Available online at www.sciencedirect.com
ScienceDirect
Mathematics and Computers in Simulation 145 (2018) 34–49
MATHEMATICS AND COMPUTERS IN SIMULATION
www.elsevier.com/locate/matcom

Original articles

Asynchronous iterative sub-structuring methods

Frédéric Magoulès*, Cédric Venet

CentraleSupélec, Université Paris-Saclay, France

Received 10 October 2014; received in revised form 17 January 2016; accepted 4 May 2016
Available online 10 August 2016

Abstract

In this paper, a new asynchronous iterative sub-structuring method is presented. This method is based on a classical sub-structuring approach, but during the iterations of the algorithm, at the end of each iteration the synchronisation between the processors is here removed leading to totally asynchronous iterations. The mathematical proof of the convergence of this new asynchronous iterative sub-structuring method is first introduced, followed by an example of the parallel implementation. Numerical results performed on a three dimensional test case illustrate the robustness, performance and efficiency of the asynchronous version over its synchronous counterpart.

Keywords: Sub-structuring methods; Domain decomposition methods; Iterative methods; Asynchronous iterations; Parallel computing

1. Introduction

The traditional scheme for parallel iterative algorithms is called *synchronous iterations* [38,20]. This describes a method where a new iteration is only started when all the data from the previous iteration has been received. Initially this scheme has been used with *synchronous communication*. This means that data can only be exchanged when the source and destination processes have finished their computation. In most case this leads to a lot of wasted time. An improvement, allowing to overlap the communication time with the computation time, is to use *asynchronous communication*. The only difference is that data can be received by a process while it is still computing the previous iteration. Thus the data computed by a process is sent as soon as the iteration computation is finished. This allows to reduce the communication time a little bit. Further down this path there is *flexible communication*. In this approach the data is sent as soon as possible, i.e., it is sent in parts before the end of the iteration. This is only useful if the data is computed progressively. However, when used it reduces the communication time a lot since the quantity of data to exchange at the end of an iteration is greatly reduced. All these synchronous iterations use the same mathematical model and have the same convergence properties as their sequential counterparts. They have been widely studied and are often simply called parallel iterative algorithms, synchronous being omitted. They are very efficient when used with well balanced workloads and short communication times. However, these conditions are hard and expensive to achieve especially as the number of cores used increases.

* Corresponding author.
E-mail address: frederic.magoules@hotmail.com (F. Magoulès).

http://dx.doi.org/10.1016/j.matcom.2016.05.009
0378-4754/© 2016 International Association for Mathematics and Computers in Simulation (IMACS). Published by Elsevier B.V. All rights reserved.

Another kind of iterative algorithm can help to solve these scalability problems. Called first *chaotic relaxations*, it is usually designed by *asynchronous iterations*. The initial work on this scheme was done by Chazan and Miranker [11] and Donnelly [12] and has generated a lot of further research. Among the earliest studies, one can cite the work of Miellou [35,34], Baudet [4], Bertsekas and Tsitsiklis [5,6], El Tarazi [13,14]. There are three commonly used schemes for asynchronous iterations. The first one is called *totally asynchronous iterations*. It sets very few conditions on the iterations and communications except that they must never stop indefinitely. In their book [6], Bertsekas and Tsitsiklis introduce a mathematical model of these as well as several convergence theorems widely applicable. In particular a very general convergence theorem based on a set of imbricated boxes has been used as the basis of numerous convergence results in various domains. This is the scheme which is considered in this paper. The second scheme is called *partially asynchronous iterations*. It is based on the assumption that the communication time is bounded and that each process does at least one iteration every given period. Some useful algorithms which do not converge under totally asynchronous iterations can be proven to converge with these assumptions, see for example Refs. [41,40]. The third scheme is called *flexible asynchronous iterations*. This class of algorithm is more recent [18] and includes in particular the asynchronous two-stage iterative methods [17]. As in the synchronous iterations case, flexible communication means that the data is sent as soon as possible, but due to the strength of asynchronous iteration, it is more general. Data can also be integrated into an iteration as soon as received without waiting for the start of a new iteration. Moreover partial data can also be used, i.e., even if only part of the data from a process (or an approximation of it) has been received, this part can be used immediately without waiting for the rest.

Asynchronous algorithms have been used to solve a variety of numerical problems including the non-exhaustive list: nonlinear fixed points and optimisation problems for partial differential equations and ordinary differential equations [1], flow electrophoresis problems [10], obstacle problems [39], option pricing problems [19,9], and references therein. In this paper, only linear fixed point problems are considered with totally asynchronous iterations. The goal is to solve a linear system of partial differential equations arising from a finite element method. The first asynchronous algorithms used to solve this type of problems are those using contracting operators: Jacobi, Richardson, Successive Over Relaxation method, fixed step gradient descent, etc. More generally, the convergence of fixed point iterations have been studied for arbitrary contracting operators [25,6]. Other asynchronous algorithms for this fixed point problem include the additive Schwarz method [16,21] and more generally the multi-splitting method [7,8] as first introduced by O'Leary and White [36].

The plan of this paper is as follows. Synchronous iterative methods with a particular focus on the Jacobi algorithm, are first introduced in Section 2, followed in Section 3 by their asynchronous counterpart. In Section 4 the classical sub-structuring method is presented, together with a simple parallel implementation. The new asynchronous version of this sub-structuring method is then introduced in Section 5. After the mathematical demonstration of the convergence of this new method, a possible parallel implementation is presented. Numerical results performed on an academic problem, illustrate in Section 6 the robustness, performance and efficiency of this new asynchronous sub-structuring method. Finally, Section 7 concludes this paper.

2. Synchronous iterative methods

2.1. Sequential iterative methods

Iterative methods are based on successive applications of a function on an initial guess of the solution until the result is close enough of the solution [38,20]. The solution $x \in \mathbb{R}^n$ of a linear system $Ax = b$ is here analysed, with $A \in \mathbb{R}^{n\times n}$, a nonsingular $n \times n$ matrix with coefficients in $\mathbb{R}$, and with $b \in \mathbb{R}^n$, the right hand side vector. In the following, the quantity $x_0 \in \mathbb{R}^n$ denotes an arbitrary given value, and x^* denotes the exact solution. To solve this linear system, the matrix A is split into two matrices M and N such as $A = M - N$ where M is a nonsingular matrix. With this splitting, the linear system $Ax = b$ can be rewritten in the form $Mx = Nx + b$ or equivalently $x = M^{-1}Nx + M^{-1}b$ (or similarly $x = Tx + c$, where $T = M^{-1}N$ and $c = M^{-1}b$). The iterative algorithm considered here could be defined by the iterations:

$$Mx(k+1) = Nx(k) + b$$

or equivalently:

$$x(k+1) = Tx(k) + c$$

where $k \in \mathbb{N}$ denotes the iteration number. The sequential implementation of this algorithm is quite simple and is illustrated in Algorithm 1.

