ELSEVIER Journal of Computational and Applied Mathematics 60 (1995) 47–62

JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS

Asynchronous parallel methods for enclosing solutions of nonlinear equations

Andreas Frommer[a,*], Hartmut Schwandt[b]

[a] *Fachbereich Mathematik, Bergische Universität GH Wuppertal, Gaußstraße 20, D-42097 Wuppertal, Germany*
[b] *Fachbereich Mathematik, Technische Universität Berlin, Straße des 17. Juni 135, D-10623 Berlin, Germany*

Received 30 October 1993; revised 22 March 1994

Abstract

We consider interval arithmetic based parallel methods for enclosing solutions of nonlinear systems of equations, where processors are allowed to proceed asynchronously. We present a general study on the convergence of asynchronous iterations in interval spaces and apply them to a variety of methods for the nonlinear equations case. The convergence results turn out to be very similar to those known for synchronous methods. Several practical examples on shared memory architectures are included. The asynchronous methods sometimes perform substantially better than their synchronous counterparts.

Keywords: Asynchronous iterations; Nonlinear equations; Enclosure methods; Parallel computing

1. Introduction

Let $f: \mathbb{R}^n \to \mathbb{R}^n$ be a nonlinear mapping having a zero $x^* \in \mathbb{R}^n$, i.e.,

$$f(x^*) = 0. \tag{1}$$

This paper considers special methods for calculating enclosures of x^* iteratively. Our primary interest is in *parallel* methods which can be run *asynchronously*. Asynchronous methods recently have attracted a lot of attention (see [3, 5, 7, 12–15, 18–20, 22, 25, 28–30, 33, 39, 40], for example). Their importance lies in the fact that they achieve minimal idle times on the individual processors of a parallel computer by allowing them to continue their iteration independently from each other, even if some processors have not yet finished the current updating. Several practical examples in the noninterval case (see [4, 5, 10, 16, 20]) show that asynchronous methods can significantly outperform their synchronous counterparts.

* Corresponding author. E-mail: frommer@math.uni-wuppertal.de.

In this paper we will first derive several rather generally formulated convergence results for asynchronous iterations on interval spaces (Section 2). We will then apply these results to various well-known enclosure methods for solutions of (1) in Section 3. Finally, Section 4 contains numerical experiments obtained on a Cray Y-MP and a Sequent Symmetry S81 shared memory parallel computer. In these examples the asynchronous methods sometimes perform substantially better than the synchronous variants.

The remaining part of this section is devoted to the introduction of notation and other preliminaries.

Interval quantities will always be typeset in boldface letters. We denote $\mathbb{IR}$ the set of all real compact intervals $\boldsymbol{a} = [\bar{a}, \underline{a}]$, where $\underline{a} \leqslant \bar{a}$. $\mathbb{R}$ is considered a subspace of $\mathbb{IR}$ by identifying $a \in \mathbb{R}$ with the interval $[a, a]$. Similarly, we denote $\mathbb{IR}^n$ and $\mathbb{IR}^{n \times n}$ the space of all interval vectors and interval matrices, respectively. So if $\boldsymbol{x} \in \mathbb{IR}^n$ then $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)^{\mathrm{T}}$ with $\boldsymbol{x}_i \in \mathbb{IR}$ and if $\boldsymbol{A} \in \mathbb{IR}^{n \times n}$ then $\boldsymbol{A} = (\boldsymbol{a}_{ij})$ with $\boldsymbol{a}_{ij} \in \mathbb{IR}$.

We assume that the reader is familiar with the basics of interval arithmetic as described, for example, in [1] or [31]. For $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{IR}$ the quantity $q(\boldsymbol{a}, \boldsymbol{b}) := \max\{|\underline{a} - \underline{b}|, |\bar{a} - \bar{b}|\} \in \mathbb{R}$ is a metric, the Hausdorff-distance. $\mathbb{IR}$ is complete with respect to this metric and we will always assume that $\mathbb{IR}^n$ is equipped with the corresponding product topology. Moreover if $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{IR}^n$ we use q also to denote the real vector $q(\boldsymbol{x}, \boldsymbol{y}) := (q(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, q(\boldsymbol{x}_n, \boldsymbol{y}_n))^{\mathrm{T}}$.

2. Asynchronous iterations in interval spaces

Let

$$\boldsymbol{h} : \mathbb{IR}^n \to \mathbb{IR}^n, \qquad \boldsymbol{h}(\boldsymbol{x}) = (\boldsymbol{h}_1(\boldsymbol{x}), \ldots, \boldsymbol{h}_n(\boldsymbol{x}))^{\mathrm{T}}, \tag{2}$$

be an operator on $\mathbb{IR}^n$. In our context, $\boldsymbol{h}$ should be regarded as some fixed point equation for f in (1), i.e., x^* in (1) satisfies $\boldsymbol{h}(x^*) = x^*$.

Assume that we are given a parallel computer with processors $P_1, \ldots, P_p$ each of which can read and write data to a common shared memory. Let $\{1, \ldots, n\} = \bigcup_{t=1}^{p} J_t$. Then a fixed point iteration for $\boldsymbol{h}$ is given by the following pseudocode for processor P_t

```
repeat
  read($\boldsymbol{x}$)
  calculate $\boldsymbol{x}_i^{\text{new}} := \boldsymbol{h}_i(\boldsymbol{x}),\ i \in J_t$
  overwrite $\boldsymbol{x}_i$ with $\boldsymbol{x}_i^{\text{new}},\ i \in J_t$.
```

So each processor updates components belonging to "his" range J_t. A first crucial point for the efficiency of this algorithm thus is, that one should be able to evaluate a block of components $\boldsymbol{h}_i$, $i \in J_t$, independently from the other components $\boldsymbol{h}_j$, $j \notin J_t$. This is the case, of course, if all components of $\boldsymbol{h}$ are given explicitly. However, in the methods to be considered later, this will only exceptionally be the matter. Rather, we will have an at least partly implicit definition of the components of $\boldsymbol{h}$ as a solution of a certain linear system. If this system is block diagonal, components corresponding to one block can still be evaluated independently from the other blocks. A more detailed discussion of this issue will be given in Remark 3.1 at the end of Section 3.

The second important point is that there is no synchronization between the processors in the above algorithm. Usually, processors will work on their individual loops at different speeds (since, e.g., the sets J_t may have a different cardinality). In the presence of synchronization some processors would thus be idle waiting for the slowest of them to finish their updates. Since we do not synchronize, these idle times will be minimized, but for a given time different processors will now have performed different numbers of their respective loops. The resulting overall process may be described by an asynchronous iteration according to the following definition (see [3]).

Definition 2.1. For $k = 1,2,\dots$ let be given nonempty sets $I^k \subseteq \{1,\dots,n\}$ and n-tuples $(s_1(k),\dots,s_n(k))$ of nonnegative integers. Suppose that the following three conditions hold:

(i) $s_i(k) \leqslant k-1$ for $i = 1,\dots,n$, $k = 1,2,\dots$,

(ii) $\lim_{k\to\infty} s_i(k) = \infty$ for $i = 1,\dots,n$,

(iii) for every $i \in \{1,\dots,n\}$ the set $\{k \mid i \in I^k\}$ is unbounded.

Then the iterative method which, starting with an initial guess $\boldsymbol{x}^0 \in \mathbb{R}^n$, calculates the iterates $\boldsymbol{x}^k$ according to

$$x_i^k = \begin{cases} x_i^{k-1} & \text{if } i \notin I^k, \\ \boldsymbol{h}_i(x_1^{s_1(k)},\dots,x_n^{s_n(k)}) & \text{if } i \in I^k \end{cases} \tag{3}$$

is termed *asynchronous iteration* for $\boldsymbol{h}$.

