E. Lyager

all extremal
ı. The algo-
. determines
ases).
a hypercube
mely, linear
of the func-
;ed (i.e., has

.s.

Derivation
ıtations. In-

llstellen mit

ıction of n
ı. 1–15.
ıge of func-

14 (1974),

Science
ı (DIKU)

Interval Computations
No 3, 1994

A Parallel Method for Linear Interval Equations

Kaj Madsen and Ole Toft

The sign-accord algorithm of Rohn for solving systems of linear interval equations is considered. This method is motivated by a theoretical analysis of the problem. A parallel version of the method is developed and tested using rounded interval arithmetic. Theoretical considerations as well as numerical tests demonstrate that the parallel method is efficient (i.e. the speedup is almost linear) if an appropriate number of parallel processors is allocated. This number can be estimated before the beginning of the (parallel) computational process.

Параллельный метод для линейных интервальных уравнений

К. Мадсен, О. Тофт

Рассматривается знакосогласованный алгоритм Рона для решения систем линейных уравнений, обоснованный теоретическим анализом задачи. Параллельная версия метода разработана и протестирована с использованием интервальной арифметики с округлением. Как теоретические соображения, так и численные примеры показывают, что параллельный метод эффективен (т. е. увеличение быстродействия почти линейно), если выделено нужное число параллельных процессоров. Это число можно оценить до начала (параллельного) вычислительного процесса.

1 Introduction

The problem of solving a linear system of equations with inaccurate coefficients and inaccurate values of the right-hand side occurs in many technical and economical situations. Let us give a few examples.

- Measuring instruments cannot be absolutely accurate, so all values that we get from measurements are measured within some (known) uncertainty. Therefore, if the measured value is $\tilde{x}$, and the guaranteed accuracy of this measuring instrument is Δ, then the only thing that we know about the actual value x of the measured quantity is that x belongs to an interval $[\tilde{x} - \Delta, \tilde{x} + \Delta]$.
- In economic situations, linear equations often describe the relationship between prices, rates, etc. However, at any given moment of time, prices, rates, etc., are not fixed; they can vary a little bit (depending on the place or on the specific bargain). So, for each variable, we have an *interval* of possible values.
- In many real-life situations, we have a *non-linear* dependency between the quantities. However, one of the main ideas of solving nonlinear equations is to reduce them to a sequence of easier-to-solve linear ones. So, many numerical methods that deal with interval uncertainty (e.g., an interval version of Newton-Raphson's iteration) have solution of a linear system as a subproblem.

To determine exact bounds on the solution set of a system of linear equations can be quite costly. In fact, for known methods, the amount of computations needed may increase exponentially in the square of the number of variables (for a detailed description, see Section 2). Therefore it makes sense to use parallel computing in order to reduce the calculation time.

In this paper we consider the *sign-accord algorithm* of Rohn [15]. We extend the method by using rounded interval arithmetic, and we show how to parallelize the resulting algorithm. We provide arguments that in many situations, our algorithm will have a nearly linear speedup. These arguments are confirmed through extensive numerical testing. In this paper, we present only a few test results. For a more complete description of numerical experiments, the reader is referred to Toft [16] ([16] also describes the results of extensive testing of the sequential sign-accord algorithm).

In Section 2 we shortly review Rohn's method, and describe how to modify it so that it would include rounded interval arithmetic. Section 3 describes and analyzes the parallel strategy, and in Section 4 several typical numerical test results are presented. Finally, some concluding remarks are given in Section 5.

2 The sequential sign-accord algorithm

In this section we first give a precise formulation of the problem to be solved, then we describe the sign-accord algorithm, and finally we demonstrate how to extend this method to rounded interval arithmetic.

2.1 Special notations

We assume that the reader is familiar with Interval Analysis and traditional notations (a complete description is given, e.g., in Toft [16]). In this subsection, we will only describe a few special denotations.

The *midpoint* a_c of an interval $a^I = [\underline{a}, \overline{a}]$ is the center element defined as

$$a_c := \frac{1}{2}(\underline{a} + \overline{a}).$$

The *radius* $\mathrm{rad}(a^I)$ of a^I is

$$\mathrm{rad}(a^I) := \frac{1}{2}(\overline{a} - \underline{a}).$$

This definition is generalized to interval vectors and matrices by taking midpoints and radii componentwise.

For an interval vector b^I the *set of corner vectors* $\mathrm{Corn}(b^I)$ is defined as

$$\mathrm{Corn}(b^I) := \left\{ b \in \Re^n \;\middle|\; b_i \in \{\underline{b}_i,\ \overline{b}_i\},\ i = 1, \ldots, n \right\} \tag{1}$$

(here, $\Re^n$ denotes a set of all n-dimensional vectors $b = (b_1, \ldots, b_n)$ whose components b_i are real numbers).

For an interval matrix A^I the *set of corner matrices*, $\mathrm{Corn}(A^I)$, is defined similarly.

An n-dimensional *sign vector* is a vector having components with values 1 or -1. The set of all such vectors is denoted by $\mathcal{S}^n$:

$$\mathcal{S}^n := \left\{ y \in \Re^n \;\middle|\; y_i \in \{1,\ -1\},\ i = 1, \ldots, n \right\}.$$

If for two vectors, $x \in \Re^n$, $y \in \mathcal{S}^n$, the predicate

$$signaccord(x, y) := \ \forall i \in \{1, \ldots, n\} :\ x_i y_i \geq 0 \tag{2}$$

is true then we say that the *signs* of x's and y's are in *accord.*

$D_y = \mathrm{diag}(y_1, \ldots, y_n)$ denotes a diagonal matrix with vector y in the diagonal.