Algorithm 1: Sequential iterative algorithm

```
$x = x_0$;
$NbIter = 0$;
while $NbIter < MaxIter$ do
    $NbIter$++;
    $tmp = Tx + c$;
    $x = tmp$;
    if $Convergence\ detected$ then
        return;
    end
end
```

With the previous form, the scalar Jacobi algorithm, for instance, corresponds to the choice $M = D$, $N = -(L + U)$, where D denotes the diagonal part, U the strict upper part, and L the strict lower part of matrix A. The speed of convergence of the algorithm can be determined using the following lemma.

Lemma 1. *The bounded value of the error between the exact solution x^* and the approximate solution $x(k)$ at iteration k is given by:*

$$\forall \epsilon > 0, \quad \left\| x(k) - x^* \right\| \leq (\rho(T) + \epsilon)^k \left\| x_0 - x^* \right\| \tag{1}$$

where $\rho(T)$ denotes the spectral radius of the matrix T, and $\epsilon \in \mathbb{R}^+$.

Proof. The reader is referred to [2] for a detailed proof of Eq. (1). □

Lemma 1 leads to the following lemma.

Lemma 2. *The iterative algorithm converges for any x_0, if and only if, $\rho(T) < 1$.*

Proof. If $\rho(T) < 1$, taking $\epsilon > 0$ such as $\rho(T) + \epsilon < 1$ and substituting in Eq. (1) implies:

$$\lim_{k \to \infty} \left\| x(k) - x^* \right\| = 0$$

or in other words, $\lim_{k \to \infty} x(k) = x^*$ which is the definition of the convergence of the algorithm.

Reciprocally, if $\rho(T) \geq 1$, a sequence $\{y(k)\}_{k \in \mathbb{N}}$ can be defined by:

$$\forall k \in \mathbb{N}, \quad y(k) = x(k) - x^*.$$

Replacing x by y in the iterative algorithm leads to:

$$\begin{aligned} y(k+1) + x^* &= T(y(k) + x^*) + c \\ y(k+1) &= Ty(k) + \left(Tx^* + c - x^*\right) \\ y(k+1) &= Ty(k) \end{aligned}$$

because by definition of the matrix T and the vector c, x^* is a fixed point of

$$\mathcal{T} : x \mapsto Tx + c.$$

This leads to the relation

$$y(k) = T^k y(0).$$

Since $\rho(T)$ is an eigenvalue of T (from Perron–Frobenius theorem), setting v an eigenvector associated with the eigenvalue $\rho(T)$, and choosing $y(0) = v$ gives

$$\begin{aligned} y(k) &= T^k y(0) \\ &= T^k v \\ &= \rho(T)^k v \end{aligned}$$

which does not tend to zero. In other words the algorithm does not converge when $\rho(T) \geq 1$ which concludes the proof. □

2.2. Parallel iterative methods

Synchronous parallel iterative methods do not lead to particular difficulties. For instance, the matrix T and the vectors x, c can be sliced in N_s horizontal bands, namely T_s, x_s, c_s defined by:

$$T = \begin{pmatrix} T_1 \\ \vdots \\ T_{N_s} \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ \vdots \\ x_{N_s} \end{pmatrix}, \quad x_0 = \begin{pmatrix} x_{1,0} \\ \vdots \\ x_{N_s,0} \end{pmatrix} \quad \text{and} \quad c = \begin{pmatrix} c_1 \\ \vdots \\ c_{N_s} \end{pmatrix}.$$

Each process works on one band of the matrix T as illustrated for instance in Algorithm 2 for the process s, $\forall s \in 1, \ldots, N_s$. In Algorithm 2, x is defined as a vector of dimension n and x_s is defined as a vector of dimen-

Algorithm 2: Synchronous parallel iterative algorithm

```
Set $T_s$, $c_s$;
$x = x_0$;
$NbIter = 0$;
while $NbIter < MaxIter$ do
    $NbIter$++;
    $x_s = T_s x + c_s$;
    // gather $x_s$ $\forall i = s \ldots N_s$ into $x$
    MPI_AllGather($x$,$x_s$);
    if Global Convergence detected then
        return;
    end
end
```

sion n_s with n_s denoting the number of rows of the band T_s. An important theorem is now recalled.

Theorem 1. *The convergence of the parallel algorithm is equivalent to the convergence of its sequential counterpart.*

Proof. The reader is referred to [2] for a detailed proof of this theorem. □

3. Asynchronous iterative methods

Several authors have investigated the behaviour of asynchronous iterative methods as first introduced by [6]. In order to make the link with the new asynchronous iterative sub-structuring method proposed in this paper and to prove the convergence of this new method, some theorems are now recalled or demonstrated for classical asynchronous iterative methods.

The solution $x^* \in \mathbb{R}^n$ of a linear system $Ax = b$, with $A \in \mathbb{R}^{n\times n}$, a nonsingular $n \times n$ matrix with coefficients in $\mathbb{R}$, and with $b \in \mathbb{R}^n$, the right hand side vector, can be written as the solution of the fixed point of the function $f : X \mapsto X$. Let N_s denote the number of processes. For each $s \in \{1, \ldots, N_s\}$, X can be decomposed as $X = X_1 \times X_2 \times \cdots \times X_{N_s}$ with $\times$ the Cartesian product. For a given s and $x \in X$, $f_s : X \mapsto X_s$ is defined as the sth component of $f(x) = (f_1^T(x), \ldots, f_{N_s}^T(x))$. Applying these notations to the linear system $Ax = b$, gives

$X = \mathbb{R}^n$, $X_s = \mathbb{R}^{n_s}$, where $\sum_{s=1}^{N_s} n_s = n$; n_s being the number of rows of the matrix T allocated to the process s. The fixed point mapping on X_s can be written as: $f_s(x) = T_s x + c_s$. An asynchronous execution of these iterations is defined (for each process s) as:

$$x_s(k+1) = \begin{cases} f_s\left(x_1(\tau_1^s(k)), \dots, x_j(\tau_j^s(k)), \dots, x_{N_s}(\tau_{N_s}^s(k))\right) & \text{if } s \in s(k) \\ x_s(k) & \text{if } s \notin s(k) \end{cases}$$

where $\{s(k)\}_{k\in\mathbb{N}}$ is a sequence of nonempty subsets of $\{1, \dots, N_s\}$. It defines which processes update their component at the iteration k. For $i, j \in \{1, \dots, p\}$, $\left\{\tau_j^i(k)\right\}_{k\in\mathbb{N}}$ is a sequence of integers such that $\forall k$, $\tau_j^i(k) \le k$. $\tau_j^i(k)$ represents the iteration number of the data coming from process j and available on process i at iteration k.

As detailed in Ref. [6], two natural conditions are supposed to be satisfied:

$$\forall i \in \{1, \dots, N_s\}, \quad \operatorname{card}\{k \in \mathbb{N} | i \in s(k)\} = +\infty \tag{2}$$

and

$$\forall i, \ j \in \{1, \dots, N_s\}, \quad \lim_{k\to+\infty} \tau_j^i(k) = +\infty. \tag{3}$$

The condition (2) means that no process will definitively stop updating its components. The condition (3) corresponds to the fact that new data will always be provided to the process. In other words no process will have one of its data stuck at one iteration. In addition, the following assumption is supposed satisfied.

