recent paper [11] a general framework for both weighted additive Schwarz and multisplitting methods is developed; see, e.g., [5] for extensive bibliographical references to multisplitting methods.

One of the advantages of additive Schwarz is that each subdomain can be handled by a different processor of a parallel computer. Each processor solves a linear system, using a right hand side which includes information collected from other processors. In two-stage variants (also called inner/outer or inexact methods), an (inner) iterative method is used in each processor, until a local convergence criterion is satisfied. The only synchronization point is the wait for fresh information from the other processors. In this paper, we consider *asynchronous* (one- or two-stage) weighted additive Schwarz methods, i.e., methods in which the linear system in each processor is solved (or approximated) anew with whatever information is available at the moment, without waiting for new information from all other processors. Intuitively, if there is a large load imbalance, i.e., if some processors have substantially more work to do per iteration than others, one can expect the asynchronous version to converge faster than the synchronous one; see the discussion at the end of this section. This is confirmed by a series of extensive experiments reported in Section 4.

In the rest of this introduction, we review the weighted additive Schwarz framework introduced in [11] and formally define the asynchronous method studied in this paper. Some preliminary results and more notations are introduced in Section 2, while our convergence theorems are presented in Section 3.

DEFINITION 1.1. *Let $A \in \mathbb{R}^{n\times n}$ be nonsingular. A collection of L splittings $A = M_l - N_l \in \mathbb{R}^{n\times n}$, $l = 1,\dots,L$, and L^2 nonnegative diagonal matrices $E_{l,m} \in \mathbb{R}^{n\times n}$ such that $\sum_{m=1}^{L} E_{l,m} = I$ for $l = 1,\dots,L$ is called a* weighted additive Schwarz-type splitting *of A. Given initial approximations $x^{0,l}$, $l = 1,\dots,L$, the corresponding* weighted additive Schwarz method *computes iterates $x^{k,l}$, $l = 1,\dots,L$, by*

$$x^{k+1,l} = \sum_{m=1}^{L} E_{l,m} y^{k,m}, \quad k = 0,1,\dots,$$

where

$$M_m y^{k,m} = N_m x^{k,m} + f, \quad m = 1,\dots,L.$$

As was shown in [11], this class of methods comprises the classical multisplittings from [17] ($E_{l,m} = E_m$ for all l) as well as the algebraic additive Schwarz methods from [8, 18, 19, 20] (where all entries of all $E_{l,m}$ are either 0 or 1).

DEFINITION 1.2. *Let $A \in \mathbb{R}^{n\times n}$ be nonsingular. Let a collection of L splittings $A = M_l - N_l$, $l = 1,\dots,L$ be given. For all $k \in \mathbb{N}_0 := \{0,1,\dots\}$ introduce a collection of L^2 nonnegative diagonal matrices $E^{(k)}_{l,m}$ such that $\sum_{m=1}^{L} E^{(k)}_{l,m} = I$ for all l and k. Moreover, for $k \in \mathbb{N}_0$ let $I_k \subseteq \{1,\dots,L\}$ and $(s_1(k),\dots,s_L(k)) \in \mathbb{N}_0^L$ be such that*

(i) $s_l(k) \le k$ *for all l, k,*
(ii) $\lim_{k\to\infty} s_l(k) = \infty$ *for $l = 1,\dots,L$,*
(iii) *the sets $\{k \mid l \in I_k\}$ are unbounded for $l = 1,\dots,L$.*

Then, given initial approximations $x^{0,l}$, $l = 1,\dots,L$, the iterative method which computes *iterates $x^{k,l}$ according to*

$$x^{k+1,l} = \begin{cases} x^{k,l} & \text{if } l \notin I_k \\ \sum_{m=1}^{L} E_{l,m}^{(k)} y^{s_m(k),m} & \text{if } l \in I_k, \end{cases} \tag{1.2}$$

where

$$y^{s_m(k),m} = M_m^{-1}(N_m x^{s_m(k),m} + f) \tag{1.3}$$

is termed an asynchronous weighted additive Schwarz method (AWAS) *for solving* (1.1).

Note that in this definition the weighting matrices $E_{l,m}^{(k)}$ are allowed to depend on the iteration level k. This is important in practice as the following discussion shows.

Weighted additive Schwarz methods provide a convenient framework to study methods which rely on a (possibly overlapping) block distribution of variables. In a parallel computing environment with L processors, each processor P_m is assigned to compute one intermediate result $y^{k,m}$ per iteration and to accumulate the intermediate results from the other processors to get 'its own' iterate $x^{k+1,m}$. The fact that we can have a different set of weighting matrices $E_{l,m}$, $l = 1, \ldots, L$ on each processor allows us to keep different approximations to the solution on the overlapping parts. Also, an (overlapping) block decomposition means that most diagonal entries of each $E_{l,m}$ are zero so that corresponding entries of $y^{k,m}$ need not be computed. Typically, then, the work of each processor per iteration is basically given by (approximately) solving a subsystem of the size of the respective block. If the time for computing the solutions of the subsystems differs from processor to processor, we end up with idle times during each iterative step if processors are forced to wait for the slowest to finish. The description of the asynchronous iteration (1.2), together with conditions (i)–(iii), is the standard description of the iterative process which arises when, instead of waiting, processors are allowed to start the next iteration, taking 'older' iterates as input from those processors which have not yet finished computing the current ones; see, e.g., [1, 4, 7, 10, 13]. If in such a context, there are several (delayed) approximations available for the same component (due to overlaps), it appears sensible to privilege the most recent one. Such a choice is made by setting the corresponding diagonal entry of the appropriate weighting matrix $E_{l,m}^{(k)}$ to 1, thus justifying the use of k-dependent weighting matrices in our asynchronous model.

2. Notation and auxiliary results.

In $\mathbb{R}^n$ and $\mathbb{R}^{n\times n}$ the relation $\geq$ denotes the natural componentwise partial ordering. In addition, for $x, y \in \mathbb{R}^n$ we write $x > y$ if $x_i > y_i$, $i = 1, \ldots, n$. A vector $x \geq 0$ ($x > 0$) is called *nonnegative* (*positive*). Similarly, $A \in \mathbb{R}^{n\times n}$ is called *nonnegative* if $A \geq O$.

A nonsingular matrix $A = (a_{ij}) \in \mathbb{R}^{n\times n}$ is termed *M-matrix*, if $a_{ij} \leq 0$ for $i \neq j$ and $A^{-1} \geq O$. Alternatively, instead of $A^{-1} \geq O$ we can equivalently require $Au > 0$ for some vector $u > 0$; see, e.g., [3].

