Improving the Performance of a Verified Linear System Solver Using Optimized Libraries and Parallel Computation

Mariana Kolberg, Gerd Bohlender and Dalcidio Claudio

Pontifícia Universidade Católica do Rio Grande do Sul, PPGCC
Av. Ipiranga, 6681 – 90619-900 – Porto Alegre, Brazil
Phone: +55 51 3320-3611 Fax: +55 51 3320-3621
Universität Karlsruhe, Institut für Angewandte und Numerische Mathematik
Kaiserstr. 12 – 76128 – Karlsruhe, Germany
Phone: +49 721 608 2049 Fax: +49 721 608 6679
{mkolberg,dalcidio}@inf.pucrs.br, bohlender@math.uka.de
topics of the conference list: Numerical algorithms for CS&E, Parallel or Distributed Computation and Cluster Computation.

Abstract. A parallel version of the self-verified method for solving linear systems was presented in [19, 18]. In this research we propose improvements aiming at a better performance. The idea is to implement an algorithm that uses technologies as MPI communication primitives associated to libraries as LAPACK, BLAS and C-XSC, aiming to provide both self-verification and speed-up at the same time. The algorithms should find an enclosure even for ill-conditioned problems. In this scenario, a parallel version of a self-verified solver for dense linear systems appears to be essential in order to solve bigger problems. Moreover, the major goal of this research is to provide a free, fast, reliable and accurate solver for dense linear systems.

1 Introduction

Many real problems are simulated and modeled using dense linear systems of equations like $Ax = b$ with an $n \times n$ matrix $A \in \mathbb{R}^{n \times n}$ and a right hand side $b \in \mathbb{R}^n$. This is true for functional linear equations that occur like partial differential equations and integral equations that appear in several problems of Physics and Engineering [4]. Many different numerical algorithms contain this task as a subproblem.

There are numerous methods and algorithms which compute 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 rigor if only floating-point approximations are used. These problems become especially difficult if the matrix A is ill conditioned. The use of self-verified methods can lead to more reliable results [12]. Verified computing provides an interval result that surely contains the correct result [20, 17]. Like that the algorithm also proves the existence and uniqueness of the solution of the problem. The algorithm will, in general, succeed in finding an enclosure of the correct solution. If the solution is not found, the algorithm will let the user know.

The use of verified computing makes it possible to find the correct result. However, finding the verified result often increases the execution times dramatically [24]. The research already developed shows that the execution time of verified algorithms are much larger than the execution time of algorithms that do not use this concept [14, 13].

To compensate the lack of performance of such verified algorithms, some works suggest the use of midpoint-radius arithmetic to achieve a better performance, since its implementation can be done using only floating-point operations [26, 27]. This would be a way to increase the performance of verified methods.

The advent of parallel computing and its impact in the overall performance of many algorithms on numerical analysis can be seen in the past years [8]. The use of clusters plays an important role in such a scenario as one of the most effective manner to improve the computational power without increasing costs to prohibitive values. The parallel solutions for linear solvers found in the literature explore many aspects and constraints related to the adaptation of the numerical methods to high performance environments [5, 25, 10, 31, 22, 11]. However, those implementations do not deal with verified methods. In the field of verified computing many important contributions have been done in the last years. Some works related to verified solvers for dense linear systems [9, 17, 12, 28, 23] can be found in the literature. However, only a few papers on verified solvers for dense systems together with parallel implementations were found [16, 30, 27], but these authors implement other numerical problems or use a parallel approach for other architectures than clusters.

The new algorithms should find an enclosure even for very ill-conditioned problems. Moreover, the major goal of this research is to provide a free, fast, reliable and accurate solver for dense linear systems.

New algorithms based on the C-XSC methods were implemented, but using just libraries like BLAS and LAPACK to achieve better performance. The idea of reducing the switching of rounding mode presented by Bohlender was implemented as well as an optimization of the residuum based on the INTLAB method. In other words, the new implementations try to join the best aspects of each library.

To ensure that an enclosure will be found, interval arithmetic was used. Several implementations and tests were done to find the most appropriate arithmetic to be used.

Aiming at a better performance, the algorithm was parallelized using the libraries SCALAPACK and PBLAS. The idea of using popular and highly optimized libraries to gain performance will also be maintained in the parallel version.

