An Accurate and Efficient Selfverifying Solver for Systems with Banded Coefficient Matrix

Carlos Hölbig[a], Walter Krämer[b] and Tiarajú Diverio[c].

[a]Universidade de Passo Fundo and PPGC-UFRGS,
Passo Fundo - RS, Brazil
E-mail: holbig@upf.br

[b]University of Wuppertal,
Wuppertal, Germany
E-mail: kraemer@math.uni-wuppertal.de

[c]Instituto de Informática and PPGC at UFRGS,
Porto Alegre - RS, Brazil
E-mail: diverio@inf.ufrgs.br

In this paper we discuss a selfverifying solver for systems of linear equations $Ax = b$ with banded matrices A and the future adaptation of the algorithms to cluster computers. We present an implementation of an algorithm to compute efficiently componentwise good enclosures for the solution of a sparse linear system on typical cluster computers. Our implementation works with point as well as interval data (data afflicted with tolerances). The algorithm is implemented using C-XSC library (a C++ class library for extended scientific computing). Our goal is to provide software for validated numerics in high performance environments using C-XSC in connection with the MPICH library. Actually, our solver for linear system with banded matrices runs on two different clusters: ALiCE[1] at the University of Wuppertal and LabTeC[2] at UFRGS. Our preliminary tests with matrix-matrix multiplication show that the C-XSC library needs to be optimized in several ways to be efficient in a high performance environment (up to now the main goal of C-XSC was functionality and portability, not speed). This research is based on a joint research project between German and Brazilian universities (BUGH, UKA, UFRGS and PUCRS) [5].

1. Introduction

For linear systems where the coefficient matrix A has band structure the general algorithm for solving linear systems with dense matrices is not efficient. Since an approximate inverse R is used there this would result in a large overhead of storage and computation time, especially if the bandwidth of A is small compared with its dimension. To reduce this overhead, we will replace the approximate inverse by an approximate LU-decomposition of A which needs memory of the same order of magnitude only as A itself. Then we will have to solve systems with triangular banded matrices (containing point data) in interval arithmetic. This seems to be a trivial task and several methods have been developed using such systems and simple forward and backward substitution in interval arithmetic, see e.g. [8], [12]. However, at this point there appears suddenly a very unpleasant effect which makes the computed intervals blow up very rapidly in many cases. This effect is known in literature as *wrapping effect* (see e.g. [13]) and was recognized first in connection with the verified solution of ordinary initial value problems. However it is a common problem within interval arithmetic and may show

[1]Cluster with 128 Compaq DS10 Workstations, 616 MHz Alpha 21264 processors, 2 MB cache, Myrinet multistage crossbar connectivity, 1 TB disc space and 32 GB memory.
[2]Cluster with 20 Dual Pentium III 1.1 GHz (40 nodes), 1 GB memory RAM, HD SCSI 18 GB and Gigabit Ethernet. Cluster server (front-end) with Dual Pentium IV Xeon 1.8 GHz, 1 GB memory RAM, HD SCSI 36 GB and Gigabit Ethernet. LaBTeC Cluster Homapage: http://www.inf.ufrgs.br/LabTeC

up whenever computations in $\mathbb{IR}^n$, $n > 1$, are performed, e.g. if an interval vector is multiplied repeatedly by matrices or more general if a function $f : \mathbb{IR}^n \to \mathbb{IR}^n$ is applied repeatedly to an argument x, $x_n = f(f(\cdots f(f(x))\cdots))$. It should be noted that the wrapping effect is not related to roundoff errors or any other sources of errors in an algorithm (like truncation errors, discretization errors and the like) but it is introduced solely by interval arithmetic itself (though any additional errors may contribute to an increase of the wrapping effect).

2. The Algorithms

The algorithms implemented in our work were described in [9] and can be applied to any system of linear equations which can be stored in the floating point system on the computer. They will, in general, succeed in finding and enclosing a solution or, if they do not succeed, will tell the user so. In the latter case, the user will know that the problem is very ill conditioned or that the matrix A is singular.

In the implementation in C-XSC, there is the chance that if the input data contains large numbers or if the inverse of A or the solution itself contain large numbers, an overflow may occur, in which case the algorithms may crash. In practical applications, this has never been observed, however. This could also be avoided by including the floating point exception handling which C-XSC offers for IEEE floating point arithmetic [2].