For a given matrix $A = (a_{ij}) \in \mathbb{R}^{n\times n}$, its *comparison matrix* $\langle A\rangle = (\alpha_{ij}) \in \mathbb{R}^{n\times n}$ is defined by

$$\alpha_{ij} = \begin{cases} |a_{ii}| & \text{if } i = j \\ -|a_{ij}| & \text{if } i \neq j \end{cases} .$$

A is called an *H-matrix* if $\langle A\rangle$ is an M- matrix.

H-matrices are always nonsingular; see, e.g., [3]. According to our previous remark on M-matrices, $A = (a_{ij})$ being an H-matrix is characterized by the existence of a positive vector u such that $\langle A\rangle u > 0$. Writing this componentwise yields

$$|a_{ii}|u_i > \sum_{j=1, j\neq i}^{n} |a_{ij}|u_j \ , \quad i = 1, \ldots, n \ .$$

Therefore, H-matrices may be viewed as generalized diagonally dominant matrices with weights u_i. Special families of H-matrices include strictly diagonally dominant matrices (take $u = (1, \ldots, 1)^T$) as well as irreducibly diagonally dominant matrices or weakly Ω-diagonally dominant matrices; see [9, 16, 23, 24].

The absolute value $|A|$ of $A = (a_{ij}) \in \mathbb{R}^{n\times n}$ is again defined componentwise, i.e., $|A| = (|a_{ij}|) \in \mathbb{R}^{n\times n}$.

Given $u > 0,\ u \in \mathbb{R}^n$, we define the weighted max-norm $||\cdot||_u$ in $\mathbb{R}^n$ by $||x||_u = \max_{1\leq i\leq n} |x_i/u_i|$.

LEMMA 2.1. *Let $A \in \mathbb{R}^{n\times n}$, $u \in \mathbb{R}^n,\ u > 0$, and $\theta > 0$ such that*

$$|A|u \leq \theta u \ .$$

Then, $||A||_u \leq \theta$. In particular, $||Ax||_u \leq \theta||x||_u$ for all $x \in \mathbb{R}^n$.

Proof. For $x \in \mathbb{R}^n$ we have

$$(Ax)_i = \sum_{j=1}^{n} a_{ij}x_j = \sum_{j=1}^{n} a_{ij}u_j \left(\frac{x_j}{u_j}\right) \ ,$$

so that

$$\begin{aligned} |(Ax)_i| \leq \sum_{j=1}^{n} |a_{ij}|u_j \cdot \left|\frac{x_j}{u_j}\right| &\leq ||x||_u \cdot \sum_{j=1}^{n} |a_{ij}|u_j \\ &\leq ||x||_u \cdot \theta \cdot u_i \ , \end{aligned}$$

from which we get $||Ax||_u \leq \theta||x||_u$. $\square$

A representation $A = M - N,\ A, M, N \in \mathbb{R}^{n\times n}$ is termed a *splitting* of A if M is nonsingular. A splitting $A = M - N$ is termed *regular* if $M^{-1} \geq O$ and $N \geq O$, *weak regular* if $M^{-1} \geq O$ and $M^{-1}N \geq O$, and *H-compatible* if $\langle A\rangle = \langle M\rangle - |N|$; see [13]. Note that regular implies weak regular.

3. Convergence theory. We show that the asynchronous method (1.2) converges in two important cases, namely, when the coefficient matrix A is monotone, i.e., when $A^{-1} \geq O$, and when A is an H-matrix and the splittings are chosen in the appropriate manner. These results represent asynchronous counterparts of similar convergence theorems given in [11] for one- and two-stage variants of synchronous algebraic additive Schwarz methods. Our main result can be summarized as follows.

THEOREM 3.1. *Let the conditions of Definition 1.2 be satisfied. Then, for any set of initial vectors $x^{0,l}$, $l = 1, \ldots, L$, the AWAS method* (1.2) *converges to x^*, the solution of* (1.1)*, in the following two cases.*
(a) *$A^{-1} \geq O$ and each splitting $A = M_m - N_m$ is weak regular, $m = 1, \ldots, L$.*
(b) *A is an H-matrix and $\langle A\rangle \leq \langle M_m\rangle - |N_m|$, $m = 1, \ldots, L$.*

This main result will follow as a special case of our more general Theorem 3.3 given later. We point out that cases (a) and (b) have a wide range of applicability to discretizations of elliptic boundary value and other problems [3, 23]. In particular, (a) includes the case of A

being an M-matrix with the splittings corresponding to various classical (Jacobi- and Gauss-Seidel type) point and block iterative methods. Similarly, (b) includes analogous situations where A is an H-matrix. Yet another special case of (b) arises for H-compatible splittings; see, e.g., [5, 11, 12], and the references given therein.

Let us collect together the iterates $x^{k,l}$ from (1.2) into a vector

$$\boldsymbol{x}^k = \left((x^{k,1})^T, \ldots, (x^{k,L})^T\right)^T \in \mathbb{R}^{Ln}$$

and define

$$\boldsymbol{c}^{(k)} = \left(\left(\sum_{m=1}^{L} E_{1,m}^{(k)} M_m^{-1} f \right)^T, \ldots, \left(\sum_{m=1}^{L} E_{L,m}^{(k)} M_m^{-1} f \right)^T \right)^T \in \mathbb{R}^{Ln} . \tag{3.1}$$

Let us use the notation $[\,]_l$ to denote the l-th block component in $\mathbb{R}^n$ of a vector in $\mathbb{R}^{Ln}$, so that, for example,

$$[\boldsymbol{c}^{(k)}]_l = \sum_{m=1}^{L} E_{l,m}^{(k)} M_m^{-1} f .$$

By setting $T_m = M_m^{-1} N_m, \;\; m = 1, \ldots, L$, and

$$\boldsymbol{H}^{(k)} = \begin{pmatrix} E_{1,1}^{(k)} T_1 & \cdots & E_{1,L}^{(k)} T_L \\ \vdots & \ddots & \vdots \\ E_{L,1}^{(k)} T_1 & \cdots & E_{L,L}^{(k)} T_L \end{pmatrix} \in \mathbb{R}^{Ln \times Ln} , \tag{3.2}$$

we can rewrite (1.2) as

$$[\boldsymbol{x}^{k+1}]_l = \begin{cases} [\boldsymbol{x}^k]_l & \text{if } l \notin I_k \\ \left[\boldsymbol{H}^{(k)}([\boldsymbol{x}^{s_1(k)}]_1^T, \ldots, [\boldsymbol{x}^{s_L(k)}]_L^T)^T\right]_l + [\boldsymbol{c}^{(k)}]_l & \text{if } l \in I_k . \end{cases} \tag{3.3}$$

