ST-HEC: Reliable and Scalable Software for Linear Algebra Computations on High End Computers

James Demmel (U California, Berkeley) and Jack Dongarra (U Tennessee, Knoxville)

1 Introduction

There is inadequate software support for High Performance Computing (HPC), a fact cited in the call for this proposal and in numerous recent Federal and Academy reports [48][Chap. 2], [47][Chap. 4], [77][App. A], [65][Chap. 5], [62][Chap. 5], [57][p. 44]. Among other deficiencies, software is considered too hard to use, too inefficient (too low a fraction of peak and/or not scalable), or both. The need for better numerical libraries that encapsulate complicated and widely used algorithms is discussed. The linear algebra libraries LAPACK [3] and ScaLAPACK [16] are frequently mentioned as positive examples.

We, the principal designers of LAPACK and ScaLAPACK, propose a number of significant improvements to these libraries. These libraries are widely used: There are over 42M web hits at www.netlib.org (to the associated libraries LAPACK, ScaLAPACK, CLAPACK and LAPACK95), and they have been adopted by many vendors as the basis of their own libraries: SGI/Cray, IBM, HP-Convex, Fujitsu, NEC, NAG and IMSL, and the Mathworks (producers of Matlab). Therefore our proposed changes will have large impact.

We summarize these improvements below. We have identified them by a combination of our own algorithmic research, an on-going user and vendor survey (at icl.cs.utk.edu/lapack-survey.html), the anticipation of the demands and opportunities of new architectures and modern programming languages, and finally the enthusiastic participation of the research community in developing and offering improved versions of existing Sca/LAPACK codes. Indeed, papers proposing new algorithms typically compare their performance with that of Sca/LAPACK, and over the years several researchers have developed better algorithms that they would like to provide to us. In some cases they have even done the same level of careful software engineering and testing that we insist on; in other cases we must do this; in yet other cases much mathematical development remains. The opportunity to harvest this bounty of good ideas (and free labor) is not to be missed. We attach supporting letters from individuals and companies.

Putting more of LAPACK into ScaLAPACK. Sec. 2 documents that ScaLAPACK contains a small subset of the functionality of LAPACK. For example, there is no ScaLAPACK support for symmetric matrices using minimal storage ($\approx n^2/2$ instead of n^2), parallel versions of only the oldest LAPACK algorithms for the symmetric eigenvalue decomposition (EVD) and singular value decomposition (SVD), and no support for the generalized nonsymmetric EVD. See Sec. 2 for a list of current LAPACK functions that we propose to parallelize and incorporate into ScaLAPACK.

Better Numerical Algorithms for Sca/LAPACK. Sec. 3 lists improvements to algorithms in LAPACK and subsequently ScaLAPACK. A new algorithm might improve speed, accuracy, or memory efficiency, but only occasionally are all three criteria simultaneously maximized by the same algorithm (if we add the criterion "simplicity of interface" it becomes even harder to maximize all criteria simultaneously). Therefore, when there is a strong tradeoff among these criteria, we will include more than one algorithm, with appropriate documentation and "switches" for the user to pick the appropriate algorithm. For example, since the fastest algorithm with the "standard" accuracy for the SVD (the MRRR algorithm of Parlett/Dhillon described below) is significantly faster than the most accurate algorithm with the "standard" speed (the Drmač/Veselič algorithm described below) we will include both. Indeed, both algorithms are faster than than "standard" algorithm based on bidiagonalization followed by QR iteration, which is taught in most textbooks!

