Department of Computer Science

University of Manchester

Manchester M13 9PL, England

Technical Report Series

UMCS–93–1–5

G.A. Hedayat

Numerical Linear Algebra and Computer Architecture: An Evolving Interaction

Numerical Linear Algebra and Computer Architecture: An Evolving Interaction*

G.A. Hedayat†
Centre for Novel Computing
Department of Computer Science, University of Manchester,
Oxford Road, Manchester M13 9PL, UK
gholam@cs.man.ac.uk

Abstract

The aim of this report is to highlight the evolving interaction between computer architecture on the one hand and the design and analysis of algorithms for numerical linear algebra on the other. This report can best be described as a collection of pointers to aspects of the development processes of numerical linear algebra, and to various existing surveys and software. The rich variety of activity in design, analysis, comparison and evaluation of algorithms, new and old, is examined in the context of rapid technological and architectural advance. The characteristics of algorithms for linear algebra are discussed without going into mathematical details: There are references to important surveys that provide such details. Reflections on the practices of numerical linear algebra will, it is hoped, point to possibilities for and limitations of automating the "mapping" of algorithms onto high performance computers.

†The author greatly acknowledges the support of the Science and Engineering Research Council (SERC Grant GR/F/04292) for this work.

Technical Reports issued by the Department of Computer Science, Manchester University, are available by anonymous ftp from ml.cs.man.ac.uk (130.88.13.4) in the directory /pub/TR. The files are stored as PostScript, in compressed form, with the report number as filename. Alternatively, reports are available by post from The Computer Library, Department of Computer Science, The University, Oxford Road, Manchester M13 9PL, U.K.

1 Introduction

Imagine yourself in 1947. With the prospect of the first digital computer on the horizon, there is an increased interest within the mathematical community in numerical methods. The theoretical debate centers around the impact of *round-off error* on the accuracy of solutions. The model problem is the numerical inversion of matrices of high order: i.e. size greater than 10×10. The algorithm under consideration is none other than Gauss's elimination method. The theoretical battle is joined by such notables as von Neumann, Goldstine [110] and Turing [139]. The purpose is to derive a rigorous *error estimate* due to round-off for the solution of sets of linear equations and inverting of matrices of high order.

One reason for taking up this subject, von Neumann and Goldstine argue, is the considerably diverging opinion as to the precision required when inverting matrices of order greater than 10. Turing states that

> "the belief that the elimination method and other 'direct' methods of solution lead to large errors has been responsible for a recent search ..."

for iterative methods. Theoretical resolution of the question had to await Wilkinson's work in the early 1960's [143], yet these early papers established a technique and standard far exceeding in importance the detailed result. A new era in numerical linear algebra had begun, characterized by the impact of the computational hardware platform on design, analysis, and evaluation of algorithms.

The aim of this report is to highlight features of the interaction between computer architecture and the design and analysis of algorithms for numerical linear algebra. There is no attempt to present a detailed historical survey of the field over the last 50 years. For one, the amount of work done in the area makes it impossible to do justice to such a survey. A recent bibliography on parallel and vector numerical algorithms, by Ortega, Voigt and Romine [112] contains more than 2000 references. Moreover, there are excellent surveys and monographs covering specific subdomains of the field. These will be referenced, as appropriate. This report can best be described as a collage that aims to give pointers to certain aspects of the development processes of numerical linear algebra, existing surveys and software. Spin-offs in the development of tools and compilers, and in the impact of requirements for numerical linear algebra on hardware design, though interesting, are not addressed here. Needless to say, the choice of issues and references are tainted by the author's personal bias.

2 Numerical Linear Algebra

According to Parlett [115], numerical linear algebra begins with three basic tasks:

- Solve systems of linear equations, $Ax = b$.
- Compute eigenvalues and eigenvectors of A.
- Solve least square problems, $\min \| b - Ax \|_2$ over all x.

Associated with these tasks are various matrix factorization algorithms, such as LU, Cholesky, QR, SVD, Schur, and generalized Schur. Often, one deals with a sequence of factorizations depending on a parameter. This leads to the problem of updating the factorizations. In order to assess the quality of a computed solution, it is necessary to estimate the *condition* of solution [145]. An excellent comprehensive reference is Golub and Van Loan's text on Matrix Computation [69].

The vitality of the field of matrix computation stems from its importance to a wide area of scientific and engineering applications on the one hand, and the advances in computer technology on the other. As early as 1958, Lanczos [93] observed that

> "it is not advisable to assume that there is only one method by which a certain numerical problem can be solved. It is more desirable to have a collection of programming procedures at our disposal which can be adjusted to the specific demands of the situation".

Therein lies the reason for the multiplicity of algorithms that are used for an apparently small number of tasks.

The structural properties of the underlying matrices is probably the most important factor in influencing the choice of algorithm, and is thus afforded the most coverage in texts on numerical linear algebra. An eigenvalue solver suitable for *dense* matrices is not necessarily appropriate for *sparse* matrices. The size of matrices involved in computational tasks is often decisive in the selection between *iterative* or *direct* methods. In numerical linear algebra, structural properties, such as *symmetry, positive definiteness, bandedness*, etc., when properly taken into account, may result in more efficient and/or stable procedures.

Great progress has been made in numerical linear algebra since the 1950's. There are now effective software packages for the basic tasks outlined above for sequential machines [45, 135, 62, 51, 52, 54, 66, 87]. Significant advances have been made in developing a portable linear algebra library, LAPACK [5], for high performance, shared memory, vector and parallel computers. There is ongoing work to extend LAPACK to distributed memory architectures [7].

The diversity of hardware platforms, from workstation to high performance computers, virtual versus physical organization of memory, uniform versus nonuniform memory access in multiprocessor systems, has frequently provided the impetus for new research. The rapid pace of change in hardware and software technology has led to the reappraisal of methods that had previously been deemed uncompetitive, the rediscovery of old techniques, and the design of new (or restructuring of existing) algorithms.

3 Stability

3.1 Error Analysis

Backward error analysis and the stability of algorithms of numerical linear algebra were developed and elucidated by James Wilkinson [143]. In the 11^{th} von Neumann lecture, Wilkinson gave a historical survey of round-off error analysis [145]. In this address the two essential features of modern error analysis were described as:

- The errors in computation are interpreted in terms of perturbations in the given matrix for which error bounds are given. The computed quantities are *exact* for this perturbed matrix.
- There is no attempt to compare computed quantities with those which would have been obtained by exact computation.

The advantage of this approach is that round-off errors are treated on the same basis as errors in the original data, and the extensive tools of perturbation theory can be used in investigating the stability of algorithms.

Wilkinson also observed that, in papers by von Neumann and Goldstine [110] in 1947 and Givens [67] in 1954, both landmarks in error analysis, the use of fixed point arithmetic led to the slow pace of development in error analysis. The essence of their techniques was obscured by the repeated need to rescale in order to avoid overflow. It also led to confusion over the use of *pivoting* in the factorization of positive definite matrices, as presented in [110]. The development of floating point hardware, and the adoption of floating point arithmetic as the mode of operation, paved the way for rapid progress in modern error analysis. Wilkinson adds [145]:

> "with freedom to concentrate on essentials, it soon became apparent that the most common requirement for stability in matrix factorization methods for the solution of linear systems, is some control of the size of the elements in the factor relative to elements of the matrix A .... Moreover, the simplest way of guaranteeing this control is to use transformations based on orthogonal group".

Wilkinson remarks that, in the case of unitary similarities, a general error analysis covers many algorithms, thus reducing the length of proofs [144]. He sums up the importance of *a priori* error analysis as:

> "The main object of error analysis is to expose the potential instabilities, if any, of an algorithm, so that with the insight thus obtained one may be led to improved algorithms."

He cites the QR algorithm as a striking example of its success.