Denote $\boldsymbol{x}^* = (x^*, \ldots, x^*) \in \mathbb{R}^{Ln}$, where $x^* = A^{-1} f \in \mathbb{R}^n$. It was shown in [11] that $\boldsymbol{x}^*$ is a fixed point of all equations $\boldsymbol{x} = \boldsymbol{H}^{(k)} \boldsymbol{x} + \boldsymbol{c}^{(k)}$, with $\boldsymbol{H}^{(k)}$ of the form (3.2) and $\boldsymbol{c}^{(k)}$ of the form (3.1). The following theorem therefore is a special case of Theorem 3.2 in [13].

THEOREM 3.2. *For $l = 1, \ldots, L$ let $||\cdot||_l$ be a norm on $\mathbb{R}^n$, let $a \in \mathbb{R}^L$, $a > 0$ and denote $||\cdot||_a$ the weighted max-norm on $\mathbb{R}^{Ln}$ given by*

$$||\boldsymbol{x}||_a := \max_{l=1,\ldots,L} \left\{ \frac{1}{a_l} ||[\boldsymbol{x}]_l||_l \right\} .$$

Assume that there exists a constant $\theta \in [0, 1)$ such that for all $k = 1, 2, \ldots$,

$$||\boldsymbol{H}^{(k)} \boldsymbol{x}||_a \leq \theta \cdot ||\boldsymbol{x}||_a \text{ for all } \boldsymbol{x} \in \mathbb{R}^{Ln} . \tag{3.4}$$

Then the asynchronous iteration (3.3) *converges to $\boldsymbol{x}^*$ for every starting vector $\boldsymbol{x}^0$.*

The following new result generalizes the corresponding result for the synchronous case, given as Theorem 4.3 in [11].

THEOREM 3.3. *Assume that there exists $u \in \mathbb{R}^n$, $u > 0$ such that*

$$|T_m| u \leq \theta u, \; \theta \in [0, 1) \; \text{ for } m = 1, \ldots, L. \tag{3.5}$$

Then $\lim_{k\to\infty} \boldsymbol{x}^k = \boldsymbol{x}^*$ *for the asynchronous iterates of (3.3).*

Proof. Denote by $\boldsymbol{u} = (u, \ldots, u) \in \mathbb{R}^{Ln}$. Then,

$$\left[|\boldsymbol{H}^{(k)}|\boldsymbol{u}\right]_l = \sum_{m=1}^{L} |E_{l,m}^{(k)} T_m| u = \sum_{m=1}^{L} E_{l,m}^{(k)} |T_m| u \le \theta \sum_{m=1}^{L} E_{l,m}^{(k)} u = \theta u \ ,$$

which results in $|\boldsymbol{H}^{(k)}|\boldsymbol{u} \le \theta \boldsymbol{u}$. Thus, by Lemma 2.1 we get

$$||\boldsymbol{H}^{(k)}\boldsymbol{x}||_{\boldsymbol{u}} \le \theta ||\boldsymbol{x}||_{\boldsymbol{u}} \quad \text{for all } \boldsymbol{x} \in \mathbb{R}^{Ln} \ . \tag{3.6}$$

But $||\boldsymbol{x}||_{\boldsymbol{u}} = \max_{l=1,\ldots,L} ||[\boldsymbol{x}]_l||_u$, so that (3.6) shows that the crucial assumption (3.4) of Theorem 3.2 is met (with $a = (1, \ldots, 1)^T$). ∎

To prove Theorem 3.1 all that remains to be shown is that in cases (a) and (b), the basic assumption (3.5) in Theorem 3.3 is fulfilled. To that end denote $e = (1, \ldots, 1)^T$ the vector with all unit entries in $\mathbb{R}^n$. A straightforward computation now shows that we can take $u = A^{-1}e$ for (a) and $u = \langle A \rangle^{-1} e$ for (b); see also Corollaries 4.4 and 4.5 in [11].

Another consequence of Theorem 3.3 is a convergence result for two–stage variants. In these variants, the solution of (1.3) is approximated using $p(m)$ iterations of an (inner) iterative method defined by the splittings $M_m = F_m - G_m$, $m = 1, \ldots, L$. It can be seen that after $p(m)$ iterations with an initial vector $x^{s_m(k),m}$, the result is the vector

$$\hat{y}^{s_m(k),m} = (F_m^{-1}G_m)^{p(m)} x^{s_m(k),m} + \sum_{\nu=0}^{p(m)-1} (F_m^{-1}G_m)^{\nu} F_m^{-1}(N_m x^{s_m(k),m} + f),$$

which can be written in the form (1.3) with the unique splitting $A = \hat{M}_m - \hat{N}_m$, induced by the iteration matrix

$$\hat{T}_m = (F_m^{-1}G_m)^{p(m)} + \sum_{\nu=0}^{p(m)-1} (F_m^{-1}G_m)^{\nu} F_m^{-1}, \tag{3.7}$$

namely, $\hat{M}_m = M_m \left(I - (F_m^{-1}G_m)^{p(m)}\right)^{-1}$, and $\hat{N}_m = A - \hat{M}_m$, satisfying $\hat{T}_m = \hat{M}_m^{-1}\hat{N}_m$; see, e.g., [2, 5, 11, 13].

Abusing the notation, we drop the hats in these last identities so that two-stage variants of the AWAS can be described by the same equations (3.1)–(3.3). Thus Theorem 3.3 applies as long as the hypotheses (3.5) are satisfied. This yields the following result.

THEOREM 3.4. *Let the conditions of Definition 1.2 be satisfied. Then, for any set of initial vectors* $x^{0,l}$, $l = 1, \ldots, L$, *and for any choice of inner iterations* $p(m) \ge 1$, $m = 1, \ldots, L$, *the two-stage AWAS method* (1.2) *with iteration matrices* (3.7) *converges to* x^*, *the solution of* (1.1), *in the following two cases.*
(a) $A^{-1} \ge O$ *and each of the splittings* $A = M_m - N_m$ *and* $M_m = F_m - G_m$ *is weak regular, with* $F_m^{-1}N_m \ge 0$, $m = 1, \ldots, L$.
(b) A *is an* H*-matrix,* $\langle A \rangle \le \langle M_m \rangle - |N_m|$, *and the splittings* $M_m = F_m - G_m$ *are* H*-compatible,* $m = 1, \ldots, L$.