Highlights of proposed speed improvements include up to 10x faster for large nonsymmetric EVD using the SIAM Linear Algebra Prize winning work of Braman/Byers/Mathias in [21, 22]; extensions of this work to the generalized nonsymmetric EVD; broad propagation of an improved version of the Parlett/Dhillon algorithm [71, 72, 74, 73, 49] for the symmetric tridiagonal EVD and bidiagonal SVD, replacing $O(n^3)$ or $\approx O(n^{2.3})$ algorithms with easily parallelized $O(n^2)$ algorithms; incorporating "successive band reduction" of Bischof/Lang [15] to change asymptotically all BLAS2 operations (Level 2 Basic Linear Algebra Subroutines) to faster, cache-optimizable BLAS3 operations for the SVD and symmetric EVD; and incorporating recursive data structures to also achieve BLAS3 speeds for solving symmetric linear systems in packed format [52, 42]. Highlights of the proposed accuracy improvements include higher precision iterative refinement for linear systems, based on our recent release of the new BLAS standard [63, 9, 20, 19]; Drmač's new SVD routine which gets even tiny singular values with high accuracy [31], and better pivoting techniques for symmetric linear systems [7, 54, 55].

Extending the functionality of Sca/LAPACK. In Sec. 4 we discuss a number of useful new functions, many of which come from user requests. Highlights include updating and downdating factorizations when the matrix is changed slightly; a new much more efficient algorithm for the special nonsymmetric EVD consisting of a companion matrix or block companion matrix (the former is used for polynomial root finding in eg Matlab, and we lower the complexity from $O(n^3)$ to $O(n^2)$; the latter is used to solve polynomial eigenvalue problems, and we also significantly improve the complexity); specialized quadratic eigenvalue problems frequently arising in mechanics and control; and matrix functions like the square root and exponential; We have also had a number of user requests for out-of-core algorithms (when the matrices do not fit in main memory, and reside on disk).

Improving Ease of Use. In Sec 5 we discuss how better interfaces using features of up-to-date programming languages are important to hide some of complex distributed data structures and other features, and make Sca/LAPACK easier to use. Based on the detailed justifications in Sec. 5, and results from our on-going user survey we currently propose to maintain a single F77 source (to simplify maintaining the large code base), but provide "wrappers" in other languages: LAPACK95 for Fortran 95 users; CLAPACK for C users (i.e. using wrappers instead of continuing to translate LAPACK to C), and most important new wrappers in popular scripting languages like Matlab, Python, Mathematica, etc. We will leave C++ support to the many other active projects in this area, as well as Java support.

Performance Tuning. Sec 6 discusses how LAPACK can be tuned at two levels: within the BLAS, and in LAPACK itself. The BLAS can be tuned for each architecture (and even workload) by ATLAS [89, 90]. LAPACK itself has a plethora of tuning parameters at a higher level (eg at which matrix or blocksize to switch from the BLAS2 code to the BLAS3 code) which are set by calling the routine ILAENV, which in turns looks up a value in a table depending on the algorithm and input parameters. The tables returning these values to the over 1300 ILAENV calls have never been carefully tuned or studied to determine their effect on performance. We propose to apply our successful automatic tuning techniques to tuning the values returned by ILAENV to achieve as high performance on individual processors and SMPs (the building blocks of larger machines) as possible.

Tuning ScaLAPACK for very large machines is even more important. Some of the largest machines will likely be heterogeneous in performance, if only because they are shared resources. Current ScaLAPACK assumes a uniform machine for load balancing purposes. We plan to incorporate load balancing for machines with heterogeneous performance and interconnection capabilities.

Reliability and support. Sec. 7 identifies the need for a large and widely used library like Sca/LAPACK to have ongoing support and maintenance, a need mentioned by the abovementioned Federal reports. Beyond fixing the known and future bugs (eg ensuring thread safety throughout) we want to establish a formal mechanism for user feedback (enhancing the abovementioned website), for tracking bugs (eg with bugzilla), systematically using version management software (eg cvs), and organizing the code to facilitate installation (eg autoconf).

The above list of six topics is our larger vision, and so a superset of what we will likely be able to accomplish with the funding provided by this grant. Depending on the funding level and results of the ongoing user survey, we will prioritize the list to first provide the features that have the highest potential impact on the user community. Our current priorities are presented below.

