The Impact of High Performance Computing in the Solution of Linear Systems: Trends and Problems[1]

Iain S. Duff[2]

ABSTRACT

We review the influence of the advent of high performance computing on the solution of linear equations. We will concentrate on direct methods of solution and consider both the case when the coefficient matrix is dense and when it is sparse. We will examine the current performance of software in this area and speculate on what advances we might expect in the early years of the next century.

Keywords: sparse matrices, direct methods, parallelism, matrix factorization, multifrontal methods.

AMS(MOS) subject classifications: 65F05, 65F50.

---

[1]Preprint of paper to appear in *J. Computational and Applied Mathematics*. Current reports available by anonymous ftp to ftp.numerical.rl.ac.uk in directory pub/reports. This report is in file duffRAL99072.ps.gz. Report also available through URL http://www.numerical.rl.ac.uk/reports/reports.html. Also published as Technical Report TR/PA/99/41 from CERFACS, 42 Ave G. Coriolis, 31057 Toulouse Cedex, France.

[2] i.s.duff@rl.ac.uk.

Computational Science and Engineering Department
Atlas Centre
Rutherford Appleton Laboratory
Oxon OX11 0QX

November 11, 1999

Contents

1 Introduction

In view of the other papers appearing in this volume, we will study only the solution of linear equations

$$Ax = b \tag{1.1}$$

using direct methods based on a factorization of the coefficient matrix A. We will consider both the case when A is dense and when it is sparse although we will concentrate more on the latter.

Although there are several ways to factorize a matrix, we will use the LU factorization

$$PAQ \;=\; LU, \tag{1.2}$$

where P and Q are permutation matrices, L is a unit lower triangular matrix, and U is an upper triangular matrix. When A is a symmetric matrix, we use the analogous factorization

$$PAP^T = LDL^T, \tag{1.3}$$

where D is a diagonal matrix, or possibly block diagonal (with blocks of order 1 and 2) if we want a stable factorization of an indefinite matrix [21].

We discuss the building blocks for both sparse and dense factorization in Section 2 and illustrate their use in dense factorization in Section 3. We then show how such building blocks can be used in sparse factorization in Section 4 indicating how this has revolutionized the performance of sparse codes. We discuss recent attempts to harness the power of parallel computers in Section 5 before examining the current power and limitations of direct methods in Section 6. We conclude with some remarks on the future in Section 7.

A wide range of iterative, direct, and preconditioning techniques with an emphasis on the exploitation of parallelism is considered at length in the recent book by Dongarra, Duff, Sorensen, and van der Vorst [34]. A more recent bibliographic tour is presented by Duff and van der Vorst [44].

2 Building blocks

A common feature of current high performance machines is that the main obstacle to obtaining high performance is the bottleneck in getting data from the main memory to the functional units. This is true whether they are built from custom-made silicon or commodity chips and whether they are RISC processor workstations, pentium based PCs, vector processors, or shared or distributed memory parallel computers. Most machines use a high speed cache as a staging post. Data in this cache (many machines have multiple caches usually organized hierarchically but here we talk about the highest level cache) can

be transferred at low latency and high bandwidth to the functional units but the amount of data that can be stored in the cache is quite small (often less than one Mbyte).

This means that if we want to obtain high performance relative to the peak of the machine, it is necessary to reuse data in the cache as much as possible to amortize the cost of getting it to the cache from main memory. The most suitable and widely used kernels for doing this are the Level 3 BLAS for $O(n^3)$ operations involving matrices of order n. There are nine Level 3 BLAS kernels but the two that are most used in routines based on LU factorization are the matrix-matrix multiplication routine `_GEMM` and the solution of a block of right-hand sides by a triangular system, `_TRSM`, although the symmetric update routine, `_SYRK`, can be used in a symmetric factorization.

Machine	*Peak*	`_GEMM`
Meiko CS2-HA	100	88
IBM SP2	266	232
Intel PARAGON	75	68
DEC Turbo Laser	600	450
CRAY C90	952	900
CRAY T3D	150	102

Table 2.1: Performance of `_GEMM` kernel in Mflop/s on a range of machines (single processor performance). Matrices of order 500.

We show, in Table 2.1, the performance of the Level 3 BLAS kernel `_GEMM` on a range of computers with various floating-point chips and memory organizations. In many cases, this kernel attains about 90% or more of the peak performance of the chip and in every case more than 66% of the peak is achieved.

These building blocks have been discussed in detail in the paper by Dongarra and Eijkhout [32] so we do not discuss them further here other than to say that they can be used in factorization algorithms so that asymptotically the floating-point operations are all performed using these kernels.

3 Factorization of dense matrices

To some extent, the algorithm and code development for numerical linear algebra have always been driven by developments in computer architectures. The first real library of subroutines for linear algebra on dense matrices was developed in Algol by Wilkinson and Reinsch [87]. These were used as the basis for the LINPACK project where a wide range of software for solving dense systems of equations was developed in Fortran and is described in the LINPACK book [30]. The LU factorization code has been used as a

basis for the benchmarking of computers with the latest results being available on the World Wide Web [29]. The codes in the LINPACK package used Level 1 BLAS [68] and were portable over a wide range of machines. Although the Level 1 BLAS were ostensibly for vector operations, the LINPACK codes performed poorly on vector or cache-based machines. This was addressed in the development of the LAPACK package for linear algebra [14]. Codes in this package incorporated Level 2 and Level 3 BLAS ([31] and [33] respectively) and had a much improved performance on modern architectures. Many vendors of shared memory computers offer parallel versions of the BLAS and so, at this level, parallelization is trivial. However, LAPACK was not designed for parallel machines and, in particular, not for machines with distributed memory that use message passing to communicate data. This last class of machines is targeted by the ongoing ScaLAPACK project [19] that supports distributed computation using tools like the BLACS (Basic Linear Algebra Communications Routines) [86].