3.2 Stability of Fast Algorithms

In the 1980's, with the advent of VLSI hardware, and the possibility of "hard wiring" linear algebra algorithms, there was renewed interest in algorithms for the solution of *Toeplitz* systems of linear equations which arise in signal processing [69, 4, 85, 65]. Toeplitz matrices have identical elements along the diagonals. Exploiting this special structure, their solution can be found in $O(n^2)$ operations using the Levinson, Durbin, Trench, and Zohar algorithms [69]. There are also several $O(n \log^2 n)$ algorithms, based on the Gohberg-Semencule formula for the inversion of Toeplitz matrices [17]. Although these algorithms provide an attractive alternative in terms of the number of floating point operations, the question of their stability had to be resolved. To borrow a phrase from Wilkinson:

> "Attractive mathematics does not protect one from the rigors of digital computation".

Bunch [18] addressed the stability question comprehensively. He showed that, when the Toeplitz matrix is not symmetric positive definite, all the above-mentioned algorithms are unstable. In [19], he extended the concept of stability to clarify certain confusions regarding stability for positive definite Toeplitz systems. As these concepts extend to other problems in numerical linear algebra, they are reproduced below:

Definition 1 *An algorithm for solving linear equations is* **stable** *for a class of matrices $\mathcal{A}$ if for each $A \in \mathcal{A}$ and for each b, the computed solution $\hat{x}$ to $Ax = b$ satisfies $\hat{A}\hat{x} = \hat{b}$ for some $\hat{A}$ and $\hat{b}$, where $\hat{A}$ is close to A and $\hat{b}$ is close to b.*

The closeness is in the sense of some norm, say ℓ_2, and depends on the desired accuracy. In Wilkinson's definition of stability $\hat{A}$ need not belong to class $\mathcal{A}$.

Under this definition, Gaussian elimination with pivoting is stable on the class $\mathcal{A}_1$ of nonsingular matrices. Similarly, Cholesky factorization is stable for the class $\mathcal{A}_2$ of symmetric positive definite matrices. Bunch observes that, from the error analysis, one often gets a stronger result. For example, in the Cholesky case, $\hat{A}$ is not only close to A but also in $\mathcal{A}_2$. This forms the basis for the extension of the concept of stability in [19].

Definition 2 *An algorithm for solving linear equations is* **strongly stable** *for a class of matrices $\mathcal{A}$ if it is* **stable** *and $\hat{A}$ is in $\mathcal{A}$.*

According to Theorem 3 of [19], Gaussian elimination with partial or complete pivoting is strongly stable in the class of nonsingular matrices. Similarly, LU decomposition without pivoting is strongly stable in the class of nonsingular diagonally dominant matrices.

Bunch observes that most users are less interested in whether or not their computed solution solves a nearby problem exactly; they are more concerned about the *accuracy* of the computed solution. Thus, restricting his attention to well-conditioned matrices, he introduces the following, weaker notion of stability.

Definition 3 *An algorithm for solving linear equations is* **weakly stable** *for a class of matrices $\mathcal{A}$ if for each well conditioned $A \in \mathcal{A}$ and for each b the computed solution $\hat{x}$ to $Ax = b$ is such that $\frac{||x-\hat{x}||}{||x||}$ is small.*

Weak stability of an algorithm for a class $\mathcal{A}$ means that the computed solution for every well-conditioned matrix $A \in \mathcal{A}$ is accurate. Stability implies weak stability but not *vice versa*. Within this framework, Bunch [19] interprets a result of Cybenko [30] on the stability of the Levinson-Durbin algorithm for positive definite Toeplitz matrices, as follows:

> "The Levinson, Durbin, Trench, and Zohar algorithms are weakly stable for the class of symmetric positive definite Toeplitz matrices".

He observes that the algorithms may even be stable for this class, but a proof using standard backward error analysis remains elusive. For fast, $O(n \log^2 n)$ algorithms, even weak stability for the class of symmetric positive definite Toeplitz matrices remains an open question.

3.3 New Approach to Error Analysis

By combining backward error analysis with perturbation theory, we can estimate the accuracy of an algorithm. In conventional error analysis, the notion of *closeness* was in the sense of the norm. This can lead to a pessimistic estimate of the accuracy of a solution. More recent work has concentrated on the use of an component-wise notion of closeness [80, 32, 8]. The component-wise measure of error reflects the *structure* of the matrix, for example, its sparsity. A brief outline of this new perturbation theory is given in [6]: There, the interested reader will find references to the detailed treatment of error analysis in this mode for much of numerical linear algebra. The basic idea is to use $rel_A(\delta A) \equiv \max_{ij} \frac{|\delta A_{ij}|}{|A_{ij}|}$, the largest relative change in an element, instead of bounding δA by its norm. As part of the LAPACK project [6], the new approach has been applied to linear system solvers, the least squares problem, and bidiagonal SVD, among others. The algorithms are usually slight variations on the conventional algorithms, although bidiagonal SVD [32] has components which are novel. All algorithms have an in-built monitoring of their accuracy and produce useful error bounds. An interesting outcome of this approach has been to show that Jacobi's method for the symmetric eigenvalue problem is more accurate than the QR algorithm when the matrix is positive definite [34].

4 Speed of Execution

4.1 The Traditional View

A common measure of the efficiency of algorithms for numerical linear algebra is the number of floating point operations that have to be performed, expressed as a function of the problem size. Whilst this provides a quantitative metric for comparison of competing algorithms, the actual performance even on a uni-processor, is affected by such factors as the relative cost of a floating point operation (compared to that of a memory access), the organization of memory, and the capabilities of compilers. As the the architecture of the hardware platform, and its associated software environment, has been evolving rapidly over the last 40 years, it is hardly surprising that, as well as the development of new algorithms, there is constant reappraisal and restructuring of old methods. There is also a need for experimental evaluation of the efficiency of competing techniques. This phenomenon pre-dates the appearance of modern high performance supercomputers. In the preface to his classic monograph [82], Alston Householder describes the literature in the 1950's as follows:

> "Among the many papers in the literature that allegedly give new methods, whether direct or otherwise, the great majority are rediscoveries, or as in the method of Crout, present a more efficient organization of data or ordering of steps".

This is not to detract from the value of these rediscoveries, and restructurings, but rather to emphasize the scope of research in the field. Often, rediscoveries have become a basis from which to develop new results in numerical linear algebra. Lanczos, in [93], argues prophetically:

> " It may likewise happen that a certain procedure, discarded for practical reasons, reappears on the platform because certain changes in the technical design of machine make the method practically feasible, while before it commanded only theoretical interest".

As an example, let us look at the Jacobi Method for the symmetric eigenvalue problem. The method was initially developed by Jacobi, in the 19^{th} century, in order to prove that the eigenvalues of a real symmetric matrix are real. In the classical Jacobi method, the symmetric matrix is reduced to diagonal form by a sequence of elementary plane rotations, annihilating the largest off-diagonal element at each step. With the development of the digital computer, significant interest was shown in the Jacobi method as it was realized that the traditional mathematical approach of finding the characteristic equation, or the companion form by similarity transformations, was catastrophic in the presence of round-off errors [145]. Goldstine, Murray, and von Neumann [68] gave a detailed account of the classical Jacobi method, with prescriptions for techniques on how to implement square root, sine, and cosine in fixed point arithmetic.

The difficulty with the classical Jacobi method was that it required scanning of the whole matrix (for the largest element) after each step. In order to increase the efficiency of the algorithm, Gregory [72] proposed in 1953 annihilating the off-diagonal elements in a row or column cyclic fashion. Even though codes based on the method were running on the ILLIAC and UNIVAC 1103 A, the proof of convergence of row and column cyclic Jacobi, and its rate of convergence under mild restriction on the angle of rotation, did not appear until several years later, in Forsythe and Henrici [55], Henrici [77], Wilkinson [146], and others. It is interesting to note that the code for the IBM 704, for example, could compute all eigenvalues and eigenvectors of a real symmetric matrix of order ≤ 68. Typical execution time varied from about 1 minute, for an 8×8 matrix, to 25 minutes, for an 43×43 matrix [141]. Pope and Tomkin [118] introduced a threshold Jacobi method, omitting annihilation of elements smaller than some threshold, to further increase the efficiency of the row-cyclic Jacobi method. Experiments on SWAC, using a Hilbert matrix of order 60, suggested comparable accuracy to row-cyclic Jacobi, but in a shorter time. The threshold was decreased after each sweep (from an initially large value) to ensure a good rate of convergence.