Assumption 1. There is a sequence of nonempty sets $\{X(k)\}$ with

$$\cdots \subset X(k+1) \subset X(k) \subset \cdots \subset X$$

satisfying the following two conditions:

(a) *Synchronous Convergence Condition*: We have

$$f(x) \in X(k+1), \quad \forall k \text{ and } x \in X(k).$$

Furthermore, if $\left\{y^k\right\}$ is a sequence such that $y^k \in X(k)$ for every k, then every limit point of $\left\{y^k\right\}$ is a fixed point of f.

(b) *Box Condition*: For every k, there exists $X_s(k) \subset X_s$ such that

$$X(k) = X_1(k) \times X_2(k) \times \cdots \times X_{N_s}(k).$$

Theorem 2 (*Asynchronous Convergence Theorem*)**.** *If the Synchronous Convergence condition and the Box Condition of Assumption 1 hold, and the initial solution estimate $x(0) = \left(x_1(0), \dots, x_{N_s}(0)\right)$ belongs to set $X(0)$, then every limit point of $\{x(k)\}$ is a fixed point of f.*

Proof. The complete proof of Theorem 2 could be obtained in Ref. [6] page 101. □

Definition 1. A weighted maximum norm is defined by:

$$\|x\|_\infty^w = \max_{1\le i\le n} \frac{|x_i|}{w_i}$$

where $w \in \mathbb{R}^n$ is a positive vector ($w > 0$).

Lemma 3. *If $f : \mathbb{R}^n \mapsto \mathbb{R}^n$ is a contraction mapping with respect to a weighted maximum norm $\|.\|_\infty^w$ then the asynchronous convergence is guaranteed.*

Proof. In order not to overload the notations, the case $N_s = n$ is considered, i.e., that the matrix is decomposed line by line. If this converges, any other decomposition will also converge. So x_s corresponds here to the sth component of x. Take $X = \mathbb{R}^n$ and $X_s = \mathbb{R}$ for $s = 1, \dots, n$. Denote x^* the unique fixed point of f and $\alpha < 1$ its contraction modulus, i.e.,

$$\left\|f(x) - x^*\right\|_\infty^w = \left\|f(x) - f(x^*)\right\|_\infty^w \le \alpha \left\|x - x^*\right\|_\infty^w$$

and define the sets

$$X(k) = \left\{ x \in \mathbb{R}^n \mid \left\| x - x^* \right\|_\infty^w \leq \alpha^k \left\| x(0) - x^* \right\|_\infty^w \right\}.$$

By construction, this implies

$$\forall k,\ \forall x \in X(k), \quad f(x) \in X(k+1)$$

and $X(k) = \prod_{1 \leq s \leq n} X_s(k)$ with:

$$X_s(k) = \left\{ x_s \in \mathbb{R} \mid \left| x_s - x_s^* \right| \leq \alpha^k w_i \left\| x(0) - x^* \right\|_\infty^w \right\}.$$

Thus, the conditions (a) and (b) of Assumption 1 are satisfied which concludes the proof. □

Theorem 3 (*Perron–Frobenius Theorem*)**.** *Let M be a $n \times n$ nonnegative real matrix.*

- *If M is irreducible, then $\rho(M)$ is an eigenvalue of M and there exists some $w > 0$ such that $Mw = \rho(M)w$. Furthermore, such a w is unique within a scalar multiple. Finally, $\|M\|_\infty^w = \rho(M)$.*
- *$\rho(M)$ is an eigenvalue of M and there exists some $w \geq 0$, $w \neq 0$, such that $Mw = \rho(M)w$.*
- *For every $\epsilon > 0$, there exists some $w > 0$ such that $\rho(M) \leq \|M\|_\infty^w \leq \rho(M) + \epsilon$.*

Proof. The detailed proof of this theorem could be found in Ref. [6] page 148. □

The Algorithm 3 shows how an asynchronous algorithm can be implemented in parallel. The algorithm is written for the processor i.

Algorithm 3: Asynchronous parallel iterative algorithm

```
Setup $T_i, c_i$;
$x = x_0$;
$NbIter = 0$;
while $NbIter < MaxIter$ do
    $NbIter$++;
    $x_i = T_i x + c_i$;
    foreach Process $j$ which depend on $x_i$ do
        // Asynchronous send of $x_i$ to process $j$
        Send(j,$x_i$);
    end
    $x(I_i) = x_i$;
    foreach Message $m$ received do
        $x(I_{m.source}) = m.data$;
    end
    if Global Convergence detected then
        return;
    end
end
```

4. Synchronous iterative sub-structuring methods

Following a classical renumbering of the degrees of freedom, as introduced in [37], a general matrix A and right hand side vector b could be rewritten in a sub-structure form

$$A = \begin{pmatrix} A_{ii}^{(1)} & 0 & 0 & A_{ip}^{(1)} \\ 0 & \ddots & 0 & \vdots \\ 0 & 0 & A_{ii}^{(N_s)} & A_{ip}^{(N_s)} \\ A_{pi}^{(1)} & \cdots & A_{pi}^{(N_s)} & A_{pp} \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} b_i^{(1)} \\ \vdots \\ b_i^{(N_s)} \\ b_{pp} \end{pmatrix}$$

where i represents the degree of freedom inside the sub-structure, p the degree of freedom on the interface between the sub-structures. In a parallel context, the matrix A and the right hand side b are computed independently on each sub-structure s:

$$A^{(s)} = \begin{pmatrix} A_{ii}^{(s)} & A_{ip}^{(s)} \\ A_{pi}^{(s)} & A_{pp}^{(s)} \end{pmatrix}, \qquad b^{(s)} = \begin{pmatrix} b_i^{(s)} \\ b_p^{(s)} \end{pmatrix}$$

where $\sum_{s=1}^{N_s} A_{pp}^{(s)} = A_{pp}$, and $\sum_{s=1}^{N_s} b_p^{(s)} = b_p$. Each sub-structure is allocated to one process for instance.