We refer to [6,17,19] for a more detailed discussion why practical asynchronous iterations usually always fit into Definition 2.1. A very general convergence result for the asynchronous iteration (3) is the following.

Theorem 2.2. *Let $\{\boldsymbol{E}^k\}_{k=0}^{\infty}$ be a sequence of subsets of $\mathbb{R}^n$ such that*

(i) $\boldsymbol{E}^k = \boldsymbol{E}_1^k \times \cdots \times \boldsymbol{E}_n^k$ *with* $\boldsymbol{E}_i^k \subseteq \mathbb{R}$, $i = 1,\dots,n$, $k = 0,1,\dots$,

(ii) $\boldsymbol{h}(\boldsymbol{E}^k) \subseteq \boldsymbol{E}^{k+1} \subseteq \boldsymbol{E}^k$, $k = 0,1,\dots$,

(iii) $\bigcap_{k=1}^{\infty} \boldsymbol{E}^k = \{\boldsymbol{x}^*\}$.

Then, if $\boldsymbol{x}^0 \in \boldsymbol{E}^0$, the iterates $\boldsymbol{x}^k$ of the asynchronous iteration (3) *converge to $\boldsymbol{x}^*$.*

Proof. This result can be found, for example, in [6] or [40], where it is shown for self-mappings of arbitrary finite-dimensional product spaces. See also [19]. □

We now develop three important corollaries of Theorem 2.2. Let us start with:

Corollary 2.3. *Let $\|\cdot\|$ denote a weighted maximum norm in $\mathbb{R}^n$, i.e.,*

$$\|x\| := \max_{i=1}^{n} |\alpha_i x_i| \quad \textit{with } \alpha_i > 0 \textit{ fixed}, \ i = 1,\dots,n.$$

Assume that $\boldsymbol{h}$ in (2) *satisfies*

$$\|q(\boldsymbol{h}(\boldsymbol{x}),\boldsymbol{h}(\boldsymbol{y}))\| \leqslant \gamma \cdot \|q(\boldsymbol{x},\boldsymbol{y})\| \tag{4}$$

with $\gamma \in [0,1)$ *for all* $\boldsymbol{x},\boldsymbol{y} \in \mathbb{R}^n$. *Then the asynchronous iterates* (3) *converge to* $\boldsymbol{x}^*$, *the unique fixed point of* $\boldsymbol{h}$, *for any initial guess* $\boldsymbol{x}^0$.

Proof. Existence and uniqueness of $\boldsymbol{x}^*$ follow from (4) by Banach's fixed-point theorem. The other assertions follow from Theorem 2.2 by setting

$$\begin{aligned} \boldsymbol{E}^k &= \{\boldsymbol{x} \mid \|q(\boldsymbol{x},\boldsymbol{x}^*)\| \leqslant \gamma^k \cdot \|q(\boldsymbol{x}^0,\boldsymbol{x}^*)\|\} \\ &= \boldsymbol{E}_1^k \times \cdots \times \boldsymbol{E}_n^k \quad \text{with } \boldsymbol{E}_i^k = \left\{\boldsymbol{x}_i \mid q(\boldsymbol{x}_i,\boldsymbol{x}_i^*) \leqslant \frac{\gamma^k}{\alpha_i} \cdot \|q(\boldsymbol{x}^0,\boldsymbol{x}^*)\|\right\}. \qquad \square \end{aligned}$$

A special case of this corollary arises for so called *P-contractions.*

Definition 2.4. The operator $\boldsymbol{h}$ in (2) is called a *P-contraction* if there exists a real matrix $P \in \mathbb{R}^{n\times n}$, $P \geqslant 0$, with spectral radius $\rho(P)$ less than one such that

$$q(\boldsymbol{h}(\boldsymbol{x}),\boldsymbol{h}(\boldsymbol{y})) \leqslant P \cdot q(\boldsymbol{x},\boldsymbol{y})$$

for all $\boldsymbol{x},\boldsymbol{y} \in I\mathbb{R}^n$. (Here, $\leqslant$ in $\mathbb{R}^n$ and $\mathbb{R}^{n\times n}$ is to be understood componentwise.)

Using the Perron–Frobenius theory (see for example [41]) one can easily show that a *P*-contraction actually satisfies (4). (The α_i in the weighted maximum norm $\|\cdot\|$ have to be taken as the inverses of the components of the Perron vector of a positive matrix sufficiently close to *P*, see [27] for details.) Therefore we have the following corollary as a yet special case of Corollary 2.3.

Corollary 2.5. *Let* $\boldsymbol{h}$ *in* (2) *be a P-contraction. Then the asynchronous iterates* (3) *converge to* $\boldsymbol{x}^*$, *the unique fixed point of* $\boldsymbol{h}$, *for any initial guess* $\boldsymbol{x}^0$.

It is interesting to discuss Corollary 2.5 in the case where $\boldsymbol{h}$ is of linear form, i.e.,

$$\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b},$$

with $\boldsymbol{A} \in I\mathbb{R}^{n\times n}$, $\boldsymbol{b} \in I\mathbb{R}^n$. Then (see [1, Theorem 12.1]) the synchronous iteration

$$\bar{\boldsymbol{x}}^{k+1} = \boldsymbol{A}\bar{\boldsymbol{x}}^k + \boldsymbol{b}$$

converges to a unique fixed point of $\boldsymbol{h}$ for any initial guess if and only if $\rho(|\boldsymbol{A}|) < 1$, where $|\boldsymbol{A}| = (|\boldsymbol{a}_{ij}|)$ with the modulus $|\boldsymbol{a}|$ of an interval $\boldsymbol{a}$ defined by $|\boldsymbol{a}| = \max\{|a| \mid a \in \boldsymbol{a}\}$. This condition, however, precisely means that $\boldsymbol{h}$ is a *P*-contraction (with $P = |\boldsymbol{A}|$). This gives us the quite surprising result that in this linear case the asynchronous iteration (3) converges for any initial guess *if and only if* the synchronous method (2) converges for any initial guess.

Another corollary of Theorem 2.2 can be obtained by considering inclusion isotone functions.

Definition 2.6. The operator $\boldsymbol{h}$ in (2) is called *inclusion isotone* if we have

$$\boldsymbol{x},\boldsymbol{y} \in I\mathbb{R}^n,\ \boldsymbol{x} \subseteq \boldsymbol{y} \ \Rightarrow\ \boldsymbol{h}(\boldsymbol{x}) \subseteq \boldsymbol{h}(\boldsymbol{y}).$$

Inclusion isotone functions arise rather frequently in enclosure methods since the basic operations $+$, $-$, $*$, $/$ in $\mathbb{IR}$ are inclusion isotone.

Corollary 2.7. *Let $\boldsymbol{h}$ in* (2) *be inclusion isotone. Assume that*

(i) $\boldsymbol{h}(\boldsymbol{x}^0) \subseteq \boldsymbol{x}^0$,

(ii) *there exists a fixed point $\boldsymbol{x}^*$ of $\boldsymbol{h}$ with $\boldsymbol{x}^* \subseteq \boldsymbol{x}^0$.*

Then the asynchronous iterates (3) *converge to a fixed point $\tilde{\boldsymbol{x}}$ of $\boldsymbol{h}$ with $\boldsymbol{x}^* \subseteq \tilde{\boldsymbol{x}} \subseteq \boldsymbol{x}^0$. Moreover, the synchronous iteration*

$$\bar{\boldsymbol{x}}^k = \boldsymbol{h}(\bar{\boldsymbol{x}}^{k-1}), \quad \bar{\boldsymbol{x}}^0 = \boldsymbol{x}^0, \ k = 1, 2, \ldots$$

converges to the same fixed point $\tilde{\boldsymbol{x}}$.