For this work we implemented interval algorithms for solution of linear systems of equations with dense and sparse matrices. There are numerous methods and algorithms computing approximations to the solution x in floating-point arithmetic. However, usually it is not clear how good these approximations are, or if there exists a unique solution at all. In general, it is not possible to answer these questions with mathematical rigour if only floating-point approximations are used. These problems become especially difficult if the matrix A is ill conditioned.

We implemented some algorithms which answer the questions about existence and accuracy automatically once their execution is completed successfully. Even very ill conditioned problems can be solved with these algorithms. Most of the algorithms implemented here can be found in [14] and [15].

3. Band Matrices

Matrices with band structure and difference equations are closely related. There the difference equations could be rewritten equivalently as a linear system with band matrix which was triangular even. Similarly we can go in the other direction and write a triangular banded matrix as a difference equation.

The system

$$Ax = \begin{pmatrix} a_{1,1} & & & & \\ \vdots & \ddots & & 0 & \\ a_{m,1} & & \ddots & & \\ & \ddots & & \ddots & \\ 0 & & a_{n,n-m+1} & \cdots & a_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ \\ b_n \end{pmatrix} = b \tag{1}$$

is equivalent with the difference equation

$$a_{i,i-m+1}x_{i-m+1} + \cdots + a_{i,i}x_i = b_i \ , \quad i = m, \ldots, n, \tag{2}$$

of order $m-1$ with the starting values

$$x_i = (b_i - a_{i,i-1}x_{i-1} - \cdots - a_{i,1}x_1)/a_{i,i} \ , \quad i = 1, \ldots, m-1 \ . \tag{3}$$

In [9] we saw that the solution of triangular systems by interval forward substitution can result in severe overestimations. Therefore we use the relationship to difference equations and solve triangular banded systems with the method for difference equations which was presented in [9].

For general banded systems we will then apply a LU-decomposition without pivoting to the coefficient matrix A and derive an interval iteration similar to

$$[y]_{k+1} = R\diamondsuit(b - A\tilde{x}) + \diamondsuit(I - RA)[y]_k \tag{4}$$

Here, however, we will not use a full approximate inverse R but rather the iteration will be performed by solving two systems with banded triangular matrices (L and U).

A similar approach to banded and sparse linear systems can be found, e.g. , in [12]. There, however, the triangular systems were solved by interval forward and backward substitution which often results in gross overestimations as we have seen already.

For a different approach to the verified solution of linear systems with large banded or arbitrary sparse coefficient matrix see Rump, [15].

The mathematical background for the verified solution of large linear systems with band matrices is exactly the same as it was already in [6] for systems with dense matrices.

For dense systems the interval iteration (4) was derived by use of an approximate inverse R of the coefficient matrix A. This is however what we want to avoid for large banded matrices A. Therefore we chose a different approximate inverse, namely

$$R := (LU)^{-1} \approx A^{-1} \tag{5}$$

where

$$LU \approx A \tag{6}$$

is an approximate LU-decomposition of A without pivoting. Since we do not use pivoting both L and U are banded matrices again, and of course they are lower and upper triangular, resp.

The analogue of the iteration (4) now reads in our case

$$y_{k+1} = (LU)^{-1}(b - A\tilde{x}) + (I - (LU)^{-1}A)y_k \tag{7}$$

or, multiplying with LU and taking intervals:

$$LU[y]_{k+1} = \diamondsuit(b - A\tilde{x}) + \diamondsuit(LU - A)[y]_k. \tag{8}$$

Therefore we have to solve two linear systems with triangular, banded coefficient matrices, L and U, in order to compute $[y]_{k+1}$, i.e. to perform one step of the iteration (8). In each iteration we first compute an enclosure for the solution of

$$L[z]_{k+1} = \diamondsuit(b - A\tilde{x}) + \diamondsuit(LU - A)[y]_k$$

and then $[y]_{k+1}$ from

$$U[y]_{k+1} = [z]_{k+1}.$$

In both systems we do not use just plain interval forward or backward substitution, however, as discussed above, we treat the systems as difference equation and apply the corresponding method. Here again, as in [6], the inclusion test

$$[y]_{k+1} = F([y]_k) \subset [y]_k^\circ \tag{9}$$

has to be checked in the same way and if it is satisfied then the same assertions hold as in the dense case.