External Collaboration and Management. Sec. 8 describes how we will manage the operation between the 2 PIs and the numerous outside collaborators we have assembled.

Educational outreach. The recent origin of most algorithms described here means that they have not yet had easily accessible descriptions suitable for education appear, let alone been described in widely used textbooks, such as [46, 83, 29, 88]. We will remedy this by providing tutorial material, and incorporating appropriate descriptions in the textbook [29] and the widely-used on-line course notes [30].

2 Putting more of LAPACK into ScaLAPACK.

We distinguish four categories of routines that we want to include in ScaLAPACK: (1) functions for matrix types appearing in LAPACK but not yet supported in ScaLAPACK (discussed here), (2) functions for matrix types common to LAPACK and ScaLAPACK, but implemented only in LAPACK (discussed here), (3) improved algorithms for functions in LAPACK, which also need to be put in ScaLAPACK (see Sec. 3), and (4) new functions for both LAPACK and ScaLAPACK (see Sec. 4).

Table 1 compares the available data types in the latest releases of LAPACK and ScaLAPACK. After the data type description, we list the prefixes used in the respective libraries, a blank entry indicates that the corresponding type is not supported. The most important omissions in ScaLAPACK are as follows. (1) There is no support for packed storage of symmetric (SP,PP) or Hermitian (HP,PP) matrices, nor the triangular packed matrices (TP) resulting from their factorizations (using $\approx n^2/2$ instead of n^2 storage); these have been requested by users. The interesting question is what data structure to support. One possibility is recursive storage as discussed in Sec. 4 [1, 51, 52, 42]. Alternatively we could partially expand the packed storage into a 2D array in order to apply Level 3 BLAS (GEMM) efficiently. Some preliminary ScaLAPACK prototypes support packed storage for the Cholesky factorization and the symmetric eigenvalue problem [17], but they are under development and have not been rigorously tested on all of the architectures to which the ScaLAPACK library is portable. (2) ScaLAPACK only offers limited support of band matrix storage and does not specifically take advantage of symmetry or triangular form (SB,HB,TB). (3) ScaLAPACK does not support data types for the standard (HS) or generalized (HG, TG) nonsymmetric EVDs; we discuss this further below.

In Table 2, we compare the available functions in LAPACK and ScaLAPACK. We list the relevant user interfaces ('drivers') by subject and the acronyms used for the software in the respective libraries. In the ScaLAPACK column, we indicate what is currently missing. We note that most expert drivers in LAPACK (which supply extra information, such as error bounds) and their specialized computational routines are missing from ScaLAPACK and do not include them explicitly in the table.

We discuss our most important priorities for inclusion in ScaLAPACK:

1. The solution of symmetric linear systems (SYSV), combined with the use of symmetric packed storage (SPSV), will be a significant improvement with respect to both memory and computational complexity over the currently available LU factorization. It has been requested by users and is expected to be used widely. In addition to solving systems it is used to compute the inertia (number of positive, zero and negative eigenvalues) of symmetric matrices.

2. EVD and SVD routines of all kinds (standard - for one matrix - and generalized - for two matrices) are missing from ScaLAPACK. We expect to exploit the MRRR algorithm for the SVD and symmetric EVD, and new algorithms of Braman/Byers/Mathias for the nonsymmetric EVD (see Sec. 3).

3. LAPACK provides software for the linearly constrained (generalized) least squares problem, and users in the optimization community will benefit from a parallel version. In addition, algorithms for rank deficient standard least squares problems based on the SVD are missing from ScaLAPACK; it may be that a completely different algorithm based on the MRRR algorithm (see Sec. 3) may be more suitable for parallelism instead of the divide & conquer (D&C) algorithm that is fastest for LAPACK.

4. Expert drivers that provide error bounds, or other more detailed structural information about eigenvalue problems, should be provided.