The *interval hull* $\Box\mathcal{U}$ of a nonempty compact subset $\mathcal{U}$ of $\Re^n$ is defined as the "tightest" interval vector containing the set:

$$\Box\mathcal{U} := [\min\ \mathcal{U}, \max\ \mathcal{U}] \tag{3}$$

where min and max are to be understood componentwise.

2.2 The solution set

We wish to solve the linear system of interval equations

$$A^I x = b^I \tag{4}$$

where $A^I \in \Im^{n\times n}$ and $b^I \in \Im^n$. The *solution set* is defined as

$$\mathcal{X} = \left\{ x \in \Re^n \;\middle|\; Ax = b,\ A \in A^I,\ b \in b^I \right\}. \tag{5}$$

When A^I is regular (i.e., when all matrices $A \in A^I$ are regular), then the solution set is bounded (since both A^I and b^I are bounded).

We limit our goal to finding the interval hull $\Box\mathcal{X}$ of $\mathcal{X}$. This is also called the *interval solution* $x^I = [\underline{x}, \overline{x}]$ of (4):

$$x^I := \Box\mathcal{X}. \tag{6}$$

It is the "tightest" interval vector enclosing the solution set.

Fi

We first
itself, or the
a *finite* algo
$\Box\mathcal{X}$ in finitel

Oettli ar
of the soluti

Theorem 1
is

(Here, $\Im^n$ *d*

This the
longs to the
has a very
be generally
approaches

Example 1
interval solu

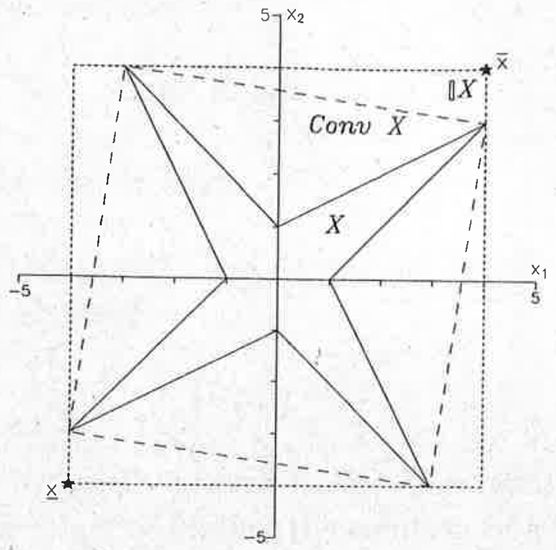

Figure 1: Example of solution set $\mathcal{X}$, Conv $\mathcal{X}$, and $\Vert\mathcal{X}$

We first give five theorems which characterise either the solution set itself, or the interval solution. These theorems lead to the construction of a *finite* algorithm (i.e., an algorithm that finds the precise expression for $\Vert\mathcal{X}$ in finitely many computational steps).

Oettli and Prager [14] provided a surprisingly simple characterization of the solution set.

Theorem 1. *Let $A^I \in \Im^{n\times n}$, $b^I \in \Im^n$. Then the solution set of $A^I x = b^I$ is*

$$\mathcal{X} = \big\{x \in \Re^n \;\big|\; |A_c x - b_c| \le \operatorname{rad}(A^I)|x| + \operatorname{rad}(b^I)\big\}. \tag{7}$$

(Here, $\Im^n$ denotes the set of all n-dimensional interval vectors.)

This theorem offers a simple way of testing whether a given point belongs to the solution set. It also expresses the fact that the solution set has a very complicated structure, since the term $|x|$ in (7) causes $\mathcal{X}$ to be generally nonconvex. Consequently, conventional linear optimization approaches cannot be used to find bounds on $\mathcal{X}$.

Example 1. Figure 1 shows the solution set, its convex hull, and the interval solution of (4) in the following example from [3]:

$$A^I = \begin{pmatrix} [2,4] & [-2,1] \\ [-1,2] & [2,4] \end{pmatrix},\ b^I = \begin{pmatrix} [-2,2] \\ [-2,2] \end{pmatrix}.$$

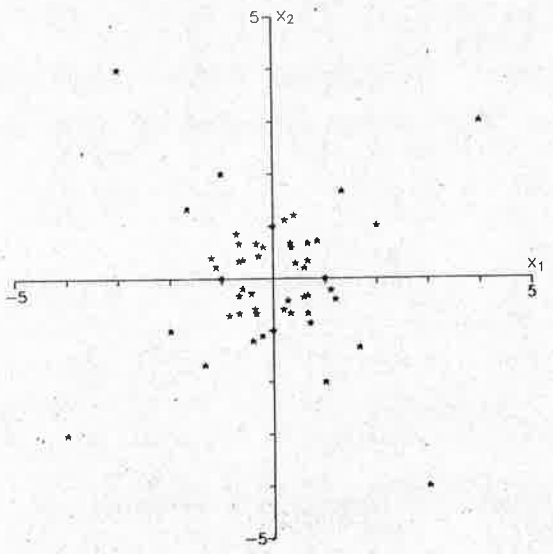

Figure 2: The set of corner solutions $\mathcal{X}_{\text{Corn}}$ corresponding to Example 1

The interval solution is

$$x^I = [\underline{x}, \overline{x}] = \|\mathcal{X} = \begin{pmatrix} [-4, 4] \\ [-4, 4] \end{pmatrix}.$$

In Figure 1 we notice the inclusions

$$\mathcal{X} \subseteq \text{Conv } \mathcal{X} \subseteq \|\mathcal{X},$$

which of course hold in general.

