A Multithreaded Verified Method for Solving Linear Systems in Dual-Core Processors

Mariana Luderitz Kolberg[1], Daniel de Angelis Cordeiro[2], Gerd Bohlender[3], Luiz Gustavo Fernandes[1] and Alfredo Goldman[4]

[1] Faculdade de Informática, PUCRS
Avenida Ipiranga, 6681 Prédio 16 - Porto Alegre, Brazil
[2] LIG Grenoble University
51, avenue Jean Kuntzmann 38330 Montbonnot Saint Martin, France
[3] Institut für Angewandte und Numerische Mathematik 2, Universität Karlsruhe
Postfach 6980, 76128 Karlsruhe, Germany
[4] Departamento de Ciência da Computação, USP
Rua do Matão, 1010. 05508-090 São Paulo, SP, Brazil
{mkolberg, gustavo}@inf.pucrs.br, Daniel.Cordeiro@imag.fr,
gold@ime.usp.br, bohlender@math.uka.de

Abstract. This paper presents a new multithreaded approach for the problem of solving dense linear systems with verified results. We propose a new method that allows our algorithm to run in a dual-core system without making any changes in the floating-point rounding mode used during the computations, i.e., each processor independently uses its own floating-point rounding strategy to do the computations. The algorithm distributes the computational tasks among the processors based on the floating-point rounding mode required by the task.

1 Introduction

In the last few years, expensive multiprocessor systems have become commodity hardware. The new technology of dual-core processors allows the execution of parallel programs in any modern computer. The available applications, however, are not fully ready to use these new technologies. Even well-studied scientific applications must be adapted in order to fully use all available processors.

Many real problems need numerical methods for their simulation and modeling. A large number of these problems can be solved through a dense linear system of equations. Therefore, the solution of systems like

$$Ax = b \tag{1}$$

with an $n \times n$ matrix $A \in \mathbb{R}^{n \times n}$ and a right hand side $b \in \mathbb{R}^n$ are very common in numerical analysis. Many different numerical algorithms contain this kind of task as a sub-problem [3].

We present a new multithreaded approach for the problem of solving dense linear systems with verified results. We propose a new method that allows our algorithm to run in a dual-core system without making any changes in the floating-point rounding mode

used during the computations, i.e., each processor independently uses its own floating-point rounding strategy to do the computations. With this strategy we do not need to pay the cost of changing the rounding strategy during the computation, which can be more than 10 times larger than a floating point operation. The algorithm distributes the computational tasks among the processors based on the floating-point rounding mode required by the task.

This text is organized as follows. Section 2 presents some mathematical concepts of verified computing, Section 3 describes briefly the implementation issues, and finally Section 4 presents some experimental results and our conclusions.

2 Mathematical Background

There are numerous methods and algorithms which compute approximations to the solution x of equation (1) 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 verified computing can lead to more reliable results [1]. It provides as solution an interval which surely contains the correct result [7]. If the solution is not correct, e.g. if the matrix is singular, the algorithm will let the user know. The requirements for achieving this goal are: interval arithmetic, high accuracy combined with well suitable algorithms.

A verified method for solving linear systems can be found in [4], where the verified method for solving linear system is based on the enclosure theory described in [6]. This method uses an interval Newton-like iteration and Brower´s fixed point theorem 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. The method can be seen in Algorithm 1.

3 Implementation Issues

Algorithms based on this method were implemented, using just libraries like BLAS and LAPACK to achieve better performance. To ensure that an enclosure will be found, interval arithmetic and directed rounding were used. Algorithms were written for point and interval input data using infimum-supremum and midpoint-radius arithmetic.

The main idea of this implementation is to use threads to explore the benefits of dual-core processors to improve the performance. The algorithm is based on interval arithmetic which needs to change the rounding mode frequently to find the enclosing solution. It is well-known that during floating-number operations, changes in the floating-point rounding mode are expensive and can impact the performance [2]. Using dual-core processors, it is possible to divide the method in two parts: one with rounding-up and another with rounding-down. The natural solution was to use dedicated threads.

Algorithm 1 Enclosure of a square linear system.

1: $R \approx A^{-1}$ {Compute an approximated 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** ($[w] \subseteq int[y]$ 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: $\quad \Sigma(A, b) \subseteq \tilde{x} + [w]$ {The solution set (Σ) is contained in the solution found by the method}
13: **else**
14: $\quad$ no verification
15: **end if**

Our parallelization methodology divides the execution of the algorithm in five supersteps, following the Bulk Synchronous Parallel model [8], and utilizes different threads to execute the operations in each rounding mode. All operations that need a particular rounding mode are executed in the same thread.