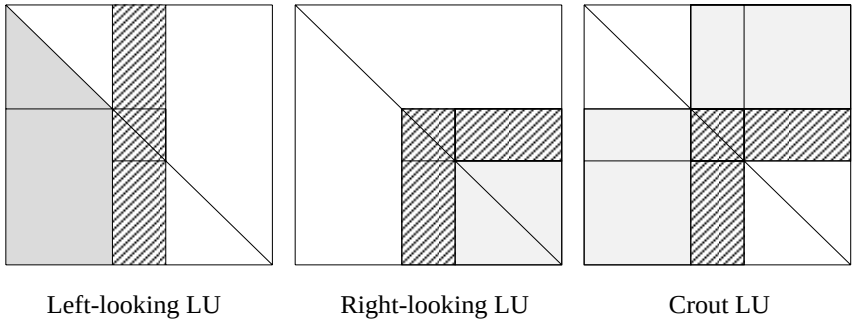

Figure 3.1: Block variants of LU decomposition.

If we view the LU factorization in a blocked or partitioned way, it becomes relatively simple to show how Level 3 BLAS can be used. We show a schematic of block LU factorization in Figure 3.1. These diagrams represent a single block stage of the factorization using three different approaches. Factorization operations are performed on the hatched regions and access is required to the other regions shaded in the matrix. For example, in the block right-looking LU factorization, the hatched block column is first factorized using the Level 2 BLAS algorithm described in Figure 3.2, the hatched block row of U is computed using the Level 3 BLAS kernel `_TRSM` and the shaded portion of the matrix updated using the `_GEMM` kernel to multiply the hatched part of the block column beneath the diagonal block with this newly computed block row of U. Algorithms of this kind are the bases for the factorization routines within the LAPACK suite that are discussed in the article by Dongarra and Eijkhout [32].

Recently, Gustavson and his collaborators [13, 58, 59] have developed a recursive way

For each column, j, of the rectangular matrix in turn do

Update the part of column j above the diagonal by solving a system whose right-hand side is the corresponding part of the original column and whose coefficient matrix is the lower triangular matrix from the previously computed columns in the block, using the Level 2 BLAS kernel _TRSV.

Update the lower part of column j by using the Level 2 BLAS kernel _GEMV to multiply the rectangular matrix corresponding to rows j to n and columns 1 to $j-1$ with the newly computed vector (and subtract this from the lower part of column j).

Choose the pivot from this newly computed lower part of column j, swop its row with row j and scale the column below the diagonal.

enddo

Figure 3.2: Level 2 factorization of rectangular block

of looking at the factorizations which has the effect of increasing the proportion of Level 3 operations and avoids the necessity for choosing block sizes as in the abovementioned block algorithms. The recursive approach can be thought of by looking at the factorization at the halfway point so that the matrix can be partitioned as

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

where matrices A_{11} and A_{21} are factorized. At this stage, a Level 3 type algorithm can be used to update the blocks A_{12} and A_{22}, and A_{22} can then be factorized using a similar recursive algorithm. Of course, the first block columns were also factorized recursively in similar fashion. An added bonus of the recursive algorithm is that access to the blocks for the Level 3 computations can be organized on contiguous data. For the earlier algorithms, the leading dimension of the arrays corresponding to the blocks is not equal to the block size. This is more noticeable in the recursive form of the Cholesky factorization.

4 Factorization of sparse matrices

The two main books discussing the direct solution of sparse linear equations are those by George and Liu [50] and by Duff, Erisman, and Reid [39]. The former restricts

its discussion to symmetric positive definite systems and emphasizes graph theoretic aspects, while the latter considers both symmetric and unsymmetric systems and includes a discussion of some of the algorithms used in the Harwell Subroutine Library (HSL) [64]. The HSL has arguably the largest number of direct sparse codes in a single library and has a few codes for iterative solution also. Information on this Library can be found in the Web pages `http://www.cse.clrc.ac.uk/Activity/HSL`. A subset of HSL is marketed by NAG as the Harwell Sparse Matrix Library (HSML). Other sparse direct software can be found through the netlib repository `http://www.netlib.org`.

When factorizing sparse matrices, it is crucial that the permutation matrices of (1.2) and (1.3) are chosen to preserve sparsity in the factors as well as to maintain stability and many algorithms have been developed to achieve this. In the general unsymmetric case, this leads to a need to compromise the numerical pivoting strategy in order to choose pivots to limit the fill-in. A common strategy for limiting fill-in, due to Markowitz [72], chooses entries so that the product of the number of other entries in the row and column of the candidate pivot is minimized. An entry is accepted as a pivot only if it is within a threshold of the largest in its column. The threshold is often an input parameter and a typical value for it is 0.1. This Markowitz-threshold strategy and a range of other similar possibilities are discussed in detail in [39]. Data structures are designed so that only the nonzero entries of the matrix and of the factors need to be held. This, coupled with the fact that it is often non-trivial to determine what part of the matrix is updated at each pivot step, has led to complicated algorithms and codes that are hard to implement efficiently on modern architectures [39].

In the symmetric case, the Markowitz analogue is *minimum degree* where one chooses as pivot a diagonal entry with the least number of entries in its row. This criterion was first proposed in 1967 [84] and has stood the test of time well. George [48] proposed a different class of orderings based on a non-local strategy of dissection. In his *nested dissection* approach, a set of nodes is selected to partition the graph, and this set is placed at the end of the pivotal sequence. The subgraphs corresponding to the partitions are themselves similarly partitioned and this process is nested with pivots being identified in reverse order. Minimum degree, nested dissection and several other symmetric orderings were included in the SPARSPAK package [49, 51]. Many experiments were performed using the orderings in SPARSPAK and elsewhere, and the empirical experience at the beginning of the 1990s indicated that minimum degree was the best ordering method for general symmetric problems. We will return to this issue of ordering when we consider parallelism in Section 5.

It is not immediately or intuitively apparent that the kernels discussed in Section 2 can be used in the factorization of sparse matrices and indeed much of the work and heuristics developed in the 1970s attempted to do just the opposite, namely to perform the basic elimination operations on as sparse vectors as possible.