Proof. From (i), (ii) and the inclusion isotonicity we get (inductively)

$$\boldsymbol{x}^* \subseteq \bar{\boldsymbol{x}}^k \subseteq \bar{\boldsymbol{x}}^{k-1}, \quad k = 1, 2, \ldots$$

so that $\lim_{k\to\infty} \bar{\boldsymbol{x}}^k =: \tilde{\boldsymbol{x}}$ exists with $\tilde{\boldsymbol{x}} \supseteq \boldsymbol{x}^*$. Let

$$\begin{aligned} \boldsymbol{E}^k &= \{\boldsymbol{x} \,|\, \tilde{\boldsymbol{x}} \subseteq \boldsymbol{x} \subseteq \bar{\boldsymbol{x}}^k\} \\ &= \boldsymbol{E}_1^k \times \cdots \times \boldsymbol{E}_n^k \quad \text{with } \boldsymbol{E}_i^k = \{\tilde{\boldsymbol{x}}_i \subseteq \boldsymbol{x}_i \subseteq \bar{\boldsymbol{x}}_i^k\}, \end{aligned}$$

observe that $\boldsymbol{h}(\boldsymbol{E}^k) \subseteq \boldsymbol{E}^{k+1} \subseteq \boldsymbol{E}^k$ for $k = 0, 1, \ldots$ and $\bigcap_{k=1}^{\infty} \boldsymbol{E}^k = \{\tilde{\boldsymbol{x}}\}$, and apply Theorem 2.2. □

Our final convergence result in this section deals with the situation, where, instead of inclusion isotonicity, we only have the property

$$\boldsymbol{x} \in \mathbb{IR}^n, \boldsymbol{x}^* \subseteq \boldsymbol{x} \ \Rightarrow \ \boldsymbol{x}^* \subseteq \boldsymbol{h}(\boldsymbol{x}), \tag{5}$$

where $\boldsymbol{x}^* \in \mathbb{IR}^n$ is supposed to be a fixed point of $\boldsymbol{h}$. It is then of very common use to improve the synchronous iterates by taking intersections with the previous iterate, i.e., one takes

$$\boldsymbol{x}^k = \boldsymbol{h}(\boldsymbol{x}^{k-1}) \cap \boldsymbol{x}^{k-1}, \quad k = 1, 2, \ldots. \tag{6}$$

We therefore consider the asynchronous iteration

$$\boldsymbol{x}_i^k = \begin{cases} \boldsymbol{x}_i^{k-1} & \text{if } i \notin I^k, \\ \boldsymbol{h}_i(\boldsymbol{x}_1^{s_1(k)}, \ldots, \boldsymbol{x}_n^{s_n(k)}) \cap \boldsymbol{x}_i^{s_i(k)} & \text{if } i \in I^k \end{cases} \tag{7}$$

with I^k, $s_i(k)$ as in Definition 2.1. Note that this iteration can break down due to empty intersection.

Theorem 2.8. *Let $\boldsymbol{h}$ satisfy* (5) *where $\boldsymbol{x}^*$ is a fixed point of $\boldsymbol{h}$ and denote $\boldsymbol{x}^k$ the iterates of* (7).

(i) *If $\boldsymbol{x}^* \subseteq \boldsymbol{x}^0$ then $\boldsymbol{x}^* \subseteq \boldsymbol{x}^k$ for all k, so that iteration* (7) *will never break down.*

(ii) *If the asynchronous iteration* (7) *breaks down due to empty intersection, then $\boldsymbol{x}^0$ does not contain $\boldsymbol{x}^*$.*

(iii) *If $\boldsymbol{x}^* \subseteq \boldsymbol{x}^0$ and $s_i(k) = k - 1$ for all $i \in I^k$, $k = 1, 2, \ldots,$ then the asynchronous iterates in* (7) *converge to some limit $\tilde{\boldsymbol{x}} \supseteq \boldsymbol{x}^*$. If $\boldsymbol{h}$ is continuous, $\tilde{\boldsymbol{x}}$ satisfies*

$$\tilde{\boldsymbol{x}} = \boldsymbol{h}(\tilde{\boldsymbol{x}}) \cap \tilde{\boldsymbol{x}}. \tag{8}$$

Proof. We show (i) by induction. Indeed, $\boldsymbol{x}^* \subseteq \boldsymbol{x}^0$ is true by assumption. If up to some k we have $\boldsymbol{x}^* \subseteq \boldsymbol{x}^l$ for $l = 0, 1, \ldots, k-1$ we have $\boldsymbol{x}^* \subseteq (\boldsymbol{x}_1^{s_1(k)}, \ldots, \boldsymbol{x}_n^{s_n(k)})^{\mathrm{T}}$ so that (5) yields

$$\boldsymbol{x}^* \subseteq \boldsymbol{h}(\boldsymbol{x}_1^{s_1(k)}, \ldots, \boldsymbol{x}_n^{s_n(k)})$$

and thus

$$\boldsymbol{x}^* \subseteq \boldsymbol{h}(\boldsymbol{x}_1^{s_1(k)}, \ldots, \boldsymbol{x}_n^{s_n(k)}) \cap \boldsymbol{x}^l, \quad l = 0, \ldots, k-1.$$

Reading the last inclusion componentwise with $l = s_i(k)$ yields $\boldsymbol{x}^* \subseteq \boldsymbol{x}_i^k$ for $i \in I^k$, whereas for $i \notin I_k$ we have $\boldsymbol{x}_i^* \subseteq \boldsymbol{x}_i^k = \boldsymbol{x}_i^{k-1}$ by the induction hypothesis. This proves (i).

(ii) is a direct consequence of (i).

To show (iii) observe that since $s_i(k) = k-1$ for $i \in I^k$ we have, together with (i),

$$\boldsymbol{x}^* \subseteq \boldsymbol{x}^k \subseteq \boldsymbol{x}^{k-1} \quad \text{for } k = 1, 2, \ldots$$

so $\lim_{k\to\infty} \boldsymbol{x}^k = \tilde{\boldsymbol{x}}$ exists with $\tilde{\boldsymbol{x}} \supseteq \boldsymbol{x}^*$. Equality (8) is trivial. □

Part (iii) of the above theorem seems less satisfactory than our previous results. It requires the additional assumption

$$s_i(k) = k-1 \quad \text{for } i \in I^k. \tag{9}$$

But even with this assumption it does not show that $\tilde{\boldsymbol{x}}$ is the same fixed point as the one we would end up at by doing a synchronous iteration.

So let us first comment that (9) is not as restrictive as it might appear. In particular, in the computational model we used to motivate Definition 2.1, requiring (9) just means that a given processor always uses the latest updates for components of the index range it is assigned to (provided the sets J_t are pairwise disjoint). So this requirement will be fulfilled most of the time in computational practice.

Theorem 2.8 is the most important theorem for dealing with methods for enclosing solutions of nonlinear equations. The next section will give a whole list of particular functions $\boldsymbol{h}$ which all satisfy (5). Typically, then, $\boldsymbol{x}^*$ is a point vector x^*, the solution of the nonlinear equation. Therefore one would like the synchronous as well as the asynchronous iterates to converge to x^*. Almost all known theoretical results to establish this for synchronous methods proceed in the same manner: They show that together with additional assumptions on $\boldsymbol{h}$ the equality

$$\boldsymbol{x} = \boldsymbol{h}(\boldsymbol{x}) \cap \boldsymbol{x}$$

is satisfied by x^* only. Since the limit point of the synchronous method satisfies the above equality one has established convergence towards x^*. But then, due to (8) we also know that the *asynchronous* iterates converge to x^*.

