HAL
open science

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

Mariana Kolberg, Daniel Cordeiro, Gerd Bohlender, Luiz gustavo Fernandes, Alfredo Goldman

To cite this version:

Mariana Kolberg, Daniel Cordeiro, Gerd Bohlender, Luiz gustavo Fernandes, Alfredo Goldman. A Multithreaded Verified Method for Solving Linear Systems in Dual-Core Processors. Workshop on State-of-the-Art in Scientific and Parallel Computing (PARA 2008), May 2008, Trondheim, Norway. pp.9. hal-00542920

HAL Id: hal-00542920

https://hal.science/hal-00542920v1

Submitted on 3 Dec 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

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

Mariana Luderitz Kolberg[1], Daniel 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. We validate our approach experimentally.

1 Introduction

Parallel computing has become a trend for the future. In the last few years, expensive multiprocessor systems have become commodity hardware. Today many users already have dual-core or even quad-core processors at home. Intel has also a prototype of a 80-core processor [25], that will be available in 5 years. Very soon multi processing capacities will be available for everyone.

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 a $n \times n$ 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 [1, 26, 23].

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.

The use of verified computing makes it possible to find the correct result. However, finding the verified result often increases the execution times dramatically [18]. 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 [9, 8].

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 [20, 21]. This would be a way to increase the performance of verified methods.

A first parallel implementation was presented using the C-XSC library and MPI for Cluster computers [12–14]. In this research, two main parts of this method (which represents the larger part of the computational costs), were studied, parallelized and optimized. These works achieved significant speed-ups, reinforcing the statement that the parallelization of such a method is an interesting alternative to increase its usability.

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 the implementation issues, Section 4 presents some experimental results, and finally Section 5 concludes de work.

2 Mathematical Background

In general, it is not possible to answer how good an approximation delivered by a conventional numeric algorithm is, when 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 [2]. It provides as solution an interval which surely contains the correct result [15]. 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 and high accuracy, combined with well suitable algorithms.

In order to ensure that an enclosure will be found, interval arithmetic was used in this implementation. 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 of these sequential versions showed that the

midpoint-radius algorithm needs approximately the same time to solve a linear system with point data as for interval input data. For infimum-supremum algorithm, the cost for solving a linear system with interval input data becomes much higher, since the interval multiplication must be implemented and the optimized functions from BLAS [6] cannot be used. Therefore, midpoint-radius arithmetic was chosen for the parallel implementation.

A verified method for solving linear systems can be found in [7], where the verified method for solving linear system is based on the enclosure theory described in [22]. 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 $[x]_{k+1} \mathring{\subset} [x]_k$ (the $\mathring{\subset}$ 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.

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**

3 Implementation Issues

The IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985) [10] is the *de facto* standard for floating-point computation and is adopted by all the major microprocessor manufacturers [11]. The standard defines both the representation and operations in floating-point numbers.

At runtime, it is possible to configure a microprocessor to perform all the floating-point operations using one of the following rounding modes: round to nearest, round toward zero, round toward $+\infty$ (rounding-up), and rounding toward $-\infty$ (rounding-down). Using the right round mode is very important to some scientific computations, although it is well-known that changes in the rounding mode are expensive and can impact the performance [3].

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. 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.

Our parallelization methodology divides the execution of the algorithm in five super-steps, following the Bulk Synchronous Parallel model [24], 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 [17] 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.

Algorithms based on this method were implemented using just libraries like BLAS and LAPACK [16] to achieve better performance. To ensure that an enclosure will be found, interval arithmetic and directed rounding were used [19]. Algorithms were written for point and interval input data using midpoint-radius arithmetic.

Inter-thread synchronization is done using POSIX shared memory and semaphores primitives. Threads are created and managed using the standard POSIX threads library [5].

4 Results

In order to verify the benefits of these optimizations, three different experiments were performed. The first concerns the correctness of the result. The second experiment was done to evaluate the speed-up improvement brought by the proposed method. The last test uses a real problem to compare both accuracy and execution time.

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.

4.1 Accuracy

Once modifications were done in the algorithm, we conducted some experiments with well-conditioned and with ill-conditioned matrices to confirm that there were no changes on the accuracy of the result. For well-conditioned matrices, this implementation delivers a very accurate result. A well-known example of ill-conditioned matrix are the Boothroyd/Dekker matrices [7] that are defined by the following formula:

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

For $n = 10$ this matrix has a condition number of $1.09 \cdot 10^{+15}$. The result found by this parallel solver compared with the sequential and with the exact result is presented in Table 1.