Next, consider all real systems that can be formed by choosing the coefficient matrix from the set of corner matrices of A^I, and the right-hand side from the set of corner vectors of b^I. We define the *set of corner solutions* $\mathcal{X}_{\text{Corn}}$ as

$$\mathcal{X}_{\text{Corn}} := \left\{ x \in \Re^n \;\middle|\; Ax = b,\ A \in \text{Corn}(A^I),\ b \in \text{Corn}(b^I) \right\}. \tag{8}$$

Clearly this is a finite subset of the solution set $\mathcal{X}$.

Figure 2 shows the set of corner solutions of Example 1. Comparing Figure 1 and Figure 2 we see that the corner solutions span the convex hull of the solution set. This natural result is generalised in the following theorem which is proved in Toft [16].

Theorem
solutions

As a simp

Theorem
solutions

This Theo
it states t
equations
algorithm
solutions
find all co
Since each
be $O(n^3 2^n$

The fo
costly met

2.3 E

It turns ou
into equal
extreme s

It can be s

one can sh
2^n points)

$\mathcal{X}_{\text{Extr}} =$

Theorem 2. *Let $A^I \in \Im^{n\times n}$ be regular, $b^I \in \Im^n$. Then the set of corner solutions of $A^I x = b^I$ satisfies*

$$\operatorname{Conv} \mathcal{X}_{\text{Corn}} = \operatorname{Conv} \mathcal{X}. \tag{9}$$

As a simple consequence we have:

Theorem 3. *Let $A^I \in \Im^{n\times n}$ be regular, $b^I \in \Im^n$. Then set of corner solutions of $A^I x = b^I$ satisfies*

$$\square\mathcal{X}_{\text{Corn}} = \square\mathcal{X}. \tag{10}$$

This Theorem was originally proved in [5]. Theorem 3 is important, since it states that the problem of finding the solution of a set of linear interval equations is a *finite* problem (in the sense that it can be solved by a *finite* algorithm). All we need to do to describe $\square\mathcal{X}$ is to compute the corner solutions and then find the interval hull of these. However, in order to find all corner solutions we may have to solve a total of 2^{n^2+n} systems. Since each system costs $O(n^3)$ the total complexity of this method would be $O(n^3 2^{n^2+n})$.

The following subsections show that we can do much better than this costly method.

2.3 Extreme solutions

It turns out that points x for which Oettli and Prager's inequality (7) turn into equality have an interesting property. Let us call such a point an *extreme solution*, and let $\mathcal{X}_{\text{Extr}}$ denote the *set of extreme solutions*:

$$\mathcal{X}_{\text{Extr}} := \{x \in \Re^n \mid |A_c x - b_c| = \operatorname{rad}(A^I)|x| + \operatorname{rad}(b^I)\}. \tag{11}$$

It can be shown that $\mathcal{X}_{\text{Extr}}$ is a subset of $\mathcal{X}_{\text{Corn}}$. Introducing the sign vector

$$y := \operatorname{sgn}(A_c x - b_c)$$

one can show that $\mathcal{X}_{\text{Extr}}$ can be described as the following set (with at most 2^n points):

$$\mathcal{X}_{\text{Extr}} = \{x \in \Re^n \mid A_c x - b_c = D_y(\operatorname{rad}(A^I)|x| + \operatorname{rad}(b^I)),\ y \in \mathcal{S}^n\}. \tag{12}$$

```
Step 1  $x^I := \emptyset$
        for each $y \in \mathcal{S}^n$ do
Step 2      find the extreme solution $x_y$
Step 3      $x^I := \square(x^I \cup \{x_y\})$
```

Figure 3: Algorithm for computing x^I using extreme solutions

Using the next two theorems of Rohn [15], we can then reduce our problem to finding these points.

Theorem 4. *Let $A^I \in \Im^{n\times n}$ be regular, $b^I \in \Im^n$. For each $y \in \mathcal{S}^n$ the nonlinear equation in (12) has exactly one extreme solution $x = x_y$, and*

$$\text{Conv}\ \mathcal{X}_{\text{Extr}} = \text{Conv}\ \{x_y \mid y \in \mathcal{S}^n\} = \text{Conv}\ \mathcal{X}. \tag{13}$$

Similarly to the case of corner solutions, we can easily deduce the following simple consequence from this Theorem.

Theorem 5. *Let $A^I \in \Im^{n\times n}$ be regular, $b^I \in \Im^n$. Then the set $\mathcal{X}_{\text{Extr}}$ of extreme solutions for $A^I x = b^I$ satisfies*

$$\square\mathcal{X}_{\text{Extr}} = \square\mathcal{X}. \tag{14}$$

Theorem 5 motivates the algorithm shown in Figure 3. Since the set $\mathcal{X}_{\text{Extr}}$ contains at most 2^n elements, the algorithm appears less costly than the algorithm using all corner solutions (assuming, of course, that we know a relatively simple way to find an extreme solution).

2.4 Finding an extreme solution

In order to find the extreme solution x_y for a given $y \in \mathcal{S}^n$ we must solve the equation

$$A_c x - b_c = D_y\big(\text{rad}(A^I)|x| + \text{rad}(b^I)\big).$$

```
Step 1  $z = \mathrm{sgn}(A_c{}^{-1} b_y)$ (recommended)
        repeat
Step 2      solve $A_{yz} x = b_y$
Step 3      if $signaccord(x, z)$ then
                $x_y := x$
            else
                $k := \min\{j \in \{1, \ldots, n\} \mid z_j x_j < 0\}$
                $z_k := -\, z_k$
        until $signaccord(x, z)$
```

Figure 4: The sign-accord algorithm for finding the extreme solution x_y for a given y

Let's introduce another sign vector $z := \mathrm{sgn}\, x \in \mathcal{S}^n$. In terms of z, $|x| = D_z x$, so the above equation can be rewritten as

$$\big(A_c - D_y \mathrm{rad}(A^I) D_z\big) x = b_c + D_y \mathrm{rad}(b^I).$$

If we denote the matrix of this system by A_{yz} and the right hand side by b_y, then we can further simplify the system. Namely, x is an extreme solution that corresponds to y if and only if z and x satisfy the following two conditions:

$$\begin{gathered} A_{yz} x = b_y \\ signaccord(x, z),\ z \in \mathcal{S}^n. \end{gathered} \tag{15}$$

(the second condition is equivalent to the constraint $|x| = D_z x$). Thus the problem is reduced to finding the correct sign vector z.