The proof follows again from verifying (3.5) with $u = A^{-1}e$ for (a) and $u = \langle A \rangle^{-1}e$ for (b). Details can be found in [11], Corollaries 5.2 and 5.3.

Finally, we can also consider non-stationary two-stage variants where $p(m)$, the number of inner iterations, also depends on k, the outer iteration level. We thus have $p = p(m, k)$ instead of $p = p(m)$. In this case, each matrix $\hat{T}_m$ in (3.7) depends on k. The convergence of the asynchronous method can be proved under the same hypothesis and in the same manner as in Theorem 3.4. We refer the reader to [13], where a similar result was given for asynchronous block methods and multisplittings.

4. Numerical results. Many sets of computational experiments have been carried out on a distributed memory CRAY T3D (256 processors) and a shared memory CRAY J90 (16 processors) at the Konrad-Zuse-Zentrum für Informationstechnik (ZIB), Berlin. Details about these computers and about our implementation of the asynchronous and synchronous codes are given in an appendix.

The numerical experiments reported here are based on the following second order elliptic boundary value problem

$$-L(u) := (a(x)u_x)_x + (b(y)u_y)_y + \alpha u = g,$$

on a rectangle $\Omega = [0,1] \times [0,s]$ with Dirichlet boundary conditions. We consider the coefficients

$$a(x) = 1 + 0.02x, \;\; b(y) = 1 + 0.002y\,.$$

The parameter α is chosen from the set $\{0.01, 0.1, 1.0\}$ in order to test different degrees of difficulty. We prescribe a solution

$$u^*(x,y) = x + y \quad \text{for } 0 \le x \le 1,\; 0 \le y \le s,$$

and define the righthand side $g := -L(u^*)$, in order to being able to control the quality of the computed iterates.

The five point discretization with central differences and a mesh size of $h = 1/(p+1)$ leads for $s = (q+1)\cdot h$ to the $q \times q$ block tridiagonal coefficient matrix

$$A = (-B_{j-1}, A_j, -B_j)_{j=1}^{q}$$

of order $n = pq$ with diagonal $p \times p$ blocks

$$B_j = b\left(\tfrac{2j+1}{2}h\right)\cdot I$$

and tridiagonal $p \times p$ blocks

$$A_j = \left(-a(\tfrac{2i-1}{2}h),\; a(\tfrac{2i-1}{2}h) + a(\tfrac{2i+1}{2}h) + b(\tfrac{2j-1}{2}h) + b(\tfrac{2j+1}{2}h) + \alpha,\; -a(\tfrac{2i+1}{2}h)\right)_{i=1}^{p}.$$

Thus, the discretized system is (1.1), where f contains the discretized values of g and the contributions of the boundaries. The matrix A thus constructed is strictly diagonally dominant, and due to the sign of its entries, it is an M-matrix, and thus we have $A^{-1} \ge O$. Since u^* is linear, the solution x^* of the discretized system is made up from the discrete values of u^*, i.e., $x^*(i,j) = u^*(ih, jh),\ 1 \le i \le p, 1 \le j \le q$.

We introduce a simple domain decomposition of L overlapping rectangles

$$\begin{aligned}
\Omega_i &= [0,1] \times [l_i, r_i],\; i = 1,\ldots,L,\\
l_i &= \beta_{i-1} - ov,\; r_i = \beta_i + 1 + ov,\\
l_1 &= 0,\; r_L = q+1,
\end{aligned}$$

for various choices of the β_i which indicate the 'raw' subdomain borders. Every subdomain is identified with one processor, i.e., the number L of subdomains is identical to the number of processors used. Each rectangle consists of $r_i - l_i - 1$ grid lines and two additional artificial boundaries l_i, r_i, except for the first left and last right 'true' boundaries. Thus, the number of grid lines in the overlap between two rectangles (subdomains) is $2ov$ and the number of lines in each subdomain is as follows.

- subdomain (processor) 1: $\beta_1 + ov - 1$;
- subdomain (processor) i: $\beta_i - \beta_{i-1} + 2ov, \quad 2 \le i \le L-1$;
- subdomain (processor) L: $q - \beta_L + ov - 1$.

The diagonal blocks of A associated with each subdomain Ω_l define the matrices M_l of Definition 1.2. With these choices, the matrices M_l are also strictly diagonally dominant M-matrices and the corresponding splittings $A = M_l - N_l$ are regular splittings, $l = 1, \ldots, L$. The weighting matrices $E^{(k)}_{l,l}$ have diagonal entries 1 in all positions corresponding to Ω_l and 0 otherwise, independently of k. For the neighboring subdomains Ω_{l-1} and Ω_{l+1} we take the diagonal entries in $E^{(k)}_{l,l-1}$ and $E^{(k)}_{l,l+1}$ to be 1 on that part of the domain which does not intersect with Ω_l and 0 otherwise, again independently of k. The choice for the other weighting matrices $E^{(k)}_{l,m}$ is irrelevant, since $y^{l,m}$ in $M_l y^{l,m} = N_l x^{l,m} + f$ only depends on the entries in $x^{l,m}$ belonging to $\Omega_{l-1} \cup \Omega_l \cup \Omega_{l+1}$.

q	dom. dec.	it_a	c_a	t_a	W_a	it_s	c_s	t_s	W_s	Q
135	11-12-18	172-215	86.73	5.48	15.84	189	82.81	10.55	7.85	0.52
136	11-12-19	167-215	87.88	5.57	15.79	189	83.31	10.69	7.79	0.52
137	11-12-20	171-227	91.67	5.79	15.82	189	83.50	10.87	7.68	0.53
138	11-12-21	167-229	92.61	5.85	15.83	189	84.24	10.77	7.82	0.54
139	11-12-22	168-237	95.85	6.06	15.83	189	84.51	11.01	7.68	0.55
140	11-12-23	166-239	96.92	6.13	15.81	189	84.87	11.13	7.62	0.55
141	11-12-24	169-251	101.6	6.41	15.85	189	84.73	11.42	7.42	0.56
142	11-12-25	163-264	106.7	6.74	15.84	189	85.86	11.29	7.60	0.60
143	11-12-26	162-253	102.4	6.47	15.83	189	85.36	11.63	7.34	0.56
144	12-13-12	193-211	88.20	5.55	15.88	191	84.54	9.79	8.63	0.57
145	12-13-13	194-212	89.16	5.63	15.85	191	84.66	9.95	8.51	0.57

TABLE 4.1
Moderate load imbalance with larger last domain, CRAY J90, BJ, $\alpha = 0.1$, $L = 16$, $p = 2000$, $itin = 4$, $ov = 1$

We report results of experiments for two-stage synchronous and asynchronous weighted additive Schwarz methods of the form (3.7). We use the Block Jacobi (BJ) method as the inner iteration with each diagonal block corresponding to variables from one grid line. This implies that the inner splittings are also regular splittings, and thus the matrices and splittings used in the numerical results in this section satisfy the hypotheses of the convergence theorems in Section 3.