The most obvious way of using dense kernels in sparse factorization is to order the

sparse matrix so that its nonzero entries are clustered near the diagonal (called bandwidth minimization) and then regard the matrix as banded, since zeros within the band soon fill-in. However, this is normally too wasteful as even the high computational rate of the Level 3 BLAS does not compensate for the extra work. A variable band format is used to extend the range of applicability of this technique. A related, but more flexible scheme, is the frontal method (for example, [36]) which owes its origin to computations using finite elements. However, all these techniques require that the matrix can be ordered to obtain a narrow band or frontwidth. Duff [35] gives several instances of how dense techniques can be used in sparse factorizations including the then newly developed multifrontal techniques. The principal advantage of multifrontal techniques over a (uni)frontal approach is that they can be used in conjunction with any ordering scheme so that sparsity can be preserved.

A fundamental concept in sparse matrix factorization is an *elimination tree*. The elimination tree is defined for any sparse matrix whose sparsity pattern is symmetric. For a sparse matrix of order n, the elimination tree is a tree on n nodes such that node j is the father of node i if entry (i, j), $j > i$ is the first entry below the diagonal in column i of the lower triangular factor. An analogous graph for an unsymmetric patterned sparse matrix is the directed acyclic graph [24, 54].

Sparse Cholesky factorization by columns can be represented by an elimination tree. This can either be a left-looking (or fan-in) algorithm, where updates are performed on each column in turn by all the previous columns that contribute to it, then the pivot is chosen in that column and the multipliers calculated; or a right-looking (or fan-out) algorithm where, as soon as the pivot is selected and multipliers calculated, that column is immediately used to update all future columns that it modifies. The terms left-looking and right-looking are discussed in detail in the book [34]. Either way, the dependency of which columns update which columns is determined by the elimination tree. If each node of the tree is associated with a column, a column can only be modified by columns corresponding to nodes that are descendants of the corresponding node in the elimination tree.

One approach to using higher level BLAS in sparse direct solvers is a generalization of a sparse column factorization. Higher level BLAS can be used if columns with a common sparsity pattern are considered together as a single block or supernode and algorithms are termed column-supernode, supernode-column, and supernode-supernode depending on whether target, source, or both are supernodes (for example, [27]).

An alternative to the supernodal approach for utilizing Level 3 BLAS within a sparse direct code is a multifrontal technique [43]. In this approach, the nonzero entries of the pivot row and column are held in the first row and column of a dense array and the outer-product computation at that pivot step is computed within that dense submatrix. The dense submatrix is called a *frontal matrix*. Now, if a second pivot can be chosen from within this dense matrix (that is there are no nonzero entries in its row and column in the sparse matrix that lie outside this frontal matrix), then the operations for this pivot

can again be performed within the frontal matrix. In order to facilitate this multiple elimination within a frontal matrix, an assembly tree is preferred to an elimination tree where, for example, chains of nodes are collapsed into a single node so that each node can represent several eliminations. Indeed sometimes we artificially enlarge the frontal matrices so that more pivots are chosen at each node and the Level 3 BLAS component is higher. Thus the kernel of the multifrontal scheme can be represented by the computations

$$F_{11} = L_1 U_1 \tag{4.4}$$

and

$$F'_{22} \leftarrow F_{22} - F_{21} {U_1}^{-1} {L_1}^{-1} F_{12}. \tag{4.5}$$

performed within the dense frontal matrix

$$\begin{bmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{bmatrix}.$$

The Schur complement, F'_{22} (4.5), is then sent to the parent node in the tree where it is summed with contributions from the original matrix and the other children to form another dense submatrix on which similar operations are performed at the father node. The effectiveness of this approach on RISC based machines can be seen from the results in Table 4.2 where the code `MA41` is a multifrontal code in the HSL [8]. Here the overall performance of the sparse code is always more than half that of the `DGEMM` kernel.

Computer	Peak	`DGEMM`	`MA41`
DEC 3000/400 AXP	133	49	34
HP 715/64	128	55	30
IBM RS6000/750	125	101	64
IBM SP2 (Thin node)	266	213	122
MEIKO CS2-HA	100	43	31

Table 4.2: Performance in Mflop/s of multifrontal code `MA41` on matrix BCSSTK15, from the Rutherford-Boeing Collection [40], on a single processor of a range of RISC processors. For comparison, the performance of `DGEMM` on a matrix of order 500 is given.

Several authors have experimented with these different algorithms (right-looking, left-looking, and multifrontal) and different blockings. Ng and Peyton [74] favour the left-looking approach and Amestoy and Duff [8] show the benefits of Level 3 BLAS within a multifrontal code on vector processors. Rothberg and Gupta [81] find that, on cache-based machines, it is the blocking that affects the efficiency (by a factor of 2 to 3) and the algorithm that is used has a much less significant effect. Demmel, Eisenstat, Gilbert, Li,

and Liu [27] have extended the supernodal concept to unsymmetric systems although, for general unstructured matrices, they cannot use regular supernodes for the target columns and so they resort to Level 2.5 BLAS, which is defined as the multiplication of a set of vectors by a matrix where the vectors cannot be stored in a two-dimensional array. By doing this, the source supernode can be held in cache and applied to the target columns or blocks of columns of the "irregular" supernode, thus getting a high degree of reuse of data and a performance similar to the Level 3 BLAS.

It is very common to solve sparse least-squares problems by forming the normal equations

$$A^T A x = A^T b \tag{4.6}$$

and to use a sparse solution scheme for symmetric positive definite systems on these resulting equations. There are, however, other methods for solving the least-squares problem. The most robust uses a QR factorization of the coefficient matrix. This factorization can also be implemented as a multifrontal method and codes have been developed by [1, 12, 73].

5 Parallel computation

In contrast to the situation with iterative methods where, in addition to vector operations, often only matrix-vector products are required, the kernel computations for sparse direct methods are far more complicated. Nevertheless, the benefits that can be obtained from successful parallelization can be much greater. Indeed, Duff and van der Vorst [44] claim that the ratio of this benefit is in proportion to 1:5:10 for iterative methods, direct sparse factorizations, and direct dense factorizations respectively. That is, we might expect gains five times as great due to the parallelization of a direct sparse code than an iterative one. The reason for this is similar to the reason why direct methods, when properly formulated, can be so efficient on RISC based or vector machines. This is due to the kernel (as we discussed in the last three sections) being a dense matrix - dense matrix multiply. We saw the benefit of using Level 3 BLAS in sparse factorization for RISC based machines in Section 4. It is also of benefit in a parallel setting to combine pivot steps and to work not with rows and columns but with block rows and columns. Clearly, the use of such block techniques and higher level BLAS allow us to obtain parallelism at the level of the elimination operations themselves.