Remark:

If we compute the LU-decomposition with Crout's algorithm, then we can get the matrix $\diamondsuit(LU-A)$ virtually for free, since the scalar products which are needed here have to be computed in Crout's algorithm anyway.

4. Tests and Results

A very well known set of ill conditioned test matrices for linear system solvers are the $n \times n$ Hilbert matrices H_n with entries $(H_n)_{i,j} := \frac{1}{i+j-1}$. As a test problem, we report the results of our program for the linear systems $H_n x = e_1$, where e_1 is the first canonical unit vector. Thus the solution x is the first column of the inverse H_n^{-1} of the Hilbert matrix H_n. We give results for the cases $n = 10$ and

$n = 20$. Since the elements of these matrices are rational numbers which can not be stored exactly in floating point, we do not solve the given problems directly but rather we multiply the system by the least common multiple lcm_n of all denominators in H_n. Then the matrices will have integer entries which makes the problem exactly storable in IEEE floating point arithmetic. For $n = 10$, we have $lcm_{10} = 232792560$ and for $n = 20$, we have $lcm_{20} = 5342931457063200$.

For the system $(lcm_{10}H_{10})x = (lcm_{10}e_1)$, the program computes the result showed in (10), which is the exact solution of this ill conditioned system.

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{pmatrix} = \begin{pmatrix} [\ 1.000000000000000E+002,\ 1.000000000000000E+002] \\ [-4.950000000000000E+003, -4.950000000000000E+003] \\ [\ 7.920000000000000E+004,\ 7.920000000000000E+004] \\ [-6.006000000000000E+005, -6.006000000000000E+005] \\ [\ 2.522520000000000E+006,\ 2.522520000000000E+006] \\ [-6.306300000000000E+006, -6.306300000000000E+006] \\ [\ 9.609600000000000E+006,\ 9.609600000000000E+006] \\ [-8.751600000000000E+006, -8.751600000000000E+006] \\ [\ 4.375800000000000E+006,\ 4.375800000000000E+006] \\ [-9.237800000000000E+005, -9.237800000000000E+005] \end{pmatrix} \tag{10}$$

For the system $(lcm_{20}H_{20})x = (lcm_{20}e_1)$, the program computes the enclosures (here an obvious short notation for intervals is used) showed in (11), which is an extremely accurate enclosure for the exact solution (the exact solution components are the integers within the computed intervals).

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \\ x_{16} \\ x_{17} \\ x_{18} \\ x_{19} \\ x_{20} \end{pmatrix} = \begin{pmatrix} [\ 3.999999999999999E+002,\ 4.000000000000001E+002] \\ [-7.980000000000002E+004, -7.979999999999998E+004] \\ [\ 5.266799999999999E+006,\ 5.266800000000001E+006] \\ [-1.716099000000001E+008, -1.716098999999999E+008] \\ [\ 3.294910079999999E+009,\ 3.294910080000001E+009] \\ [-4.118637600000001E+010, -4.118637599999999E+010] \\ [\ 3.569485919999999E+011,\ 3.569485920000001E+011] \\ [-2.237302782000001E+012, -2.237302781999999E+012] \\ [\ 1.044074631599999E+013,\ 1.044074631600001E+013] \\ [-3.700664527560001E+013, -3.700664527559999E+013] \\ [\ 1.009272143879999E+014,\ 1.009272143880001E+014] \\ [-2.133234304110001E+014, -2.133234304109999E+014] \\ [\ 3.500692191359999E+014,\ 3.500692191360001E+014] \\ [-4.443186242880001E+014, -4.443186242879999E+014] \\ [\ 4.316238064511999E+014,\ 4.316238064512001E+014] \\ [-3.147256922040001E+014, -3.147256922039999E+014] \\ [\ 1.666194841079999E+014,\ 1.666194841080001E+014] \\ [-6.044040109800001E+013, -6.044040109799999E+013] \\ [\ 1.343120024399999E+013,\ 1.343120024400001E+013] \\ [-1.378465288200001E+012, -1.378465288199999E+012] \end{pmatrix} \tag{11}$$