3 Better numerical algorithms.

In Section 2, we have given an outline of how ScaLAPACK needs to be extended in order to cover the functionalities of LAPACK. However, LAPACK itself needs to be updated in order to include recent developments since its last release. Whereever possible, we plan to extend these functionalities subsequently

	LAPACK	SCALAPACK
general band	GB	GB, DB
general (i.e., unsymmetric, in some cases rectangular)	GE	GE
general matrices, generalized problem	GG	GG
general tridiagonal	GT	DT
(complex) Hermitian band	HB	
(complex) Hermitian	HE	HE
upper Hessenberg matrix, generalized problem	HG	
(complex) Hermitian, packed storage	HP	
upper Hessenberg	HS	LAHQR only
(real) orthogonal, packed storage	OP	
(real) orthogonal	OR	OR
positive definite band	PB	PB
general positive definite	PO	PO
positive definite, packed storage	PP	
positive definite tridiagonal	PT	PT
(real) symmetric band	SB	
symmetric, packed storage	SP	
(real) symmetric tridiagonal	ST	ST
symmetric	SY	SY
triangular band	TB	
generalized problem, triangular	TG	
triangular, packed storage	TP	
triangular (or in some cases quasi-triangular)	TR	TR
trapezoidal	TZ	TZ
(complex) unitary	UN	UN
(complex) unitary, packed storage	UP	

Table 1: Data types supported in LAPACK and ScaLAPACK. A blank entry indicates that the corresponding format is not supported in ScaLAPACK.

to ScaLAPACK. The expected benefits higher accuracy and/or speed in the solution of linear systems and eigensolvers. We list each set of improvements in our current priority order (highest first).

3.1 Algorithmic improvements for the solution of linear systems

1. The recent developments of extended precision arithmetic [63, 9, 20, 19] in the framework of the new BLAS standard allow the use of higher precision iterative refinement to improve computed solutions. We have recently shown how to modify the classical algorithm of Wilkinson [91, 55] to compute not just an error bound measured by the infinity (or max) norm, but a componentwise relative error bound, i.e. a bound on the number of correct digits in each component. Both error bounds can be compute for a tiny $O(n^2)$ extra cost after the initial $O(n^3)$ factorization [32].

2. Gustavson, Kågström and others have recently proposed a new set of *recursive data structures* for dense matrices [51, 52, 42]. These data structures represent a matrix as a collection of small rectangular blocks (chosen to fit inside the L1 cache), and then stores these blocks use ones of several “space filling curve” orderings. The idea is that the data structure, and associated recursive matrix algorithms, are *cache oblivious* [43], that is they optimize cache locality without any explicit blocking of the sort conventionally done in LAPACK and ScaLAPACK, or any of the tuning parameters (beyond the L1 cache size).

	LAPACK	SCALAPACK
Linear Equations	GESV (LU) POSV (Cholesky) SYSV (LDL^T)	PxGESV PxPOSV missing
Least Squares (LS)	GELS (QR) GELSY (QR w/pivoting) GELSS (SVD w/QR) GELSD (SVD w/D&C)	PxGELS missing driver missing driver missing
Generalized LS	GGLSE (GRQ) GGGLM (GQR)	missing missing
Symmetric EVD	SYEV (inverse iteration) SYEVD (D&C) SYEVR (RRR)	PxSYEV missing missing
Nonsymmetric EVD	GEES (HQR) GEEV (HQR + vectors)	missing driver missing driver
SVD	GESVD (QR) GESDD (D&C)	PxGESVD missing
Generalized Symmetric EVD	SYGV (inverse iteration) SYGVD (D&C)	PxSYGVX missing
Generalized Nonsymmetric EVD	GGES (HQZ) GGEV (HQZ + vectors)	missing missing
Generalized SVD	GGSVD (Jacobi)	missing

Table 2: LAPACK codes and corresponding parallel version in ScaLAPACK. Underlying LAPACK algorithm shown in parentheses. "Missing" means both drivers and computational routines are missing. "Missing driver" means that underlying computational routines are present.