One important advantage of the presented algorithm is the ability to find a solution even for very ill-conditioned problems (in tests on personal computers an enclosure could be found for condition number up to 10^{15}) while most algorithms may lead to an incorrect result when it is too ill-conditioned (above condition number 10^{8}). Our main contribution is to increase the use of verified computing through its optimization and parallelization, once without parallel techniques it becomes the bottleneck of an application.

2 Verified Computing

One possibility to implement verified computing is using interval arithmetic combined with suitable algorithms.

2.1 Interval Arithmetic

Let $\mathbb{R}$ denote the set of real numbers and $\mathbb{PR}$ the power set over $\mathbb{R}$. The two most frequently used representations for intervals over $\mathbb{R}$, are the infimum-supremum representation

$$[a_1, a_2] := \{x \in \mathbb{R} : a_1 \leq x \leq a_2\} \text{ for some } a_1, a_2 \in \mathbb{R},\ a_1 \leq a_2, \tag{1}$$

where $\leq$ is the partial ordering $x \leq y$ and the midpoint-radius representation

$$\langle a, \alpha \rangle := \{x \in \mathbb{R} : |x - a| \leq \alpha\} \text{ for some } a \in \mathbb{R},\ 0 \leq \alpha \in \mathbb{R}. \tag{2}$$

The two representations are identical for real intervals (not for floating-point intervals), whereas for complex intervals the first representation are rectangles, the second one represents discs in the complex plane.

Today mostly the infimum-supremum arithmetic is used. There are two main reasons for that. First, the standard definition of midpoint-radius arithmetic causes overestimation for multiplication and division, and second, the computed midpoint of the floating point result of an operation is, in general, not exactly representable in floating point, thus again causing overestimation and additional computational effort. However, in [27], Rump shows that the overestimation of operations using midpoint-radius representation compared to the result of the corresponding power set operation is limited by at most a factor 1.5 in radius.

In the computer implementation of interval arithmetic, special care has to be taken for the rounding [27]. Both infimum-supremum und midpoint-radius have advantages and disadvantages. Some intervals are better represented in one arithmetic and have an overestimation in other.

As presented in [27], the main point in using midpoint-radius arithmetic is that no case distinctions, switching of rounding mode in inner loops, etc. are necessary, only pure floating point matrix multiplications. And for those the fastest algorithms available may be used, for example, BLAS. The latter bear the striking advantages that

1. they are available for almost every computer hardware, and that
2. they are individually adapted and tuned for specific hardware and compiler configurations.

This gives an advantage in computational speed which is difficult to achieve by other implementations.

2.2 Suitable Algorithm

As mentioned before, we have to ensure that the mathematical properties of interval arithmetic as well as high accuracy arithmetic be held if we want to achieve self-verification. Based on that, we used Algorithm 1 as the starting point of a first parallel version. This algorithm is based on the verified method fully described in [12] and will, in general, succeed in finding and enclosing a solution or, if it does not succeed, will let the user know. In the latter case, the user will know that the problem is likely to be very ill-conditioned or that the matrix A is singular. In this case, the user can try to use higher precision arithmetic.

Algorithm 1 Enclosure of a square linear system

1: $R \approx A^{-1}$ {Compute an approximate inverse using LU-Decomposition algorithm}
2: $\tilde{x} \approx R \cdot b$ {compute the approximation of the solution}
3: $[z] \supseteq R(b - A\tilde{x})$ {compute enclosure for the residuum}
4: $[C] \supseteq (I - RA)$ {compute enclosure for the iteration matrix}
5: $[w] := [z]$, $k := 0$ {initialize machine interval vector}
6: **while not** ($[w] \overset{\circ}{\subset} [y]$ or $k > 10$) **do**
7: $\quad [y] := [w]$
8: $\quad [w] := [z] + [C][y]$
9: $\quad k++$
10: **end while**
11: **if** $[w] \subseteq int[y]$ **then**
12: $\quad \Sigma(A, b) \subseteq \tilde{x} + [w]$ {The solution set (Σ) is contained in the solution found by the method}
13: **else**
14: $\quad$ no verification
15: **end if**

These enclosure methods are based on the following interval Newton-like iteration:

$$x_{k+1} = Rb + (I - RA)x_k, k = 0, 1, ... \tag{3}$$

This equation is used to find a zero of $f(x) = Ax - b$ with an arbitrary starting value x_0 and an approximate inverse $R \approx A^{-1}$ of A. If there is an index k with $[x]_{k+1} \overset{\circ}{\subset} [x]_k$ (the $\overset{\circ}{\subset}$ operator denotes that $[x]_{k+1}$ is included in the interior of $[x]_k$), then the matrices R and A are regular, and there is a unique solution x of the system $Ax = b$ with $x \in [x]_{k+1}$. We assume that $Ax = b$ is a dense square linear system and we do not consider any special structure of the elements of A.