Applying such a renumbering to the Jacobi algorithm leads to:

$$T = \begin{pmatrix} T_{ii}^{(1)} & & & & & T_{ip}^{(1)} \\ & \ddots & & & & \vdots \\ & & T_{ii}^{(s)} & & & T_{ip}^{(s)} \\ & & & \ddots & & \vdots \\ & & & & T_{ii}^{(N_s)} & T_{ip}^{(N_s)} \\ T_{pi}^{(1)} & \cdots & T_{pi}^{(s)} & \cdots & T_{pi}^{(N_s)} & T_{pp} \end{pmatrix} \quad \text{and} \quad c = \begin{pmatrix} c_i^{(1)} \\ \vdots \\ c_i^{(s)} \\ \vdots \\ c_i^{(N_s)} \\ c_p \end{pmatrix}.$$

The matrix T and right hand side c can be distributed on each process s as:

$$T^{(s)} = \begin{pmatrix} T_{ii}^{(s)} & T_{ip}^{(s)} \\ T_{pi}^{(s)} & T_{pp}^{(s)} \end{pmatrix} \quad \text{and} \quad c^{(s)} = \begin{pmatrix} c_i^{(s)} \\ c_p^{(s)} \end{pmatrix} \tag{4}$$

where:

$$\sum_{s=1}^{N_s} T_{pp}^{(s)} = T_{pp} \quad \text{and} \quad \sum_{s=1}^{N_s} c_p^{(s)} = c_p. \tag{5}$$

In the case of the Jacobi algorithm, where $M = D$ and $N = -(L + U)$, the matrices M and N can be expressed as:

$$M = \begin{pmatrix} M_{ii}^{(1)} & 0 & 0 & 0 \\ 0 & \ddots & 0 & \vdots \\ 0 & 0 & M_{ii}^{(N_s)} & 0 \\ 0 & \cdots & 0 & M_{pp} \end{pmatrix}, \qquad N = \begin{pmatrix} N_{ii}^{(1)} & 0 & 0 & N_{ip}^{(1)} \\ 0 & \ddots & 0 & \vdots \\ 0 & 0 & N_{ii}^{(N_s)} & N_{ip}^{(N_s)} \\ N_{pi}^{(1)} & \cdots & N_{pi}^{(N_s)} & N_{pp} \end{pmatrix}$$

which are distributed on each process s as:

$$M^{(s)} = \begin{pmatrix} M_{ii}^{(s)} & M_{ip}^{(s)} \\ M_{pi}^{(s)} & M_{pp}^{(s)} \end{pmatrix}, \qquad N^{(s)} = \begin{pmatrix} N_{ii}^{(s)} & N_{ip}^{(s)} \\ N_{pi}^{(s)} & N_{pp}^{(s)} \end{pmatrix} \tag{6}$$

with:

$$\sum_{s=1}^{N_s} M_{pp}^{(s)} = M_{pp} \quad \text{and} \quad \sum_{s=1}^{N_s} N_{pp}^{(s)} = N_{pp}. \tag{7}$$

Making the product of matrix $M^{(-1)}$ and N leads, after identification with matrix T, to the following relations:

$$\begin{aligned} T_{ii}^{(s)} &= M_{ii}^{(s)^{-1}} N_{ii}^{(s)} \\ T_{ip}^{(s)} &= M_{ii}^{(s)^{-1}} N_{ip}^{(s)} \\ T_{pi}^{(s)} &= M_{pp}{}^{-1} N_{pi}^{(s)} \\ T_{pp} &= M_{pp}{}^{-1} N_{pp}. \end{aligned}$$

It is important to notice that $T_{pi}^{(s)}$ and T_{pp} require the complete (assembled) value of M_{pp}, i.e., the diagonal part of the interface matrix $A_{pp} = \sum_{s=1}^{N_s} A_{pp}^{(s)}$ must be assembled between the sub-structures. Since

$$\sum_{s=1}^{N_s} N_{pp}^{(s)} = N_{pp} \quad \text{and} \quad \sum_{s=1}^{N_s} T_{pp}^{(s)} = T_{pp}$$

this leads to the relation

$$T_{pp}^{(s)} = {M_{pp}}^{-1} N_{pp}^{(s)}.$$

The following vectors can now be defined:

$$y^{(s)} = \begin{pmatrix} x_i^{(s)} \\ x_p^{(s)} \end{pmatrix} \quad \text{and} \quad y = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(s)} \\ \vdots \\ y^{(N_s)} \end{pmatrix} \tag{8}$$

with $y^{(s)}$ being the part of the solution computed by the process s. The mapping function g can be defined by:

$$g : y \mapsto g(y) = ({g^{(1)}}^T(y), \dots, {g^{(s)}}^T(y), \dots, {g^{(N_s)}}^T)$$
$$g^{(s)}(y) = T^{(s)} \begin{pmatrix} x_i^{(s)} \\ \sum_{q=1}^{N_s} x_p^{(q)} \end{pmatrix} + c^{(s)}$$

and the fixed point problem reduces to the iterative procedure on each process s:

$$y^{(s)}(k+1) = g^{(s)}(y(k)). \tag{9}$$

The following theorem can now be introduced.

Theorem 4. *The synchronous parallel iterations of* (9) *lead to the same operations as the sequential iterative Algorithm* 1. *As a consequence the convergence condition remains the same, i.e.,* $\rho(T) < 1$.

Proof. Substituting Eq. (8) in Eq. (9) and using Eq. (6) gives:

$$\begin{cases} x_i^{(s)}(k+1) &= T_{ii}^{(s)} x_i^{(s)}(k) + T_{ip}^{(s)} \sum_{q=1}^{N_s} x_p^{(q)}(k) + c_i^{(s)} \\ x_p^{(s)}(k+1) &= T_{pi}^{(s)} x_i^{(s)}(k) + T_{pp}^{(s)} \sum_{q=1}^{N_s} x_p^{(q)}(k) + c_p^{(s)}. \end{cases}$$

Defining $x_p(k)$ by $\sum_{s=1}^{N_s} x_p^{(s)}(k)$, substitution in the previous equations gives:

$$\begin{cases} x_i^{(s)}(k+1) &= T_{ii}^{(s)} x_i^{(s)}(k) + T_{ip}^{(s)} x_p(k) + c_i^{(s)} \\ x_p^{(s)}(k+1) &= T_{pi}^{(s)} x_i^{(s)}(k) + T_{pp}^{(s)} x_p(k) + c_p^{(s)} \end{cases}$$

and after summation over all the processes

$$x_p(k+1) = \sum_{s=1}^{N_s} \left(T_{pi}^{(s)} x_i^{(s)}(k) + T_{pp}^{(s)} x_p(k) + c_p^{(s)} \right).$$

Taking into account the zero blocks of the matrix T and the relation $\sum_{s=1}^{N_s} T_{pp}^{(s)} = T_{pp}$ lead to the equivalent form:

$$\begin{pmatrix} x_i^{(1)}(k+1) \\ \vdots \\ x_i^{(s)}(k+1) \\ \vdots \\ x_i^{(N_s)}(k+1) \\ x_p(k+1) \end{pmatrix} = T \begin{pmatrix} x_i^{(1)}(k) \\ \vdots \\ x_i^{(s)}(k) \\ \vdots \\ x_i^{(N_s)}(k) \\ x_p(k) \end{pmatrix} + c$$

which corresponds to the sequential Algorithm 1. □

An implementation of the synchronous parallel iterative sub-structuring method is proposed in Algorithm 4 for the process s.