There is also a very coarse level at which parallelism can be exploited. At this coarsest level, which is similar to the subdivision of a problem by domain decomposition, we use techniques for partitioning the matrix. These are often designed for parallel computing and are particularly appropriate for distributed memory computers. Indeed, partitioning methods are often only competitive when parallelism is considered. The PARASPAR package [88] uses a preordering to partition the original problem. The MCSPARSE package

[47, 53] similarly uses a coarse matrix decomposition to obtain an ordering to bordered block triangular form.

However, the main level of parallelism that we wish to discuss here is at a level intermediate between these two and is due to the sparsity of the matrix being factorized. Clearly, there can be substantial independence between pivot steps in sparse elimination. For example, if the matrix were a permutation of a diagonal matrix all operations could be performed in parallel. Two matrix entries a_{ij} and a_{rs} can be used as pivots simultaneously if a_{is} and a_{rj} are zero. These pivots are termed *compatible*. This observation [22] has been the basis for several algorithms and parallel codes for general matrices. When a pivot is chosen all rows with entries in the pivot column and all columns with entries in the pivot row are marked as ineligible and a subsequent pivot can only be chosen from the eligible rows and columns. In this way, a set of say k independent pivots is chosen. If the pivots were permuted to the beginning of the matrix, this $k \times k$ pivot block would be diagonal. The resulting elimination operations are performed in parallel using a rank k update. The procedure is repeated on the reduced matrix. The algorithms differ in how the pivots are selected (clearly one must compromise criteria for reducing fill-in in order to get a large compatible pivot set) and in how the update is performed.

Alaghband [2] uses compatibility tables to assist in the pivot search. She uses a two-stage implementation where first pivots are chosen in parallel from the diagonal and then off-diagonal pivots are chosen sequentially for stability reasons. She sets thresholds for both sparsity and stability when choosing pivots. Davis and Yew [25] perform their pivot selection in parallel, which results in the nondeterministic nature of their algorithm because the compatible set will be determined by the order in which potential compatible pivots are found. Their algorithm, D2, was designed for shared-memory machines and was tested extensively on an Alliant FX/8.

The Y12M algorithm [89] extends the notion of compatible pivots by permitting the pivot block to be upper triangular rather than diagonal, which allows them to obtain a larger number of pivots, although the update is more complicated. For distributed memory architectures, van der Stappen, Bisseling, and van de Vorst [85] distribute the matrix over the processors in a grid fashion, perform a parallel search for compatible pivots, choosing entries of low Markowitz cost that satisfy a pivot threshold, and perform a parallel rank-k update of the reduced matrix, where k is the number of compatible pivots chosen. They show high speedups (about 100 on 400 processors of a PARSYTEC SuperCluster FT-400) although the slow processor speed masks the communication costs on this machine. Their code was originally written in OCCAM, but they have since developed a version using PVM [67].

In the context of reduced stability of the factorization due to the need to preserve sparsity and exploit parallelism, it is important that sparse codes offer the possibility of iterative refinement both to obtain a more accurate answer and to provide a measure of the backward error. Demmel and Li [69] try to avoid the dynamic data structures required

by numerical pivoting by using the algorithm of Duff and Koster [41, 42] to permute large entries to the diagonal prior to starting the factorization. They also suggest computing in increased precision to avoid some of the problems from this compromise to stability pivoting.

An important aspect of these approaches is that the parallelism is obtained directly because of the sparsity in the system. In general, we exhibit this form of parallelism through the assembly tree of Section 4 where operations at nodes which are not on the same (unique) path to the root (that is none is a descendant of another) are independent and can be executed in parallel (see, for example, [37, 70]). The set of pivots discussed above could correspond to leaf nodes of such a tree. The tree can be used to schedule parallel tasks. For shared memory machines, this can be accomplished through a shared pool of work with fairly simple synchronizations that can be controlled using locks protecting critical sections of the code [6, 37, 38]. One of the main issues for an efficient implementation on shared memory machines concerns the management of data, which must be organized so that book-keeping operations such as garbage collection do not cause too much interference with the parallel processing.

A problem with the minimum degree ordering is that it tends to give elimination trees that are not well balanced and so not ideal for use as a computational graph for driving a parallel algorithm. The elimination tree can be massaged [71] so that it is more suitable for parallel computation but the effect of this is fairly limited for general matrices. The beauty of dissection orderings is that they take a global view of the problem; their difficulty until recently has been the problem of extending them to unstructured problems. Recently, there have been several tools and approaches that make this extension more realistic [76]. The essence of a dissection technique is a bisection algorithm that divides the graph of the matrix into two partitions. If node separators are used, a third set will correspond to the node separators. Recently, there has been much work on obtaining better bisections even for irregular graphs. Perhaps the bisection technique that has achieved the most fame has been spectral bisection [76]. In this approach, use is made of the Laplacian matrix that is defined as a symmetric matrix whose diagonal entries are the degrees of the nodes and whose off-diagonals are -1 if and only if the corresponding entry in the matrix is nonzero. This matrix is singular because its row sums are all zero but, if the matrix is irreducible, it is positive semidefinite with only one zero eigenvalue. Often the same software is used for the dissection orderings as for graph partitioning. Two of the major software efforts in this area are CHACO [62] and METIS [66].

A currently favoured approach is for the dissection technique to be used only for the top levels and the resulting subgraphs to be ordered by a minimum degree scheme. This hybrid technique was described some time ago [52] but was discredited because of the poor performance of nested dissection techniques on irregular graphs at that time. However, because of the much better implementations of dissection orderings as discussed above, this hybrid technique is included in many current implementations (for example, [17, 63]).