3 Tools and Solvers for Dense Linear Systems of Equations

3.1 Optimized Tools and Solvers

LAPACK [21] is a fortran library for numerical linear algebra. This package includes numerical algorithms for the more common linear algebra problems in scientific computing (solving linear equations, linear least squares, and eigenvalue problems for dense and banded systems).

The numerical algorithms in LAPACK are based on BLAS routines. The BLAS (Basic Linear Algebra Subprograms [6]) are routines that provide standard building blocks for performing basic vector and matrix operations. The Level 1 BLAS perform scalar, vector and vector-vector operations, the Level 2 BLAS perform matrix-vector operations, and the Level 3 BLAS perform matrix-matrix operations. BLAS routines are efficient, portable, and widely available. They utilize block-matrix operations, such as matrix-multiply in inner loops to achieve high performance. These operations improve the performance by increasing the granularity of the computations and keeping the most frequently accessed subregions of a matrix in the fastest level of memory [7].

The ScaLAPACK (or Scalable LAPACK) library includes a subset of LAPACK routines redesigned for distributed memory computers. Like LAPACK, the ScaLAPACK routines are based on block-partitioned algorithms in order to minimize the frequency of data movement between different levels of the memory hierarchy. (For such machines, the memory hierarchy includes the off-processor memory of other processors, in addition to the hierarchy of registers, cache, and local memory on each processor.)

The fundamental building blocks of the ScaLAPACK library are distributed memory versions (PBLAS) of the Level 1, 2 and 3 BLAS, and a set of Basic Linear Algebra Communication Subprograms (BLACS) for communication tasks that arise frequently in parallel linear algebra computations. In the ScaLAPACK routines, all interprocessor communication occurs within the PBLAS and the BLACS.

3.2 Verified Tools and Solvers

There is a multitude of tools and algorithms that provide verified computing. Among them, an option is C-XSC (C for eXtended Scientific Computing) [17]. C-XSC is a free and portable programming environment for C and C++ programming languages, offering high accuracy and automatic verified results. This programming tool allows the solution of many standard problems with reliable results. The Matlab toolbox for self-verified algorithms, INTLAB [15], is also an option. Like C-XSC, it also provides interval arithmetic for real and complex data including vectors and matrices, interval arithmetic for real and complex sparse matrices, rigorous real interval standard functions, rigorous complex interval standard functions, rigorous input/output, accurate summation, dot product and matrix-vector residuals, multiple precision interval arithmetic with error bounds, and more. However, this solver can be used just together with the commercial MATLAB environment, which can increase the costs to prohibitive values.

3.3 Comparison

Each tool has its advantages and disadvantages, some are more accurate and some are faster. Another important aspect is the availability of these tools. A comparison among C-XSC, INTLAB and LAPACK [1] was performed.

Both C-XSC and LAPACK are free libraries and can be used for educational and scientific purposes. They can be downloaded from internet and used without any restrictions. INTLAB, otherwise, cannot be used freely. INTLAB itself has an open source, but to be able to use INTLAB you have to buy the commercial product MATLAB [29]. This is a big disadvantage which can make the costs of using INTLAB prohibitively large.

C-XSC and INLAB present an interval as result. Both present enclosures of the exact result. In several test cases, C-XSC presented a point interval, while INTLAB presented some uncertainty digits. The verified result makes possible to evaluate how good the floating-point approximation is. LAPACK in the other hand, does not provide a verified result, just an approximation.

As expected, LAPACK presented the best performance. However it gives just an approximation of the correct result, and not an inclusion as INTLAB and C-XSC. INTLAB is based on BLAS, therefore it presents also a good performance comparing with C-XSC. The performance presented by C-XSC is not so optimal because the algorithm uses special variables (data type dotprecision), which are simulated in software to achieve high accuracy.

The results show that C-XSC has the most reliable results and the highest accuracy. LAPACK is the one that presents the best performance, but results are not verified, and in some cases less accurate. INTLAB is the best compromise between performance and accuracy. However, as said before, it requires Matlab which is not free. The tests show that the method used in C-XSC is a good choice, but it should be optimized to gain performance.