In the tables, we denote by it_a, it_s, the number of outer iterations in the asynchronous and synchronous case, respectively, and by $itin$ that of inner iterations. This means that $p(m)$ from Section 3 is always $itin$, independently of m, except in Table 4.6. The quantities t_a, t_s are the computing times (in seconds) for asynchronous and synchronous runs, respectively. In some examples we also note the respective CPU times c_a and c_s, i.e., the total *accumulated* CPU time used by all processors. The quotient of asynchronous and synchronous times is defined by $Q = t_a/t_s$. By $W_s = c_s/t_s$ and $W_a = c_a/t_a$, we denote the work load, i.e., the efficiency times the number of processors. We indicate the distribution of the grid lines a processor receives, i.e., the number of grid lines of the respective subdomain, in the following form, where for simplicity of the notation we assume L is even.

- domain 1 - (domain i - domain $i+1$ (i even, $2 \le i < L-1$)) - domain L.
  For example, for $L = 8$, 7-(10-8)-15 means 7, 10, 8, 10, 8, 10, 8, 15.
- domain 1 - domain i ($2 \le i \le L-1$) - domain L.
  For example, for $L = 8$, 7-3-5 means 7, 3, 3, 3, 3, 3, 3, 5.

As a convergence criterion we use a modified relative difference of two iterates

$$\max_{1\leq i\leq p, 1\leq j\leq q} \left\{ \frac{\left|x_{i,j}^{k+1} - x_{i,j}^{k}\right|}{\max\left\{\left|x_{i,j}^{k}\right|, 10^{-300}\right\}} \right\} < 10^{-14}.$$

All experiments were run with this stopping criterion and we always observed a relative error with respect to the prescribed solution x^* of the same order of magnitude, namely, below 10^{-14} on the CRAY T3D and below 10^{-13} on the CRAY J90.

The first set of results are reported in Table 4.1. Here, we test several rather small values of q on $L = 16$ subdomains (processors). The first 15 subdomains are almost equally sized while the last one is moderately larger. In this example, the asynchronous version is significantly better even for almost equally sized domains ($q = 144$, 145). The mean work load W_s in the synchronous case is limited to less than 9 processors while it is almost perfect for the asynchronous version. These results clearly illustrate that asynchronous methods promise to be useful in the presence of load imbalance.

L	dom. dec.	it_a	c_a	t_a	W_a	it_s	c_s	t_s	W_s	Q
2	1027-5	287-16204	158.8	79.43	2.00	287	85.39	79.35	1.08	1.00
3	515-516-5	320-9145	136.5	45.64	2.99	297	89.61	42.12	2.13	1.08
4	344-345-6	334-6136	127.6	31.99	3.99	304	92.10	29.52	3.12	1.08
5	259-260-5	325-4884	122.3	24.70	4.95	300	93.90	22.59	4.16	1.09
6	207-208-9	326-3483	118.5	19.82	5.98	301	94.94	18.64	5.09	1.06
7	173-174-9	330-2861	115.6	16.56	6.98	304	95.61	15.98	5.98	1.04
8	149-150-7	327-2690	117.3	14.71	7.97	305	99.86	14.74	6.78	1.00
9	131-132-5	328-2630	120.6	13.43	8.98	302	103.7	13.69	7.57	0.98
10	116-117-12	332-1719	113.2	11.37	9.95	303	97.87	11.69	8.37	0.97
11	105-106-9	328-1742	114.2	10.42	10.96	302	100.7	11.27	8.94	0.92
12	96-97-6	329-1834	117.2	9.80	11.96	303	104.3	10.85	9.61	0.90
13	88-89-9	338-1584	123.2	9.51	12.96	307	108.7	10.70	10.16	0.89
14	81-82-15	340-1219	126.5	9.06	13.95	310	112.6	10.19	11.05	0.89
15	76-77-7	328-1546	132.1	8.91	14.82	309	115.6	10.02	11.54	0.89
16	71-72-9	331-1389	132.8	8.47	15.68	309	119.1	10.61	11.22	0.80

TABLE 4.2
Strong load imbalance with small last domain, CRAY J90, BJ, $\alpha = 0.01$, $q = 1026$, $p = 100$, $itin = 20$, $ov = 1$

In the experiments reported in Table 4.2, we keep $q = 1026$ fixed, while we vary the number of subdomains (processors) from $L = 2$ to $L = 16$. As indicated, the domain decomposition consists of $L - 1$ almost equally sized domains while the last domain is rather small. As the domain decompositions are rather arbitrary, the table should not be interpreted in the sense of a speedup with respect to the number of processors. The comparison of the columns for the accumulated CPU times c_a and c_s yields a measure of the amount of work load in both cases. The columns W_a, W_s illustrate again the characteristic effect of asynchronous methods: all processors are busy almost all the time while in the synchronous case, far less processors can be used in the mean.

It can be appreciated in Table 4.2 that the asynchronous version becomes more advantageous with an increasing number of processors and also a smaller quotient between the size of the smallest and the largest domain, i.e., a *better* load balance, which still remains, however, a strong imbalance. The following explanation has been confirmed by a large number of other tests. The tests for small numbers of processors are characterized by an extreme load imbalance (1:200 for $L = 2$ down to 1:20 for $L = 7$). Then, the last, small domain does not receive frequently enough updates from its much larger neighbor. This effect is enforced by the number of 20 inner iterations which seems to be too large in this context. A large number of inner iterations can inhibit a frequent exchange of new data which is necessary for asynchronous processes to be efficient.

