Optimizing a Parallel Self-verified Method for Solving Linear Systems

Mariana Kolberg, Lucas Baldo, Pedro Velho,
Luiz Gustavo Fernandes and Dalcidio Claudio

Faculdade de Informática, PUCRS
Avenida Ipiranga, 6681 Prédio 16 - Porto Alegre, Brazil
{mkolberg, lbaldo, pedro, gustavo, dalcidio}@inf.pucrs.br

Abstract. Solvers for linear equation systems are commonly used in many different kinds of real applications which deal with large matrices. Nevertheless, two key problems appear to limit the use of linear system solvers to a more extensive range of real applications: computing power and solution correctness. In a previous work, we proposed a method which employs high performance computing techniques together with verified computing techniques in order to eliminate the problems mentioned above. This paper presents an optimization of a previously proposed parallel self-verified method for solving dense linear systems of equations. Basically, improvements are related to the way communication primitives were employed and to the identification of the points in the solver algorithm in which mathematical accuracy is needed to achieve reliable results.

1 Introduction

Self-verified methods compute for the given problem a highly accurate inclusion of the solution and automatically prove the existence and uniqueness of a true result within the given enclosure interval [4]. Many real problems need numerical methods for their simulation and modeling. The use of self-verified methods can increase the quality of the result, but also the execution time [5]. It is known that interval arithmetic is more time consuming then real arithmetic, since the computation must be performed on two bounding values in each step. Finding the verified result often increases dramatically the execution times [5]. The use of parallel computing is a typical approach to improve the computational power and thus to solve large problems in reasonable time.

However, in some numerical problems, accuracy is essential. For instance, even in a simple linear system like presented in [2], the solution achieved with IEEE double precision arithmetic is completely wrong indicating the need for the use of verified computation. One solution for this problem can be found in [3], where the verified method for solving linear system using the C-XSC library is based on the enclosure theory described in [6]. Enclosure methods characteristics can be observed through the analysis if the Newton-like iteration (Equation 1):

$$X_{k+1} = Rb + (I - RA)x_k, k = 0, 1, ... \quad (1)$$

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, that if there is an index k with $[x]_{(k+1)} \subset [x]_k$ ($[x]_{k+1}$ 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 the linear system $Ax = b$ must be dense, square and we do not consider any special structure of the elements of A.

A parallel self-verified linear equation solver proposed in [1] uses as base the algorithm 1. This algorithm describes the implementation of the enclosure methods theory explained before.

Algorithm 1 Compute an enclosure for the solution of the square linear system $Ax = b$.

1: $R \approx (mid[A])^{-1}$ {Compute an approximate inverse using LU-Decomposition algorithm}
2: $\tilde{x} \approx R \cdot mid[b]$ {compute the approximation of the solution}
3: $[z] \supseteq R([b] - [A]\tilde{x})$ {compute enclosure for the residuum (without rounding error)}
4: $[C] \supseteq (I - RA)$ {compute enclosure for the iteration matrix (without rounding error)}
5: $[w] := [z]$, $k := 0$ {initialize machine interval vector}
6: **while** $[w] \subseteq int[y]$ 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: $\quad \Sigma([A], [b]) \subseteq \tilde{x} + [w]$
13: **else**
14: $\quad$ "no verification"
15: **end if**

This paper presents an optimization of the parallel version of this algorithm presented in [1]. Two points of optimization were focused: the parts in which mathematical accuracy is relevant and the communication cost. Our main contribution is to point out the advantages and drawbacks of the self-verified computation usage, speeding-up the performance of the previous parallel self-verified linear solver.

This paper is organized as follows: next section briefly describes the optimization strategies for the parallel self-verified method. The analysis of the results obtained through those optimizations is presented in section 3. Finally, the conclusion and some future works are highlighted in the last section.

2 Optimization strategies

After the analysis of the original work, two possible optimization points were detected: the use of the mathematical accuracy primitives and the use of the communication primitives. It is important to mention that the implementation description of the parallel self-verified method will not be discussed here. Readers interested in a more detailed explanation of the parallel version of the algorithm should read [1].