4 Parallel Approach

To implement the parallel version of Algorithm 1, we used an approach for cluster architectures with message passing programming model (MPI) and the highly optimized library PBLAS and SCALAPACK. Clusters of computers are considered a good option to achieve better performance without using parallel programming models oriented to very expensive (but not frequently used) machines. A parallel version for this algorithm runs on distributed processors, requiring communication among the processors connected by a fast network and the communication library.

The self-verified method presented above is divided in several steps. By tests, the computation of R, the approximate inverse of matrix A, takes more than 50% of the total processing time. Similarly, the computation of the interval matrix $[C]$ that contains the exact value of $I - RA$ (iterative refinement) takes more than 40% of the total time, since matrix multiplication requires O(n^3) execution time, and the other operations are mostly vector or matrix-vector operations which require at most O(n^2). Both operations could be implemented using SCALAPACK (R calculation) and BLAS (C calculation).

A parallel version of the self-verified method for solving linear systems was presented in [19, 18]. In this paper we propose the following improvements aiming at a better performance:

- Calculation of R using just floating-point operations;
- Avoid the use of C-XSC elements that could slow down the execution;
- Use of the fast and highly optimized libraries: BLAS and LAPACK in the first sequential version (for the parallel version PBLAS and SCALAPACK respectively);
- Use of both interval arithmetics: infimum-supremum and midpoint-radius (as proposed by Rump [27]);
- Use of techniques to avoid the switching of rounding mode in infimum-supremum operations (proposed by Bohlender [2, 3]).

To find the best arithmetic for this method, the sequential algorithms for point and interval input data were written using both infimum-supremum and midpoint-radius arithmetic. The performance tests showed that the midpoint-radius algorithm needs approximately the same time to solve a linear system with interval input data, while the infimum-supremum algorithm needs much more time in this case, since the interval multiplication must be implemented and optimized functions from BLAS cannot be used. Therefore, midpoint-radius arithmetic was chosen for the parallel implementation.

The parallel implementation uses the following SCALAPACK/BLAS routines in the algorithm 1:

- SCALAPACK
  - *pdgetrf*: for the LU decomposition (matrix R on step 1);
  - *pdgetri*: to find the approximative inverse matrix (matrix R on step 1).
- PBLAS
  - *pdgemm*: matrix-matrix multiplication (matrix C on step 4);
  - *pdgemv*: matrix-vector multiplication (many steps: 2, 3 and 8 to find the vectors x, z and w);.

It is important to remember that behind every midpoint-radius interval operation more than one floating-point operation should be done using the appropriate rounding mode. The matrix multiplication $R * A$ from step 4 needs the following operations:

Algorithm 2 Midpoint-radius matrix multiplication in $\mathbb{F}^n$

1: $\tilde{c_1} = \bigtriangledown(R \cdot mid(A))$
2: $\tilde{c_2} = \bigtriangleup(R \cdot mid(A))$
3: $\tilde{c} = \bigtriangleup(\tilde{c_1} + 0.5(\tilde{c_2} - \tilde{c_1}))$
4: $\tilde{\gamma} = \bigtriangleup(\tilde{c} - \tilde{c_1}) + |R| \cdot rad(A)$

In this case, the routine PDGEMM of PBLAS would be called three times, one for step 1, one for step 2, and one for step 4.

5 Results

5.1 Performance

Performance analysis of this parallel solver was carried out varying the order of input matrix A. Matrices with three different orders were used as test cases: 2.500×2.500, 5.000×5.000, and 10.000×10.000. For each of those matrices, executions with the number of processors varying from 1 to 64 were performed. All matrices were specifically generated for these experiments and are composed of random numbers.

The results presented in this section were obtained on the HP XC6000, the high performance computer of the federal state Baden-Württemberg. The HP XC6000 is a distributed memory parallel computer with 128 nodes all in all; 108 nodes consist of 2 Intel Itanium2 processors with a frequency of 1.5 GHz and 12 nodes consist of 8 Intel Itanium2 processors with a frequency of 1.6 GHz. All nodes own local memory, local disks and network adapters. Thus the theoretical peak performance of the system is 1.9 TFLOPS. The main memory above all compute nodes is about 2 TB. All nodes are connected to the Quadrics QsNet II interconnect that shows a high bandwidth of more than 800 MB/s and a low latency. The basic operating system on each node is HP XC Linux for High Performance Computing (HPC), the compiler used was the Intel icc 10.0 and the MKL 10.0.011 was used for an optimized version of libraries SCALAPACK and PBLAS.