For an $n \times n$ symmetric matrix there are $\frac{n(n-1)}{2}$ strictly upper triangular elements. A cyclic Jacobi method visits each off-diagonal element exactly once in any sweep, in a predetermined order. The question of the convergence of more general cyclic Jacobi methods was satisfactorily settled only if thresholds were used [144]. Nazareth [109] gives examples where an element is pushed around ahead of the sequence of rotations and never annihilated.

On sequential machines, however, the competition to Jacobi and Jacobi-like methods came from another quarter: The QR algorithm for symmetric matrices turned out to be more efficient. If we simply count the number of floating point operations, then two sweeps of Jacobi are roughly equivalent to a complete QR, including the accumulation of transformations [69]. Thus QR became the method of choice for most problems.

4.2 Reordering and Parallelism

In the early 1970's, with the appearance of parallel computers, such as the ILLIAC IV, there was a renewed interest in the Jacobi method for the symmetric eigenvalue problem. It was shown by Sameh [126] that it is possible to modify the Jacobi method so as to annihilate several off-diagonal elements simultaneously, whereas, in the practical QR algorithm, the shift, in each iteration reduces the potential parallelism of the algorithm. To maximize the degree of parallelism, it is desirable to perform simultaneously as many noninteracting Jacobi rotations as possible. With wider acceptance of parallel processing in the 1980's, many authors analyzed seemingly different "parallel Jacobi orderings" for SIMD, MIMD architectures as well as for systolic arrays [107, 16, 102, 25, 137]. Because many of these orderings were developed for systolic arrays or SIMD machines, they also tried to minimize data movement, by restricting it to nearest neighbour communication, and to make it uniform between successive stages. In [102], Luk and Park showed that, under certain assumptions, the well known Jacobi orderings are mathematically equivalent and, in [103], they proved the convergence for two parallel orderings.

It is evident that the search for new Jacobi orderings with high degrees of parallelism as a by-product has led to new theoretical questions regarding convergence of these methods. Recently, Shroff and Schreiber made a significant contribution by uncovering a large class of provably convergent parallel cyclic Jacobi and block Jacobi methods [134]. They introduce a class of orderings, which they called "P–Wavefront", that include all orderings related to row-cyclic ordering through any known convergence-preserving transformations. They show that the class of P–Wavefront orderings is characterized by simple properties that can be verified in polynomial time. Moreover, they show that there are some parallel block Jacobi methods that fall in to this class. However, they observe that there is a provably convergent ordering in Nazareth's class [109] that is not a P–Wavefront ordering; similarly some of the Shroff-Schreiber orderings, notably block wavefront ordering, is not in Nazareth's class. Finally, they observe that there are orderings for which no proof of convergence exists. As an example, the Brent-Luk ordering [16] for even values of n is cited.

The same trends are noticeable in computing the SVD, CSD, GSVD, QRD, and Schur decompositions. Here, algorithms first suggested by Hestenes in [78] and Kogbetliantz in [89] were developed and adapted for the new parallel machines. Again, the main motivation for selection (as in the case of Jacobi), was the potential speedup available due to the high degree of parallelism of these algorithms [39, 101, 134].[1]

In summary, the cyclic Jacobi method for the symmetric eigenvalue problem, and Jacobi-like methods for SVD, have once again become competitive because they offer potential parallelism that may be exploitable on parallel hardware, once a suitable parallel ordering of Jacobi rotations has been chosen so as to maximize the number of simultaneous rotations. The new theoretical questions, regarding the equivalence and convergence of a class of parallel cyclic Jacobi and block Jacobi methods, have been successfully addressed by several authors.

[1] It is interesting to note that Jacobi-like methods were also adopted for implementation of SVD on mini-computers by Nash [108]. This had probably more to do with the storage demand of Jacobi-like methods. One can think of each annihilation step as a 2×2 eigenvalue (or SVD) sub-problem, reducing the workspace required.

4.3 Reordering and Rate of Convergence

There is a subtle distinction between reordering the computation in cyclic Jacobi and, say, the more familiar reordering of matrix multiplication [46]. In the latter case, the numerical properties of the reordered algorithm are exactly the same as the original, down to and including the effects of round-off error. However, in the cyclic Jacobi algorithm, for example, the question of convergence of an arbitrary cyclic Jacobi remains an open question [134]. This distinction, in turn, influences the degree to which restructuring can be automated for all the algorithms of numerical linear algebra. If, in addition to ordering for parallelism, *blocking* is introduced (in order to reduce the impact of non-uniform access to various levels of memory), then the distinction becomes even more pronounced.

An explicit analysis of the impact of reordering (so as to increase parallelism) on the rate of convergence of an algorithm was carried out by Kuo and Chan [91]. The multicolor ordering scheme is often used to increase the parallelism of iterative methods for the solution of linear equations arising from discretization of elliptic partial differential equations [111]. Kuo and Chan use a unified two color Fourier analysis to analyze the impact of the red/black ordering on the convergence rate of SOR, SSOR, ILU and MILU preconditioners for preconditioned iterative methods, and, finally, multigrid (MG) methods. They show that, whereas the rate of convergence for SOR and MG methods with the red/black ordering does not deteriorate, the SSOR, and ILU, MILU preconditioned iterative methods are inferior when compared to the corresponding natural order. This two color Fourier analysis allows the analytical determination of the optimal relaxation parameter for red/black SSOR, and the eigenstructure of the preconditioned linear operator with SSOR, ILU, and MILU preconditioners. They show that the condition number of the preconditioned operator with red/black ordering is, in general, one order of magnitude higher than that of its naturally ordered counterpart. Hence, the convergence rate is sacrificed in order to increase parallelism. Kuo and Chan pose the fundamental question:

> “What is the inherent feature of these algorithms that make them behave so differently with respect to reordering?”

They observe that, in the case of the red/black SSOR, ILU, and MILU preconditioners, the question can partly be answered by the fact that red/black preconditioners use only local information, whereas the naturally ordered preconditioners make use of some global information. For further qualitative discussion of the relationship between rate of convergence, degree of parallelism, and local versus global features of an algorithm, see [21].

The search for preconditioners that are easy to parallelize, with satisfactory convergence rates, has thus become an important research topic in the iterative solution of linear systems; see, for example, [92, 22, 23, 123].

5 Storage Requirements

Consider a PDE problem (Euler Equations) in a three dimensional domain. If there are 5 state variables discretized over a $50 \times 50 \times 50$ grid, the resulting matrices have size $n = 625000$

with bandwidth of approximately 25000. The general consensus is that iterative linear system solvers are the method of choice for such a problem: One of the main attractions is that only the product of the matrix times a vector is needed to find an approximate solution, and thus the matrix need not be stored in its entirety. Similarly, for eigenvalue problem associated with large matrices, often only a few eigenvalues are of interest. Lanczos method, first suggested in the 1950's, involves partial tridiagonalization of a large sparse symmetric matrix for computing some of its eigenvalues [69]. The advantage of Lanczos algorithm over, say, QR is that no intermediate, full submatrices are generated and that acceptable approximations to extreme eigenvalues tend to emerge long before the process of tridiagonalization is complete. Thus Lanczos algorithm, already recognized as an important method for large sparse matrix computation, has become an intense focus of research activity in recent years. For a survey of recent results see [58].