Current empirical evidence would suggest that these schemes are at least competitive with minimum degree on some large problems from structural analysis [17, 79] and from financial modelling [18]. In these studies, dissection techniques outperform minimum degree by on average about 15% in terms of floating-point operations for Cholesky factorization using the resulting ordering, although the cost of these orderings can be several times that of minimum degree and may be a significant proportion of the total solution time [17]. We show, in Table 5.3 the effect of the hybrid ordering within the MUMPS code (see Section 6) on one of the PARASOL test examples. AMD is an ordering produced by the Approximate Minimum Degree code of [7], and HYBRID is an ordering from METIS that combines nested dissection and minimum degree. We see that the gains from the HYBRID ordering are even more dramatic than those mentioned above with about half the number of operations required for factorization with the HYBRID ordering than with AMD. We also note, from the results in Table 5.3, that the parallelism is better for the HYBRID ordering.

	Analysis Phase	
	Entries in factors	Operations
AMD	1.13×10^8	1.28×10^{11}
HYBRID	8.53×10^7	6.72×10^{10}

	Factorization		Solve	
No. procs	AMD	HYBRID	AMD	HYBRID
1	687	307	12.0	10.1
2	408(1.7)	178(1.7)	7.5(1.6)	5.4(1.9)
4	236(2.9)	82(3.7)	6.7(1.8)	4.2(2.4)
8	143(4.8)	58(5.3)	4.2(2.9)	2.6(3.9)
16	112(6.1)	36(8.5)	2.9(4.1)	1.9(5.3)

Table 5.3: Effect of ordering on Problem BMWCRA_1 from the PARASOL test set. Matrix of order 148,770 with 5,396,386 entries in the lower triangle. Elapsed times in seconds on an ORIGIN 2000. Speedups in parentheses.

In recent years, the performance of sparse direct codes has increased considerably. The improvement is not from the approach used (fan-in, fan-out, multifrontal) but rather because of the use of blocking techniques and two-dimensional mappings. The benefit of using higher level BLAS kernels, coupled with increases in local memory and the communication speed of parallel processors, have at last made the solution of large sparse systems feasible on such architectures. We now review some of the recent performance figures from several different implementations. Dumitrescu *et al.* [45] record a performance of over 360 Mflop/s on 32 nodes of an IBM SP1 using a two-dimensional block fan-

in algorithm. Rothberg [80] has implemented a block fan-out algorithm using two-dimensional blocking and obtains a performance of over 1.7 Gflop/s on 128 nodes of an Intel Paragon, which is about 40% of the performance of the `_GEMM` kernel on that machine. A 2-D block fan-out algorithm has been coupled with some block mapping heuristics to obtain a performance of over 3 Gflop/s for a 3-D grid problem on a 196-node Intel Paragon [82]. A similar type of 2-dimensional mapping is used [57] in an implementation of a multifrontal method, where much of the high performance is obtained through balancing the tree near its root and using a highly tuned mapping of the dense matrices near the root to allow a high level of parallelism to be maintained. Although the headline figure of nearly 20 Gflop/s on the CRAY T3D was obtained on a fairly artificial and essentially dense problem, large sparse problems from structural analysis were factorized at between 8 and 15 Gflop/s on the same machine for which a tuned `_GEMM` code will execute at around 50 Gflop/s. This code is available in compiled form on an IBM SP2 [56] and source code versions of a portable implementation are available from the authors [55]. More recently, Li and Demmel [69] have been developing a version of the SuperLU code [26] for distributed memory machines and the MUMPS multifrontal code [10], developed within the EU PARASOL Project, also targets message passing architectures.

Partly because of the success of fast and parallel methods for performing the numerical factorization, other phases of the solution are now becoming more critical on parallel computers. The package [61] executes all phases in parallel, and there has been much recent work in finding parallel methods for performing the reordering. This has been another reason for the growth in dissection approaches (for example, see [65, 77]). Parallelism in the triangular solve can be obtained either using the identical tree to the numerical factorization [12] or by generating a tree from the sparsity pattern of the triangular factor [15]. However, in order to avoid the intrinsically sequential nature of a sparse triangular solve, it is possible to hold the denser but still sparse L^{-1}, or better a partitioned form of this to avoid some of the fill-in that would be associated with forming L^{-1} explicitly [5]. Various schemes for this partitioning have been proposed to balance the parallelism (limited by the number of partitions) with the fill-in (for example, [3, 4, 75]) and, more recently, the selective inversion of submatrices produced by a multifrontal factorization algorithm has been proposed [78].

6 Current situation

There is no question that direct sparse matrix algorithms and codes based on them have "come of age". Gone are the days when the only sparse codes that were generally available could be found in the HSL. We have already remarked on the PSPASES code for symmetric positive definite systems by Gupta and others [56, 55] and the SuperLU code for general unsymmetric sparse systems by Demmel and Li [27, 69]. Both these projects

have developed code for distributed memory computers.

The MUMPS code [9, 11] implements a parallel multifrontal technique for distributed memory computers and is part of the EU PARASOL Project[1] whose goal was to build and test a portable library for solving large sparse systems of equations on distributed memory systems. The PARASOL software is written in Fortran 90, uses MPI for message passing, and is available from teh Web page `http://www.pallas.de/parasol/`. The solvers developed in this Project are two domain decomposition codes by Bergen and ONERA, a multigrid code by GMD, and the MUMPS code.

Dobrian, Kumfert, and Pothen [28] have studied the use of an object oriented approach to design sparse direct solvers and O-O is used by Ashcraft in his SPOOLES package [16]. Yang and his co-workers have developed a sparse direct package S* for distributed memory computers [46] and there are a number of commercial offerings that can be found through Web searches.

7 Future trends

There seems a never ending demand for the solution of larger and larger problems. For example, some problems from the ASCI program in the United States have dimensions of several million and animal breeders are now solving systems of 20 to 30 million degrees of freedom.

Clearly the size of problem that can be solved by a direct method is very dependent on the matrix structure. For example, a diagonal or tridiagonal system pose no problems when the dimension increases and indeed, if fill-in can be kept low, it is usually possible to solve very large problems by sparse direct factorization. However, for the discretization of three-dimensional partial differential equations, the limitations of direct methods become all too apparent. Although problems from finite-element discretizations of order nearly one million have been solved by MUMPS [11], in my opinion, the most promising set of techniques for the solution of really large problems are those that combine both direct and iterative methods. This can be viewed as a sophisticated preconditioning for an iterative method and is discussed in greater length in the article by Saad and van der Vorst [83].