As other example, we compute an enclosure for a very large system. We take the symmetric Toeplitz matrix with five bands having the values $1, 2, 4, 2, 1$ and on the right hand side we set all components of b equal to 1. Then the program produces the following output for a system of size $n = 200000$ (only the first ten and last ten solution components are printed):

```
Dimension  n = 200000
Bandwidths l,k : 2 2
A = 1 2 4 2 1
change elements ? (y/n) n
b = =1
change elements ? (y/n) n
```

```
x =
  1: [  1.860146067479180E-001,  1.860146067479181E-001 ]
  2: [  9.037859550210300E-002,  9.037859550210302E-002 ]
  3: [  7.518438200412189E-002,  7.518438200412191E-002 ]
  4: [  1.160876404875081E-001,  1.160876404875082E-001 ]
  5: [  1.003153932563721E-001,  1.003153932563722E-001 ]
  6: [  9.427129202687645E-002,  9.427129202687647E-002 ]
  7: [  1.028361799416204E-001,  1.028361799416205E-001 ]
  8: [  1.005240450090008E-001,  1.005240450090009E-001 ]
  9: [  9.874921290539136E-002,  9.874921290539138E-002 ]
 10: [  1.004617422430963E-001,  1.004617422430964E-001 ]

199990: [  1.001953939326196E-001,  1.001953939326197E-001 ]
199991: [  1.004617422430963E-001,  1.004617422430964E-001 ]
199992: [  9.874921290539136E-002,  9.874921290539138E-002 ]
199993: [  1.005240450090008E-001,  1.005240450090009E-001 ]
199994: [  1.028361799416204E-001,  1.028361799416205E-001 ]
199995: [  9.427129202687645E-002,  9.427129202687647E-002 ]
199996: [  1.003153932563721E-001,  1.003153932563722E-001 ]
199997: [  1.160876404875081E-001,  1.160876404875082E-001 ]
199998: [  7.518438200412189E-002,  7.518438200412191E-002 ]
199999: [  9.037859550210300E-002,  9.037859550210302E-002 ]
200000: [  1.860146067479180E-001,  1.860146067479181E-001 ]

max. rel. error =  1.845833860422451E-016  at i = 3
max. abs. error =  2.775557561562891E-017  at i = 1
min. abs. x[3] = [ 7.518438200412189E-002,  7.518438200412191E-002 ]
max. abs. x[1] = [ 1.860146067479180E-001,  1.860146067479181E-001 ]
```

5. Integration between C-XSC and MPI Libraries

As part of our research, we did the integration between C-XSC and MPI libraries on cluster computers. This step is necessary and essential for the adaptation of our solvers to high performance environments. This integration was developed using, with first tests, algorithms for matrix multiplication in parallel environments of cluster computers. Initially, we did some comparations about the time related to the computational gain using parallelization, the parallel program performance depending on the matrix order, and the parallel program performance using a larger number of nodes. We also studied some other information like the memory requirement in each method to verify the performance relation with the execution time and memory.

We want to join the high accuracy given by C-XSC with the computational gain provided by parallelization. This parallelization was developed with the tasks division among various nodes on the cluster. These nodes execute the same kind of tasks and the communication between the nodes and between the nodes and the server uses message passing protocol.

Measures and tests were made to compare the routines execution time in C language, C using MPI library, C using C-XSC library and C using C-XSC and MPI libraries. The developed tests show that simple and small changes in the traditional algorithms can provide an important gain in the performance [11]. We observed the way that the processor pipeline is used, and we notice that it is decisive for the results. Based in these tests, we could also observe that the use of just 16 nodes is enough for this multiplication.

In the results obtained with these tests, the execution time of the algorithms using C-XSC library are much larger than the execution time of the algorithms that do not use this library. Even in this tests, it is possible to conclude that the use of high accuracy operations make the program slower. It shows that the C-XSC library need to be optimized to have an efficient use on clusters, and make it possible to obtain high accuracy and high performance in this kind of environment.

6. Conclusions

In our work we provide the development of selfverifying solvers for linear systems of equations with sparse matrices and the integration between C-XSC and MPI libraries on cluster computers. Our preliminary tests with matrix multiplication show that the C-XSC library needs to be optimized to be efficient in a High Performance Environment (up to now the main goal of C-XSC was functionality and portability, not speed). Actually we are working in to finish this integration and in the development of parallel software tools for validated numerics in high performance environments using the C++ class library C-XSC in connection with the MPICH library.