Algorithm 4: Synchronous parallel iterative sub-structuring algorithm

```
Setup $T^{(s)}$, $c^{(s)}$;
$x_i^{(s)} = x_i^{(s)}(0)$;
$x_p^{(s)} = x_p^{(s)}(0)$;
$NbIter = 0$;
while $NbIter < MaxIter$ do
    $NbIter$++;
    // Summation of the distributed $x_p^{(s)}$
    MPI_AllReduce($x_p$,$x_p^{(s)}$,MPI_SUM);
    // Local computation of $y^{(s)}(k+1) = g^{(s)}(y(k))$
    $x_p^{(s)} = T_{pi}^{(s)} * x_i^{(s)} + T_{pp}^{(s)} * x_p + c_p^{(s)}$;
    $x_i^{(s)} = T_{ii}^{(s)} * x_i^{(s)} + T_{ip}^{(s)} * x_p + c_i^{(s)}$;
    if Global Convergence detected then
        return;
    end
end
```

5. Asynchronous iterative sub-structuring methods

Synchronous iterative sub-structuring methods have been widely used in engineering applications including structural mechanics [15,24]. In this section, we propose a new variant of sub-structuring methods where totally asynchronous iterations are considered. A new theorem is first introduced showing the asynchronous convergence of the iterative sub-structuring methods under some conditions.

Theorem 5. *If the matrix T satisfies the three conditions*

$$\rho\left(|T|\right) < 1 \tag{10}$$

$$\sum_{s=1}^{N_s} T_{pp}^{(s)} = T_{pp} \tag{11}$$

$$\sum_{s=1}^{N_s} \left|T_{pp}^{(s)}\right| = \left|T_{pp}\right| \tag{12}$$

then the associated sub-structuring method converges asynchronously.

To demonstrate Theorem 5 a proposition is first introduced.

Proposition 1. *For any matrix $T \in \mathbb{R}^{n\times n}$ there exist $\alpha \in \mathbb{R}$, $\beta \in \mathbb{R}$ and $w \in \mathbb{R}^n$ such that*

$$0 < \alpha < \beta < 1, \quad w > 0 \quad \textit{and} \quad |T|\,w \le \alpha w. \tag{13}$$

Proof. Applying Theorem 3 on $|T|$ gives:

$$\forall \epsilon > 0,\ \exists w \in \mathbb{R}^n,\ w > 0 \text{ such that } \quad \rho(|T|) \le \|\,|T|\,\|_\infty^w \le \rho(|T|) + \epsilon.$$

Taking $\epsilon = \frac{1-\rho(|T|)}{2}$ and choosing one w which satisfies the previous relation and choosing $\alpha = \|\,|T|\,\|_\infty^w$ and $\beta = 1 - \frac{1-\alpha}{2}$ gives:

$$\alpha = \|\,|T|\,\|_\infty^w = \max_x \frac{\|\,|T|\,x\,\|_\infty^w}{\|x\|_\infty^w} \ge \frac{\|\,|T|\,w\,\|_\infty^w}{\|w\|_\infty^w} = \|\,|T|\,w\,\|_\infty^w = \max_i \frac{\sum_j |t_{ij}|\,w_j}{w_i}$$

and leads to,

$$\forall i, \quad \alpha \ge \frac{\sum_j |t_{ij}|\,w_j}{w_i} = \frac{(|T|\,w)_i}{w_i}$$

which concludes the proof. □

The demonstration of Theorem 5 is now presented.

Proof. According to Proposition 1, we choose α, β and w, such that Eq. (13) is satisfied. The decomposition of w on the sub-structures and the interfaces gives:

$$w = \begin{pmatrix} w_i^{(1)} \\ \vdots \\ w_i^{(s)} \\ \vdots \\ w_i^{(N_s)} \\ w_p \end{pmatrix}.$$

For each sub-structure s, the vector $\theta^{(s)}$ is defined by:

$$\theta^{(s)} = \frac{1}{\beta}\left(\left|T_{pi}^{(s)}\right| \quad \left|T_{pp}^{(s)}\right|\right)\begin{pmatrix} w_i^{(s)} \\ w_p \end{pmatrix} + \frac{\beta-\alpha}{N_s\beta} w_p. \tag{14}$$

Since $w > 0$, $\beta > 0$ and $\beta - \alpha > 0$ this implies $\forall s, \theta^{(s)} > 0$. For each sub-structure the quantity $x_p^{(s)*}$ is defined by:

$$x_p^{(s)*} = \begin{pmatrix} T_{pi}^{(s)} & T_{pp}^{(s)} \end{pmatrix}\begin{pmatrix} x_i^{(s)*} \\ x_p^* \end{pmatrix} + c_p^{(s)}.$$

The sum of these quantities gives

$$\sum_{s=1}^{N_s} x_p^{(s)*} = x_p^*.$$

The quantity $\Delta(k) = \beta^k \Delta$ (read β power k here) where

$$\Delta = \max\left(\left\|x_0 - x^*\right\|_\infty^w,\ \max_{s=1,\dots,N_s}\left\|x_p^{(s)}(0) - x_p^{*(s)}\right\|_\infty^{\theta^{(s)}}\right)$$

is defined together with the following sets (for each sub-structure s):

$$\begin{aligned}
X_i^{(s)}(k) &= \left\{x_i^{(s)} \in X_i^{(s)}, \left|x_i^{(s)} - x_i^{(s)*}\right| \leq \Delta(k) w_i^{(s)}\right\}\\
X_p^{(s)}(k) &= \left\{x_p \in X_p, \left|x_p - x_p^{*(s)}\right| \leq \Delta(k)\theta^{(s)}\right\}\\
Y^{(s)}(k) &= X_i^{(s)}(k) \times X_p^{(s)}(k)
\end{aligned}$$

where $X_i^{(s)}$ (resp. X_p) denotes the space associated with the degree of freedom of the sub-structure s (resp. the interface). Two additional useful sets are defined by

$$\begin{aligned}
X_p(k) &= \left\{x_p \in X_p, \left|x_p - x_p^*\right| \leq \Delta(k) w_p\right\}\\
Y(k) &= Y^{(1)}(k) \times Y^{(2)}(k) \times \cdots \times Y^{(N_s)}(k).
\end{aligned}$$

In the following, the relation: $\forall k,\ y \in Y(k) \Rightarrow g(y) \in Y(k+1)$ will be derived.