Table 1. Results for the Boothroyd/Dekker 10x10 matrix

	exact result	solution with threads	sequential solution
x[0]	0.0	[-0.0000023,0.0000022]	[-0.0000023,0.0000034]
x[1]	1.0	[0.9999785,1.0000223]	[0.9999672,1.0000221]
x[2]	-2.0	[-2.0001191,-1.9998557]	[-2.0001182,-1.9998255]
x[3]	3.0	[2.9995540,3.0004640]	[2.9993194,3.0004611]
x[4]	-4.0	[-4.0014757,-3.9985808]	[-4.0014680,-3.9978331]
x[5]	5.0	[4.9960976,5.0040557]	[4.9940374,5.0040392]
x[6]	-6.0	[-6.0099871,-5.9903856]	[-6.0099531,-5.9853083]
x[7]	7.0	[6.9783176,7.0225133]	[6.9668534,7.0224518]
x[8]	-8.0	[-8.0472766,-7.9544496]	[-8.0471964,-7.9303270]
x[9]	9.0	[8.9097973,9.0935856]	[8.8620181,9.0934651]

As expected for an ill-conditioned problem, the accuracy of the results is not as good as the results for well-conditioned problems. It is important to remark that even if the result has an average diameter of $4.436911 \cdot 10^{-02}$, it is an inclusion. In other words, it is a verified result.

The tests generated by the Boothroyd/Dekker formula presented almost the same accuracy on both versions (sequential and multithreaded). Actually, for this example, the result of the parallel version is a narrower interval than the result of the sequential version. As required by the algorithm, both results contain the exact result. This indicates that the rounding was affected by the use of threads (possibly by changes in the sequence of operations), however there was no loss of accuracy of the results.

4.2 Performance

We now present the execution times of both sequential and multithreaded algorithms. Tests were executed for matrix dimensions from 1,000 to 10,000. As we had a very small standard deviation (less than 1.82 in the worst case; the error bars did not even appear) we just run 10 simulations for each dimension.

In order to validate the proposed parallelization schema, only the computation of the enclosure for the iteration matrix (the C component showed in Algorithm 1) was parallelized. The computation of C is the most costly operation performed by the algorithm (complexity $O(n^3)$), so this operation is the first hot spot to be considered in order to improve the performance for this method. Also, our first experiments showed that, for this particular case, the parallelization of the other parts of this algorithm suffered from poor performance mainly due to the overhead introduced by lock contention. The fine-grained access to the memory performed by these operations caused, very often,

a dispute for the locks that protect the same region of memory between the rounding threads.

A more detailed study on where the use of multi-threaded parallelism can effectively improve the execution time of other parts still needs to be done. There is a clear trade-off between the overhead incurred by thread synchronization and the performance gain. On this paper our major goal was to do a proof-of-concept on the use of threads to avoid context changes.

Figure 1 shows the execution times for the C computation (enclosure for the iteration matrix). In this figure, it is possible to see a significant reduction in the execution time. The speed-ups are even super linear on the C computation (the heaviest part of the computation), with speed-ups up to 2.80. We suppose that this is due to cache effects as with two processors there are less cache misses. In the sequential version, all matrix elements must be loaded in the cache to compute C with rounding-up, and after that, again, to compute it with rounding-down. If the entire matrix does not fit in the cache, there will be many cache misses for each rounding mode. Since both threads use the same data at the same time, the multithreaded version allows a more effective utilization of the available cache memory, resulting in a better (and even super linear) speed-up as expected.

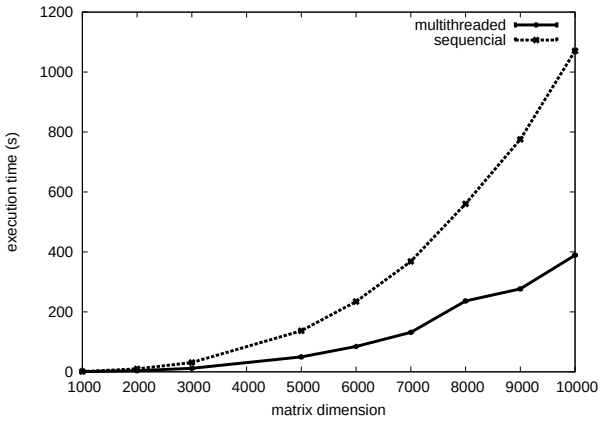

Fig. 1. Time for C computation

Despite the fact of threads are being used only in the C computation, it was possible to obtain a reasonable global speed-up that can be seen in Figure 2. The gain for solving a system with dimension $10,000$ was approximately 40%, which is a considerable result for this kind of platform.