L	dom. dec.	it_a	c_a	t_a	W_a	it_s	c_s	t_s	W_s	Q
1	65	11-11	1.93	1.92	1.00	11	2.56	1.72	1.49	1.12
2	34-35	24-24	5.15	2.57	2.00	17	3.94	1.87	2.11	1.37
3	24-25-24	22-23	5.83	1.95	2.99	16	4.39	1.42	3.10	1.37
4	18-19-21	22-23	6.82	1.72	3.95	16	5.17	1.30	3.97	1.32
5	15-16-18	22-25	8.23	1.67	4.94	16	5.90	1.22	4.83	1.37
6	13-14-16	22-23	8.92	1.51	5.92	16	6.61	1.18	5.59	1.28
7	12-13-12	23-24	10.28	1.51	6.80	16	7.25	1.08	6.74	1.40
8	10-11-17	21-27	12.30	1.56	7.89	17	8.79	1.34	6.56	1.16
9	10-11-10	23-24	12.39	1.42	8.74	17	9.28	1.12	8.26	1.27
10	9-10-12	23-26	14.06	1.44	9.77	17	10.16	1.15	8.84	1.25
11	8-9-16	21-28	16.31	1.51	10.91	17	11.11	1.31	8.48	1.15
12	8-9-11	23-25	16.41	1.39	11.79	17	11.74	1.15	10.17	1.21
13	7-8-18	21-30	20.16	1.58	12.78	18	13.65	1.57	8.69	1.00
14	7-8-14	21-27	19.49	1.42	13.77	18	14.30	1.39	10.31	1.02
15	7-8-10	25-28	21.72	1.47	14.74	18	15.06	1.35	11.16	1.09
16	6-7-21	20-31	25.20	1.64	15.27	21	19.11	2.43	7.85	0.67

TABLE 4.3

Small number of grid lines, more diagonally dominant problem, CRAY J90, BJ, $\alpha = 1.0$, $q = 63$, $p = 2000$, $itin = 10$, $ov = 1$

The set of experiments reported in Table 4.3 might appear as a counterexample to the characteristic effects discussed for the Tables 4.1 and 4.2. Here, in almost all cases the synchronous version is clearly better, except for $L = 16$. Again, the work load is almost perfect for the asynchronous case and unsatisfactory in the synchronous case, depending on the degree of load imbalance. The domain decomposition ranges from a satisfactory load balance (12-13-12, 34-35) to an evident imbalance (8-9-16, 6-7-21) with a larger last domain. Obviously, the advantage of the synchronous version is reduced by an increasing imbalance. The explanation for this behavior is given again by the number of inner iterations $itin = 10$ which is by far too large with respect to the rather small number of outer iterations in this example: there is not enough communication for the asynchronous case to be efficient.

For the experiments reported in Table 4.4, we modify two parameters with respect to those in Table 4.3. We reduce the diagonal dominance from $\alpha = 1.0$ to $\alpha = 0.1$, observing no significant effect (we omit the corresponding results). We further reduce $itin$ to 4. From the discussion of the previous tables we expect this modification to significantly increase the efficiency of the asynchronous method and this is confirmed by Table 4.4. Table 4.5 contains results for the same example as in Table 4.4, computed on a CRAY T3D. Now, absolute times are much larger since the T3D processors are slower, but the qualitative behavior is unchanged. In other experiments with $itin = 4$, varying values of p and q and various domain decompositions we have witnessed similar results than those in Tables 4.4 and 4.5. For example, the asynchronous method was about 20% faster than the synchronous method, using $L = 16$ processors. For brevity, we do not provide the corresponding timings.

In the experiments reported in Table 4.6, we try to improve the load balance by chosing the number of inner iterations depending on the subproblem size. Here, the even numbered domains are roughly three times larger than the odd numbered domains. So, in order to improve the load balance, the odd numbered processors perform three times as many inner iterations (12 instead of 4 and 3 instead of 1) as the even numbered processors; cf. [5]. This

L	dom. dec.	it_a	c_a	t_a	W_a	it_s	c_s	t_s	W_s	Q
1	65	171-171	12.58	12.58	1.00	171	17.25	11.48	1.50	1.10
2	34-35	183-185	16.81	8.40	2.00	176	18.37	8.19	2.24	1.04
3	24-25-24	185-191	20.62	6.88	3.00	178	21.67	6.87	3.16	1.00
4	18-19-21	185-198	24.26	6.11	3.97	180	25.70	6.35	4.05	0.96
5	15-16-18	184-200	28.00	5.61	4.99	181	29.26	6.13	4.77	0.92
6	13-14-16	184-203	31.96	5.35	5.98	182	32.61	6.13	5.32	0.87
7	12-13-12	199-205	36.34	5.21	6.98	187	36.31	5.57	6.52	0.94
8	10-11-17	177-224	42.00	5.28	7.95	189	42.10	6.74	6.25	0.78
9	10-11-10	208-218	46.12	5.14	8.97	189	43.87	5.48	8.01	0.94
10	9-10-12	198-222	50.42	5.07	9.95	189	48.37	6.29	7.69	0.81
11	8-9-16	177-234	56.55	5.17	10.94	196	54.79	7.24	7.56	0.71
12	8-9-11	202-231	60.92	5.11	11.93	197	57.80	6.58	8.78	0.78
13	7-8-18	169-245	67.67	5.24	12.91	193	62.08	8.02	7.74	0.65
14	7-8-14	185-237	70.66	5.08	13.92	194	64.69	8.26	7.83	0.62
15	7-8-10	210-235	74.92	5.03	14.88	194	67.83	7.28	9.32	0.69
16	6-7-21	170-276	91.53	5.81	15.77	206	77.40	11.01	7.00	0.83

TABLE 4.4
Small number of grid lines, less diagonally dominant problem, less inner iterations, CRAY J90, BJ, $\alpha = 0.1$, $q = 63$, $p = 2000$, $itin = 4$, $ov = 1$

L	dom. dec.	it_a	t_a	it_s	t_s	Q
2	34-35	165-170	73.90	160	71.53	1.03
4	18-19-21	159-189	41.38	163	42.23	0.98
8	10-11-17	137-262	28.26	170	34.92	0.81
16	6-7-21	136-656	35.37	186	47.30	0.75

TABLE 4.5
Small number of grid lines, less diagonally dominant problem, less inner iterations, CRAY T3D, BJ, $\alpha = 0.1$, $q = 63$, $p = 2000$, $itin = 4$, $ov = 1$

does not seem to have any measurable effect as compared to the case where $itin$ is constant across the processors ($itin = 1$ and $itin = 4$), neither for the synchronous nor for the asynchronous algorithm. The processors on smaller domains perform almost the same total number of iterations and, therefore, do not produce faster approximations.