Figure 1 presents a comparison of the speed-ups achieved for the tested matrices. The proposed parallel solution presents a good scalability and improves the performance of the application. As expected, the larger the input matrix, the better is the speed-up achieved.

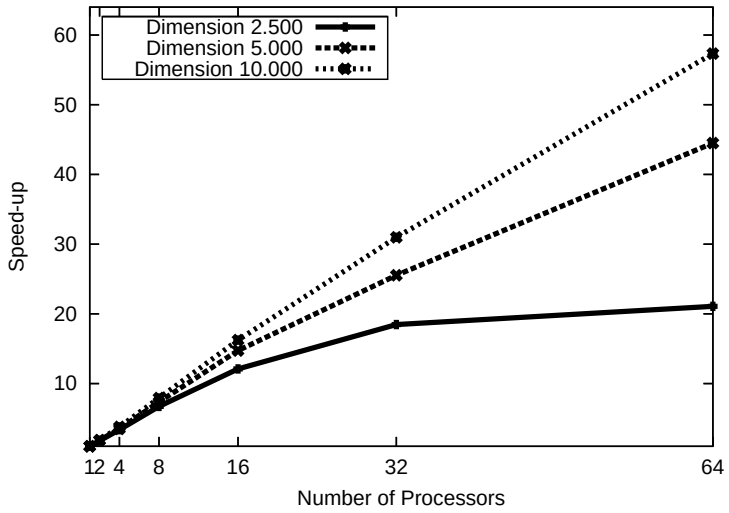

Fig. 1. Experimental results - Speed-up

For larger dimensions, the speed-up is almost linear. In some cases, like for dimension 10.000 and 16 processors, we found a super linear speed-up. This is possible due to cache effects resulting from the different memory hierarchies of a modern computer. The size of accumulated caches from different processors can also change, and sometimes all data can fit into caches and the memory access time reduces dramatically, leading to a drastic speed-up. It is understandable that this effect occurs in this implementation since PBLAS and SCALAPACK were optimized to make the best possible use of cache hierarchies.

Results for more than 64 processors were not presented for these test cases since beyond this threshold the performance started to drop down. As can be seen in figure 2, for 64 processors, the efficiency drops significatively, for the test case with dimension 2.500, it drops under 35%.

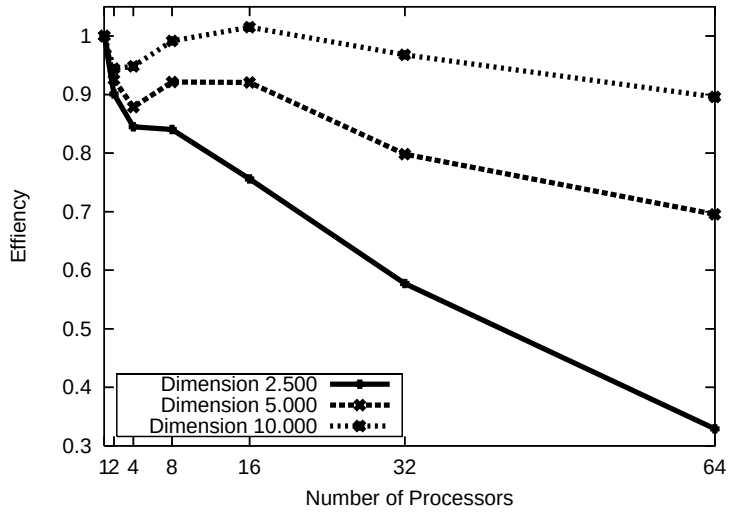

Fig. 2. Experimental results - Efficiency

5.2 Accuracy

The accuracy depends on the condition number of the matrix A. For well conditioned problems, the new algorithm may deliver a very accurate result with up to 16 correct digits.

For example supposing A is a 10x10 point matrix with condition number $4.06 \cdot 10^{+02}$ and b is a vector where each position has the value 1.0. The developed algorithm will delivery the following solution for x:

The new implementation also finds an inclusion for ill-conditioned dense matrices, but the accuracy may vary depending on the condition number as presented in table 2.