The reported benefits of these data structures and associated algorithms to which they apply is usually slightly higher peak performance on large matrices, and a faster increase towards peak performance as the dimension grows. Sometimes slightly modified tuned BLAS are use for operations on matrices assumed to be in L1 cache. The biggest payoff by far is for factoring symmetric matrices stored in packed format, where the current LAPACK routines are limited to the performance of BLAS2, which do $O(1)$ flops per memory reference, whereas the recursive algorithms can use the faster BLAS3, which do $O(n)$ flops per memory reference, and so can be optimized to hide slower memory bandwidth and latencies

The drawback of these algorithms is their use of a completely different and rather complicated data structure, which only a few expert users could be expected to use. That leaves the possibility of copying the input matrices in conventional column-major (or row-major) format into the recursive data structure. Furthermore, they are only of benefit for "one-sided factorizations" (LU, LDL^T, Cholesky, QR), but none of the "two-sided factorizations" needed for the EVD or SVD. (There is a possibility they might be useful when no eigenvectors or singular vectors are desired; see below.)

We will incorporate the factorization of symmetric packed matrices using the recursive data structures into LAPACK, copying the usual data structure in-place to the recursive data structure. The copying costs $O(n^2)$ in contrast to the overall $O(n^3)$ operation count, so the asymptotic speeds should be the same. We will explore the use of the recursive data structures for other parts of LAPACK, but for the purposes of ease of use, we will keep the same column-major interface data structures that we have now.

3. Ashcraft, Grimes and Lewis [7] proposed a variation of Bunch-Kaufman factorization for solving symmetric indefinite systems $Ax = b$ by factoring $A = LDL^T$ with different pivoting. The current Bunch-

Kaufman factorization is backward stable for the solution of $Ax = b$ but can produce unbounded L factors. Better pivoting provides better accuracy for applications requiring bounded L factors, like optimization and the construction of preconditioners [55, 27]. In addition, Livne [64] has developed alternative equilibration strategies for symmetric indefinite matrices whose accuracy benefits we need to evaluate.

4. A Cholesky factorization with diagonal pivoting [54, 55] that avoids a breakdown if the matrix is nearly indefinite/rank-deficient is valuable for optimization problems (and has been requested by users), and is also useful for the high accuracy solution of the symmetric positive definite EVD, see below. For both this pivoting strategy and the one proposed above by Askcraft/Grimes/Lewis, published results indicate that on uniprocessors (LAPACK), the extra search required (compared to Bunch-Kaufman) has a small impact on performance. This may not be the case for distributed memory (ScaLAPACK), in which the extra searching and pivoting may involve nonnegligible communication costs; we will evaluate this.

5. Progress has been made in the development of new algorithms for computing or estimating the condition number of tridiagonal [34, 53] or triangular matrices [41]. These algorithms play an important role in obtaining error bounds in matrix factorizations and we plan to evaluate and incorporate the most promising algorithms in a future release.

3.2 Algorithmic improvements for the solution of eigenvalue problems

Algorithmic improvements to the current LAPACK eigensolvers concern both accuracy and performance.

1. Braman, Byers, and Mathias proposed in their SIAM Linear Algebra Prize winning work [21, 22] an up to 10x faster Hessenberg QR-algorithm for the nonsymmetric EVD. This is the bottleneck of the overall nonsymmetric EVD, for which we expect significant speedups. Byers recently spent a sabbatical visiting PI Demmel where he did much of the software engineering required to convert his prototype into LAPACK format. We also expect an extension of this work with similar benefits to the QZ algorithm for Hessenberg-triangular pencils, with collaboration from Mehrmann (see the attached letter). We will also be able to exploit these techniques to accelerate our routines for (block) companion matrices; see Sec. 4.