$itin$	it_a	t_a	it_s	t_s
1	602-1792	15.00	605	14.93
1-3	602- 849	14.99	605	14.89
2	309- 919	12.01	312	12.06
4	159- 472	10.61	162	10.77
4-12	159- 177	10.59	159	10.57

TABLE 4.6
Load balance by a different number of inner iterations, CRAY T3D, BJ, dom. dec. $67-(196-66)-195$, $q = 1024$, $L = 8$, $p = 50$, $\alpha = 0.1$, $itin = 4$, $ov = 1$

In Table 4.7 we illustrate how the algorithms scale with the number of processors. By scaling we mean that when the problem size is increased in the same proportion as the number of processors, then the work per processor (and iteration) remains relatively constant. The last processor receives only 3 grid lines. The results almost perfectly scale (except for $L = 2$) in both the synchronous and the asynchronous versions. In addition, the asynchronous version is always faster. Similar results were obtained by chosing a large last domain.

L	q	it_a	t_a	it_s	t_s	Q
2	38	33-1251	10.22	34	11.33	0.90
4	124	26-1283	10.87	34	15.35	0.71
8	296	26-1287	10.88	34	15.35	0.71
16	640	26-1328	11.30	34	15.35	0.74
32	1328	26-1329	11.30	34	15.35	0.74
64	2704	26-1332	11.31	34	15.36	0.74
128	5656	26-1332	11.31	34	15.36	0.74
256	10960	26-1332	11.31	34	15.36	0.74

TABLE 4.7
Scaling, CRAY T3D, BJ, $\alpha = 1.0$*, dom. dec. 41-(52-42)-3 (39-3 for* $L = 2$*),* $p = 1000$*,* $itin = 4$*,* $ov = 1$

While we have reported results on asynchronous weighted additive Schwarz, our theory is also valid for asynchronous multisplitting methods. We have performed extensive experiments with multisplitting methods using a weight of $1/2$ in the variables that overlap, and have found that the results are very similar for Schwarz and multisplitting in the synchronous case, while asynchronous multisplitting is slower by a factor of sometimes more than two than synchronous multisplitting. We also point out that the results for the multisplitting method again scale quite well. Multisplitting with weighting seems to become very sensible to asynchronous effects; cf. [5] where the weights are all 0 or 1. The weighting with possibly very old iterates from larger neighboring subdomains seems to slow down considerably the iteration process.

inner method	$itin$	it_a	t_a	it_s	t_s	Q
Block Jacobi	24	59- 63	80.11	44	57.58	1.39
Point Jacobi	24	74- 79	75.49	65	66.83	1.13
Point Gauss-Seidel	24	57- 61	55.69	44	41.90	1.33
Block Jacobi	4	172-184	46.46	165	41.71	1.11
Point Jacobi	4	314-335	67.24	312	64.24	1.05
Point Gauss-Seidel	4	170-182	35.19	167	32.20	1.09

TABLE 4.8
Different inner iteration methods, CRAY T3D, $q = 128$*,* $p = 2000$*,* $L = 8$*,* $\alpha = 0.1$*,* $ov = 1$*, dom. dec. 19-20-19*

We close our discussion by a short remark on the use of other inner iteration methods. Table 4.8 contains a few results for block Jacobi, point Jacobi as well as point Gauss-Seidel as the inner iteration. Here we have chosen an example with an almost ideal load balance, an ideal situation for a synchronous iteration. The timings obtained are consistent with what is expected in these cases. Block Jacobi needs fewer outer iterations than point Jacobi, but a larger run time. Point Gauss-Seidel converges in a similar number of outer iterations as Block Jacobi. In this case, since we have tridiagonal matrices, once the factorization is performed for Block Jacobi, the arithmetic work for all three methods is comparable. The run time of point Gauss-Seidel, however, is shortest, since this method is amenable to more efficient pipelining and caching than Block Jacobi. Even in this case of almost perfect load balance, the asynchronous versions are only slightly slower, in particular, when the number of inner iterations is reduced appropriately (here from $itin = 24$ to $itin = 4$).

5. Conclusion. The numerical results presented illustrate that asynchronous versions of weighted additive Schwarz methods can be advantageous to compensate the effect of load imbalance in parallel computations. This is important for the frequent case of domain decompositions with subdomains of different size. Unfortunately, it does not seem to be possible to indicate a criterion for an appropriate degree of load imbalance or a particular distribution of subdomains for the asynchronous versions to be always faster. We can note, however, that in all experiments of asynchronous weighted additive Schwarz methods we carried out, even in the case of a good load balance, the quotient of wall clock times $Q = t_a/t_s$ never left the interval between roughly 0.5 and 1.5 (larger for asynchronous multisplittings). The amount of communication seem to have a more important (and more plausible) influence on the computation speed. Reducing the number of inner iterations or changing from a block method with a larger amount of work per step to a different inner iteration method, can lead to a noticeable speedup by the use of an asynchronous method. Finally, we note that, at least in our experiments, asynchronous multisplitting methods with weight 1/2 have not been competitive.

Appendix: computing environment. We briefly describe the two parallel computers used in our experiments, namely the CRAY T3D and the CRAY J90, with relevant information related to our implementation.

On the T3D processors are allocated exclusively to a job, i.e., a job is executed in dedicated mode (possible perturbations mostly result from actions involving the front-end computer, i.e., from the operating system, the scheduling or interrupts caused, e.g., by I/O).

Communication is implemented with the CRAY specific (and therefore fast) `shmem` message passing library. This library allows for direct and unsynchronized memory access between different processors with specific `put` and `get` routines. Although, principally, `put` is faster, we only used the `get` routine as the latter enables us to implement very easily asynchronous communication: every processor executes a `get` to the memory of another processor whenever it needs data.

The *asynchronous* version is fully asynchronous, in particular with respect to convergence control which does not need global synchronization: after every iteration, each processor tests for its local convergence criterion and places the results in a logical variable. If the criterion is satisfied, it starts to get the corresponding variable from other processors. As soon as it encounters a variable whose value is *false*, it stops the convergence control and proceeds to the next iteration.