One of the most promising techniques uses graph partitioning to subdivide the problem, solves the local subproblems by direct methods and uses an iterative method to couple the blocks in the partitioning. This approach is very similar to methods used in the solution of problems from discretizations of PDEs using domain decomposition which can be viewed as permuting the matrix to bordered block diagonal form. However additional preconditioners are used both for the Schur complement and also a coarse preconditioner

[1] For more information on the PARASOL project, see the web site at `http://www.genias.de/projects/parasol/index.html`.

for the overall problem. A good discussion of these preconditioners can be found in the thesis of Luiz Carvalho [23].

It is interesting to surmise what the trends will be. Personally I feel that languages like Fortran 90 combine sufficient object orientation with clarity and efficiency although there is certainly an increasing population who find the lure of novel object oriented languages irresistible. Old techniques continue to be rediscovered as in the revamping of interpretative code approaches, originally developed in the 1970s [60], by Grund [20] who has had some success in solving problems from chemical engineering. The exploitation of sparsity in the right-hand side, for some time pursued in the context of electronic circuit applications and power systems, is now becoming a very powerful tool in the rehabilitation of simplex methods for linear programming.

Acknowledgements

I would like to than my colleagues Patrick Amestoy, Jacko Koster, Xiaoye Li, John Reid, and Jennifer Scott for some helpful remarks on a draft of this paper.

References

[1] M. Adlers. Computing sparse orthogonal factors in MATLAB. Technical Report LiTH-MAT-R-1998-19, Linköping University, 1999.

[2] G. Alaghband. Parallel sparse matrix solution and performance. *Parallel Computing*, 21(9):1407–1430, 1995.

[3] F. L. Alvarado, A. Pothen, and R. Schreiber. Highly parallel sparse triangular solution. In A. George, J. R. Gilbert, and J. W. H. Liu, editors, *Graph Theory and Sparse Matrix Computation*. Springer-Verlag, 1993.

[4] F. L. Alvarado and R. Schreiber. Optimal parallel solution of sparse triangular systems. *SIAM J. Scientific Computing*, 14:446–460, 1993.

[5] F. L. Alvarado, D. C. Yu, and R. Betancourt. Partitioned sparse A^{-1} methods. *IEEE Trans. Power Systems*, 3:452–459, 1990.

[6] P. R. Amestoy. Factorization of large sparse matrices based on a multifrontal approach in a multiprocessor environment. INPT PhD Thesis TH/PA/91/2, CERFACS, Toulouse, France, 1991.

[7] P. R. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Analysis and Applications*, 17(4):886–905, 1996.

[8] P. R. Amestoy and I. S. Duff. Vectorization of a multiprocessor multifrontal code. *Int. J. of Supercomputer Applics.*, 3:41–59, 1989.

[9] P. R. Amestoy, I. S. Duff, and J.-Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. Technical Report RAL-TR-1998-051, Rutherford Appleton Laboratory, 1998. To appear in special issue of *Comput. Methods in Appl. Mech. Eng.* on Domain Decomposition and Parallel Computing.

[10] P. R. Amestoy, I. S. Duff, and J.-Y. L'Excellent. Multifrontal solvers within the PARASOL environment. In B. Kågström, J. Dongarra, E. Elmroth, and J. Waśniewski, editors, *Applied Parallel Computing, PARA'98*, Lecture Notes in Computer Science, No. 1541, pages 7–11, Berlin, 1998. Springer-Verlag.

[11] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. Technical Report RAL-TR-1999-059, Rutherford Appleton Laboratory, 1999.

[12] P. R. Amestoy, I. S. Duff, and C. Puglisi. Multifrontal QR factorization in a multiprocessor environment. *Numerical Linear Algebra with Applications*, 3(4):275–300, 1996.

[13] B. S. Andersen, F. Gustavson, A. Karaivanov, J. Wasniewski, and P. Y. Yalamov. LAWRA - Linear Algebra With Recursive Algorithms. Technical Report UNIC-99-01, UNI-C, Lyngby, Denmark, 1999.

[14] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide, second edition*. SIAM Press, 1995.

[15] E. C. Anderson and Y. Saad. Solving sparse triangular systems on parallel computers. *Int. J. High Speed Computing*, 1:73–95, 1989.

[16] C. Ashcraft and R. Grimes. SPOOLES: An object-oriented sparse matrix library. In *Proceedings of the Ninth SIAM Conference on Parallel Processing*, 1999. See `http://www.netlib.org/linalg/spooles`.

[17] C. Ashcraft and J. W. H. Liu. Robust ordering of sparse matrices using multisection. *SIAM J. Matrix Analysis and Applications*, 19:816–832, 1998.

[18] A. Berger, J. Mulvey, E. Rothberg, and R. Vanderbei. Solving multistage stochastic programs using tree dissection. Technical Report SOR-97-07, Programs in Statistics and Operations Research, Princeton University, Princeton, New Jersey, 1995.

[19] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. SIAM Press, 1997.

[20] J. Borchardt, F. Grund, and D. Horn. Parallel numerical methods for large systems of differential-algebraic equations in industrial applications. Technical Report 382, Weierstraß-Institut für Angewandte Analysis und Stochastik, Berlin, 1997.

[21] J. R. Bunch, L. Kaufman, and B. N. Parlett. Decomposition of a symmetric matrix. *Numerische Mathematik*, 27:95–110, 1976.

[22] D. A. Calahan. Parallel solution of sparse simultaneous linear equations. In *Proceedings 11th Annual Allerton Conference on Circuits and System Theory, University of Illinois*, pages 729–735, 1973.

[23] L. M. Carvalho. Preconditioned Schur complement methods in distributed memory environments. INPT PhD Thesis TH/PA/97/41, CERFACS, Toulouse, France, 1997.

[24] T. A. Davis and I. S. Duff. An unsymmetric-pattern multifrontal method for sparse LU factorization. *SIAM J. Matrix Analysis and Applications*, 18(1):140–158, 1997.

[25] T. A. Davis and P. C. Yew. A nondeterministic parallel algorithm for general unsymmetric sparse LU factorization. *SIAM J. Matrix Analysis and Applications*, 11:383–402, 1990.