2. An early version of an algorithm based on Multiple Relatively Robust Representations (MRRR) [71, 72, 74, 73] for the tridiagonal symmetric eigenvalue problem (STEGR) was incorporated into LAPACK version 3. This algorithm promised to replace the prior $O(n^3)$ QR algorithm (STEQR) or $\approx O(n^{2.3})$ divide & conquer (STEDC) algorithm with an $O(n^2)$ algorithm. In fact, it should have cost $O(nk)$ operations to compute the nk entries of k n-dimensional eigenvectors, the minimum possible work, in a highly parallel way. In fact the algorithm in LAPACK v.3 did not cover all possible eigenvalue distributions, and resorted to a slower and less accurate algorithm based on classical inverse iteration for “difficult” (highly clustered) eigenvalue distributions. The inventors of MRRR, Parlett and Dhillon, have continued to work on improving this algorithm, and very recently have proposed a solution for the last hurdle [75] and now pass our tests for the most extreme examples of highly multiple eigenvalues. (These arose in our tests from very many large “glued Wilkinson matrices”, constructed so that large numbers of mathematically distinct eigenvalues agreed to very high accuracy, much more than double precision. The proposed solution involves randomization, making small random perturbations to an intermediate representation of the matrix to force all eigenvalues to disagree in at least 1 or 2 bits.) Given the solution to this last hurdle, we can propagate this algorithm to all the variants of the symmetric EVD (banded, generalized, packed, etc.) in LAPACK (this has NPACI funding through the end of 2004). For this proposal we will go beyond this and parallelize this algorithm for the corresponding ScaLAPACK symmetric EVD routines. Currently ScaLAPACK only has parallel versions of the oldest, least efficient (or least accurate) LAPACK routines, because we had been waiting for the completion of the sequential MRRR algorithm. This final MRRR algorithm

requires some care at load balancing because the Multiple Representations used in MRRR represent subsets of the spectrum based on how clustered they are, which may or may not correspond to a good load balance. Initial work by our collaborators in this area is very promising [14].

3. The MRRR algorithm can and should also be applied to the SVD, replacing the current $O(n^3)$ or $\approx O(n^{2.3})$ bidiagonal SVD algorithms with an $O(n^2)$ algorithm. The necessary theory and a preliminary prototype implementation have been developed [49]. Grosser, the author of [49] visited us to help with this development, and we expect a visit from the same team to help again (see the attached letter from Lang, who was Grosser's adviser).

4. There are three phases in the EVD (or SVD) of a dense or band matrix: (1) reduction to tridiagonal (or bidiagonal) form, (2) the subsequent tridiagonal EVD (or bidiagonal SVD), and (3) backtransforming the eigenvectors (or singular vectors) of the tridiagonal (or bidiagonal) to correspond to the input matrix. If many (or all) eigenvectors (or singular vectors) are desired the bottleneck had been phase 2. But now that the MRRR algorithm promises to make phase 2 cost just $O(n^2)$ in contrast to the $O(n^3)$ costs of phases 1 and 3, our attention turns to these phases. In particular, Howell and Fulton [44] recently devised a new variant of reduction to bidiagonal form for the SVD, that has the potential to eliminate half the memory references, by reordering the floating point operations (flops). Howell and Fulton fortunately discovered this algorithm during the deliberations of the recent BLAS standardization committee (led by PI Dongarra, and participated in by PI Demmel), because they required new BLAS routines to accomplish this, which we added to the standard (routines GEMVT and GEMVER [19]). We call these routine BLAS2.5, because they do many more than $O(1)$ but fewer than $O(n)$ flops per memory reference. Preliminary tests indicate speed ups of up to nearly 2x.

5. For the SVD, when only left or only right singular vectors are desired, there are other variations on phase 1 to consider, that reduce both floating point operations and memory references [11, 76]. Initial results indicate reduced operation counts by a ratio of up to .75, but at the possible cost of numerical stability for some singular vectors. We will evaluate these for possible incorporation.