From here, k is fixed and y is supposed to belong to $Y(k)$. This implies

$$\forall s,\ x_i^{(s)} \in X_i^{(s)} \quad \text{and} \quad x_p^{(s)} \in X_p^{(s)}.$$

The quantity x_p is defined by $x_p = \sum_{s=1}^{N_s} x_p^{(s)}$ and we will show that $x_p \in X_p$.

$$\begin{aligned}
\left|x_p - x_p^*\right| &\leq \sum_{s=1}^{N_s} \left|x_p^{(s)} - x_p^{(s)*}\right|\\
&\leq \sum_{s=1}^{N_s} \Delta(k)\theta^{(s)}\\
&= \Delta(k) \sum_{s=1}^{N_s} \theta^{(s)}\\
&= \Delta(k) \frac{1}{\beta} \sum_{s=1}^{N_s} \left(\left(\left|T_{pi}^{(s)}\right| \quad \left|T_{pp}^{(s)}\right| \right) \begin{pmatrix} w_i^{(s)} \\ w_p \end{pmatrix} + \frac{\beta - \alpha}{N_s} w_p \right)\\
&= \frac{\Delta(k)}{\beta} \left(\sum_{s=1}^{N_s} \left|T_{pi}^{(s)}\right| w_i^{(s)} + \sum_{s=1}^{N_s} \left|T_{pp}^{(s)}\right| w_p + \sum_{s=1}^{N_s} \frac{\beta - \alpha}{N_s} w_p \right) && \text{(15a)}\\
&= \frac{\Delta(k)}{\beta} \left(\sum_{s=1}^{N_s} \left|T_{pi}^{(s)}\right| w_i^{(s)} + \left|T_{pp}\right| w_p + (\beta - \alpha) w_p \right) && \text{(15b)}\\
&\leq \frac{\Delta(k)}{\beta} \left(\alpha w_p + (\beta - \alpha) w_p \right) && \text{(15c)}\\
&= \Delta(k) w_p. && \text{(15d)}
\end{aligned}$$

The hypothesis of Theorem 5 allows the passage from (15a) to (15b). Relation (13) allows the passage from (15b) to (15c) by remarking that

$$\sum_{s=1}^{N_s} \left|T_{pi}^{(s)}\right| w_i^{(s)} + \left|T_{pp}\right| w_p$$

is the restriction of $|T|\, w$ to X_p. Eq. (15d) means that $x_p \in X_p$.

We will show that $g(y) \in Y(k+1)$:

$$g(y) \in Y(k+1) \iff \forall s, g^{(s)}(y) \in Y^{(s)}(k+1)$$
$$\iff \forall s, \begin{cases} \begin{pmatrix} T_{ii}^{(s)} & T_{ip}^{(s)} \end{pmatrix} \begin{pmatrix} x_i^{(s)} \\ x_p \end{pmatrix} + c_i^{(s)} \in X_i^{(s)}(k+1) \\ \begin{pmatrix} T_{pi}^{(s)} & T_{pp}^{(s)} \end{pmatrix} \begin{pmatrix} x_i^{(s)} \\ x_p \end{pmatrix} + c_p^{(s)} \in X_p^{(s)}(k+1). \end{cases}$$

The first condition is proven by:

$$\begin{aligned}
\left| \begin{pmatrix} T_{ii}^{(s)} & T_{ip}^{(s)} \end{pmatrix} \begin{pmatrix} x_i^{(s)} \\ x_p \end{pmatrix} + c_i^{(s)} - x_i^{(s)*} \right| &= \left| \begin{pmatrix} T_{ii}^{(s)} & T_{ip}^{(s)} \end{pmatrix} \begin{pmatrix} x_i^{(s)} - x_i^{(s)*} \\ x_p - x_p^* \end{pmatrix} \right| \\
&\le \begin{pmatrix} \left|T_{ii}^{(s)}\right| & \left|T_{ip}^{(s)}\right| \end{pmatrix} \begin{pmatrix} \left|x_i^{(s)} - x_i^{(s)*}\right| \\ \left|x_p - x_p^*\right| \end{pmatrix} \\
&\le \begin{pmatrix} \left|T_{ii}^{(s)}\right| & \left|T_{ip}^{(s)}\right| \end{pmatrix} \begin{pmatrix} \Delta(k) w_i^{(s)} \\ \Delta(k) w_p \end{pmatrix} \\
&\le \Delta(k) \left(\left|T_{ii}^{(s)}\right| w_i^{(s)} + \left|T_{ip}^{(s)}\right| w_p \right) && \text{(16a)} \\
&\le \Delta(k) \beta w_i^{(s)} && \text{(16b)} \\
&= \Delta(k+1) w_i^{(s)}.
\end{aligned}$$

Relation (13) allows the passage from (16a) to (16b) by remarking that $\left|T_{ii}^{(s)}\right| w_i^{(s)} + \left|T_{ip}^{(s)}\right| w_p$ is the restriction of $|T|\, w$ to $X_i^{(s)}$.

The second condition is proven by:

$$\begin{aligned}
\left| \begin{pmatrix} T_{pi}^{(s)} & T_{pp}^{(s)} \end{pmatrix} \begin{pmatrix} x_i^{(s)} \\ x_p \end{pmatrix} + c_p^{(s)} - x_p^{(s)*} \right| &= \left| \begin{pmatrix} T_{pi}^{(s)} & T_{pp}^{(s)} \end{pmatrix} \begin{pmatrix} x_i^{(s)} - x_i^{(s)*} \\ x_p - x_p^* \end{pmatrix} \right| \\
&\le \begin{pmatrix} \left|T_{pi}^{(s)}\right| & \left|T_{pp}^{(s)}\right| \end{pmatrix} \begin{pmatrix} \left|x_i^{(s)} - x_i^{(s)*}\right| \\ \left|x_p - x_p^*\right| \end{pmatrix} \\
&\le \begin{pmatrix} \left|T_{pi}^{(s)}\right| & \left|T_{pp}^{(s)}\right| \end{pmatrix} \begin{pmatrix} \Delta(k) w_i^{(s)} \\ \Delta(k) w_p \end{pmatrix} && \text{(17a)} \\
&\le \Delta(k) \left(\beta \theta^{(s)} - \frac{\beta - \alpha}{N_s} w_p \right) && \text{(17b)} \\
&\le \Delta(k) \beta \theta^{(s)} \\
&= \Delta(k+1) \theta^{(s)}.
\end{aligned}$$

Relation (14) allows the passage from (17a) to (17b). From their definition, it is easy to see that:

$$\lim_{k \to \infty} \Delta(k) = 0$$
$$\lim_{k \to \infty} Y(k) = \left\{ y^* \right\}.$$

The sets $Y(k)$ and $Y^{(s)}(k)$ met the requirement of Assumption 1. And since Δ has been chosen so that the initial solution estimate belongs to $Y(0)$, Theorem 2 assures the convergence of the asynchronous iterative sub-structuring algorithm. □

The reader may wonder about the conditions (11) and (12) which might be difficult to ensure in practice. The following proposition suggests an easy and suitable choice for $T_{pp}^{(s)}$.

Proposition 2. *With an algebraic partitioning of the (assembled) matrix T, each block $T_{pp}^{(s)}$ can be defined in each sub-structure s, as $\frac{T_{pp}}{\alpha^{(s)}}$, in such a way that the sum of $\alpha^{(q)}$ (over all the sub-structures q, sharing the degree of freedom p with the sub-structure s) is equal to one. As a direct consequence, the two relations hold:*

$$T_{pp} = \sum_{s=1}^{N_s} T_{pp}^{(s)}$$

$$|T_{pp}| = \sum_{s=1}^{N_s} |T_{pp}^{(s)}|$$

and the parallel asynchronous iterative sub-structuring method converges if the synchronous iterative (Jacobi) method converges.

Proof. According to the algebraic partitioning suggested in Proposition 2, the two conditions (11) and (12) are immediately obtained.