Among iterative algorithms for the solution of large linear systems, variants of the conjugate gradient (CG) method stand out. Hestenes and Stiefel [79] proposed the conjugate gradient method (CG) in 1952 for the solution of positive definite linear systems. The true potential of the technique was not appreciated until the 1970's. Since then, considerable research effort in numerical linear algebra has been devoted to the generalization of CG to non-Hermitian and indefinite linear systems. Saad [123] makes the interesting observation that the rise in popularity of Krylov Subspace Methods, of which CG is an example, coincided with the appearance of the first supercomputers in the mid 1970's. The most widely used CG-type method is the generalized minimum residual algorithm (GMRES) of Saad and Schultz [124]. GMRES and other related schemes suffer from the fact that work and storage requirements grow linearly with the number of iterations, often requiring restarts, with subsequent slow convergence. For this reason, in the late 1980's, research focussed on methods that can be implemented with a low (and roughly constant) storage requirement per iteration. The new algorithms proposed with this characteristic are related to the non-symmetric Lanczos process. For an excellent recent survey, see [58]. An earlier survey paper, by Axelsson [10], deals with preconditioning of iterative methods. Saad surveys modern iterative techniques, such as Krylov subspace methods, and their implementation on supercomputers in [123].

6 Patterns of Data Access

6.1 Impact of Memory Organization

The diversity of organization of memory in high performance computers had a significant impact on the efficiency of the algorithms of numerical linear algebra. It has led to reappraisal, restructuring and the development of new algorithms. Dongarra *et al.* [46] give an illustration of this phenomenon by considering the implementation of matrix multiplication on the Cyber 205. The Cyber 205 has a *virtual memory* giving the programmer the illusion of a very large address space. In systems with virtual memory, data is stored on pages, and the operating system manages the memory. In the Cyber 205, one page contains 65536 words. Thus, for example, in a 1024×1024 matrix, 64 columns of the matrix can be stored on one such page. In [46], it is assumed that the available main memory consists of 16 pages. If an accessed item of data does not reside on a page in the main memory, then a *page fault* occurs, and the appropriate page is fetched from secondary store, incurring a significant time penalty. Thus, efficient implementation requires organizing computation so as to reuse a segment of data residing on a page, before moving on to access the next page. The example considered is matrix multiplication, $C = AB + C$, where all matrices are assumed to be 1024×1024, thus requiring 48 pages of store. It is assumed that matrices are stored column-wise, as is normal in Fortran. In [46], the authors first compare the inner-product approach, (ijk), with the column-wise update of matrix C, (jki), ignoring vectorization. The I/O time due to page faults in the ijk approach is estimated at 93 days, whereas, in the jki approach, it reduce to 2.25 hours. Even though the jki approach is clearly superior to the ijk approach, the I/O time is still significantly higher than the vectorized CPU time (approximately tens of seconds). To reduce the I/O time even further, the authors use block-column matrix multiplication, with a block size of 1024×256, approximately equal to the size of one page. The I/O time is reduced to the same order of magnitude as the CPU time (approximately 70 seconds).

These various implementations are simply reorderings of computation, equivalent even down to round-off errors. The access pattern is the sole determining factor in the efficiency of the latter implementation, compared to the first two. With improved clock speeds in central processing units, hiding the latency of access to various levels of memory is crucial to the efficient implementation of algorithms. Lest the reader is left with the wrong impression that this phenomenon is peculiar to high performance computers, I quote the following observation by Forsythe, from a 1953 paper concerning the choice of algorithm for the solution of systems of linear equations on SWAC, then a recent entry onto the computer scene [56]:

> "These are much faster than previous computers, but the speed of arithmetic operations and of access to a small store of data (high speed memory) has been accelerated a great deal more than the operations of input, output, and access to a large store of data. The resulting change in the relative costs of different operations has a profound effect on the choice of computing method …certainly the optimal method depends on the problem, the machine, the operator, and the coder."

After discussing the various iterative and direct methods of solution, he adds:

> "With any method the partitioning of A may greatly increase the speed by enabling subsidiary matrix inversion to take place entirely within the high speed memory."

As an example, he refers to a Rand Corporation report entitled: 'Matrix algebra programs for the UNIVAC', and concludes with this far-sighted question:

> "One usually thinks of Gaussian elimination on the submatrix block. Would other methods on the blocks be preferable?"

6.2 Block Algorithms

The idea of block algorithms for linear algebra on machines with paged virtual store was addressed again in the 1960's and 1970's in the works of McKellar and Coffman [104], Trivedi

[138], and Du Croz *et al.* [50]. The interest and activity in developing block algorithms was increased sharply with the appearance of high performance multiprocessors containing a hierarchy of memory.

Block algorithms rely on accessing large submatrices between different levels of memory. Agarwal and Gustavson designed block algorithms for LU decomposition to take advantage of the cache memory, vector registers, and multiprocessing capabilities of the IBM 3090 [1]; (see also [121], for related results). A numerical linear algebra library based on block methods was designed and analyzed for the Cedar machine at the University of Illinois [60] in the mid 1980's. Calahan [20] introduced a block oriented linear equation solver on the CRAY-2 uniprocessor with a software managed local memory. Daydé and Duff, in [31], compared the performance of block LU factorization on three vector multiprocessors. Demmel and Higham investigated the stability of block algorithms in [36] and, with Schreiber, analyzed the stability of block LU factorization in [37]. Van Loan, in [140], considered a block version of the Jacobi method for singular value decomposition. Bischof and Van Loan [15] introduced the WY representation for block Householder QR factorization algorithm. A memory efficient WY representation for the products of Housholder transformation was later introduced by Schreiber and Van Loan [132]; see also the related results by Schreiber and Parlett [131].

This is by no means a complete list of references to block algorithms, their implementation, and comparisons between various machines. It is indicative of the intense activity in design, restructuring and implementation of block linear algebra algorithms. For further work, the interested reader is referred to the comprehensive survey in [60], and the monograph [46].

Block algorithms derive their efficiency by reuse of data, through reordering of computation. Similar to the matrix multiplication example on the Cyber 205, the effect of loop interchange on the access pattern of LU decomposition was studied in detail in [44] and [113]. The variants came to be known as *right looking*, *left looking*, and *Crout* versions of the algorithm. Typical performance figures and comparisons between the variants are given in [46] and [31], for high performance shared memory computers. An important architecture-dependent parameter is the *block size*, which is influenced by the cache size, the number and length of any vector registers, and the memory bandwidth. These are usually determined by detailed experimental procedures.

The data access pattern plays an equally important role in the efficient implementation of numerical linear algebra algorithms on distributed memory architectures. Here, due to the explicit (and often static) nature of partitioning and scheduling of data and computation, the problem manifests itself as the determination of a partition of data and code so as to reduce the overhead due to nonlocal data access, while maintaining a good balance of load across the multiple processors.

The lack of software environments to assist in the above tasks means that the algorithm designer has to take into account architecture-dependent factors, such as the *topology of interconnection* and the *compute/communication ratio*, as well as partitioning by blocks, row or columns. For examples, see [57, 63, 64, 73, 74, 96, 106, 119, 122, 136, 9, 12, 16, 26, 27, 53, 83, 60, 75]. Timing models of the performance of algorithms include communication complexity (based on assumptions regarding the start-up time plus the cost of transmitting a byte of data) as well as the topology of the interconnection network [84].

6.3 LAPACK, BLAS and BLACS

It is evident that tuning each algorithm for specific hardware, taking into account data access patterns and characteristics of memory organization, is both time-consuming and results in a loss of portability, reminiscent of the era of assembly language programming. Hence, in an attempt to obviate these problems, and in probably the largest collaborative effort in the design, analysis, and evaluation of numerical linear algebra software, the LAPACK project was launched [33, 6]. Its aim was, in part, to reduce the cost of data movement by carefully restructuring the code, and to ensure portability across a class of architectures.