Table 1. Midpoint-radius and the equivalent infimum-supremum result

res	Midpoint	Radius
x[0] =	-1.81645396742261731e-05	3.30189340310308432e-18
x[1] =	-1.81103764097278691e-06	1.27042988556363316e-18
x[2] =	-3.50284396008609789e-06	8.65422361039543574e-19
x[...] =	...	...
x[8] =	-2.65213537238245186e-06	1.12671706774209183e-18
x[9] =	3.01161008622528871e-05	4.81485187791373545e-18

res	Infimum	Supremum
x[0] =	-0.00001816453967423	-0.00001816453967422
x[1] =	-0.00000181103764097	-0.00000181103764097
x[2] =	-0.00000350284396009	-0.00000350284396009
x[...] =	...	...
x[8] =	-0.00000265213537238	-0.00000265213537238
x[9] =	0.00003011610086225	0.00003011610086226

Table 2. Relation between condition number and diameter

Condition number	diameter
10^1	10^{-14}
10^2	10^{-13}
10^3	10^{-12}
10^4	10^{-11}
10^5	10^{-10}
10^6	10^{-9}
10^7	10^{-8}
10^8	10^{-7}
10^9	10^{-6}

It is important to mention that this relation between condition number and diameter of the resulting interval was found for a special class of matrix: square, dense with random numbers.

A well-known example of ill-conditioned matrix are the Boothroyd/Dekker matrices that are defined by the following formula:

$$A_{ij} = \binom{\text{n+i-1}}{\text{i-1}} \times \binom{\text{n-1}}{\text{n-j}} \times \tfrac{n}{i+j-1}, b_i = i, \forall i, j = 1..n,$$

For $n = 10$ this matrix has a condition number of $1.09 \cdot 10^{+15}$. The result found by this parallel solver is presented in table 3.

As expected for an ill-conditioned problem, the accuracy of the results is not the same as for a well-conditioned problem. It is important to remark that even if the result

Table 3. Results for the Boothroyd/Dekker 10x10 matrix

res	Midpoint	Radius	Infimum	Supremum
x[0]	5.4711703e-07	2.8569585e-06	-0.0000023	0.0000034
x[1]	9.9999473e-01	2.7454332e-05	0.9999672	1.0000221
x[2]	-1.9999718e+00	1.4633209e-04	-2.0001182	-1.9998255
x[3]	2.9998902e+00	5.7081064e-04	2.9993194	3.0004611
x[4]	-3.9996506e+00	1.8174619e-03	-4.0014680	-3.9978331
x[5]	4.9990383e+00	5.0008707e-03	4.9940374	5.0040392
x[6]	-5.9976307e+00	1.2322380e-02	-6.0099531	-5.9853083
x[7]	6.9946526e+00	2.7799205e-02	6.9668534	7.0224518
x[8]	-7.9887617e+00	5.8434700e-02	-8.0471964	-7.9303270
x[9]	8.9777416e+00	1.1572349e-01	8.8620181	9.0934651

has an average diameter of $4.436911 \cdot 10^{-02}$, it is an inclusion. In other words, it is a verified result.

5.3 Real Problem

For a real problem test, the used matrix is from the application of the Hydro-Quebec power systems' small-signal model, used for power systems simulations. This problem uses a square 1484 x 1484 real unsymmetric matrix, with 6110 entries (1126 diagonals, 2492 below diagonal, 2492 above diagonal) as can be seen in figure 3.

Matrix QH1484: Quebec Hydroelectric Power System

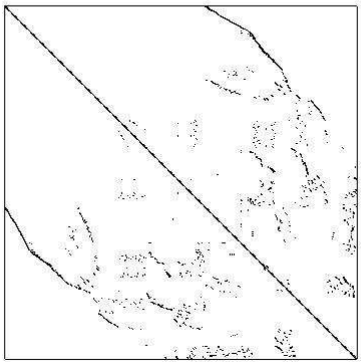

Fig. 3. Structure view of matrix QH1484

The presented solver was written for dense systems, therefore, this sparse systems will be treated as a dense system. No special method or data storage was used/done concerning the sparsity of this systems.