The last condition to be satisfied is the condition (10), but this condition is the same as for the synchronous iterative (Jacobi) algorithm.

Three conditions of Theorem 5 are thus satisfied; this theorem ensures the convergence of the asynchronous iterative sub-structuring method, which concludes the proof. □

Remark 1. More suitable algebraic partitioning approaches might be considered as the ones introduced in [30,33] for Dirichlet-to-Neumann maps [31,32], an issue under current investigation.

A possible parallel implementation is presented in Algorithm 5.

Algorithm 5: Asynchronous parallel iterative sub-structuring algorithm

```
Setup $T^{(s)}, c^{(s)}$;
$x_i^{(s)} = x_i^{(s)}(0)$;
$x_p = x_p(0)$;
$NbIter = 0$;
while $NbIter < MaxIter$ do
    $NbIter$++;
    $x_p^{(s)} = T_{pi}^{(s)} * x_i^{(s)} + T_{pp}^{(s)} * x_p + c_p^{(s)}$;
    foreach Process $j$ do
        // Asynchronous, non blocking send
        Send(j,$x_p^{(s)}$);
    end
    $x_i^{(s)} = T_{ii}^{(s)} * x_i^{(s)} + T_{ip}^{(s)} * x_p + c_i^{(s)}$;
    foreach Message $m$ received do
        $x_p^{(I_{m.source})} = m.data$;
    end
    $x_p = \sum_{j=1}^{N_s} x_p^{(j)}$
    if Global Convergence detected then
        Return;
    end
end
```

6. Numerical results

The synchronous and asynchronous iterative sub-structuring solvers presented before have been implemented in Alinea [27], an advanced linear algebra library we have developed for massively parallel computations. The

Table 1
Convergence behaviour of the iterative row-band Jacobi algorithm (top: synchronous, bottom: asynchronous).

# processors	# iter	Comm (s)	Comp (s)	Total (s)
8	975	63.166	2.546	65.749
32	975	576.257	1.050	577.345
64	975	1491.225	0.894	1492.162
8	8179	1.065	19.07	20.393
32	3868	8.574	3.241	11.986
64	2237	67.447	3.139	70.677

synchronous and asynchronous communications are managed by JACK [28], an asynchronous communication kernel library we have developed on top of MPI.

The test cases are run on a cluster of sixty four computers. The characteristics of the computers are pentium 4 D (dual-core) at 3.4 GHz with 1Go of RAM running under Windows XP. The interconnected network is a switched, star shaped 10 Mb/s Ethernet network. The version of the MPI library used is MPICH2. The farm used for the experiments consists of 64 computers. In order to avoid communications bottleneck between the several cores of one processor, which will advantage our new asynchronous sub-structuring method, it is important to limit the experiments with one thread per processor only. If not, our method will outperform the synchronous version by an order of magnitude, but this will not be fair comparison.

In order to illustrate the convergence behaviour of the new asynchronous iterative sub-structuring methods proposed in this paper, several numerical experiments are now conducted. The test case consists of a 3D Poisson problem: $\Delta u = f$, within a domain Ω with homogeneous Dirichlet boundary conditions equal to one on the boundary $\partial\Omega$. The domain Ω has the shape of a cube $(-1, +1) \times (-1, 1) \times (-1, 1)$. The domain is meshed with a regular 3D grid composed of regular hexahedra, with a mesh size h equal to $h = 1/32$. Lagrange P_1 finite elements are considered for the discretisation, leading to the linear system $Kx = b$ with K a sparse matrix with 35 937 rows.

The matrix K is assembled and is first split into 8, 32, and 64 row-band sub-matrices. The fixed point problem is solved with the parallel iterative Jacobi algorithm. Each sub-matrix is allocated to a different processor. A residual criteria equal to 10^{-8} is considered on the quantity $\|x(k+1) - x(k)\|$ as the convergence criteria, see [3]. The results are reported in Table 1 and include the number of iterations, the communication time for the data transfer during the iterations, the computational time of the iterations, and the total time equal to the communication time plus the computational time. It is well known that the row-band synchronous parallel iterative Jacobi is not scalable when increasing the number of processors, since the data transfer between the processors represents a real drawback. Anyway, this simulation presents some interests to compare the behaviour of the synchronous version versus the asynchronous version. As we can see the key advantage of the asynchronous version is to reduce significantly the communication time, despite an increase of the number of iterations. This leads for the asynchronous version to a total time reduction by an order of magnitude compared to the synchronous version.

The parallel iterative sub-structuring method is now considered to solve the fixed point problem. The associated matrix is now split using the Metis [23,22] software into 8, 32, and 64 sub-matrices. When partitioning the (assembled) matrix K by Metis, each block $K_{pp}^{(s)}$ has been set to $\frac{K_{pp}}{\alpha^{(s)}}$ in each sub-matrix s, such as the sum of $\alpha^{(q)}$ over all the sub-matrices q sharing the degree of freedom p with the sub-matrix s, is equal to one. This algebraic partitioning guarantees the convergence of the asynchronous iterative sub-structuring method, if the synchronous Jacobi method converges. Each sub-matrix is allocated to a different processor. The convergence results are reported in Table 2. The obtained results confirm the efficiency of the sub-structuring approach compared to the row-band Jacobi approach, especially in the communication time. The synchronous parallel iterative sub-structuring methods are clearly scalable upon the number of processors (as expected). In addition, the asynchronous version is much faster (in time) than the synchronous version.

Remark 2. It is well known that the Jacobi iteration, is not a scalable numerical method. This lack of numerical scalability (as a stand-alone solver) is reflected by the large number of iterations for the synchronous method. For this reason, Jacobi is generally rather used as a smoother within a multilevel algorithm, an issue under our current

Table 2
Convergence behaviour of the iterative sub-structuring algorithm (top: synchronous, bottom: asynchronous).

# processors	# iter	Comm (s)	Comp (s)	Total (s)
8	975	6.692	1.992	8.712
32	975	5.220	0.490	5.725
64	975	3.461	0.214	3.684
8	1509	0.262	3.269	3.548
32	1851	0.443	1.733	2.190
64	3511	0.568	0.966	1.545

investigation to extend the asynchronous sub-structuring approach to Lagrangian [26] based domain decomposition methods [29].

7. Conclusions

In this paper, the extension to asynchronous iterations of sub-structuring method is presented. The mathematical proof of the convergence of this new method under the asynchronous iterations conditions is demonstrated.

In the numerical experiments, in synchronous mode, the sub-structuring method appears much faster than a parallel Jacobi method despite having the same number of iterations. In an asynchronous mode, its convergence rate is less affected by the asynchronous iterations, the method outperforms both synchronous sub-structuring methods (and of course synchronous and asynchronous Jacobi method).

Acknowledgements

The authors acknowledge partial financial support from the CRESTA project (European Commission), the Institut de Recherche Technologique SystemX (Pôle de Compétitivité, France), and GENCI (Grand Equipement National de Calcul Intensif, France) for the computer time used during this long-term research.