3. Enclosure methods for nonlinear equations

In this section we will present different operators $\boldsymbol{h} : \mathbb{IR}^n \to \mathbb{IR}^n$ the fixed points of which are solutions of the nonlinear equation

$$f(x) = 0, \qquad f : D \subseteq \mathbb{R}^n \to \mathbb{R}^n. \tag{10}$$

These operators will satisfy (5) with $\boldsymbol{x}^* = x^*$ being a zero of f. The synchronous iterations based on these operators represent well-known Newton-like interval methods for nonlinear equations. As we will see, their asynchronous variants can also all be used to calculate enclosures for a solution of (10) according to Theorem 2.8.

Throughout the whole section we assume that f is Fréchet-differentiable on an interval vector $\boldsymbol{x}^0 \in \mathrm{I}\mathbb{R}^n$ containing a solution x^* of (10). The methods to be considered all rely on the basic relation

$$f(m) = f(m) - f(x^*) = F'(m,x^*)(m - x^*). \tag{11}$$

Here, $m \in \boldsymbol{x}^0$ is arbitrary and the matrix $F'(m,x^*) \in \mathbb{R}^{n\times n}$ is chosen such that (11) holds. For example, we can take

$$F'(m,x^*) = (f'_{ij}(m,x^*)) = \left(\frac{\partial f_i(\xi^i)}{\partial x_j}\right) \tag{12}$$

with

$$\xi^i = (1 - \theta_i)m + \theta_i x^*, \quad \theta_i \in [0,1] \quad \text{for } i = 1,\dots,n,$$

according to the mean value theorem.

Another possible choice for $F'(m,x^*)$ is to take slopes defined by

$$f'_{ij}(m,x^*) = \begin{cases} \dfrac{f_i(m_1,\dots,m_j,x^*_{j+1},\dots,x^*_n) - f_i(m_1,\dots,m_{j-1},x^*_j,\dots,x^*_n)}{m_j - x^*_j} & \text{if } m_j \neq x^*_j, \\ \dfrac{\partial f_i(m_1,\dots,m_j,x^*_{j+1},\dots,x^*_n)}{\partial x_j} & \text{if } m_j = x^*_j. \end{cases} \tag{13}$$

Of course, in either case $F'(m,x^*)$ is not known explicitly (because we do not know x^*). However, using interval arithmetic, we can compute an interval matrix $\boldsymbol{F}'(m,\boldsymbol{x})$ with the property

$$F'(m,x^*) \in \boldsymbol{F}'(m,\boldsymbol{x}) \quad \text{if } m, x^* \in \boldsymbol{x} \subseteq \boldsymbol{x}^0.$$

For example in the case of (12), denoting $\partial f_i(\boldsymbol{x})/\partial x_j$ an interval arithmetic evaluation of $\partial f_i/\partial x_j$, we know by the inclusion isotonicity of interval arithmetic that

$$f'_{ij}(m,x^*) = \frac{\partial f_i(\xi^i)}{\partial x_j} \in \frac{\partial f_i(\boldsymbol{x})}{\partial x_j}, \tag{14}$$

since $\xi^i = (1 - \theta_i)m + \theta_i x^* \in \boldsymbol{x}$. Similarly, in the case of (13) we can apply the mean value theorem componentwise to obtain

$$f'_{ij}(m,x^*) \in \frac{\partial f_i(\boldsymbol{x}_1,\dots,\boldsymbol{x}_j,m_{j+1},\dots,m_n)}{\partial x_j}.$$

Alternatively in the case of (13) and for special functions f, such as rational functions, we also can compute (a different) $\boldsymbol{F}'(m,\boldsymbol{x})$ recursively, see [24].

We now decompose

$$F'(m,x^*) = M(m,x^*) - N(m,x^*) \tag{15}$$

with, for $m, x^* \in \boldsymbol{x}$,

$$M(m,x^*) \in \boldsymbol{M}(m,\boldsymbol{x}), \qquad N(m,x^*) \in \boldsymbol{N}(m,\boldsymbol{x}), \qquad F'(m,x^*) \in \boldsymbol{F}'(m,\boldsymbol{x}). \tag{16}$$

Assuming that $M(m,x^*)$ is nonsingular, we obtain from (11)

$$x^* = m - M(m,x^*)^{-1}\{N(m,x^*)(m - x^*) + f(m)\}.$$

Using (15),(16) and $x^* \in \boldsymbol{x}$ we thus get

$$x^* \in \boldsymbol{h}(\boldsymbol{x})$$

with the *Newton-like* operator

$$\boldsymbol{h}(\boldsymbol{x}) = m - \mathrm{LES}(\boldsymbol{M}(m,\boldsymbol{x}), \boldsymbol{N}(m,\boldsymbol{x})(m - \boldsymbol{x}) + f(m)). \tag{17}$$

Here, $\mathrm{LES}(\boldsymbol{A},\boldsymbol{b}) \in \mathrm{I}\mathbb{R}^n$ with $\boldsymbol{A} \in \mathrm{I}\mathbb{R}^{n\times n}$ and $\boldsymbol{b} \in \mathrm{I}\mathbb{R}^n$ denotes any interval "linear" equation solver, i.e.,

$$\mathrm{LES}(\boldsymbol{A},\boldsymbol{b}) \supseteq \{x \in \mathbb{R}^n \mid \exists A \in \boldsymbol{A}, b \in \boldsymbol{b}\colon Ax = b\}.$$

For example, $\mathrm{LES}(\boldsymbol{A},\boldsymbol{b})$ could be the result of the interval Gaussian algorithm (see [1]).

The vector m appearing in (17) is a function of $\boldsymbol{x}$, i.e., $m = m(\boldsymbol{x})$ with $m(\boldsymbol{x}) \in \boldsymbol{x}$. Often one chooses m to be the midpoint vector of $\boldsymbol{x}$. Also note that $\boldsymbol{h}$ in (17) is continuous if $m(\boldsymbol{x})$, $\boldsymbol{M}(m(\boldsymbol{x}),\boldsymbol{x})$ and $\boldsymbol{N}(m(\boldsymbol{x}),\boldsymbol{x})$ depend continuously on $\boldsymbol{x}$ and if $\mathrm{LES}(\boldsymbol{A},\boldsymbol{b})$ is continuous with respect to both its entries (as this is the case for the interval Gaussian algorithm).

Particular cases take $\boldsymbol{M}(m,\boldsymbol{x})$ to be the diagonal or lower triangular part of $\boldsymbol{F}'(m,\boldsymbol{x})$. In that case we get the familiar Newton–Jacobi or Newton–Gauss–Seidel operator (see [1]). Other methods in this class include those considered in [34,36] and the multistep methods from [21,37].