4.3 Real Application

For a real problem test, the used matrix is from the application of Alfven Spectra in Magnetohydrodynamics [4], used for Plasma physics. Large nonsymmetric generalized

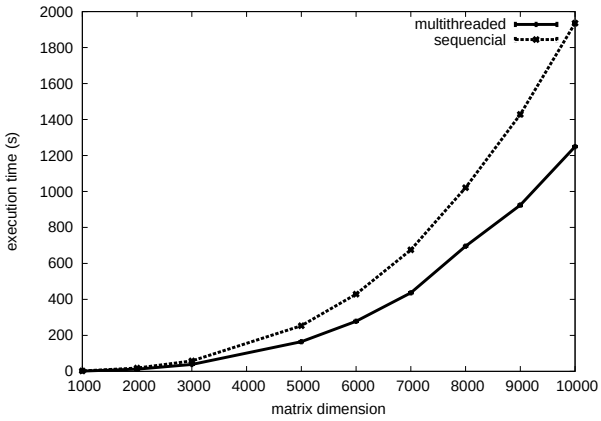

Fig. 2. Total execution time

matrix eigenvalue problems arise in the modal analysis of dissipative magnetohydrodynamics (MHD). The MHD system combines Maxwell's and fluid flow equations. The physical objective of these MHD systems is to derive nuclear energy from the fusion of light nuclei. The plasmas generated exhibit both the characteristics of an ordinary fluid and special features caused by the magnetic field. The study of linearized motion in MHD has contributed significantly to the understanding of resistive and nonadiabatic MHD plasma phenomena such as plasma stability, wave propagation and heating.

This problem uses a square $3,200$ x $3,200$ real symmetric indefinite matrix, with $18,316$ entries ($3,200$ diagonals, $7,558$ below diagonal, $7,558$ above diagonal).

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 $2.02 \cdot 10^{13}$ are presented in Table 2.

Table 2. Results for MHD3200B: Alfven Spectra in Magnetohydrodynamics

res	Midpoint	Radius	Infimum	Supremum
x[0]	$6.1031675 \cdot 10^{-01}$	$6.7758771 \cdot 10^{-17}$	$6.10316758737 \cdot 10^{-01}$	$6.10316758738 \cdot 10^{-01}$
x[1]	$1.0954139 \cdot 10^{-01}$	$5.3302866 \cdot 10^{-14}$	$1.095413995491 \cdot 10^{-01}$	$1.095413995492 \cdot 10^{-01}$
x[2]	$5.4993982 \cdot 10^{-02}$	$6.1055585 \cdot 10^{-18}$	$5.499398270479 \cdot 10^{-02}$	$5.499398270480 \cdot 10^{-02}$
x[3]	$7.0470230 \cdot 10^{+05}$	$4.221512 \cdot 10^{-07}$	$7.04702304574 \cdot 10^{+05}$	$7.04702304575 \cdot 10^{+05}$

Despite it is an ill-conditioned problem, the average diameter of the interval results found by this threaded solver was $1.87 \cdot 10^{-5}$. This is a very accurate result for such an ill-conditioned problem.

The execution time for solving this systems of linear equations using the sequential version was 90.80 seconds and with the new multithreaded solution was 70.03 seconds. The gain in C computation was around 45% from 48.53 to 26.41 seconds.

5 Conclusions

We have shown one possible use of dual-core computers when rounding is needed. We presented a separation of the calculation with 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.

Our experiments showed that we can minimize the performance overhead of changes in the floating-point rounding mode using multi-core machines. Although the inter-thread synchronization primitives could potentially downgrade the overall performance, we showed that coarse-grained operations combined with this minimization in the number of changes in the rounding mode can lead to a significantly better performance.

The fine control in the scheduling of threads provided by the Linux kernel (through the processor affinity mechanism described in Section 3) allows the use of any number of threads and still guarantees that each available processor will use the minimum number possible of changes in the floating-point rounding mode. We believe that this makes our parallelization method flexible enough to be used in the parallelization of other numerical problems.