One way of achieving efficiency is to express the algorithm in terms of a set of commonly occurring primitive *kernels*, which could be tuned to a particular architecture. The advantage of this approach is that efficiency is achieved whilst maintaining clarity, portability, modularity, and maintenance of software. The Basic Linear Algebra Subprograms (BLAS), proposed by Lawson *et al*. [95], pursued this goal and soon became a *de facto* standard, being used in a wide range of software, including LINPACK [45]. The original (Level 1) BLAS performed low level vector operations, such as inner product and sum of a vector with a multiple of another (SAXPY). With the appearance of the CRAY-1, with only one path between memory and the vector registers, it soon became evident that data reuse in registers was essential to overcome this bottleneck and achieve what Dongarra refers to as "super-vector" performance [44]. Level 2 BLAS, rich in matrix-vector operation, were proposed to improve register reuse in such vector machines [42]. Experiences with parallel architectures with a hierarchy of memory indicated that, in order to maximize data reuse, it was necessary to introduce primitive kernels that have a higher ratio of floating point operations to data movement [59, 60]. Level 3 BLAS, rich in matrix-matrix operations were thus proposed [40, 41], and are now a *de facto* standard for the design of numerical linear algebra software, such as LAPACK, on high performance architectures [6]. Since March 1992, LAPACK has been publically available through Netlib [2]. It is composed of more than half a million lines of Fortran 77 code. The first revisions were made in June 1992. For an overall perspective on the design philosophy of LAPACK see [35, 5]. Efforts are underway [7] to develop LAPACK for distributed memory architectures.

Similar to the development of BLAS, the need to port the core of LAPACK to distributed memory architectures gave impetus to the development of the Basic Linear Algebra Communication Subprograms (BLACS) [47]. The first workshop on BLACS was held in March 1991, in Houston, and considered preliminary proposals for a set of basic communication routines for use in linear algebra mathematical software. The motivation, as in BLAS, is to increase portability, efficiency and modularity, at a high level of abstraction. BLACS is targeted to mathematical software experts and people with experience in large scale computations, and efforts are underway to achieve a *de facto* standard. The underlying assumption is that the core routines of the LAPACK library written in Fortran77, are ported to distributed memory machines in the Single Program Multiple Data (SPMD) style of programming [86], and changes to the algorithm and code are kept to a minimum. Among the issues under consideration are: the subset of LAPACK that support BLACS; the target machine (SIMD versus MIMD); the data

[2] For background about netlib see: "Distribution of Mathematical Software Via Electronic Mail", by Jack J. Dongarra and Eric Grosse, in Communications of the ACM, Vol 30, 1987, pp 403-407.

layout (column wrap versus block wrap); multiple partitions of the machine; multiple processes or users per node; and, finally, the important issue of scalability in terms of problem size, number of processors, size of memory per processor, and the number of problem instances [47]. In terms of data layout, recent experience has shown that higher performance can be achieved for dense problems when utilizing data distribution that views the processor nodes as lying in a logical two dimensional grid [48]. Schreiber has recently investigated the data layout question for a sparse direct solver on a highly parallel machine [133]. The surprising result is that the performance of a column oriented approach to sparse Cholesky factorization does not scale as the number of processors increases. Thus, from scalability considerations, as well, a two dimensional data layout is shown to be favorable.

Other recent developments, such as Fortran-D [81] and High Performance Fortran (HPF) with data distribution directives [49], and the availability of distributed memory architectures with a single address space, are bound to have an impact on the future development of BLACS.

7 Related Surveys

One of the earliest surveys of numerical linear algebra is the 1958 article by White [141], reviewing various techniques for computing eigenvalues and eigenvectors of a matrix, with the corresponding codes written for a wide range of machines, such as the ILLIAC, UNIVAC 1103A, IBM 650 and 704, and SWAC. With the appearance of parallel and vector machines, parallel numerical algorithms were surveyed in the 1970's by Miranker [105], Sameh [127] and Heller [76]. More recently, Dongarra, Gustavson and Karp discuss implementation issues for dense linear algebra algorithms on vector pipeline machines [44]. In 1985 Ortega and Voigt [111] gave a comprehensive review of the solution of partial differential equation on vector and parallel machines. An overall perspective of parallel algorithms for dense linear algebra computations is presented by Gallivan, Plemmons and Sameh [60]. Progress in parallel algorithms for sparse linear systems is surveyed by Heath, Ng, and Peyton [75], with a special focus on the direct method for solution of positive definite linear systems. Dongarra, Duff, Sorenson and van der Vorst [46] give an overview of the solution of linear systems on vector and shared memory computers. An extensive bibliography of parallel and vector numerical algorithm has been compiled by Ortega, Voigt and Romine [112]. Iterative methods on supercomputers are addressed by Saad [123]. Freund, Golub and Nachtigal [58] give a comprehensive survey of the recent advances in CG-type algorithms and Krylov subspace methods for non-Hermitian matrices. Finally, the theory and practice of the Multi-Frontal method for solution of large sparse symmetric positive definite linear systems is reviewed by Liu in [98].

8 Conclusion

It is reported that Poincaré once said:

> "mathematics has to reflect upon itself".

In this report an attempt has been made to highlight interesting features of the evolving interaction between computer architecture and numerical linear algebra. The rich varieties of activity in design, analysis, comparison and evaluation of algorithms, new and old, have been examined in the context of rapid technological and architectural advances. The characteristics of algorithms for numerical linear algebra have been discussed, with reference to important surveys that "bridge the gap" to specific details. Some of the methods mentioned here, e.g. to analyze algorithms, improve parallelism and reduce memory access, are now incorporated into programming tools and compilers. For a sample of developments in this area, see [61, 71, 2, 3, 11, 81, 90, 94, 129, 114, 117, 130, 147]. The automation of efficient implementation of numerical linear algebra algorithms on high performance computers is a challenging area of current research. It is hoped that these reflections on the practice of numerical linear algebra will give insight concerning the possibilities for, and limitations of, automating the process of mapping algorithms onto high performance computers.
Acknowledgments. The author greatly acknowledges the support of the Science and Engineering Research Council (SERC Grant GR/F/04292) for this work, and wishes to thank Professor John Gurd, Dr. Nick Higham, Dr. Michael O'Boyle, and Mark Bull for many helpful comments and suggestions.

References

[1] Agarwal R., Gustavson F., *A parallel implementation of matrix multiplication and LU factorization on IBM 3090*, Aspects of Computation on Asynchronous Parallel Processors, M.H.Wright, ed. North-Holland, Amsterdam, 1989.

[2] Allen J.R., Callahan D., Kennedy K., *Automatic decomposition of scientific programs for parallel execution*, Tech.Report,TR86-42, Dept. of Computer Science, Rice University, Houston, TX, 1986. Also ACM POPL 14, January 1987.

[3] Allen J.R., Kennedy K., *Automatic translation of Fortran programs to vector form*, ACM Trans. on Programming Languages and Systems, 9, 1987.

[4] Ahmed H., Delosme J., Morph M.,*Highly concurrent computing structures for matrix arithmetic and signal processing*, Computing,15 1982.

[5] Anderson E., *et al.*, *LAPACK Users' Guide*, SIAM Publications, Philadelphia, 1992.

[6] Anderson E., *et al.*, *LAPACK: A portable linear algebra library for high performance computers*, Proc.Supercomputing'90, IEEE Computer Society Press, Los Alamitos, CA, 1990.

[7] Anderson E., *et al.*, *LAPACK for distributed memory architectures:progress report*, Proc.5th SIAM Conference on Parallel Processing for Scientific Computing, SIAM Publications, Philadelphia, 1991.

[8] Arioli M., Demmel J., Duff I.S., *Solving sparse linear system with sparse backward error*, SIAM J.Matrix Anal.Appl., Vol 10, 2, 1989.

[9] Ashcraft C., Eisenstat S., Liu J.,*A fan-in algorithm for distributed sparse numerical factorization*, SIAM J. Sci. Statis. Computing, 11(3), 1990.

[10] Axelsson O., *A survey of preconditioned iterative iterative methods for linear system of algebraic equations*, BIT, 25, 1985.

[11] Balasundaram V., *Interactive parallelization of numerical scientific programs*, PhD Thesis, Dept. of Computer Science, Rice University,Houston, TX, 1989.