To derive another class of methods, let $M(m,\boldsymbol{x})$ be an arbitrary, nonsingular *real* matrix. Then we obtain from (11), setting $F'(m,x^*) = M(m,\boldsymbol{x}) - (M(m,\boldsymbol{x}) - F'(m,x^*))$, the relation

$$x^* = m - M(m,\boldsymbol{x})^{-1}\{(M(m,\boldsymbol{x}) - F'(m,x^*))(m - x^*) + f(m)\}. \tag{18}$$

Hence, if as before $F'(m,x^*) \in \boldsymbol{F}'(m,\boldsymbol{x})$ for $m, x^* \in \boldsymbol{x}$ we get

$$x^* \in \boldsymbol{h}(\boldsymbol{x})$$

with the *Krawczyk-like* operator

$$\boldsymbol{h}(\boldsymbol{x}) = m - \mathrm{LES}(M(m,\boldsymbol{x}), (M(m,\boldsymbol{x}) - \boldsymbol{F}'(m,\boldsymbol{x}))(m - \boldsymbol{x}) + f(m)). \tag{19}$$

Often one takes $M(m,\boldsymbol{x})$ as the midpoint matrix of $\boldsymbol{F}'(m,\boldsymbol{x})$. In contrast to the Newton-like operator (17), the Krawczyk-like operator requires the solution of a "linear" system with a *real* (noninterval) coefficient matrix. Krawczyk-like methods were considered in [2,23,35–37], for example.

Krawczyk's original operator is defined by

$$\boldsymbol{h}(\boldsymbol{x}) = m - Y(m,\boldsymbol{x})f(m) + (I - Y(m,\boldsymbol{x})\boldsymbol{F}'(m,\boldsymbol{x})(m - \boldsymbol{x})). \tag{20}$$

Here $Y(m,\boldsymbol{x})$ is usually chosen to be independent from m and $\boldsymbol{x}$ but close to the inverse of the midpoint matrix of $\boldsymbol{F}'(m,\boldsymbol{x})$. For $x^*, m \in \boldsymbol{x}$ we obtain $x^* \in \boldsymbol{h}(\boldsymbol{x})$ again. This follows from (18), substituting $Y(m,\boldsymbol{x})$ for $M(m,\boldsymbol{x})^{-1}$.

Asynchronous iterations based on the operators presented so far are characterized by two major particularities which we will discuss in the following two remarks.

Remark 3.1. We want to parallelize iterations with the above operators $\boldsymbol{h}$ by distributing (blocks of) components of $\boldsymbol{h}$ to the individual processors. For reasons of efficiency, it should therefore be possible that (blocks of) components of $\boldsymbol{h}$ can be evaluated independently from each other. The Newton- and Krawczyk-like operators (17) and (19) both involve the solution of "linear" systems with the coefficient matrix $\boldsymbol{M}(m,\boldsymbol{x})$. Hence, in both cases, (block) components of $\boldsymbol{h}$ can be evaluated independently only if $\boldsymbol{M}(m,\boldsymbol{x})$ is (block)diagonal.

Remark 3.2. A given component of $\boldsymbol{x}$ appears several times in each component $\boldsymbol{h}_i$ of any of the operators $\boldsymbol{h}$ presented in this section. For example, in the Newton-like operator (17) it appears in $\boldsymbol{M}(m,\boldsymbol{x})$ as well as $\boldsymbol{N}(m,\boldsymbol{x})$ and in the factor $(m(\boldsymbol{x}) - \boldsymbol{x})$. Therefore, in order to actually implement the asynchronous iteration (7) on a shared memory computer, each processor would have to make a private copy of the current iterate and then use this copy during one updating procedure. Of course, it could be more economic in an asynchronous iteration not to make these private copies and to always access the current iterate stored in the shared memory. But then it will happen that a processor uses different values for the same component of $\boldsymbol{x}$ when evaluating some (block) component of $\boldsymbol{h}$. Consequently the resulting iteration should be expressed as

$$\boldsymbol{x}_i^k = \begin{cases} \boldsymbol{x}_i^{k-1} & \text{if } i \notin I^k, \\ \boldsymbol{g}_i(\boldsymbol{x}_1^{s_1(1,k)}, \ldots, \boldsymbol{x}_1^{s_1(l,k)}, \ldots, \boldsymbol{x}_n^{s_n(1,k)}, \ldots, \boldsymbol{x}_n^{s_n(l,k)}) \cap \boldsymbol{x}_i^{s_i(l+1,k)} & \text{if } i \in I^k. \end{cases} \tag{21}$$

Here the sets I^k are as in Definition 2.1 and l is such that any component $\boldsymbol{x}_i$ of $\boldsymbol{x}$ appears at most l times in $\boldsymbol{h}$. (Consequently, if a given component $\boldsymbol{x}_i$ actually appears $l_i < l$ times in $\boldsymbol{h}$, the second line of (21) contains $l - l_i$ dummy entries for $\boldsymbol{x}_i$.) For all i, j and k we have $s_i(j,k) \leqslant k-1$ and

$$\boldsymbol{g}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n, \ldots, \boldsymbol{x}_n) = \boldsymbol{h}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n).$$

The properties of iteration (21) can substantially differ from the asynchronous iteration (7). In particular, even if $\boldsymbol{h}$ satisfies (5) and $\boldsymbol{x}^* \subseteq \boldsymbol{x}^0$ it can happen that we end up with $\boldsymbol{x}^* \not\subseteq \boldsymbol{x}^k$ for some k. As we will see from the subsequent discussion, in order to ensure $\boldsymbol{x}^* \subseteq \boldsymbol{x}^k$, $k = 0, 1, \ldots$, in (21) for the operators presented in this section, we basically have to guarantee that for $i = 1, \ldots, n$ the component $m_i(\boldsymbol{x})$ of the "midpoint" we use will really lie in all different interval components we use for $\boldsymbol{x}_i$. We now discuss this point in more detail.

Let us first observe that, as in the proof of Theorem 2.8, we obtain $\boldsymbol{x}^{k+1} \subseteq \boldsymbol{x}^k$ in (21) if $s(l+1,k) = k-1$ for $i \in I^k$. This condition is again rather automatically fulfilled in practice.

In order to guarantee $x^* \in \boldsymbol{x}^k$ for all k we have to assume that we use only one value for each component of $m(\boldsymbol{x})$ when evaluating $\boldsymbol{g}_i$ in (21). Let us associate this value with the index $j = l$, i.e., we take $m(\boldsymbol{x}_1^{s_1(l,k)}, \ldots, \boldsymbol{x}_n^{s_n(l,k)})$. The way we derived all operators $\boldsymbol{h}$ of this section shows that $x^* \in \boldsymbol{x}^k$ can be guaranteed for (21) only if for $i = 1, \ldots, n$ we have

$$\boldsymbol{x}_i^{s_i(l,k)} \subseteq \boldsymbol{x}_i^{s_i(j,k)}, \quad j = 1, \ldots, l-1. \tag{22}$$

The relation (22) implies that the computed $m_i \equiv m_i(\boldsymbol{x}_i^{s_i(l,k)})$ lie within all intervals $\boldsymbol{x}_i^{s_i(j,k)}$ used when evaluating $\boldsymbol{g}$ so that the analogue of (14) still holds. In order to guarantee (22) we only have to make sure that $s_i(l,k) \geqslant s_i(j,k)$ for $j = 1, \ldots, l-1$ and $i = 1, \ldots, n$. Practically, this means that we should determine the "midpoint" m we use in evaluating a (block) component of $\boldsymbol{h}$ only after all other occurrencies of $\boldsymbol{x}$ in this (block) component have already been evaluated.

4. Numerical examples

In this section we present several numerical experiments on two different shared memory parallel computers: A Cray Y-MP with 8 vector processors (peaking at 333 MFlops each) and a Sequent Symmetry S81 with 24 Intel 80386 processors together with 80387 and Weitek WTL 3167 floating point accelerators.[1]

On both computers we used interval arithmetic implemented in software through the iv2ftn preprocessor (see [38]). From a source written in Fortran style, but allowing for interval operations, this preprocessor generates Fortran 77 code where interval arithmetic operations are simulated on the basis of the existing floating point arithmetic (here Cray and IEEE, resp.). One thus obtains a correct machine interval arithmetic in the sense of [1].