Theorem 4 expresses the fact that for each y there exists exactly one $x = x_y$ satisfying the two conditions in (15). To find this point x_y, Rohn [15] proposes the *sign-accord algorithm* shown in Figure 4.

The starting value of z in Step 1 is a recommended value. However, other starting values can be used, because Rohn [15] has proved that if A^I is regular then the algorithm is finite for an arbitrary starting value of z. For each starting value, at most 2^n steps of type Step 2 will be undertaken. Experiments show that for the chosen starting value, the average computation time is smaller than for other possible values. Thus, the recommended starting value is an important part of Rohn's algorithm.

In the best case where only one computation step per extreme solution is taken, the cost of the algorithm in Figure 3 is $O(n^3 2^n)$ operations. In the worst case it is $O(n^3 2^{2n})$.

2.5 Reducing the number of extreme solutions

The problem is now reduced to computing 2^n vectors that satisfy the conditions (15). It is, however, often possible to reduce this number even further by exploiting information about the inverse interval matrix.

The reduction is based on the following result of Rohn [15].

Theorem 6. *Let $A^I \in \Im^{n\times n}$ be regular, $b^I \in \Im^n$. Then for each i, $1 \leq i \leq n$, the following two properties are true for the interval solution $[\underline{x}, \overline{x}]$ of $A^I x = b^I$:*

1. *There exist sign vectors $y, z \in \mathcal{S}^n$ and an extreme solution x_y, for which $(x_y)_i = \underline{x}_i$, signaccord$(x_y, z)$, and*

$$(A_{yz}^{-1})_{ij} y_j \leq 0,\ j = 1, \ldots, n. \tag{16}$$

2. *There exist sign vectors $y, z \in \mathcal{S}^n$ and an extreme solution x_y, for which $(x_y)_i = \overline{x}_i$, signaccord(x_y, z), and*

$$(A_{yz}^{-1})_{ij} y_j \geq 0,\ j = 1, \ldots, n. \tag{17}$$

In many real-life situations, the signs of some components of the inverse interval matrix $(A^I)^{-1}$ are constant (i.e., either for every $A \in A^I$ the (i,j) component B_{ij} of the inverse matrix $B = A^{-1}$ is positive, or for every $A \in A^I$, $B_{ij} < 0$). Such components of $(A^I)^{-1}$ will be called *sign stable*. If a matrix $(A^I)^{-1}$ has sign-stable components, then it is possible to reduce the number of extreme solutions x_y which need to be considered. If, for instance, $(A^I)^{-1}_{ij}$ is sign stable, then the value of y_j needed to find $\underline{x}_i$ or $\overline{x}_i$ is given by the corresponding inequality of Theorem 6. Thus, if $(A^I)^{-1}_{ij}$ is sign stable for p different values of j, then the number of sign vectors y

(and hence t
order to finc
of the invers
number of e
description c

In summ
ities of Thec

In practice,
matrix. Hc
Rohn [15] s
A^I i.e. assu

where $C =$

Rohn [1
test.

Theorem

where $C =$

Then al

2.6 Fi

In summar
found usin

2.7 Co

To assure t
achieved o

(and hence the number of extreme solutions that we need to compute in order to find $\underline{x}_i$ or $\overline{x}_i$) is reduced to 2^{n-p}. In many cases, all components of the inverse interval matrix $(A^I)^{-1}$ are sign stable. In such cases, the number of extreme solutions needed for finding x^I is $2n$. For a detailed description of the selection procedure we refer to Toft [16].

In summary, if $\mathcal{Y}$ is the set of sign vectors y selected using the inequalities of Theorem 6 then the interval solution is determined by

$$x^I = [\![\{x_y \mid y \in \mathcal{Y}\}.$$

In practice, of course, we do not know the signs of the inverse interval matrix. However, an enclosure may provide the necessary information. Rohn [15] suggests the following enclosure (assuming *strong regularity* of A^I i.e. assuming that the spectral radius ρ of $|A_c^{-1}|\text{rad}(A^I)$ is less than 1):

$$\underline{B} = A_c^{-1} - C|A_c^{-1}|, \tag{18}$$

$$\overline{B} = A_c^{-1} + C|A_c^{-1}| \tag{19}$$

where $C = R(I-R)^{-1}$, $R = |A_c^{-1}|\text{rad}(A^I)$, and I is the identity matrix.

Rohn [15] provides the following sufficient criterion which is easy to test.

Theorem 7. *Let $A^I \in \Im^{n\times n}$ be strongly regular, and let*

$$C|A_c{}^{-1}| \leq |A_c{}^{-1}| \tag{20}$$

where $C = R(I-R)^{-1}$, $R = |A_c^{-1}|\text{rad}(A^I)$.

Then all components of $(A^I)^{-1}$ are sign stable.

2.6 Final theoretical algorithm

In summary, the final algorithm is given by Figures 4 and 5, where $\mathcal{Y}$ is found using Theorem 6, (18), (19), and, probably, also (20).