[12] Bischof C., *Computing the singular value decomposition on a distributed system of vector processors*, Tech.Report TR86-798, Dept. of Computer Science, Cornell University, Ithaca, NY, 1986.

[13] Bischof C., *A parallel QR factorization algorithm with controlled local pivoting*, SIAM J. Sci. Statist. Comput., 12, 1, 1991.

[14] Bischof C., Lewis J., Pierce D., *Incremental condition estimation for sparse matrices*, SIAM J. Matrix Anal. Appl., 11(4), 1990.

[15] Bischof C., Van Loan C., *The WY representation for products of Householder matrices*, SIAM J. Sci. Statist. Comput., 8, 1987.

[16] Brent R., Luk F., *The solution of singular-value and symmetric eigenvalue problems on multiprocessors arrays*, SIAM J. Sci. Statist. Comput., 6, 1985.

[17] Brent R.P., Gustavson F.G., Yun D.Y.Y., *Fast solution of Toeplitz systems of equation and computation of Pade' approximants*, J.Algorithms, Vol 1, 1980.

[18] Bunch J.R., *Stability of Methods for solving Toeplitz systems of equation*, SIAM J. Sci. Stat. Comp., Vol 6, 1985.

[19] Bunch J.R., *The Weak and Strong Stability of Algorithms in Numerical Linear Algebra*, Linear Algebra and its Appl., Vol 88/89, 1987.

[20] Calahan D., *Block oriented local memory based linear equation solution on the CRAY-2:uniprocessor algorithm*, Proc.Intl.Conf.Par.Processing, IEEE Computer Society Press, NY, 1986.

[21] Chan T.F., *Hierarchical algorithms and architectures for parallel scientific computing*, Proc.ICS 90, ACM Press, NY, 1990.

[22] Chan T.F., *A note on the efficiency of domain decomposed incomplete factorization*, SIAM J. Sci. Statis. Computing, 11(4), 1990.

[23] Chan T.F., Resasco D., *A framework for the analysis of domain decomposition preconditioners*, Proc. First Internat. Symp. on Domain Decomposition Methods for Partial Differential Equations, SIAM, Philadelphia, PA, 1988.

[24] Chartres B.A., *Adaptation of the Jacobi method for a computer with magnetic-tape backing store*, Comput.J., 5, 1962.

[25] Chen K-W., Irani K.B., *A Jacobi Algorithm and its implementation on parallel computers*, Proc. 18th Annual Allerton Conference on Communication, Control, and Computing, Illinois, 1980.

[26] Chu E., George A., *Gaussian elimination with partial pivoting and load balancing on a multiprocessor*, Parallel Computing, 5, 1987.

[27] Chu E., George A., *QR factorization of a dense matrix on a Hypercube multiprocessor*, SIAM J. Sci. Statis. Computing, 11(5), 1990.

[28] Chu E., George A., Liu J.W.-H., Ng E.G.-Y., *User's Guide for SPARSPAK-A: Waterloo sparse linear equations package*, Tech. Report CS-84-36, University of Waterloo, Waterloo, Ontario, 1984.

[29] Concus P., Golub G.H., Meurant G., *Block preconditioning for the conjugate gradient method*, SIAM J. Sci. Statist. Comput., 6, 1985.

[30] Cybenko G., *The numerical stability of Levinson-Durbin algorithm for Toeplitz systems of equations*, SIAM J. Sci. Statis. Comput., 1980.

[31] Daydé M.J., Duff I.S., *Use of parallel Level 3 BLAS in LU factorization on three vector multiprocessors the Alliant FX/80,the CRAY-2, and the IBM 3090 VF*, Proc.ICS 90, ACM Press, NY, 1990.

[32] Demmel J., Kahan W., *Accurate singular values of bidiagonal matrices*, SIAM J. Sci. Statist. Comput., 11, 5, 1990.

[33] Demmel J., *et al.*, *Prospectus for the development of a linear algebra library for high performance computers*, Tech.Report TM-97, Mathematics and Computer Science Div., Argonne National Laboratory, Argonne, IL, 1987.

[34] Demmel J., Veselić K., *Jacobi's method is more accurate than QR*, Computer Science Dept., Tech.Rep. 468, Courant Institute, NY, 1989.

[35] Demmel J., Kahan W., Dongarra J., *On designing portable high performance Numerical libraries*, Tech. Report, July 1991, Available through Netlib. Also in Numerical Analysis 1991, Proceedings of the 14th Dundee Conference, Eds, Griffiths D. F., Watson G.A., Longman Scientific and Technical, Essex, UK, 1992.

[36] Demmel J.W., Higham N.J., *Stability of block algorithms with fast level 3 BLAS*, Numerical Analysis Report 188, Department of Mathematics, University of Manchester, August 1991, To appear in ACM TOMS.

[37] Demmel J.W., Higham N.J., Schreiber R.S., *Block LU factorization*, Numerical Analysis Report 207, Department of Mathematics, University of Manchester, February 1992. Also as LAPACK Working Note 40, Technical Report CS-92-149, Department of Computer Science, University of Tennessee.

[38] Delosme J., Ipsen I., *Design methodology for systolic arrays*, Proc. SPIE Symposium, 1986.

[39] De Rijk P., *A one-sided Jacobi algorithm for computing the singular value decomposition on a vector computer*, SIAM J. Sci. Statis. Computing, 10(2), 1989.

[40] Dongarra J., *Workshop on Level 3 BLAS*, Tech.Report TM-89,Mathematics and Computer Science Div., Argonne National Laboratory, Argonne, IL, 1987.

[41] Dongarra J., Croz J.D., Duff I., Hammarling S., *A proposal for a set of Level 3 basic linear algebra subprogram*, Tech.Report TM-88, Mathematics and Computer Science Div. Argonne National Laboratory, Argonne, IL, 1987.

[42] Dongarra J., Croz J.D., Hammarling S., Hanson R., *A proposal for an extended set of Fortran basic linear algebra subprogram*, Tech.Report TM-41, Mathematics and Computer Science Div. Argonne National Laboratory, Argonne, IL, 1984.

[43] Dongarra J., Eisenstat S., *Squeezing the most out of an algorithm in CRAY Fortran*, ACM Trans. Math. Software, 10, 1984.

[44] Dongarra J., Gustavson F., Karp A., *Implementing linear algebra algorithms for dense matrices on a vector pipeline machine*, SIAM Review,26,1984.

[45] Dongarra J., *et al.*, *Linpack user's guide*, SIAM Publications, 1979.

[46] Dongarra J., Duff I., Sorensen D., van der Vorst H., *Solving linear systems on vector and shared memory computers*, SIAM, Philadelphia, 1991.

[47] Dongarra J.J., *LAPACK Working Note 34: Workshop on the BLACS*, Technical Report TN 37996-1301. Computer Science Department, University of Tennessee, February 1992. Available through Netlib.

[48] Dongarra J.J., van de Geijn R.A., *LAPACK Working Note 37: Two Dimensional Basic Linear Algebra Communication Subprograms*, October 1991, Available through Netlib.

[49] *DRAFT: High Performance Fortran Specification*, High Performance Fortran Forum, Version 0.1, August 1992, Rice University, Available through FTP.

[50] Du Croz J., *et al.*, *Solving large full sets of linear equations in a paged virtual store*, ACM Trans. Math. Software, 7, 1981.

[51] Duff I., Reid J., *MA27- A set of Fortran subroutines for solving sparse symmetric sets of linear equations*, Tech. Report AERE R 10533, Harwell, 1982.

[52] Duff I., *MA32- A package for solving sparse unsymmetric systems using the frontal method*, Report R10079, HMSO, AERE Harwell, 1981.

[53] Eisenstat S.C. , *et al.*, *Modified cyclic algorithms for solving triangular systems on distributed memory multiprocessors*, SIAM J. Sci. Statist. Comput., 9, 1988.