All our examples are based on the nonlinear Dirichlet problem

$$\begin{aligned} &\Delta u(x,y) = u(x,y) + u^2(x,y) \quad \text{for } (x,y) \in \Omega := (0,1)\times(0,1),\\ &u \equiv 1 \quad \text{for } (x,y) \in \partial\Omega. \end{aligned} \tag{23}$$

This continuous problem is discretized using the five point difference star at N^2 equidistant mesh points (ih, jh), $i, j = 1, \ldots, N$, $h = 1/(N+1)$. We then obtain the following system of nonlinear equations in $\mathbb{R}^{N\cdot N}$ (denoting $\phi(u_{i,j}) := h^2(u_{i,j} + u_{i,j}^2)$):

$$f_{i,j}(u) = 4u_{i,j} + \phi(u_{i,j}) - u_{i,j-1} - u_{i,j+1} - u_{i-1,j} - u_{i+1,j} = 0, \quad i, j = 1, \ldots, N, \tag{24}$$

with $u_{i,j} = 1$ if $i \in \{0, N+1\}$ or $j \in \{0, N+1\}$. The component $u_{i,j}$ of the solution of the discretized problem is an approximation for the value of $u(ih, jh)$ in the continuous problem. In Eq. (24) variables and function components are indexed in a natural manner by the double subscript (i, j). We will keep this indexing scheme during this section. Be aware that whenever $i \in \{0, N+1\}$ or $j \in \{0, N+1\}$ in $u_{i,j}$ (or later in $\boldsymbol{x}_{i,j}$) we are referring to the known boundary value 1.

The almost linear function f in (24) is an M-function (see [32]), and if o and e denote the vector with all 0 and 1 entries, respectively, then $F(o) \leqslant o$ and $o \leqslant F(e)$. Thus, by [32, Theorem 13.5.2] we know that (24) has a solution x^* with $o \leqslant x^* \leqslant e$. Therefore, in all our examples we took as our initial guess the interval vector with all components equal to $[0, 1]$, containing the zero x^* of F.

In order to calculate enclosures for x^* in all examples we used Newton-like operators of the form (17). The interval matrices $\boldsymbol{F}'(m,\boldsymbol{x})$ were obtained by an interval arithmetic evaluation of $F'(x)$ (see (12)). So, since $\boldsymbol{F}'(m,\boldsymbol{x})$ does not depend on m here, we simply write $\boldsymbol{F}'(\boldsymbol{x})$ from now on and, similarly, $\boldsymbol{M}(\boldsymbol{x})$ and $\boldsymbol{N}(\boldsymbol{x})$.

According to Remark 3.1 at the end of Section 3 we have to choose $\boldsymbol{M}(\boldsymbol{x})$ with a (block)diagonal structure to be able to evaluate (block) components of the Newton-like operator (17) independently. We will consider two different such choices for $\boldsymbol{M}(\boldsymbol{x})$.

In our first example, the splitting $\boldsymbol{F}'(\boldsymbol{x}) = \boldsymbol{M}(\boldsymbol{x}) - \boldsymbol{N}(\boldsymbol{x})$ is obtained as follows: Using p processors, we decompose our $N \times N$-grid as evenly as possible into p non-intersecting blocks of consecutive rows. Each processor is then assigned one of these blocks of rows, which means that it has to

[1] Access to the CRAY was through the HLRZ at the Kernforschungsanlage Jülich, Germany. The Sequent was available at the ACRF at Argonne National Laboratory, USA. We thank both institutions for their support.

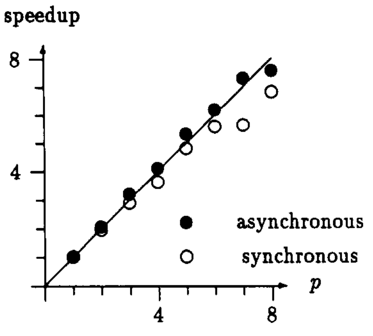

Fig. 1. Speedup on Cray, first example, $N = 30$.

perform updates for all grid points lying in the corresponding block. All grid points are ordered according to the well-known red–black ordering scheme. In the synchronous case we then use the familiar red–black Gauss–Seidel splitting $F'(x) = M(x) - N(x)$, i.e., $M(x)$ is the lower triangular part of $F'(x)$ (assuming that the variables and the components of f are numbered via the red–black-ordering). In pseudocode for one processor, the resulting iteration is then given as follows.

```
repeat
  for $i \in$ myrows
    for $j = 1, \ldots, N$
      if $(i, j)$ is *red* then
        $x_{i,j} = m_{i,j} - (-x_{i-1,j} - x_{i+1,j} - x_{i,j-1} - x_{i,j+1} + 4m_{i,j} + \phi(m_{i,j}))/(4x_{i,j} + \phi(x_{i,j}))$

  synchronize
  for $i \in$ myrows
    for $j = 1, \ldots, N$
      if $(i, j)$ is *black* then
        $x_{i,j} = m_{i,j} - (-x_{i-1,j} - x_{i+1,j} - x_{i,j-1} - x_{i,j+1} + 4m_{i,j} + \phi(m_{i,j}))/(4x_{i,j} + \phi(x_{i,j}))$
  synchronize
```

Here, myrows contains the rows a processor is assigned to. The updating scheme given for $x_{i,j}$ is just an explicit formula for the (i, j)-component of the Newton-like operator $h(x) = m - \text{LES}(M(x), N(x)(m - x) + f(m))$. (Since $M(x)$ is lower triangular we took LES to be the familiar forward substitution process. We also used the fact that several terms involving $m_{i,j}$ cancel when calculating $N(x)(m - x) + f(m)$.)

In the synchronous case we thus have to synchronize twice in each iteration, once after the updating of the grid points of each of the two colours. In the asynchronous case, these synchronizations are left out.

Figs. 1 and 2 report the speedups obtained for the synchronous and asynchronous implementations on the Cray and the Sequent, respectively. Here, the speedup for p processors is defined as the ratio of the wall clock time of the synchronous algorithm on one processor to the wall clock time of

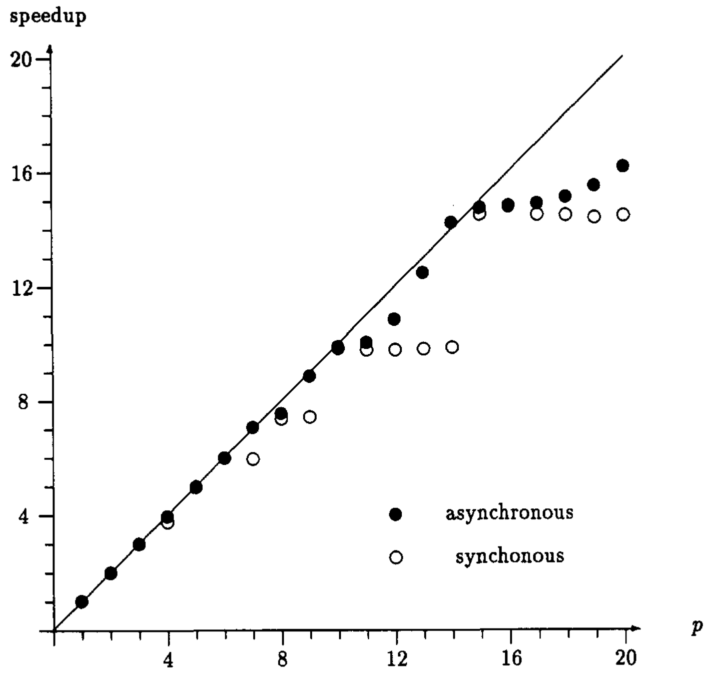

Fig. 2. Speedup on Sequent, first example, $N = 30$.