The first elements of the result vector found for this problem with conditional number $5.57 \cdot 10^{17}$ are the following:

Table 4. Results for QH1484: Quebec Hydroelectric Power System

res	Midpoint	Radius	Infimum	Supremum
x[0]	$-4.1415310 \cdot 10^{+00}$	$2.0242898 \cdot 10^{-07}$	$-4.1415312 \cdot 10^{+00}$	$-4.1415308 \cdot 10^{+00}$
x[1]	$-2.1936961 \cdot 10^{+00}$	$2.3526014 \cdot 10^{-07}$	$-2.1936964 \cdot 10^{+00}$	$-2.1936959 \cdot 10^{+00}$
x[2]	$-4.1417322 \cdot 10^{+00}$	$2.0242898 \cdot 10^{-07}$	$-4.1417324 \cdot 10^{+00}$	$-4.1417320 \cdot 10^{+00}$
x[3]	$-2.1954030 \cdot 10^{+00}$	$2.3526014 \cdot 10^{-07}$	$-2.1954032 \cdot 10^{+00}$	$-2.1954028 \cdot 10^{+00}$
x[...]	...	...	...	...

Despite it is an ill-conditioned problem, the average diameter of the interval results found by this solver was $1.26 \cdot 10^{-8}$. This is a very accurate result for such an ill-conditioned problem.

The execution time for solving this systems of linear equations using 8 processors was 1.4193 seconds

6 Results and Conclusions

New sequential algorithms based on a verified method were implemented, but using just libraries like BLAS and LAPACK to achieve better performance. The idea of reducing the switching of rounding mode presented by Bohlender was implemented as well as an optimization of the residuum based on the INTLAB method. In other words, the new implementations join the best aspects of each library.

To ensure that an enclosure will be found, interval arithmetic was used. To find the best arithmetic for this method, the sequential algorithms for point input data were written using both infimum-supremum and midpoint-radius arithmetic. The performance tests showed that the midpoint-radius algorithm needs approximately the same time to solve a linear system with interval input data, while the infimum-supremum algorithm needs much more time in this case, since the interval multiplication must be implemented and optimized functions from BLAS cannot be used. Therefore, midpoint-radius arithmetic was chosen for the parallel implementation.

Aiming at a better performance, the algorithm was parallelized using the libraries SCALAPACK and PBLAS.

The performance results showed that the parallel implementation leads to nearly perfect speed-up in a wide range of processor numbers for large dimensions. This is a very significant result for clusters of computers.

One important advantage of the presented algorithm is the ability to find a solution even for ill-conditioned problems while most algorithms may lead to an incorrect result when it is too ill-conditioned (above condition number 10^8). The accuracy of the results

in many cases depends on the condition number. However, the result of this method is always an inclusion, given the guarantee that the correct result is inside the interval.

Our main contribution is to provide a free, fast, reliable and accurate solver for dense linear systems and to increase the use of verified computing through its optimization and parallelization, once without parallel techniques it becomes the bottleneck of an application.

Among the ideas for future work the use of interval input data is the first to be implemented. The parallelization of a verified method for solving sparse matrices is also a goal for the future. Since many real problems are modeled using these systems, it is a natural choice to implement such methods, joining the benefits of verified and parallel computing.

References

1. E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.
2. G. Bohlender. Faster Interval Computations Through Minimized Switching of Rounding Modes. 2002. presented at PUCRS, Porto Alegre, Brazil.
3. G. Bohlender, M. Kolberg, and D. Claudio. Modifications to Expression Evaluation in C-XSC. Technical report, preprint BUW-WRSWT 2005/5, Universität Wuppertal, DE, 2005. presented at SCAN04, Fukuoka, Japan.
4. D. M. Claudio and J. M. Marins. *Cálculo Numérico Computacional: Teoria e Prática*. Editora Atlas S. A., São Paulo, 2000.
5. J.J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A van der Vorst. *Numerical Linear Algebra for High-performance Computers*. SIAM press, Philadelphia, USA, 1998.
6. J.J. Dongarra, j. Du Croz, I. S. Duff, and S. Hammarling. A set of Level 3 Basic Linear Algebra Subprograms. *ACM trans. Math. Soft.*, 16:1–17, 1990.
7. J.J. Dongarra, R. Pozo, and D.W. Walker. LAPACK++: A design overview of object-oriented extensions for high performance linear algebra. *Supercomputing '93. Proceedings*, pages 162–171, 1993.
8. I. S. Duff and H. A. van der Vorst. Developments and Trends in the Parallel Solution of Linear Systems. Technical Report RAL TR-1999-027, CERFACS, Toulouse, France, 1999.
9. A. Facius. *Iterative solution of linear systems with improved arithmetic and result verification*. PhD thesis, University of Karlsruhe, Germany, 2000.
10. A. Frommer. *Lösung linearer Gleichungssysteme auf Parallelrechnern*. Vieweg-Verlag, Universität Karlsruhe, Germany, 1990.
11. P. Gonzalez, J. C. Cabaleiro, and T. F. Pena. Solving Sparse Triangular Systems on Distributed Memory Multicomputers. In *Proceedings of the Sixth Euromicro Workshop on Parallel and Distributed Processing*, pages 470–478. IEEE Press, January 1998.
12. R. Hammer, D. Ratz, U. Kulisch, and M. Hocks. *C++ Toolbox for Verified Scientific Computing I: Basic Numerical Problems*. Springer-Verlag, New York, Inc., Secaucus, NJ, USA, 1997.
13. C. A. Hölbig, P. S. Morandi Júnior, B. F. K. Alcalde, and T. A. Diverio. Selfverifying Solvers for Linear Systems of Equations in C-XSC. In *Proceedings of Parallel and Distributed Programming (PPAM)*, volume 3019, pages 292–297, 2004.
14. C. A. Hölbig, W. Krämer, and T. A. Diverio. An Accurate and Efficient Selfverifying Solver for Systems with Banded Coefficient Matrix. In *Proceedings of Parallel Computing (PARCO)*, pages 283–290, Germany, September 2003.