2.7 Controlling rounding errors

To assure that the correct mathematical results are contained in the results achieved on a computer, all calculations must be performed using *rounded*

```
Step 1  find the set $\mathcal{Y}$ of selected sign vectors
        $x^I := \emptyset$
        **for** each $y \in \mathcal{Y}$ **do**
Step 2      find the extreme solution $x_y$
Step 3      $x^I := \Box(x^I \cup \{x_y\})$
```

Figure 5: Algorithm for computing x^I using a reduced number of extreme solutions

interval arithmetic (in which the result of applying an arithmetic operation to two intervals takes into consideration additional rounding error). Since we want to implement our algorithm in rounded interval arithmetic, we must thus modify each step of the above-described algorithm accordingly. Let's describe how we modify these steps.

- *Computation of* A_c^{-1}: By finding A_c using rounded interval arithmetic and then applying an inversion procedure using interval Gaussian elimination we obtain an interval matrix $fl(A_c^{-1})$. Preconditioning with the midpoint inverse is used.

- *Solution of* $A_{yz}x = b_y$: Using interval Gaussian elimination with partial pivoting we find an interval vector $fl(x)$. Again preconditioning with the midpoint inverse is used. The result of the sign-accord algorithm is then an interval vector $fl(x_y)$.

- *Verification of* $\rho\big(|A_c^{-1}|\text{rad}(A^I)\big) < 1$: This criterion is used in the procedure for reducing the number of extreme solutions. Here an interval method for finding eigenvalues could be applied. However, we are not aware of any available implementation. Therefore we have been experimenting with different scaled infinity norms providing upper bounds for the spectral radius. Unfortunately such methods seem to be rather pessimistic in practical cases.

  As a result, in our current algorithm, we simply compute the spectral radius using floating point arithmetic. Therefore, there is no interval guarantee at this point in our present implementation.

- *Determining the predicate $signaccord(x, z)$*: x is found using rounded interval arithmetic, leading to an including interval $fl(x)$. If this interval vector has one or more components that contain 0 as an interior point, then how do we decide whether the sign-accord criterion is satisfied? We have not been able to find a satisfactory answer to this question; therefore we use a procedure where the predicate tested is

  $$\exists x \in fl(x) : signaccord(x, z). \tag{21}$$

  This method is safe when the computed solutions are sign stable, and it can find sign unstable solutions. If, however, a sign unstable interval vector is accepted as the extreme solution then we have no guarantee that it is the solution, and a warning is given to the user.

Thus there are two unresolved problems in our implementation. The problem of including the spectral radius of $|A_c^{-1}|\mathrm{rad}(A^I)$ can be resolved using an interval method for eigenvalues. The other problem, the usage of the sign-accord algorithm in cases where the solutions $fl(x)$ may have zero (or "almost zero") components, is, however, harder to solve. There is no guarantee that the algorithm stops at the correct point, and apparently this problem has not been addressed in the literature. Further research is needed at this point.

3 The parallel algorithm

It follows from the previous section that finding exact bounds on the solution set of a system of linear interval equations can be a rather costly affair. Fortunately the sign-accord algorithm is suitable for parallel computing, which gives hope of a reduction of the time required to solve a given problem. The parallel computer we have in mind is of the type Multiple Instruction–Multiple Data (MIMD), see Akl [1]. Data is exchanged between the processors through an interconnection network. When we use terms like *send* and *receive* we mean transmitting data on the interconnecting network.

3.1 The parallel strategy

The foundation of our strategy for the parallel algorithm is the observation that *the extreme solutions can be found independently*. Therefore several extreme solutions are found in parallel, each being calculated using the *sequential* sign-accord algorithm.

Of course the method could also have been parallelized in other respects. For example many parallel algorithms for solving a linear system of equations have been proposed in recent years, see for instance Vorst and Dooren [17]. However, such methods would involve a much more complicated setup of the processors and much more communication between the processes than the larger grained strategy indicated above.

Therefore we suggest a *central algorithm* where a *master process* communicates with a number of *slave processes*. There is no communication between the slaves. The master process finds the set $\mathcal{Y}$ of selecting sign vectors and distribute the relevant *jobs* (a *job* is to find one extreme solution) to the slave processes, one at the time. The slaves solve their problems by using the sequential sign-accord algorithm, and then send the relevant results back to the master which gathers the extreme solutions to the final solution.

The algorithm for the master process is shown in Figure 6. After $\mathcal{Y}$ has been found initializing data is sent to the slaves. More specifically this data is A^I, b^I, A_c^{-1}, and $\det(A_c)$. In Step 3 idle slaves are repeatedly told which extreme solutions to find, as pointed out by the y-vectors. To ensure that only free slaves get jobs the slaves only get a y-vector upon request. In Step 4 the master waits until all the slaves are ready to report their results, and when this is the case the results are collected in Step 5. Note that the master does not carry out any calculations other than generating the sign vectors.

The algorithm for the slaves is shown in Figure 7. After receiving initializing data from the master each slave repeatedly asks the master for work. If the master does not request the result then the sign vector y is received and the corresponding extreme solution found. Obviously, there is no reason to return each extreme solution to the master immediately after it has been found by a slave. Each slave can keep its own local result r^I which expresses the interval hull of all the extreme solutions found so far.

observation
efore several
ed using the

in other re-
near system
ce Vorst and
nore compli-
between the

rocess com-
munication
ing sign vec-
ne solution)
ir problems
the relevant
to the final

6. After $\mathcal{Y}$
cifically this
eatedly told
. To ensure
on request.
report their
tep 5. Note
generating

r receiving
master for
vector y is
sly, there is
iately after
al result r^I
und so far.

```
Step 1  find the set $\mathcal{Y}$ of selecting sign vectors
Step 2  send initializing data to slaves
Step 3  repeat
            find next $y \in \mathcal{Y}$ and drop $y$ from $\mathcal{Y}$
            wait for a slave requesting work
            send $y$ to the slave
        until $\mathcal{Y}$ is empty

Step 4  wait for slaves to finish
Step 5  $x^I := \emptyset$
        for $i := 1$ to $s$ do
            request result from slave $i$
            receive result $r^I$ from slave $i$
            $x^I := \square(x^I \cup r^I)$
```

Figure 6: Algorithm for the master process

```
Step 1  receive initializing data from master
Step 2  $r^I := \emptyset$
Step 3  repeat
            request work from master
            if master requests result
Step 4          send result $r^I$ to master
            else
Step 5          receive $y$ from master
                find the extreme solution $x^I_y$
                $r^I := \square(r^I \cup x^I_y)$
        until result is reported
```

Figure 7: Algorithm for the slave processes

This (local) result is then reported back to the master upon request.