References

[1] J.M. Bahi, Asynchronous iterative algorithms for nonexpansive linear systems, J. Parallel Distrib. Comput. 60 (2000) 92–112.
[2] J.M. Bahi, S. Contassot-Vivier, R. Couturier, Parallel Iterative Algorithms: From Sequential to Grid Computing, CRC Press, Boca Raton, USA, 2007.
[3] J.M. Bahi, S. Contassot-Vivier, R. Couturier, F. Vernier, A decentralized convergence detection algorithm for asynchronous parallel iterative algorithms, IEEE Trans. Parallel Distrib. Syst. 16 (1) (2005) 4–13.
[4] G.M. Baudet, Asynchronous iterative methods for multiprocessors, J. ACM 25 (2) (1978) 226–244.
[5] D.P. Bertsekas, Distributed asynchronous computation of fixed points, Math. Program. 27 (1983) 107–120.
[6] D.P. Bertsekas, J.N. Tsitsiklis, Parallel and Distributed Comput.: Numer. Methods, Prentice-Hall, 1989.
[7] R. Bru, L. Elsner, M. Neumann, Models of parallel chaotic iteration methods, Linear Algebra Appl. 103 (1988) 175–192.
[8] R. Bru, V. Migallon, J. Penadés, Chaotic methods for the parallel solution of linear systems, Comput. Syst. Eng. 6 (4–5) (1995) 385–390.
[9] M. Chau, R. Couturier, J.M. Bahi, P. Spitéri, Asynchronous grid computation for American options derivatives, Adv. Eng. Softw. 60 (2013) 136–144.
[10] M. Chau, P. Spiteri, R. Guivarch, H.C. Boisson, Parallel asynchronous iterations for the solution of a 3D continuous flow electrophoresis problem, Comput. & Fluids 37 (2008) 1126–1137.
[11] D. Chazan, W. Miranker, Chaotic relaxation, Linear Algebra Appl. 2 (2) (1969) 199–222.
[12] J.D.P. Donnelly, Periodic chaotic relaxation, Linear Algebra Appl. 4 (2) (1971) 117–128.
[13] M. El Tarazi, Some convergence results for asynchronous algorithms, Numer. Math. 39 (1982) 325–340.
[14] M. El Tarazi, Algorithmes mixtes asynchrones. Étude de la convergence monotone, Numer. Math. 44 (1984) 363–369.
[15] C. Farhat, F.X. Roux, Implicit Parallel Processing in Structural Mechanics, North Holland, 1994.
[16] A. Frommer, H. Schwandt, D.B. Szyld, Asynchronous weighted additive Schwarz methods, Electron. Trans. Numer. Anal. 5 (1997) 48–61.
[17] A. Frommer, D.B. Szyld, Asynchronous two-stage iterative methods, Numer. Math. 69 (1994).
[18] A. Frommer, D.B. Szyld, Asynchronous iterations with flexible communication for linear systems, Calc. Parallèles 10 (1998) 421–429.
[19] T. Garcia, M. Chau, P. Spitéri, Synchronous and asynchronous distributed computing for financial option pricing, in: ICCSA, Vol. 2, 2011, pp. 664–679.
[20] A. Greenbaum, Iterative Methods for Solving Linear Systems, Society for Industrial and Appl. Math., Philadelphia, PA, USA, 1997.

[21] R. Guivarch, L. Giraud, Asynchronous additive Schwarz preconditioners for flexible Krylov solvers on convection-diffusion problems, in: Quatrième Sminaire sur l'Algorithmique Numérique Appliquée aux Problèmes Industriels, Calais, 2003.

[22] G. Karypis, METIS - Serial graph partitioning and fill-reducing matrix ordering, 2014.

[23] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM J. Sci. Comput. 20 (1) (1998) 359–392.

[24] P. Le Tallec, Domain Decomposition Methods in Computational Mechanics, North Holland, 1994.

[25] L. Lei, Convergence of asynchronous iteration with arbitrary splitting form, Linear Algebra Appl. 113 (1989) 119–127.

[26] Y. Maday, F. Magoulès, Absorbing interface conditions for domain decomposition methods: a general presentation, Comput. Methods Appl. Mech. Engrg. 195 (29–32) (2006) 3880–3900.

[27] F. Magoulès, A.-K. Cheik Ahamed, Alinea: An advanced linear algebra library for massively parallel computations on graphics processing units, Int. J. High Perform. Comput. Appl. 29 (3) (2015) 284–310.

[28] F. Magoulès, G. Gbikpi-Benissan, JACK: An asynchronous communication kernel library for iterative algorithms, J. Supercomput. (2016).

[29] F. Magoulès, F.-X. Roux, Lagrangian formulation of domain decomposition methods: a unified theory, Appl. Math. Model. 30 (7) (2006) 593–615.

[30] F. Magoulès, F.-X. Roux, L. Series, Algebraic way to derive absorbing boundary conditions for the Helmholtz equation, J. Comput. Acoust. 13 (3) (2005) 433–454.

[31] F. Magoulès, F.-X. Roux, L. Series, Algebraic Dirichlet-to-Neumann mapping for linear elasticity problems with extreme contrasts in the coefficients, Appl. Math. Model. 30 (8) (2006) 702–713.

[32] F. Magoulès, F.-X. Roux, L. Series, Algebraic approximation of Dirichlet-to-Neumann maps for the equations of linear elasticity, Comput. Methods Appl. Mech. Engrg. 195 (29–32) (2006) 3742–3759.

[33] F. Magoulès, F.-X. Roux, L. Series, Algebraic approach to absorbing boundary conditions for the Helmholtz equation, Int. J. Comput. Math. 84 (2) (2007) 231–240.

[34] J.C. Miellou, Algorithmes de relaxation chaotique à retards, RAIRO 1 (1975) 55–82.

[35] J.C. Miellou, Itérations chaotiques à retards. Études de la convergence dans le cas d'espaces partiellement ordonnés, C. R. Acad. Sci., Paris 280 (1975) 233–236.

[36] D.P. O'Leary, R.E. White, Multi-splittings of matrices and parallel solution of linear systems, SIAM J. Algebr. Discrete Methods 6 (4) (1985) 630–640.

[37] J.S. Przemieniecki, Matrix Structural Analysis of Substructures, Vol. 1, AIAA, 1963, pp. 138–147.

[38] Y. Saad, Iterative Methods for Sparse Linear Syst., PWS, 1995.

[39] P. Spitéri, M. Chau, Parallel asynchronous Richardson method for the solution of obstacle problem, High Perform. Comput. Syst. Appl. (2002).

[40] J.N. Tsitsiklis, On the stability of asynchronous iterative processes, Theory Comput. Syst. 20 (1) (1987) 137–153.

[41] J.N. Tsitsiklis, D. Bertsekas, M. Athans, Distributed asynchronous deterministic and stochastic gradient optimization algorithms, IEEE Trans. Automat. Control 31 (9) (1986) 803–812.