Table 1
Number of iterations on Cray, first example

p	1	2	3	4	5	6	7	8
No. iter., sync.	1226	1226	1226	1226	1226	1226	1226	1226
Max. no. iter., async.	1226	1278	1251	1356	1251	1254	1328	1587
Min. no. iter., async.	1226	1185	1162	1093	1139	1155	966	1167

the respective algorithm on p processors. In our calculations we took $N = 30$. The iteration was stopped when the width of each component of the calculated enclosure for x^* was less than 10^{-6}. All results concerning the Cray were obtained on a dedicated machine. On the Sequent we could use up to 20 processors simultaneously.

Both figures show that the asynchronous method performs better than its synchronous counterpart. We even sometimes ($p = 3, \ldots, 7$ on the Cray, $p = 14$ on the Sequent) observe a speedup greater than p for p processors. A look at Table 1 partly explains this fact for the Cray by showing that in these cases some processors (those which have the most work to do per iteration) need less iterations than in the synchronous case.

Also note, for example, that there is no gain in the speedup on the Sequent for the synchronous iteration in the range $p = 10, \ldots, 14$. This is due to the fact that the maximum number of rows a processor has to work on is constant (and equals 3) for any p in that range. This rather drastically illustrates the (trivial) fact that the overall execution time of the synchronous method is determined by the slowest processor. In contrast to this, the speedup of the asynchronous method grows strictly monotonically with p and is quite satisfactorily close to p. The same may be observed for $p = 16,\ldots,20$, although now the asynchronous method — still better than the synchronous method — performs somewhat less well.

In our second example we consider the same Dirichlet problem (23) as before. The assignment of the grid point to the processors is also identical to the first example, but we now take a different decomposition $F'(x) = M(x) - N(x)$ from the one used before. More precisely, variables are now ordered by the zebra-ordering, i.e., *whole* rows of grid points are given alternatively red and black colours. With this ordering $F'(x)$ is a 2×2 block matrix, where both diagonal blocks are block diagonal matrices, each block of which is tridiagonal of order N and corresponds to one row of grid points. In our computation we now take $M(x)$ to be the block diagonal part of $F'(x)$. This means that in each iteration each processor has to "solve" as many tridiagonal systems of order N as it has rows assigned to it. For solving these tridiagonal systems we used the interval Gaussian algorithm (IGA). In the synchronous case we synchronized processors every time all points of one colour had been updated, so that the overall process resulted in a line Gauss–Seidel method.

As in our first example, let us describe the resulting iteration is pseudocode. We need some additional notation: x_i will describe a *whole row* of x, i.e., $x_i = (x_{i,j})_{j=1,\ldots,N}$, and similarly for m_i and $\phi(m_i) = (\phi(m_{i,j}))_{j=1,\ldots,N}$. Moreover, let $T_i(x)$ denote the tridiagonal matrix with -1 everywhere on its super- and subdiagonal and with its jth diagonal entry equal to $4x_{i,j} + \phi(x_{i,j})$. Finally, let z_i denote the N-vector with all zero entries but its first and last component which is $x_{i,0}$ ($=1$) and $x_{i,N+1}$ ($=1$), respectively. The synchronous iteration can then be written as

```
repeat
  for $i \in$ myrows
    if row $i$ is *red* then
      $x_i = m_i - \mathrm{IGA}(T_i(x), -x_{i-1} - x_{i+1} - z_i + 4m_i + \phi(m_i))$
  synchronize
  for $i \in$ myrows
    if row $i$ is *black* then
    $x_i = m_i - \mathrm{IGA}(T_i(x), -x_{i-1} - x_{i+1} - z_i + 4m_i + \phi(m_i))$
  synchronize
```

As in our first example, the asynchronous variant is obtained by simply dropping the two synchronizations in each iteration.

Due to results in [26], we can expect this method to require less iterations to achieve a given accuracy than the Newton–Gauss–Seidel method of the first example. On the other hand, one iteration is now more costly, since we have to solve tridiagonal systems. So, already in the synchronous case, it seems quite difficult to theoretically predict the actual execution time of this algorithm as compared to the Newton–Gauss–Seidel method. Numerical experiments with $N = 40$ with the synchronous algorithm on one processor showed that the line Gauss–Seidel algorithm required 1075 iterations instead of 2828 for plain Newton–Gauss–Seidel to achieve an accuracy of 10^{-6}. The execution time, however, was approximately 10% *more*. So the gain in the

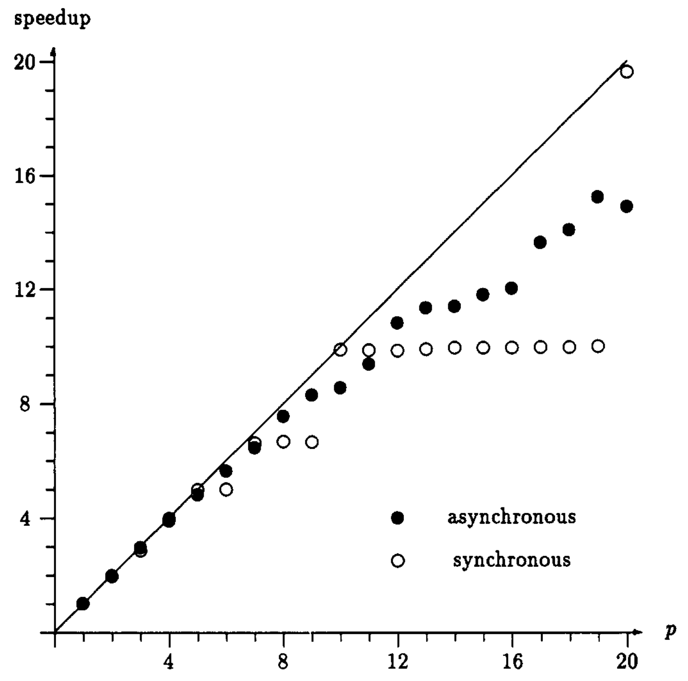

Fig. 3. Results on Sequent, second example, $N = 40$.

number of iterations is not sufficient to balance the additional work necessary to factor the tridiagonal matrix $F'(x)$ in the line Gauss–Seidel process.

Fig. 3 reports the speedup we obtained on the Sequent for the choice $N = 40$, the stopping criterion is as in the first example.

The results are similar to our first example. It is interesting to see that the synchronous method yields no gain in speedup within the whole range from $p = 10, \ldots, 19$. Within this range, there is always one processor holding two red rows and another holding two black rows, so their computational speed is governing the whole process. However, we also see that in the case where p divides N, particularly $p = 10$ and $p = 20$, where the working load is very well balanced, the synchronous method works better.

Our final example deals with a three-dimensional Dirichlet problem which is just Eq. (23) on the unit cube instead of the unit square. The discretization was done via the standard seven-point difference star on a mesh of $20 \times 20 \times 20$ equidistant grid points. We thus end up with a linear system of size 8000. The distribution of the grid points to the different processors was done by *slices*, i.e., each processor holds a $20 \times 20 \times m$ subgrid with $m \simeq 20/p$. As in our second example we used a zebra red black ordering to obtain a line Gauss–Seidel splitting of $F'(x)$ to be used in the Newton-like operator (17). Fig. 4 shows the speedup obtained on the Cray for this quite large problem. In order not to waste too much time on the dedicated machine we now modified our stopping criterion to check the width of all components of the calculated inclusion vector being less than 10^{-4}.