[54] Eisenstat S.C., Gursky M.C., Schultz M.H., Sherman A.H., *Yale sparse matrix package 1: The symmetric codes*, International J. Numerical Methods in Engineering, Vol 18, 1982, pp 1145-1151.

[55] Forsythe G.E., Henrici P., *The cyclic Jacobi method for computing the principal value of matrix*, Trans. AMS, Vol94, 1960.

[56] Forsythe G.E., *Solving linear equations can be interesting*, Bulletin of AMS, 59, 1953.

[57] Fox G., *et al.*, *Solving problems on concurrent processors*, Vol 1, Prentice Hall, Englewood Cliffs, NJ, 1988.

[58] Freund R.W., Golub G.H., Nachtigal N.M., *Iterative Solution of Linear Systems*, Acta Numerica, Cambridge University Press, pp 57-100, 1992.

[59] Gallivan K., Jalby W., Meier U., *The use of BLAS3 in linear algebra on a parallel processor with a hierarchical memory*, SIAM J. Sci. Statist. Comput., 8, 1987.

[60] Gallivan K., Plemmons R., Sameh A., *Parallel algorithms for dense linear algebra computations*, SIAM Review, March 1990.

[61] Gannon D., Jalby W., Gallivan K., *On the problem of optimizing data transfers for complex memory systems*, Proc. 1988 ICS, ACM Press, NY, 1988.

[62] Garbow B.S., Boyle J.M., Dongarra J.J., Moler C.B., *Matrix eigensystem routines- EISPACK guide extensions*, Lecture Notes in Computer Science, Vol 51, Springer-Verlag, New York, 1977.

[63] Geist A., Heath M., *Matrix factorization on a hypercube*, Hypercube multiprocessors 1986, M.T.Heath, ed., SIAM, Philadelphia, 1986.

[64] Geist A., Romine C., *LU factorization algorithms on distributed memory multiprocessor architectures*, SIAM J. Sci. Statist. Comput., 9, 1988.

[65] Gentleman W., Kung H.T., *Matrix triangularization by systolic arrays*, Proc. SPIE 298, Real Time Signal Processing, San Diego, CA, 1981.

[66] George A., Ng E., *SPARSPAK: Waterloo sparse matrix package user's guide for SPARSPAK-B*, Department of Computer Science CS-84-37, 1984.

[67] Givens J.W., *Numerical computation of the characteristic values of a real symmetric matrix*, Tech.Rep. ORNL-1574, Oak Ridge National Lab., Oak Ridge, Tenn., 1954.

[68] Goldstine H.H., Murray F.J., von Neumann J., *The Jacobi method for real symmetric matrices*, J. ACM, Vol 6, 1959.

[69] Golub G.H., Van Loan C., *Matrix Computations*, 2d Edition, The John Hopkins University Press, Baltimore, 1989.

[70] Golub G.H., Riensch C., *Singular value decomposition and least squares solutions*, Numer.Math., 14, 1970.

[71] Gornish E.H., Granston E.D., Veidenbaum A.V., *Compiler directed data prefetching in multiprocessors with memory hierarchies*, Proc.ICS 90, ACM Press, NY, 1990.

[72] Gregory R.T., *Computing eigenvalues and eigenvectors of a symmetric matrix on ILLIAC*, Math.Tables Aids Comput., Vol7, 1953.

[73] Heath M., *Parallel cholesky factorization in message passing multiprocessor environments*, Tech.Report, ORNL-6150, Oak Ridge National Lab., Oak Ridge, TN, 1985.

[74] Heath M., Romine C., *Parallel Solution of triangular systems on distributed memory multiprocessors*, SIAM J. Sci. Statist. Comput., 9, 1988.

[75] Heath M., Ng E., Peyton B., *Parallel algorithms for sparse linear system*, In Parallel Algorithms for Matrix Computation, SIAM, Philadelphia, 1990.

[76] Heller D., *A survey of parallel algorithms for numerical linear algebra*, SIAM Review, 20, 1978.

[77] Henrici P., *On the speed of convergence of cyclic and quasi-cyclic Jacobi method for computing eigenvalues of hermitian matrices*, J. SIAM, Vol 6, 1958.

[78] Hestenes M.R., *Inversion of matrices by biorthogonalization and related results*, J. SIAM , Vol 6, 1958.

[79] Hestenes M.R., Stiefel E., *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Standards, 49, pp 409-436.

[80] Higham N.J., *How accurate is Gaussian elimination?*, Dept.Mathematics, Joint Numerical Analysis Tech.Rep.173, University of Manchester/UMIST, 1989. Also in Numerical Analysis 1989, Proceedings of the 13th Dundee Conference, Eds, Griffiths D. F. and Watson G.A., Longman Scientific and Technical, Essex, UK, 1990.

[81] Hiranandani S., Kennedy K., Tseng C.-W., *Compiler optimization for Fortran D on MIMD distributed-memory machines*, Proceedings of Supercomputing 91, Albuquerque, 1991.

[82] Householder A.S., *Theory of matrices in numerical analysis*, Dover Publications, New York, 1964.

[83] Ipsen I., Jessup E., Solving the symmetric tridiagonal eigenvalue problem on the hypercube, SIAM J. Sci. Statis. Computing, 11(2), 1990.

[84] Ipsen I., Saad Y., Schultz M., *Complexity of dense linear system solution on a multiprocessor ring*, Linear Algebra Appl., 77, 1986.

[85] Kailath T., Vieira A., Morf M., *Inversion of Toeplitz operators, innovations and orthogonal polynomials*, SIAM Review, 20, 1978.

[86] Karp A., *Programming for parallelism*, IEEE Computer, May 1987.

[87] Kincaid D.R., Oppe T.C., Respess J.R., Young D.M., *ITPACKV 2C User's guide*, Report CNA-191, Center for Numerical Analysis, University of Texas, Austin, 1984.

[88] Keyes D.E., Gropp W.D., *A comparison of domain decomposition techniques for elliptic partial differential equations*, SIAM J. Sci. Statist. Comput., 8, 1987.

[89] Kogbetliantz E., *Diagonalization of general complex matrices as a new method for solution of linear system*, International Congress on Mathematics, Amsterdam, 1954.

[90] Kumar M., *Measuring parallelism in computation-intensive scientific/engineering applications*, IEEE Trans. Computers, C-37, 1988.

[91] Kuo C.-C.J., Chan T.F., *Two-color Fourier analysis of iterative algorithms for elliptic problems with red/black ordering*, SIAM J. Sci. Statis. Computing, 11(4), 1990.

[92] Kuo C.-C.J., Chan T.F., Tong C., *Multilevel filtering elliptic preconditioners*, SIAM J. Matrix Anal. Appl., 11(3), 1990.

[93] Lanczos C., *Iterative solution of large scale linear systems*, J. SIAM, Vol 6, No 1,1958.

[94] Larus J., *Parallelism in numeric and symbolic programs*, Advances in Language and Compiler for Parallel Processing, Nicolau A., et al, eds, MIT Press, Cambridge, MA, 1991.

[95] Lawson C., et al, *Basic linear algebra subprograms for Fortran use*, ACM Trans. Math. Software, 5, 1979.

[96] Li G., Coleman T., *A parallel triangular solver on distributed memory multiprocessor*, SIAM J. Sci. Statist. Comput., 9, 1988

[97] Liu J.W.H., *The minimum degree ordering with constraints*, SIAM J. Sci. Statis. Computing, 10(6), 1989.

[98] Liu J.W.H., *The multifrontal method for sparse matrix solution: Theory and practice*, SIAM Review, Vol 34, No 1, March 1992.

[99] Lewis J.G., Peyton B.W., Pothen A., *A fast algorithm for reordering sparse matrices for parallel factorization*, SIAM J. Sci. Statis. Computing, 10(6), 1989.

[100] Lewis J.G., Simon H.D., *The impact of hardware scatter-gather on sparse Gaussian elimination*, SIAM J. Sci. Statist. Comput., 9, 1988.