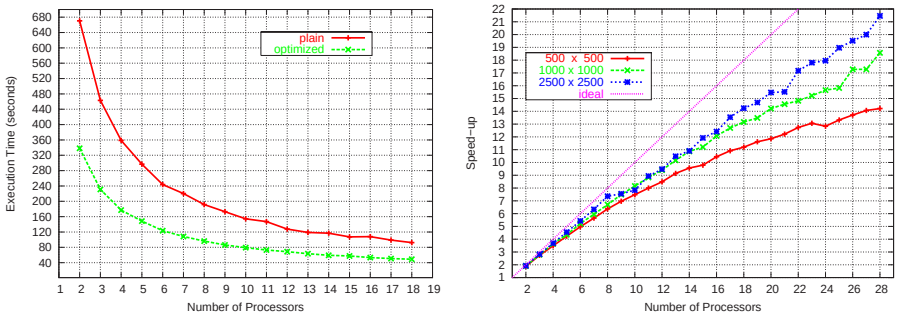
The first optimization was related to the use of high accuracy on the R computation (see algorithm 1 line 1). Since R is an approximation of the matrix A inversion, which does not need exactly results, it can be computed without high accuracy. The elimination of the use of high accuracy on the R computation does not compromise the results obtained and decreases the overall executing time (detailed results are presented in section 3).

On the other hand, the data types of the C-XSC library, which allows the self-verification and was used on the parallel approach, does not fit with the MPI library data types. In order to solve this problem, the parallel previous version used a simple, yet time consuming, solution. It is based on a pack and unpack step before sending and receiving data. In the current work, we replace the MPI library for one that already implements the pack and unpack procedures in a low level [3]. This improvement makes the data exchange among process faster, decreasing the communication cost of the parallel solution.

3 Results

The results presented in this section were obtained over the parallel environment ALiCEnext (Advanced Linux Cluster Engine, next generation) installed at the University of Wuppertal (Germany). This cluster is composed of 1024 1.8 GHz AMD Opteron processors (64 bit architecture) on 512 nodes connected by Gigabit Ethernet. ALiCEnext processors employ Linux as operating system and MPI as communication interface for parallel communication.

Performance analysis of the optimized parallel solver were carried out varying the order of input matrix A in three different grains: 500×500, $1,000 \times 1,000$ and $2,500 \times 2,500$. Figure 1(a) shows the comparison between the old parallel version (without R optimization) and the new parallel implementation (with R optimization) both using matrix A as $1,000 \times 1,000$.



(a) **(b)**

Fig. 1. Experimental Results

In figure 1(a), it can be seen that the parallel version with optimization reached lower execution times than the parallel version without optimization. In Figure 1(b),

experiments are presented varying the equation systems size. Using the matrices orders described above, it is possible to remark that the optimized solution presents good scalability. It is also important to highlight that for the largest input matrix, the speed-up achieved was around 21 for 28 processors which is a representative result for cluster platforms.

Finally, we only could carry out experiments up to 28 processors due to cluster nodes availability for our experiments. For all experiments, this number of processors was not enough to achieve the inflexion point in the speed-up curves of all three experiments, indicating that more processors could be used to improve even more the speed-ups.

4 Conclusion

This paper presented the optimization of a parallel self-verified method for solving linear systems. Two optimizations were carried out: the use of the mathematical accuracy primitives and the use of the communication primitives. The results presented a significant performance gain, for all experiments, when comparing to the old parallel implementation.

However, it is important to mention that more experiments must be carried out to verify and validate our parallel verified algorithm to a larger range of input matrices. Also, other possible optimizations in the use of verified operations must be mathematically investigated in order to guarantee that no unnecessary time consuming operations are being executed.

References

1. L. Baldo, D. Claudio, L. F. Fernandes, P. Fernandes, M. Kolberg, P. Velho, and T. Webber. Parallel Selfverified Method for Solving Linear Systems. In 7^{th} *VECPAR - International Meeting on High Performance Computing for Computational Science*. To appear, 2006.
2. G. Bohlender. *What Do We Need Beyond IEEE Arithmetic? Computer Arithmetic and Self-validating Numerical Methods* . Academic Press Professional, Inc., San Diego, CA, 1990.
3. M. Grimmer. An MPI Extension for the Use of C-XSC in Parallel Environments. Technical report, Wissenschaftliches Rechnen / Softwaretechnologie, Wuppertal, DE, 2005. http://www.math.uni-wuppertal.de/wrswt/literatur.html.
4. 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.
5. T. Ogita, S. M. Rump, and S. Oishi. Accurate Sum and Dot Product. *SIAM Journal on Scientific Computing*, 26(6):1955–1988, 2005.
6. S. M. Rump. *Kleine Fehlerschranken bei Matrixproblemen*. PhD thesis, University of Karlsruhe, Germany, 1980.