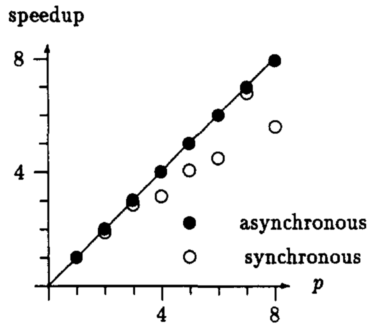

Fig. 4. Speedup on Cray, third-example, $N = 20$.

Again, the asynchronous method often works considerably better than the synchronous one and achieves a speedup very close to p on the whole range $p = 1, \ldots, 8$.

In concluding we remark that in all three examples the working load (i.e., the work required for one iteration on one processor) gets increasingly worse balanced as the number of processor increases. The reported experiments show that the asynchronous methods deal fairly well with this situation and achieve good (and sometimes optimal) speedups. The synchronous iterations, of course, can heavily suffer from load imbalancies.

We also ran our examples on the Cray in nondedicated mode. Surprisingly, the asynchronous methods then achieve an even better performance when compared to the synchronous ones. This can be explained by the fact that in multiuser mode the operating system on the Cray achieves a better throughput for the asynchronous methods which are broken up into totally independent parts.

References

[1] G. Alefeld and J. Herzberger, *Introduction to Interval Computations* (Academic Press, New York, 1983).
[2] G. Alefeld and L. Platzöder, A quadratically convergent Krawczyk-like algorithm, *SIAM J. Numer. Anal.* **20** (1983) 210–219.
[3] G. Baudet, Asynchronous iterative methods for multiprocessors, *J. ACM* **25** (1978) 226–244.
[4] R. Barlow and D. Evans, Synchronous and asynchronous iterative parallel algorithms for linear systems, *Comput. J.* **25** (1982) 56–60.
[5] D. Bertsekas, Distributed asynchronous computation of fixed points, *Math. Programming* **27** (1983) 107–120.
[6] D. Bertsekas and J. Tsitsiklis, *Parallel and Distributed Computation* (Prentice-Hall, Englewood Cliffs, NJ, 1989).
[7] R. Bru, L. Elsner and M. Neumann, Models of parallel chaotic relaxation methods, *Linear Algebra Appl.* **103** (1988) 175–192.
[8] D. Chazan and W. Miranker, Chaotic relaxation, *Linear Algebra Appl.* **2** (1969) 199–222.
[9] J. Deminet, Experience with multiprocessor algorithms, *IEEE Trans. Comput.* **C-31** (1982) 278–288.
[10] M. Dubois and F. Briggs, Performance of synchronized iterative processes in multiprocessor systems, *IEEE Trans. Software Engrg.* **SE-8** (1982) 419–431.
[11] D. El Baz, *M*-Functions and parallel asynchronous algorithms, *SIAM J. Numer. Anal.* **27** (1990) 136–140.
[12] M. El Tarazi, Some convergence results for asynchronous algorithms, *Numer. Math.* **39** (1982) 325–340.
[13] M. El Tarazi, Algorithmes mixtes asynchrones. Etude de convergence monotone, *Numer. Math.* **44** (1984) 363–369.

[14] L. Elsner, I. Koltracht and M. Neumann, On the convergence of asynchronous paracontractions with application to tomographic reconstruction from incomplete data, *Linear Algebra Appl.* **130** (1990) 65–82.

[15] L. Elsner, L. Koltracht and M. Neumann, Convergence of sequential and asynchronous nonlinear paracontractions, *Numer. Math.* **62** (1992) 305–319.

[16] D. Evans, Parallel S.O.R. iteraitve methods, *Parallel Comput.* **1** (1984) 3–18.

[17] A. Frommer, *Lösung Linearer Gleichungssysteme auf Parallelrechnern* (Vieweg, Wiesbaden, 1990).

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

[19] A. Frommer, Asynchronous iterations in partially ordered spaces, *Numer. Funct. Anal. Optim.* **12** (1991) 315–325.

[20] A. Frommer, Asynchronous iterations for enclosing solutions of fixed point problems, in: L. Atanassova and J. Herzberger, Eds., *Computer Arithmetic and Enclosure Methods* (Elsevier, Amsterdam, 1992).

[21] A. Frommer and G. Mayer, On the R-order of Newton-like methods for enclosing solutions of nonlinear equations, *SIAM J. Numer. Anal.* **27** (1990) 105–116.

[22] A. Frommer and D. Szyld, Asynchronous two-stage iterative methods, *Numer. Math.* **69** (1994) 141–154.

[23] R. Krawczyk, Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken, *Computing* **4** (1969) 187–201.

[24] R. Krawczyk and A. Neumaier, Interval slopes for rational functions and associated centered forms, *SIAM J. Numer. Anal.* **22** (1985) 604–616.

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

[26] G. Mayer, Comparison theorems for iterative methods based on strong splittings, *SIAM J. Numer. Anal.* **24** (1987) 215–227.

[27] G. Mayer, Epsilon-inflation in verification algorithms, *J. Comput. Appl. Math.* **60** (1–2) (1995) 147–169 (this issue).

[28] J.-C. Miellou, Itérations chaotiques à retards, *C. R. Acad. Sci. Paris. Ser.* **A 278** (1974) 957–960.

[29] J.-C. Miellou, Algorithmes de relaxation chaotique à retards, *Revue d'Automatiques, Informatiques et Recherche Opérationelle* **9 R-1** (1975) 55–82.

[30] J.-C. Miellou, Itérations chaotiques à retards; études de la convergence dans le cas d'espaces partiellement ordonnés, *C. R. Acad. Sci. Paris. Ser.* **A 280** (1975) 233–236.

[31] A. Neumaier, *Interval Methods for Systems of Equations* (Cambridge University Press, Cambridge, 1990).

[32] J. Ortega and W. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables* (Academic Press, New York, 1970).

[33] F. Robert, M. Charnay and F. Musy, Itérations chaotiques série-parallèle pour des équations non-linéaires de point fixe, *Apl. Mat.* **20** (1975) 1–38.

[34] H. Schwandt, The solution of nonlinear elliptic Dirichlet problems on rectangles by almost globally convergent interval methods, *SIAM J. Sci. Statist. Comput.* **6** (1985) 617–638.

[35] H. Schwandt, Krawczyk-like algorithms for the solution of systems of nonlinear equations, *SIAM J. Numer. Anal.* **22** (1985) 792–810.

[36] H. Schwandt, Interval arithmetic methods for systems of nonlinear equations arising from discretizations of quasilinear elliptic and parabolic partial differential equations, *Appl. Numer. Math.* **3** (1987) 257–287.

[37] H. Schwandt, Interval arithmetic multistep methods for nonlinear systems of equations, *Japan J. Appl. Math.* **4** (1987) 139–171.

[38] H. Schwandt, iv2ftn — an interval arithmetic preprocessor for UNIX systems, in preparation

[39] J. Shao and L. Kang, An asynchronous parallel mixed algorithm for linear and nonlinear equations, *Parallel Comput.* **5** (1987) 313–321.

[40] A. Üresin and M. Dubois, Sufficient conditions for the convergence of asynchronous iterations, *Parallel Comput.* **10** (1989) 83-92.

[41] R. Varga, *Matrix Iterative Analysis* (Prentice-Hall, Englewood Cliffs, NJ, 1962).