In the *synchronous* version, a (synchronizing) global convergence control is carried out after every iteration, using a global reduction function from the `shmem` library. In addition, a *barrier* is necessary in every iteration in order to synchronize the correct exchange of the updated artificial boundary values.

The CRAY J90 is a shared memory computer with 16 vector processors. Parallelization is implemented with CRAY autotasking. All tests on the J90 have been carried out in dedicated mode. In the synchronous version, the iteration loop contains three synchronization points, in the asynchronous version none.

We conclude with a practical remark. Tests on the J90 in nondedicated – the usual multiuser – mode suggest that the program usually gets more CPUs in the asynchronous cases than in the synchronous ones. This is due to the fact that for each parallel region in the program, the J90 dynamically creates a new process for each processor which competes with any other process on the machine. When a parallel region like a loop is terminated, the CPU is released. The next time, i.e., in the next step, the competition begins again. In other words, at synchronization points, processors are deallocated immediately in most cases. This differs from the T3D (or also machines like the PowerChallenge/Origin from SGI) where CPUs are allocated to a job and not released before the end of the job. The drawback of this effect on the J90 is that if a job does not get the number of CPUs it needs for a specific parallel region, the whole job ‘starves’: on at least one subdomain the iteration is stopped and on the other subdomains ‘wrong’ problems are solved over and over until a possible time limit.

Acknowledgement. The authors wish to thank Olof B. Widlund for his comments on an earlier version of the manuscript, and an anonymous referee for his suggestions.

REFERENCES

[1] G. M. BAUDET, *Asynchronous iterative methods for multiprocessors*, J. ACM, 25 (1978), pp. 226–244.

[2] M. BENZI, D. B. SZYLD, *Existence and Uniqueness of Splittings for Stationary Iterative Methods with Applications to Alternating Methods*, Numer. Math., 76 (1997), pp. 309–321.

[3] A. BERMAN, R. PLEMMONS, *Nonnegative matrices in the mathematical sciences*, Academic Press, New York (1979), reprinted by SIAM, Philadelphia (1994).

[4] D. P. BERTSEKAS, J. N. TSITSIKLIS, *Parallel and Distributed Computation*, Prentice Hall, Englewood Cliffs, New Jersey (1989).

[5] R. BRU, V. MIGALLÓN, J. PENADÉS, D. B. SZYLD, *Parallel, Synchronous and Asynchronous Two-stage Multisplitting Methods*, Electr. Trans. Numer. Anal., 3 (1995), pp. 24–38.

[6] M. DRYJA, O. B. WIDLUND, *Towards a Unified Theory of Domain Decomposition Algorithms for Elliptic Problems*, in: T. C. Chan, R. Glowinski, J. Périaux, O. B. Widlund (eds.), Domain Decomposition Methods - Proceedings of the Third International Symposium on Domain Decomposition Methods for Partial Differential Equations (Houston, March 1989), SIAM, Philadelphia, 1990, pp. 3–21.

[7] M. N. EL TARAZI, *Some convergence results for asynchronous algorithms*, Numer. Math., 39 (1982) pp. 325–340.

[8] D. J. EVANS, SHAO JIANPING, KANG LISHAN, *The convergence factor of the parallel Schwarz overrelaxation method for linear systems*, Parallel Comput., 6 (1988), pp. 313–324.

[9] A. FROMMER, *Generalized nonlinear diagonal dominance and applications to asynchronous iterative methods*, J. Comput. Appl. Math., 38 (1991), pp. 105–124.

[10] A. FROMMER, *On asynchronous iterations in partially ordered spaces*, Numer. Funct Anal. Optim., 12 (1991) pp. 315–325.

[11] A. FROMMER, H. SCHWANDT, *A Unified Representation and Theory of Algebraic Additive Schwarz and Multisplitting Methods*, SIAM J. Matrix Anal. Appl., 18 (1997).

[12] A. FROMMER, D. B. SZYLD, *H-splittings and two-stage iterative methods*, Numer. Math., 63 (1992), pp. 65–82.

[13] A. FROMMER, D. B. SZYLD, *Asynchronous two-stage iterative methods*, Numer. Math., 69 (1994), pp. 141–153.

[14] P. L. LIONS, *On the Schwarz alternating method I*, in: R. Glowinski, G. H. Golub, G. Meurant, J. Périaux, eds., Proceedings of the First International Symposium on Domain Decomposition Methods for Partial Differential Equations (Paris, January 7-9, 1987), SIAM, Philadelphia, 1988, pp. 1–42.

[15] P. L. LIONS, *On the Schwarz alternating method II: Stochastic Interpretation and Order Properties*, in: T. C. Chan, R. Glowinski, J. Périaux, O. B. Widlund, eds., Domain Decomposition Methods - Proceedings of the Second International Symposium on Domain Decomposition Methods for Partial Differential Equations (Los Angeles, January 14-16, 1988), SIAM, Philadelphia, 1988, pp. 47–71.

[16] J. MORÉ, *Nonlinear generalizations of matrix diagonal dominance with application to Gauss-Seidel iterations*, SIAM J. Numer. Anal., 9 (1972), pp. 357–378.

[17] D. P. O'LEARY, R. E. WHITE, *Multi-splittings of matrices and parallel solution of linear systems*, SIAM J. Algebra Discrete Methods, 6 (1985), pp. 630–640.

[18] G. RODRIGUE, *Inner/outer iterative methods and numerical Schwarz algorithms*, Parallel Comput., 2 (1985), pp. 205–218.

[19] G. RODRIGUE, KANG LISHAN, LIU YU-HUI, *Convergence and comparison analysis of some numerical Schwarz methods*, Numer. Math., 56 (1989), pp. 123–138.

[20] G. RODRIGUE, J. SIMON, *A generalized numerical Schwarz algorithm*, in: Computer Methods in Applied Sciences and Engineering VI (R. Glowinski, J.-L. Lions, eds.), Elsevier, (1984), pp. 272–281.

[21] H. SCHWARZ, *Gesammelte mathematische Abhandlungen*, Vol.2 (1890), Springer, Berlin, pp. 133–134.

[22] B. SMITH, P. BJØRSTAD, W. GROPP, *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*, Cambridge University Press (1996).

[23] R. S. VARGA, *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs, New Jersey (1962).

[24] R. S. VARGA, *On recurring theorems on diagonal dominance*, Linear Algebra Appl., 13 (1976), pp. 1–9.