6. When few or no vectors are desired, the bottleneck shifts entirely to phase 1. Bischof and Lang [15] have proposed a Successive Band Reduction (SBR) algorithm that will asymptotically (for large dimension n) change most of the BLAS2 operations in phase 1 to BLAS3 operations (see the attached letter from Lang). They report speed ups of almost 2.4x. This approach is not suitable when a large number of vectors are desired, because the cost of phase 3 is much larger per vector. In other words, depending on how many vectors are desired, we will either use the SBR approach or the one-step reduction (the Howell/Fulton variant for the SVD, and the current LAPACK code for the symmetric EVD). And if only left or only right singular vectors are desired, we might want to use the algorithms described in bullet 5. This introduces a machine-dependent tuning parameter to choose the right algorithm; we discuss tuning of this and other parameters in Sec. 6. It may also be possible to use Gustavson's recursive data structures to accelerate SBR; we will consider this.

7. Drmač and Veselić have made significant progress on the performance of the one-sided Jacobi algorithm for computing singular values with high relative accuracy [31, 40]. In contrast to the algorithms described above, theirs can compute most or all of the significant digits in tiny singular values, when these digits are determined accurately by the input data, and when the above algorithms return only roundoff noise. The early version of this algorithm introduced by PI Demmel in [33, 31] was rather slower than the conventional QR-iteration-based algorithms, and so much slower than the MRRR algorithms discussed above. But recent results reported by Drmač at [59] show that a combination of clever optimizations have finally led to an accurate algorithm that is *faster* than the original QR-iteration-based algorithm. Innovations include preprocessing by QR factorizations with pivoting, block application of Jacobi rotations, and early termination. Two immediate applications include the (full matrix) SVD and the symmetric positive-definite EVD, by first reducing to the SVD by using the Cholesky-with-pivoting algorithm discussed earlier.

8. Analogous high accuracy algorithms for the symmetric indefinite EVD have also been designed. One approach by Slapničar [79, 78] uses a J-symmetric Jacobi algorithm with hyperbolic rotations, and another one by Dopico/Molera/Moro [39] does an SVD, which "forgets" the signs of the eigenvalues, and then reconstructs the signs. The latter can directly benefit by the Drmač/Veselič algorithm above. We will investigate which of these two algorithm meets our criteria for inclusion; see the attached letter from Dopico.

4 Extending current functionality.

In this section, we outline possible extensions of the functionalities available in LAPACK and ScaLAPACK. These extensions are mostly motivated by users but also by research progress.

1. Following several user requests, we plan to include several updating facilities in a new release. While updating matrix factorizations like Cholesky, LDL^T, LU, QR [46] have a well established theory and unblocked (i.e. non cache optimized) implementations exist, e.g. in LINPACK [35], the efficient update of the SVD is a current research topic [50]. Furthermore, divide & conquer based techniques are promising for a general framework of updating eigendecompositions of submatrices, this will be a future research topic.

2. Semi-separable matrices are generalizations of the inverses of banded matrices, with the property that any rectangular submatrix lying strictly above or strictly below the diagonal has a rank bounded by a small constant. Recent research has focused on methods exploiting semiseparability, or being a sum of a banded matrix and a semiseparable matrix, for better efficiency [26, 86]. We will consider the development of such algorithms in a future release. Most exciting is the recent observation of Gu, Bini and others that a companion matrix is banded plus semiseparable, and that this structure is preserved under QR iteration to find its eigenvalues. This observation let us accelerate the standard method used in Matlab and other libraries for finding roots of polynomials from $O(n^3)$ to $O(n^2)$. Our initial rough prototype of this code starts being faster than the highly tuned LAPACK eigensolver for n between 100 and 200, and becomes arbitrarily faster for larger n. While the current algorithm (joint work with Gu) has been numerically stable on all examples we have tried so far, more work needs to be done to guarantee stability in all cases. The same technique should apply to finding eigenvalues of block companion matrices, i.e. matrix polynomials, yielding speedups proportional to the degree of the matrix polynomial.