[26] J. W. Demmel, J. R. Gilbert, and X. S. Li. SuperLU users' guide. Technical report, Computer Science Division, U. C. Berkeley, Berkeley, California, February 1995. (available from netlib).

[27] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, Xiaoye S. Li, and J. W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Analysis and Applications*, 20:720–755, 1999.

[28] F. Dobrian, G. Kumfert, and A. Pothen. Object-oriented design for sparse direct solvers. Technical Report 99-2, Institute for Computer Applications in Science and Engineering (ICASE), MS 132C, NASA Langley Research Center, Hampton, VA 23681-0001, USA, 1999.

[29] J. J. Dongarra. Performance of various computers using standard linear algebra software. Technical Report CS-89-85, University of Tennessee, Knoxville, Tennessee, 1999. Updated version at Web address `http://www.netlib.org/benchmark/performance.ps`.

[30] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart. *LINPACK User's Guide*. SIAM Press, Philadelphia, 1979.

[31] J. J. Dongarra, J. J. Du Croz, S. Hammarling, and R. J. Hanson. An extented set of Fortran Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw.*, 14:1–17, 1988.

[32] J. J. Dongarra and V. Eijkhout. Numerical linear algebra and software. Technical report, University of Tennessee, 1999. *Journal Computational and Applied Mathematics*, this volume.

[33] J. J. Dongarra, J. Du Croz, I. S. Duff, and S. Hammarling. A set of Level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw.*, 16:1–17, 1990.

[34] 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, 1998.

[35] I. S. Duff. Full matrix techniques in sparse Gaussian elimination. In G. A. Watson, editor, *Numerical Analysis Proceedings, Dundee 1981*, Lecture Notes in Mathematics 912, pages 71–84, Berlin, 1981. Springer-Verlag.

[36] I. S. Duff. Design features of a frontal code for solving sparse unsymmetric linear systems out-of-core. *SIAM J. Scientific and Statistical Computing*, 5:270–280, 1984.

[37] I. S. Duff. Parallel implementation of multifrontal schemes. *Parallel Computing*, 3:193–204, 1986.

[38] I. S. Duff. Multiprocessing a sparse matrix code on the Alliant FX/8. *J. Comput. Appl. Math.*, 27:229–239, 1989.

[39] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, Oxford, England, 1986.

[40] I. S. Duff, R. G. Grimes, and J. G. Lewis. The Rutherford-Boeing Sparse Matrix Collection. Technical Report RAL-TR-97-031, Rutherford Appleton Laboratory, 1997. Also Technical Report ISSTECH-97-017 from Boeing Information & Support Services, Seattle and Report TR/PA/97/36 from CERFACS, Toulouse.

[41] I. S. Duff and J. Koster. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. *SIAM J. Matrix Analysis and Applications*, 20(4):889–901, 1999.

[42] I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. Technical Report RAL-TR-1999-030, Rutherford Appleton

Laboratory, 1999. Also appeared as Report TR/PA/99/13, CERFACS, Toulouse, France.

[43] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear systems. *ACM Trans. Math. Softw.*, 9:302–325, 1983.

[44] 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, Rutherford Appleton Laboratory, 1999. To appear in *Parallel Computing*.

[45] B. Dumitrescu, M. Doreille, J.-L. Roch, and D. Trystram. Two-dimensional block partitionings for the parallel sparse Cholesky factorization. *Numerical Algorithms*, 16(1):17–38, 1997.

[46] C. Fu, X. Jiao, and T. Yang. Efficient sparse LU factorization with partial pivoting on distributed memory architectures. Technical report, University of california at Santa Barbara, 1997. Submitted to *IEEE Trans. on Parallel and Distributed Systems*.

[47] K. A. Gallivan, B. A. Marsolf, and H. A. G. Wijshoff. Solving large nonsymmetric sparse linear systems using MCSPARSE. *Parallel Computing*, 22:1291–1333, 1996.

[48] A. George. Nested dissection of a regular finite element mesh. *SIAM J. Numerical Analysis*, 10:345–363, 1973.

[49] A. George and J. W. H. Liu. The design of a user interface for a sparse matrix package. *ACM Trans. Math. Softw.*, 5(2):139–162, 1979.

[50] A. George and J. W. H. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Englewood Cliffs, New Jersey: Prentice-Hall, 1981.

[51] A. George, J. W. H. Liu, and E. G. Ng. User's guide for SPARSPAK: Waterloo sparse linear equations package. Technical Report CS-78-30 (Revised), University of Waterloo, Canada, 1980.

[52] A. George, J. W. Poole, and R. Voigt. Incomplete nested dissection for solving $n \times n$ grid problems. *SIAM J. Numerical Analysis*, 15:663–673, 1978.

[53] J. P. Geschiere and H. A. G. Wijshoff. Exploiting large grain parallelism in a sparse direct linear system solver. *Parallel Computing*, 21(8):1339–1364, 1995.

[54] J. R. Gilbert and J. W. H. Liu. Elimination structures for unsymmetric sparse LU factors. *SIAM J. Matrix Analysis and Applications*, 14:334–354, 1993.

[55] A. Gupta, G. Karypis, and V. Kumar. Highly scalable parallel algorithms for sparse matrix factorization. *IEEE Trans. Parallel and Distributed Systems*, 8:502–520, 1997.

[56] A. Gupta, M. Joshi, and V. Kumar. WSSMP: Watson Symmetric Sparse Matrix Package. Users Manual: Version 2.0β. Technical Report RC 20923 (92669), IBM T. J. Watson Research Centre, P. O. Box 218, Yorktown Heights, NY 10598, July 1997.

[57] A. Gupta, G. Karypis, and V. Kumar. Highly scalable parallel algorithms for sparse matrix factorization. Technical Report TR-94-63, Department of Computer Science, University of Minnesota, 1994.

[58] F. Gustavson, A. Henriksson, I. Jonsson, B. Kågström, and P. Ling. Recursive blocked data formats and BLAS's for dense linear algebra algorithms. In B. Kågström, J. Dongarra, E. Elmroth, and J. Waśniewski, editors, *Applied Parallel Computing, 4th International Workshop, PARA'98*, pages 195–206, Berlin, 1998. Springer-Verlag.