In order to improve the performance of our implementation, each worker is statically attributed to one available processor. Defining the processor affinity [5] instructs the operating system kernel scheduler to not change the processor used by one particular thread, minimizing the number of changes in the rounding mode of each CPU.

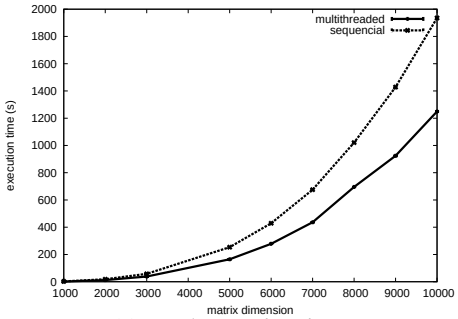
Inter-thread synchronization is done using POSIX shared memory and semaphores primitives. Threads are created and managed using the standard POSIX threads library.

4 Results and Conclusions

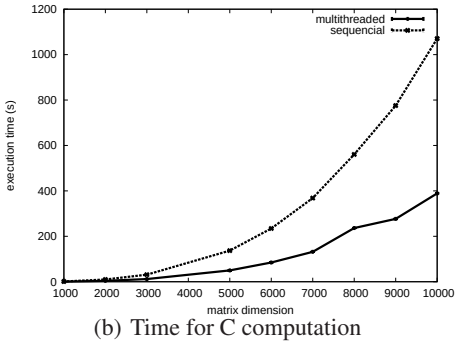
In order to verify the benefits of these optimizations, two different experiments have been performed. The first concerns the correctness of the result. Once modifications were done in the algorithm, we need to verify that it did not change the accuracy of the result. The tests generated by the Boothroyd/Dekker formula presented the same accuracy on both versions (sequential and parallel) and indicate that the parallelization did not modify the accuracy of the results. The second experiment was performed to evaluate the speed-up improvement brought by the proposed method.

We used a dedicated computer with 2 Intel Itanium2 processors of 1.6 GHz. The operating system 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 LAPACK and BLAS.

Performance analysis of the new method was carried out varying the order of input matrix A from 1,000 to 10,000. Matrix A and vector b were generated by two distinct forms: using the Boothroyd/Dekker formula [4] and random numbers, to evaluate the correctness and the performance of the new implementation respectively.



(a) Total execution time



(b) Time for C computation

We now present the execution times of both sequential and multithreaded algorithms. Figure 1 (a) shows the total execution time and (b) shows the execution times for the C computation (enclosure for the iteration matrix), both for matrix dimensions from 1,000 to 10,000. As we had a very small standard deviation (the error bars did not even appear) we just run 10 simulations for each dimension. We can notice the speed-ups when two threads are used. The speed-ups are even super linear on the C computation (the heaviest part of the calculus), we suppose that this is due to cache effects as with two processors there are less cache misses.

We have shown one possible use of dual core computers when rounding is needed. We presented a separation of the rounding up and rounding down on two separated threads in order to speed up the verified computation. The same idea can be used for other problems of the same kind.

References

1. G. Bohlender. *What Do We Need Beyond IEEE Arithmetic? Computer Arithmetic and Self-validating Numerical Methods* . Academic Press Professional, Inc., San Diego, CA, 1990.
2. G. Bohlender, M. Kolberg, and D. Claudio. Modifications to Expression Evaluation in C-XSC. Technical Report BUW-WRSWT 2005/5, Universität Wuppertal, DE, 2005. Presented at SCAN04, Fukuoka, Japan.
3. D. M. Claudio and J. M. Marins. *Cálculo Numérico Computacional: Teoria e Prática*. Editora Atlas S. A., São Paulo, 2000.
4. 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.
5. Robert Love. Kernel korner: CPU affinity. *Linux Journal*, 2003(111):8, 2003.
6. S. M. Rump. *Kleine Fehlerschranken bei Matrixproblemen*. PhD thesis, University of Karlsruhe, Germany, 1980.
7. W. L. Miranker U. Kulisch. *Computer Arithmetic in Theory and Practice*. Academic Press, New York, 1981.
8. Leslie Valiant. A Bridging Model for Parallel Computation. *Communications of the ACM*, 33(8):103–111, 1990.