The above protocol has the following drawback. When the last sign vector is sent, the master waits until *all* of the slaves are ready to report their results. Therefore, the slaves that finish first are idle until the last slave has finished. In the meantime the first slaves could just as well report their results to the master. Still, in spite of this disadvantage, we recommend the above algorithm because it is easy to implement.

As an alternative to the *dynamic* job distribution of Figure 6 we could have considered a *static* job distribution, i.e. an initial distribution of all jobs to the slaves. This would require less communication. However, as experiments have indicated, this strategy can be less efficient because of differences in the amount of work (i.e. the number of sign-accord iterations) used to find one extreme solution. Thus, some slaves could receive a bunch of difficult jobs whereas others might have only easy jobs to solve, and as a result the latter slaves would have more idle time. Therefore we recommend the dynamic distribution.

3.2 Time estimate

In this subsection we give a rough estimate of the *total computation time* T_p of the parallel algorithm when p processors are used. This is the time spent from the moment when the master process started to the moment when the final result is found. Basically this time depends only on the following parameters:

$p =$ number of processors;

$s =$ number of slaves, $s = p - 1$;

$m =$ total number of extreme solutions to be found (i.e the total number of jobs);

$c_0 =$ time used to find $\mathcal{Y}$;

$w_0 =$ time used to initialize the slaves;

$w_1 =$ average time from the moment when a request is sent from a slave to the moment when the message (i.e., the value y) is received by this slave;

$c_1 =$ average time used for to find one extreme solution;

$c_2 =$ average time used to include a solution x_y^I in r^I;

$w_2 =$ average time from the moment when a request is sent from the master to the moment when the local result r^I is received by the master;

$c_3 =$ average time used to include r^I in x^I.

Notice that w_1 depends on s, because w_1 includes the queueing time at the master processor.

Now we assume that all jobs take equal time. Of course this assumption does not always hold in practice (it would, for instance, require that all extreme solutions were found in the same number of sign-accord iterations). But very often extreme solutions are found in one iteration only, and therefore we find it useful to make the analysis under the equal time assumption. Then the maximum number of jobs done by any slave is

$$r = (m + s - 1) \text{ div } s.$$

Therefore, some slaves will be idle in the last round unless m is a multiple of s, i.e. $r = m/s$. Now the total calculation time can be estimated as the sum of the following quantities:

- the time used to initialize: $c_0 + w_0$;
- the time used in Step 3 by the most busy slave: $r*$(the time used in one repeat-loop);
- the time used to collect the results: $s * (w_2 + c_3)$.

Hence, the total computation time is

$$T_p = c_0 + w_0 + r * (w_1 + c_1 + c_2) + s * (w_2 + c_3). \qquad (22)$$

An ideal parallel method would produce a computation time which is p times smaller than that of the sequential method (if p processors are used). This goal cannot quite be obtained here because the master is waiting most of the time, so we can at best obtain $T_p = T_1/(p - 1)$.

But if m is very large compared to s and n is not too small then w_0, w_1 w_2, c_2, and c_3 are negligible compared to c_1. If, furthermore, we neglect c_0 (the time spent before the slaves need to be allocated) then the computation time T_p is proportional to r, and thus

$$T_p \approx r * c_1 \approx (r/m) * T_1 \approx T_1/s = T_1/(p-1). \tag{23}$$

If, however, m is only a few times larger than s then w_1 may be comparable with c_1, and the difference between r and m/s may be significant. In this case, the speedup is smaller.

4 Numerical results

The parallel algorithm has been implemented and tested on a Meiko Transputer System consisting of 64 T800 Inmos Transputers, where 60 are available to the user. In our tests we have used up to 33 Transputers. Typical results are presented in this section.

4.1 Measurements used

When evaluating the performance of the parallel algorithm we use the following measures: The *parallel time* is the total computation time T_p minus the initialization time c_0. The reason for excluding Step 1 of the master algorithm is that no slaves need to be allocated at that time. Therefore we also exclude Step 1 when measuring the computation time for the sequential algorithm.

The *speedup* using a total of p processors is defined as

$$\text{speedup} := \frac{\text{parallel time for sequential program}}{\text{parallel time using } p \text{ processors}}. \tag{24}$$

It follows from (22) and (23) that the best we can get is

$$\text{speedup } = p - 1.$$

The parallel time for a process can be divided into *calculation time* and *communication time*. In (22) the calculation times are denoted by c_1, c_2 and the communication times by w_1, w_2. In our timings we measure the

calculation
between the

For each
that is

ut

If the utiliz
calculating,

In order
measure the

jo

If the job di
time on calc
and the disp
of the calcu

This wa
increases.
est before t
utilization r
persion rati
waiting for

4.2 Tes

The algorit
detail dozer
results relat

calculation time, and we find the communication time as the difference between the calculation time and the communication time.