[101] Luk F., *Computing the singular value decomposition on the ILLIAC IV*, ACM Trans. Math. Software, 6, 1980.

[102] Luk F., Park H., *On parallel Jacobi orderings*, SIAM J. Sci. Statis. Computing, 10(1), 1989.

[103] Luk F., Park H., *A proof of convergence for two parallel Jacobi SVD algorithm*, IEEE Trans. Comput., 38, 1989.

[104] McKellar A., Coffman JR. E., *Organizing matrices and matrix operations for paged memory systems*, Comm. ACM, 12, 1969.

[105] Miranker W., *A survey of parallelism in numerical analysis*, SIAM Review, 13, 1971.

[106] Moler C., *Matrix computation on distributed memory multiprocessors*, Hypercube Multiprocessors 1986, M.T.Heath, ed., SIAM, Philadelphia, 1986.

[107] Modi J.J., Pryce J.D., *Efficient implementation of Jacobi's diagonalization method on the DAP*, Numer.Math., 46, 1985.

[108] Nash J.C., *A one sided transformation method for the singular value decomposition and algebraic eigenproblem*, Comput. J., 18, 1975.

[109] Nazareth L., *On the convergence of the cyclic Jacobi method*, Linear Algebra Appl., 12, 1975.

[110] von Neumann J, Goldstine H.H., *Numerical inversion of matrices of high order*, Bulletin of AMS, 1947, pp1021-1099.

[111] Ortega J., Voigt R., *Solution of partial differential equations on vector and parallel computers*, SIAM Review, 27, 1985.

[112] Ortega J., Voigt R., Romine C., *A bibliography on parallel and vector numerical algorithms*, Tech.Report, ORNL TM-10998, Oak Ridge National Laboratory, Oak Ridge, TN, 1989. Also in Parallel Algorithms for Matrix Computations, SIAM Publications, Philadelphia, 1990.

[113] Ortega J., Romine C., *The ijk forms of factorization II. Parallel systems*, Parallel Computing, 7, 2, 1988.

[114] Padua D., Wolfe M.J., *Advanced compiler optimization for supercomputers*, Comm. ACM, 29, Dec. 1986.

[115] Parlett B.N., *Some basic information on information-based complexity theory*, Bulletin of the AMS, Vol 26, No 1, 1992.

[116] Peterson V.L., *Impact of computers on aerodynamics research and development*, Proceedings of the IEEE, 72, 1984.

[117] Polychronopoulos C.D.,*Compiler optimization for enhancing parallelism and their impact on architecture design*,IEEE Trans.Computer,37,1988.

[118] Pope D.A., Tomkin C., *Maximizing functions of rotations–experiments concerning speed of diagonalization of symmetric matrices using Jacobi's method*, J. ACM, 4, 1957.

[119] Pothen A., Raghavan P., *Distributed orthogonal factorization: Given and Housholder Algorithms*, SIAM J. Sci. Statis. Computing, 10(6), 1989.

[120] Radicati G., Robert Y., *Vector and parallel CG-like algorithms for sparse nonsymmetric linear systems*, Tech. Report RR 681, IMAG, University of Grenoble, France, 1987.

[121] Radicati G., Robert Y., Sguazzero P., *Dense linear systems Fortran solver on the IBM 3090 vector multiprocessor*, Parallel Computing, 8, 1988.

[122] Romine C., Ortega J., *Parallel solution of triangular systems of equations*, Parallel Computing, 6, 1988.

[123] Saad Y., *Krylov subspace methods on supercomputers*, SIAM J. Sci. Statis. Computing, 10(6), 1989.

[124] Saad Y., Schultz M.H., *GMRES: A generalized minimal residual algorithm for solving non-symmetric linear systems*, SIAM J. Sci. Stat. Comput., Vol 7, 1986.

[125] Saltz J.H., *Aggregation methods for solving sparse triangular systems on multiprocessors*, SIAM J. Sci. Statis. Computing, 11(1), 1990.

[126] Sameh A., *On Jacobi and Jacobi-like algorithms for a parallel computer*, Math. Comp., 25, 1971.

[127] Sameh A., *Numerical parallel algorithms- a survey*, High Speed Computer and Algorithm Organization, D.Kuck, D.Lawrie, and A.Sameh, eds., Academic Press, NY, 1977.

[128] Sameh A., *et al.*, *Algorithm and applications group:Semiannual report July-December 1990*, Tech. Report CSRD 1127, Center for Supercomputing Research and Development, University of Illinois, Urbana. IL, 1991.

[129] Sarkar V., *Determining average program execution times and their variance*, Proc.SIGPLAN'89 Conference on Programming Language Design and Implementation, June 1989.

[130] Schreiber R., Dongarra J., *Automatic blocking of nested loops*, Tech. Report CS-90-108, University of Tennessee, May 1990.

[131] Schreiber R., Parlett B., *Block reflectors:theory and computation*, SIAM J. Numer. Analysis, 25, 1988.

[132] Schreiber R., Van Loan C., *A storage-efficient WY representation for products of Housholder transformations*, SIAM J. Sci. Statis. Computing, 10(1), 1989.

[133] Schreiber R., *Scalability of sparse direct solvers*, Research Institute for Advanced Computer Science Report MS T045-1. May 1992.

[134] Shroff G., Schreiber R., *On the convergence of the cyclic Jacobi method for parallel block orderings*, SIAM J. Matrix Anal. Appl., 10(3), 1989, Also , Tech Report 88-11, Dept. of Computer Science, Rensselaer Polytechnic Institute, Troy, NY, 1988.

[135] Smith B., *et al.*, *Matrix Eigensystem Routines: EISPACK Guide*, 2nd,ed., Springer Verlag, Berlin, 1976.

[136] Stewart G.W., *A parallel implementation of QR-algorithm*, Parallel Computing, 5, 1987.

[137] Stewart G.W., *A Jacobi-like algorithm for computing the Schur decomposition of a non-Hermitian matrix*, SIAM J. Sci. Statist. Comput., 6, 1985.

[138] Trivedi K., *On the paging performance of array algorithms*, IEEE Trans. Computer, C-26, 1977.

[139] Turing A.M., *Rounding-off errors in matrix processes*, Quarterly J. Mech. Appl. Math., Vol 1, 1948, pp 287-308.

[140] Van Loan C., *The block Jacobi method for computing the singular value decomposition*, Tech.Report, TR85-680, Dept. of Computer Science, Cornell University, Ithaca, NY, 1985.

[141] White P.A., *The computation of eigenvalue and eigenvectors of a matrix*, J. SIAM, Vol 6, 1958.

[142] Wijshoff H.A.G., *Data Organization in Parallel Computers*, Kluwer, Boston, 1989.

[143] Wilkinson J.H., *Error analysis of direct methods of matrix inversion*,J. ACM, Vol8, 1961.

[144] Wilkinson J.H., *The algebraic eigenvalue problem*, Oxford University Press, London, 1965.

[145] Wilkinson J.H., *Modern error analysis*, SIAM Review, Vol 13, No 4, 1971.

[146] Wilkinson J.H., *Note on the quadratic convergence of the cyclic Jacobi process*, Numer.Math., Vol 4, 1962.

[147] Wolf M.E., Lam M., *A data locality optimizing algorithm*, Proc. ACM SIGPLAN'91 Conference on Programming Language Design and Implementation, Toronto, Canada, June 1991.

[148] Worley P.H., *Limits on parallelism in numerical solution of linear partial differential equations*, SIAM J. Sci. Statist. Comput., 12, 1, 1991.

[149] Young D.P., *et al.*, *Application of sparse matrix solvers as effective preconditioners*, SIAM J. Sci. Statist. Comput., 10, 6, 1989.

[150] Yserentant H., *On the multi-level splitting of finite element spaces*, Numer. Math., 49, 1986.

[151] Zlatev Z., Wasniewski J., Schaumburg K., *Y12M- Solution of large and sparse systems of linear algebraic equations*, Vol 121, Springer-Verlag, New York, 1981.