References

1. M. Baboulin, L. Giraud, and S. Gratton. A parallel distributed solver for large dense symmetric systems: applications to geodesy and electromagnetism problems. Technical Report TR/PA/04/16, CERFACS, Toulouse, France, 2004. Preliminary version of an article published in Int. J. High Speed Computing, vol. 19, nber 4, pp 353-363, 2005.
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. 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.
4. J. G. L. Booten, P. M. Meijer, H. J. J. te Riele, and H. A. van der Vorst. Parallel Arnoldi method for the construction of a krylov subspace basis an application in magnetohydrodynamics. In W. Gentzsch and U. Harms, editors, *Vol. II: Networking and Tools*, volume 797 of *Lecture Note in Computer Science*, Berlin, 1994. Springer-Verlag. (Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking, Munich, Germany, April 1994).
5. D. R. Butenhof. *Programming with POSIX threads*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.
6. J.J. Dongarra, J. Du Croz, I. S. Duff, and S. Hammarling. A set of Level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 16:1–17, 1990.

7. 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.
8. 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 Distributted Programming (PPAM)*, volume 3019, pages 292–297, 2004.
9. 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.
10. IEEE. IEEE standard for binary floating-point arithmetic. *ACM SIGPLAN Notices*, 22(2):9–25, February 1985.
11. W. Kahan. Lecture notes on the status of IEEE standard 754 for binary floating-point arithmetic. May 1996.
12. M. Kolberg, L. Baldo, P. Velho, L. F. Fernandes, and D. Claudio. Optimizing a Parallel Self-verified Method for Solving Linear Systems. *Lecture Notes in Computer Science: Applied Parallel Computing. State of the Art in Scientific Computing 8th International Workshop, PARA 2006, Umeå, Sweden, June 18-21, 2006, Revised Selected Papers*, 3:949–955, 4699/2007 2007.
13. M. Kolberg, L. Baldo, P. Velho, T. Webber, L. F. Fernandes, P. Fernandes, and D. Claudio. Parallel Selfverified Method for Solving Linear Systems. In 7th *VECPAR - International Meeting on High Performance Computing for Computational Science*, 2006.
14. M. Kolberg, L. F. Fernandes, and D. Claudio. Dense Linear System: A Parallel Self-verified Solver . *International Journal of Parallel Programming*, 2007.
15. U. W. Kulisch and W. L. Miranker. *Computer Arithmetic in Theory and Practice*. Academic Press, Inc., Orlando, FL, USA, 1981.
16. LAPACK. Linear Algebra Package. http://www.cs.colorado.edu/ jessup/lapack/.
17. R. Love. Kernel korner: CPU affinity. *Linux Journal*, 2003(111):8, 2003.
18. T. Ogita, S. M. Rump, and S. Oishi. Accurate Sum and Dot Product. *SIAM Journal on Scientific Computing*, 26(6):1955–1988, 2005.
19. S. Oishi. Fast enclosure of matrix eigenvalues and singular values via rounding mode controlled computation. 324(1–3):133–146, February 2001.
20. S. Rump. INTLAB - INterval LABoratory. *Developments in Reliable Computing*, pages 77–104, 1998. http://www.ti3.tu-harburg.de/ rump/intlab.
21. S. Rump. Fast and Parallel Interval Arithmetic. *Bit Numerical Mathematics*, 39(3):534–554, 1999.
22. S. M. Rump. *Kleine Fehlerschranken bei Matrixproblemen*. PhD thesis, University of Karlsruhe, Germany, 1980.
23. P. Stpiczynski and M. Paprzycki. Numerical Software for Solving Dense Linear Algebra Problems on High Performance Computers. In *Proceedings of the 4th International Conference APLIMAT 2005*, pages 207–218, Slovak University of Technology, Bratislava, May 2005.
24. L. Valiant. A Bridging Model for Parallel Computation. *Communications of the ACM*, 33(8):103–111, 1990.
25. S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, T. Jacob, S. Jain, C.and Hoskote Y. Erraguntla, V.and Roberts, N. Borkar, and S. Borkar. An 80-tile sub-100-w teraflops processor in 65-nm cmos. *Solid-State Circuits, IEEE Journal of*, 43(1):29–41, 2008.
26. J. Zhang and C. Maple. Parallel solutions of large dense linear systems using mpi. *parelec*, 00:312, 2002.