For each example below we furthermore present the *utilization ratio*, that is

$$\text{utilization ratio} := \frac{\text{mean value of the calculation times for the slaves}}{\text{mean value of the parallel times for the slaves}}. \tag{25}$$

If the utilization ratio is 1 then the slaves are spending all of their time calculating, and if it is 0 then they only communicate.

In order to give a feeling of how efficient the job distribution is we also measure the *job dispersion ratio*, that is

$$\text{job dispersion ratio} := \frac{\text{dispersion of the calculation times for the slaves}}{\text{mean value of the calculation times for the slaves}}. \tag{26}$$

If the job dispersion ratio is 0 then the slaves all spend the same amount of time on calculating. Otherwise, some slaves spend more time than others and the dispersion ratio gives the dispersion relatively to the mean value of the calculation time for the slaves.

This way, the utilization ratio decreases when the job dispersion ratio increases. The reason is that the fastest slave has to wait for the slowest before the master requests the result, and this waiting time causes the utilization ratio to fall. If the utilization ratio decreases, but the job dispersion ratio stays very small, the decrease cannot be caused by slaves waiting for each other.

4.2 Test examples

The algorithm has been tested extensively. In Toft [16], we describe in detail dozens of experimental results; here, we only present a few typical results related to the following example from Gregory and Karney [7].

Example	n	Radius of A^I	# of extreme solutions	Sequential par. time (s.)
P1	5	0.030	32	0.86
P2	10	0.0045	390	47.18
P3	15	0.001	126	43.40
P4	30	0.00013	1022	2471.27

Table 1: Characteristics of the test examples

The midpoint matrix is the following:

$$A_c = \begin{pmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & \dots & \dots & 0 \\ 0 & -1 & 2 & \dots & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & -1 & 2 & -1 & 0 \\ 0 & \dots & 0 & -1 & 2 & -1 \\ 0 & \dots & 0 & 0 & -1 & 2 \end{pmatrix}, \; b_c = \begin{pmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

and the radius matrices are:

$$\begin{aligned} \left(\mathrm{rad}(A^I)\right)_{ij} &= \text{constant}, \; i,j = 1,\dots,n, \\ \left(\mathrm{rad}(b^I)\right)_i &= \text{constant}, \; i = 1,\dots,n. \end{aligned}$$

The radius of all components of the right-hand side is fixed at 0.001 for the examples presented here. The radii of all components of the coefficient matrix are as indicated in Table 1. In each of these four examples the number of iterations in the sign-accord algorithm is 1.

4.3 Results

Table 2 shows the results of applying the parallel program to Example P1. The time corresponding to 0 slaves is the sequential time. One can see that up to 4 slaves can be used reasonably efficiently, but a larger number of slaves cannot be recommended for this problem. The utilization ratio falls drastically, and since the job dispersion ratio stays small, the equation

(22) is vali
thus the ba
the commu
slaves. We
use enough
result the c
Consequent
only a very

Increasi
better spee
lized efficie
influence tl
ence w_1).
sequential
the y-vecto

In Tabl
ence is due
dispersion

Increasi
excellent c
Table 5. A
ratio is clo

Number of slaves	Parallel time (seconds)	Speedup	Utilization ratio	Job dispersion ratio
0	0.86	1.00	–	–
1	0.90	0.96	0.97	0.00
2	0.46	1.85	0.95	0.00
4	0.26	3.27	0.88	0.00
8	0.20	4.28	0.67	0.02
16	0.24	3.57	0.36	0.02
32	0.44	1.98	0.12	0.02

Table 2: Results from Example P1, n=5, 32 jobs

(22) is valid. Furthermore, we always have $r = m/s$ in this case, and thus the bad speedup is not caused by an uneven job distribution, i.e. the communication time must dominate when there is more than a few slaves. We conclude that the problem is "too easy", i.e. each job does not use enough computation time, and the number of jobs is too small. As a result the communication time dominates when many processors are used. Consequently, this kind of problem should be solved sequentially, or with only a very limited number of parallel processes.

Increasing the size of the problem and the number of jobs leads to a better speedup, as seen in Table 3. Now, up to 16 processors can be utilized efficiently; if more slaves are used, the communication time starts to influence the results (probably because of bottleneck problems that influence w_1). Notice that the time for one slave is slightly smaller than the sequential time. This is because the master is doing some work (finding the y-vectors) while the slave is working.

In Table 4 the performance is similar, but slightly worse. The difference is due to the fact that the smaller number of jobs gives a worse job dispersion rate.

Increasing the order n to 30 and the number of jobs to 1022 gives excellent conditions for a good speedup, as illustrated for Example P4 in Table 5. As shown in Table 5 ideal speedup is obtained. The utilization ratio is close to 1 and the job dispersion ratio is close to 0.

Number of slaves	Parallel time (seconds)	Speedup	Utilization ratio	Job dispersion ratio
0	47.18	1.00	–	–
1	46.95	1.00	1.00	0.00
2	23.59	2.00	0.99	0.00
4	11.93	3.95	0.98	0.01
8	6.16	7.66	0.96	0.01
16	3.42	13.80	0.90	0.02
32	2.32	20.37	0.74	0.03

Table 3: Results from Example P2, n=10, 390 jobs

Number of slaves	Parallel time (seconds)	Speedup	Utilization ratio	Job dispersion ratio
0	43.40	1.00	–	–
1	43.25	1.00	1.00	0.00
2	21.69	2.00	1.00	0.00
4	11.09	3.91	0.98	0.02
8	5.73	7.57	0.96	0.03
16	3.25	13.34	0.89	0.04
32	2.47	17.60	0.69	0.06

Table 4: Results from Example P3, n=15, 126 jobs

N
s

T

In an at
from maste
nificant diff
speedups: r
differences l

5 Cor

It has been
parallel con
notable size
too "easy"
but this is
solved sequ
therefore m
parallel cap
examples.