15. INTLAB. INTerval LABoratory. http://www.ti3.tu-harburg.de/ rump/intlab/.
16. T. Kersten. *Verifizierende rechnerinvariante Numerikmodule*. PhD thesis, University of Karlsruhe, Germany, 1998.
17. R. Klatte, U. Kulisch, C. Lawo, R. Rauch, and A. Wiethoff. *C-XSC- A C++ Class Library for Extended Scientific Computing*. Springer-Verlag, Berlin, 1993.
18. M. Kolberg, L. Baldo, P. Velho, L. F. Fernandes, and D. Claudio. Optimizing a Parallel Self-verified Method for Solving Linear Systems. In *PARA - Workshop on State-of-the-art in Scientific and Parallel Computing*, 2006.
19. M. Kolberg, L. Baldo, P. Velho, T. Webber, L. F. Fernandes, P. Fernandes, and D. Claudio. Parallel Selfverified Method for Solving Linear Systems. In 7^{th} *VECPAR - International Meeting on High Performance Computing for Computational Science*, pages 179–190, Rio de Janeiro, Brazil, 2006.
20. U. Kulisch. and W. L. Miranker. *Computer Arithmetic in Theory and Practice*. Academic Press, New York, 1981.
21. LAPACK. Linear Algebra Package. http://www.cs.colorado.edu/ jessup/lapack/.
22. G. C. Lo and Y. Saad. Iterative Solution of General Sparse Linear Systems on Clusters of Workstations. Technical Report umsi-96-117, msi, uofmad, 1996.
23. T. Ogita, S. M. Rump, and S. Oishi. Accurate Sum and Dot Product with Applications. In *2004 IEEE International Symposium on Computer Aided Control Systems Design*, LNCS, pages 152–155, Taipei, Taiwan, September 2004. IEEE Press.
24. T. Ogita, S. M. Rump, and S. Oishi. Accurate Sum and Dot Product. *SIAM Journal on Scientific Computing*, 26(6):1955–1988, 2005.
25. V. Pan and J. Reif. Fast and Efficient Parallel Solution of Dense Linear Systems. *Computers & Mathematics with Applications*, 17(11):1481–1491, 1989.
26. S. Rump. INTLAB - INterval LABoratory. *Developments in Reliable Computing*, pages 77–104, 1998. http://www.ti3.tu-harburg.de/ rump/intlab.
27. S. Rump. Fast and Parallel Interval Arithmetic. *Bit Numerical Mathematics*, 39(3):534–554, 1999.
28. S. M. Rump. Solving Algebraic Problems with High Accuracy. In *IMACS World Congress*, pages 299–300, 1982.
29. Inc. (publisher) The MathWorks. *MATLAB, The Language of Technical Computing*. see http://www.mathworks.com, 2001.
30. A. Wiethoff. *Verifizierte globale Optimierung auf Parallelrechnern*. PhD thesis, University of Karlsruhe, Germany, 1997.
31. J. Zhang and C. Maple. Parallel Solutions of Large Dense Linear Systems Using MPI. In *International Conference on Parallel Computing in Electrical Engineering, PARELEC '02*, pages 312–317. IEEE Computer Society Press, 2002.