[59] F. G. Gustavson. Recursion leads to automatic variable blocking for dense linear algebra algorithms. *IBM J. Research and Development*, 41:737–755, 1997.

[60] F. G. Gustavson, W. M. Liniger, and R. A. Willoughby. Symbolic generation of an optimal Crout algorithm for sparse systems of linear equations. *J. ACM*, 17(1):87–109, 1970.

[61] M. T. Heath and P. Raghavan. Performance of a fully parallel sparse solver. In *Proceedings of SHPCC '94, Scalable High-Performance Computing Conference. May 23-25, 1994, Knoxville, Tennessee*, pages 334–341, Los Alamitos, California, 1994. IEEE Computer Society Press.

[62] B. Hendrickson and R. Leland. The CHACO User's Guide. Version 2.0. Technical Report SAND94-2692, Sandia National Laboratories, Albuquerque, October 1994.

[63] B. Hendrickson and E. Rothberg. Improving the runtime and quality of nested dissection ordering. *SIAM J. Scientific Computing*, 20:468–489, 1998.

[64] HSL. *Harwell Subroutine Library. A Catalogue of Subroutines (Release 12).* AEA Technology, Harwell Laboratory, Oxfordshire, England, 1996. For information concerning HSL contact: Dr Nick Brealey, HSL Manager, AEA Technology Products & Systems, Culham Science Centre, Oxon OX14 3ED, England (tel: +44-1235-463404, fax: +44-1235-463480, email: hsl@aeat.co.uk).

[65] G. Karypis and V. Kumar. Parallel multilevel graph partitioning. Technical Report TR-95-036, Department of Computer Science, University of Minnesota, May 1995.

[66] G. Karypis and V. Kumar. METIS: Unstructured graph partitioning and sparse matrix ordering system. Technical Report TR 97-061, Department of Computer Science, University of Minnesota, Minnesota, 1997.

[67] J. Koster. On the parallel solution and the reordering of unsymmetric sparse linear systems. INPT PhD Thesis TH/PA/97/51, CERFACS, Toulouse, France, 1997.

[68] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Softw.*, 5:308–323, 1979.

[69] X. S. Li and J. W. Demmel. Making sparse Gaussian elimination scalable by static pivoting. In *Proceedings of Supercomputing*, Orlando, Florida, November 1998.

[70] J. W. H. Liu. On the storage requirement in the out-of-core multifrontal method for sparse factorization. *ACM Trans. Math. Softw.*, 12:249–264, 1987.

[71] J. W. H. Liu. Reordering sparse matrices for parallel elimination. *Parallel Computing*, 11:73–91, 1989.

[72] H. M. Markowitz. The elimination form of the inverse and its application to linear programming. *Management Science*, 3:255–269, Apr. 1957.

[73] P. Matstoms. Parallel sparse QR factorization on shared memory architectures. *Parallel Computing*, 21:473–486, 1995.

[74] E. G. Ng and B. W. Peyton. Block sparse Cholesky algorithms on advanced uniprocessor computers. *SIAM J. Scientific Computing*, 14:1034–1056, 1993.

[75] B. W. Peyton, A. Pothen, and X. Yuan. Partitioning a chordal graph into transitive subgraphs for parallel sparse triangular solution. Technical Report ORNL/TM-12270, Engineering Physics and Mathematics Division, Oak Ridge National Laboratory, Tennessee, December 1992.

[76] A. Pothen, H. D. Simon, and K. P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Analysis and Applications*, 11(3):430–452, 1990.

[77] P. Raghavan. Parallel ordering using edge contraction. *Parallel Computing*, 23(8):1045–1067, 1997.

[78] P. Raghavan. Efficient parallel sparse triangular solution using selective inversion. *Parallel Processing Letters*, 8(1):29–40, 1998.

[79] E. Rothberg. Exploring the tradeoff between imbalance and separator size in nested dissection ordering. Technical Report Unnumbered, Silicon Graphics Inc, 1996.

[80] E. Rothberg. Performance of panel and block approaches to sparse Cholesky factorization on the iPSC/860 and Paragon multicomputers. *SIAM J. Scientific Computing*, 17(3):699–713, 1996.

[81] E. Rothberg and A. Gupta. An evaluation of left-looking, right-looking and multifrontal approaches to sparse Cholesky factorization on hierarchical-memory machines. Technical Report STAN-CS-91-1377, Department of Computer Science, Stanford University, 1991.

[82] E. Rothberg and R. Schreiber. Improved load distribution in parallel sparse Cholesky factorization. Technical Report 94-13, Research Institute for Advanced Computer Science, 1994.

[83] Y. Saad and H. A. van der Vorst. Iterative solution of linear systems in the 20-th century. Technical report, Department of Mathematics, University of Utrecht, 1999. *Journal Computational and Applied Mathematics*, this volume.

[84] W. F. Tinney and J. W. Walker. Direct solutions of sparse network equations by optimally ordered triangular factorization. *Proc. of the IEEE*, 55:1801–1809, 1967.

[85] A. F. van der Stappen, R. H. Bisseling, and J. G. G. van de Vorst. Parallel sparse LU decomposition on a mesh network of transputers. *SIAM J. Matrix Analysis and Applications*, 14:853–879, 1993.

[86] R. C. Whaley. LAPACK Working Note 73 : Basic Linear Algebra Communication Subprograms: analysis and implementation across multiple parallel architectures. Technical Report CS-94-234, Computer Science Department, University of Tennessee, Knoxville, Tennessee, May 1994.

[87] J. H. Wilkinson and C. Reinsch. *Handbook for Automatic Computation. Volume II Linear Algebra*. Springer-Verlag, Berlin, 1971.

[88] Z. Zlatev, J. Waśniewski, P. C. Hansen, and Tz. Ostromsky. PARASPAR: a package for the solution of large linear algebraic equations on parallel computers with shared memory. Technical Report 95-10, Technical University of Denmark, Lyngby, 1995.

[89] Z. Zlatev, J. Waśniewski, and K. Schaumburg. Introduction to PARASPAR. Solution of large and sparse systems of linear algebraic equations, specialised for parallel computers with shared memory. Technical Report 93-02, Technical University of Denmark, Lyngby, 1993.