The nun
should be c
of jobs to b
Since the n
be allocatec

Number of slaves	Parallel time (seconds)	Speedup	Utilization ratio	Job dispersion ratio
0	2471.27	1.00	–	–
1	2447.76	1.01	1.00	0.00
2	1224.10	2.02	1.00	0.00
4	613.26	4.03	1.00	0.00
8	307.16	8.05	1.00	0.00
16	154.57	15.99	0.99	0.00
32	79.29	31.17	0.98	0.01

Table 5: Results from Example P4, n=30, 1022 jobs

In an attempt to reduce the communication time we tried to send jobs from master to slaves in batches of 2 or 10, but this did not make a significant difference. Neither did the use of a static distribution improve the speedups: most often the opposite was the case because of unpredictable differences between the computing times required by the individual jobs.

5 Conclusions

It has been demonstrated that the sign-accord algorithm is suitable for parallel computing. However, the problems to be solved need to have a notable size in order to utilize the parallel system well. Problems that are too "easy" (i.e., fast to solve) should therefore not be solved in parallel, but this is not a big problem, because such problems can be efficiently solved sequentially. Large systems, i.e. systems with many variables (and therefore most often many jobs in the sign-accord algorithm) utilize the parallel capacity well, and linear speedup has been obtained in several examples.

The number of parallel processors to be allocated for a specific problem should be determined in such a way that the ratio between the number of jobs to be done and the number of processors used is not too small. Since the number of jobs is known before the parallel processors need to be allocated, this initialization is possible in practice.

If an appropriate number of processors is allocated then the parallelization is very efficient, i.e. an almost linear speedup is obtained.

The use of rounded interval arithmetic guarantees that the computation results enclose the desired solution set. However, in connection with rounding errors a problem in the sign-accord algorithm has been revealed. The method is strongly based on finding the signs of computed results, but if a result interval contains 0 as an interior point then this determination cannot be made. This situation occurs, for instance, if some of the corner vectors defining the theoretical solution x^I have 0-components. Thus, there is a gap between the class of problems that can be solved *in theory* and those that can be solved *in practice* by the sign-accord algorithm. This issue and several others discussed are subject to further research.

References

[1] Akl, S. G. *The design and analysis of parallel algorithms.* Prentice Hall, 1982.

[2] Alefeld, G. and Herzberger, J. *Über die Verbesserung von Schranken für die Lösung bei Linearen Gleichungssystemen.* Angew. Informatik **3** (1971), pp. 107–112.

[3] Barth, W. and Nuding, E. *Optimale Lösung von Intervallgleichungssystemen.* Computing **12** (1974), pp. 117–125.

[4] Baumann, M. *Řešení Soustav Intervalových Lineárích Rovnic.* Master's Thesis, Charles University, Prague, 1983.

[5] Beeck, H. *Über die Struktur und Abschätzungen der Lösungsmenge von linearen Gliechungssystem mit Intervallkoeffizienten.* Computing **10** (1972), pp. 231–244.

[6] Beeck, H. *Zur Problematik der Hüllenbestimmung von Intervallgleichungssystemen.* In: Nickel, K. (ed.) "Interval Mathematics", Lecture Notes in Computer Science **29**, Springer-Verlag, 1975, pp. 150–159.

[7] Gregory, R. T. and Karney, D. L. *A collection of matrices for testing computational algorithms.* Wiley-Interscience, John Wiley & Sons, 1969.

[8] Hansen
In: Har
Oxford,

[9] Henriks
trix libr
Technic

[10] Hudak,
Math. M

[11] Neumai
Univers

[12] Nickel,
der Inte
Compu

[13] Oettli,
efficien

[14] Oettli,
linear e
sides. N

[15] Rohn, .
126 (19

[16] Toft, O
Master'
versity

[17] Vorst, I
linear a

Received: O
Revised vers

[8] Hansen, E. *On linear algebraic equations with interval coefficients.* In: Hansen, E. (ed.) "Topics in Interval Analysis", Clarenton Press, Oxford, 1969.

[9] Henriksen, T. and Toft, O. *C++ interval, matrix, and interval matrix libraries.* Report NI–92–08, Institute for Numerical Analysis, The Technical University of Denmark, 1992.

[10] Hudak, D. *Ein Eigenwertproblem für Intervall-Matrizen.* Z. Angew. Math. Mech. **64** (1984), pp. 503–505.

[11] Neumaier, A. *Interval methods for systems of equations.* Cambridge University Press, 1990.

[12] Nickel, K. *Die Überschätzung des Wertebereichs einer Funktion in der Intervallrechnung mit Anwendungen auf lineare Gleichungsysteme.* Computing **18** (1977), pp. 15–36.

[13] Oettli, W. *On the solution set of a linear system with inaccurate coefficients.* SIAM J. Numer. Anal. **2** (1965), pp. 115–118.

[14] Oettli, W. and Prager, W. *Compatibility of approximate solution of linear equations with given error bounds for coefficients and right-hand sides.* Numer. Math. **6** (1964), pp. 405–409.

[15] Rohn, J. *Systems of linear interval equations.* Linear Algebra Appl. **126** (1989), pp. 39–78.

[16] Toft, O. *Sequential and parallel solution of linear interval equations.* Master's Thesis, Institute for Numerical Analysis, The Technical University of Denmark, 1992.

[17] Vorst, H. A. and Dooren, P. (eds) *Parallel algorithms for numerical linear algebra.* North Holland, Amsterdam, 1990.

Received: October 30, 1992
Revised version: October 17, 1993

Institute for Numerical Analysis
The Technical University of Denmark
DK–2800 Lyngby
Denmark