Acknowledgement: We would like to thank the students Bernardo Frederes Krämer Alcalde and Paulo Sérgio Morandi Júnior from the Mathematic of Computation and High Performance Computing Group at UFRGS for their help in implementing and testing the C-XSC routines. This work was supported in part by DLR international bureau (BMBF, Germany) and FAPERGS (Brazil).

REFERENCES

[1] Albrecht, R., Alefeld, G., Stetter, H. J. (Eds.): *Validation Numerics – Theory and Applications*. Computing Supplementum **9**, Springer-Verlag (1993).

[2] American National Standards Institute / Institute of Electrical and Electronics Engineers: *A Standard for Binary Floating-Point Arithmetic*. ANSI/IEEE Std. 754-1985, New York, 1985.

[3] Hammer, R., Hocks, M., Kulisch, U., Ratz, D.: *C-XSC Toolbox for Verified Computing I: basic numerical problems*. Springer-Verlag, Berlin/Heidelberg/New York, 1995.

[4] Hofschuster, W., Krämer, W., Wedner, S., Wiethoff, A.: *C-XSC 2.0: A C++ Class Library for Extended Scientific Computing*. Universität Wuppertal, Preprint BUGHW - WRSWT 2001/1 (2001).

[5] Hölbig, C.A., Diverio, T.A., Claudio, D.M., Krämer, W., Bohlender, G.: Automatic Result Verification in the Environment of High Performance Computing In: IMACS/GAMM INTERNATIONAL SYMPOSIUM ON SCIENTIFIC COMPUTING, COMPUTER ARITHMETIC AND VALIDATED NUMERICS, 2002, Paris. Extended abstracts, pg. 54-55 (2002).

[6] Hölbig, C., Krämer, W.: *Selfverifying Solvers for Dense Systems of Linear Equations Realized in C-XSC*. Universität Wuppertal, Preprint BUGHW - WRSWT 2003/1, 2003.

[7] Klatte, R., Kulisch, U., Lawo, C., Rauch, M.,Wiethoff, A.: *C–XSC, A C++ Class Library for Extended Scientific Computing*. Springer-Verlag, Berlin/Heidelberg/New York, 1993.

[8] Klein, W.: *Verified Solution of Linear Systems with Band-Shaped Matrices*.DIAMOND Workpaper, Doc. No. 03/3-3/3, January 1987.

[9] Krämer, W., Kulisch, U., Lohner, R.: *Numerical Toolbox for Verified Computing II - Advanced Numerical Problems*. University of Karlsruhe (1994), see *http://www.uni-karlsruhe.de/~Rudolf.Lohner/papers/tb2.ps.gz*.

[10] Kulisch, U., Miranker, W. L. (Eds.): *A New Approach to Scientific Computation*. Proceedings of Symposium held at IBM Research Center, Yorktown Heights, N.Y., 1982. Academic Press, New York, 1983.

[11] Marquezan, C.C., Contessa, D.F., Alves, R.S., Navaux, P.O.A., Diverio, T.A.: An evolution of simple and efficient optimization techniques for matrix multiplication. In: The 2002 INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED PROCESSING TECHNIQUES AND APPLICATIONS, PDPTA 2002. Proceedings ..., Las Vegas, 24-27, June, 2002. Las Vegas: CSREA Press (2002). Vol. IV. p. 2029-2034.

[12] Miranker, W. L., Toupin, R. A.: *Accurate Scientific Computations*.Symposium Bad Neuenahr, Germany, 1985.Lecture Notes in Computer Science, No. **235**,Springer-Verlag, Berlin, 1986.

[13] Neumaier, A.: *The Wrapping Effect, Ellipsoid Arithmetic, Stability and Confidence Regions*. In [1], pp 175–190, 1993.

[14] Rump, S. M.: *Solving Algebraic Problems with High Accuracy*. Habilitationsschrift. In [10], pp 51–120, 1983.

[15] Rump, S. M.: *Validated Solution of Large Linear Systems*. In [1], pp 191–212 (1993).