3. Eigenvalue problems for matrix polynomials [45] are common in science and engineering. The most common case is the quadratic eigenvalue problem $(\lambda^2 M + \lambda D + K)x = 0$, where typically M is a mass matrix, D a damping matrix, K a stiffness matrix, λ a resonant frequency, and x a mode shape. The classical solution is to linearize this eigenproblem, asking instead for the eigenvalues of a system of twice the size: $\lambda \cdot \begin{bmatrix} 0 & I \\ M & D \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} I & 0 \\ 0 & K \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = 0$ where $y_2 = x$ and $y_1 = \lambda x$. But there are a number of ways to linearize, and some are better at preserving symmetries in the solution of the original problem or saving time than others. There has been a great deal of recent work on picking the right linearization and subsequent algorithm for its EVD to preserve desired structures, and we have had user requests to incorporate some of these structures. In particular, for the general problem $\sum_{i=0}^{k} \lambda^i \cdot A_i \cdot x = 0$, the requested cases are symmetric ($A_i = A_i^T$, arising in mechanical vibrations without gyroscopic terms), its even ($A_i = (-1)^i A_i^T$) and odd ($A_i = (-1)^{i+1} A_i^T$) variations (used with gyroscopic terms and elsewhere), and palindromic ($A_i = A_{k-i}^T$, arising in discrete time periodic and continuous time control). Recent references include [4, 5, 6, 66, 67, 68, 81, 82, 84, 12]; see the attached letter from Mehrmann.

4. Matrix functions (square root, exponential, sign function) play an important role in the solution of differential equations in science and engineering, and have been requested by users. Recent research

progress has led to the development of several new algorithms [28, 56, 80, 70, 23, 24, 25, 60, 61, 8] that could be included in a future release.

5. The eigenvalue and singular value decomposition of products and quotients of matrices plays an important role in control theory, we consider incorporating such functionalities from the software library SLICOT [13], using our improved underlying EVD algorithms (Mehrmann, see attached letter, was one of the designers of SLICOT). Following another request from the control theory community, we could incorporate efficient solvers for Sylvester and Lyapunov equations that are also currently in SLICOT.

6. Multiple user requests concern the development of out-of-core versions of matrix factorizations. ScaLAPACK prototypes [17] are under development which implement out-of-core data management for the LU, QR, and Cholesky factorizations [37, 36]. Users have asked for two kinds of parallel I/O: to a single file from a sequential LAPACK program (possible with sequential I/O in the reference implementation), and to a single file from MPI-based parallel I/O in ScaLAPACK.

5 Improving ease of use.

First we discuss language interfaces to Sca/LAPACK, and then higher level issues related to their use on possibly heterogeneous clusters.

By exploiting features of modern programming languages and making Sca/LAPACK easier to use we will be in a position to capture a new generation of users who are not interested in using Fortran 77, the current implementation language. Balanced against this are the cost in performance, memory usage or even reliability of some of these features, and the difficulty of building and maintaining one version of these very large libraries, let alone several versions in different languages. Since we do not believe that we can simultaneously maximize performance, memory efficiency, ease of use, reliability, and ease of maintenance, we have decided on the following strategy: Maintain one core version in Fortran 77, and provide wrappers in other languages just for the driver routines. Based on current user demand, these other languages will include Fortran 95 and C, as well as selected higher level languages like Matlab, Python and Mathematica (where ultimate ease of use is possible, such as typing "$x = A\backslash b$" to solve $Ax = b$ no matter what type, mathematical properties or data structure A has). Users of the Fortran 77 version will get maximum performance and memory efficiency, but worst ease-of-use. Users of the wrappers will have better ease of use and reliability, but worse performance and memory efficiency in some cases. We, the developers, will have a tractable amount of code to maintain.

We justify the above mentioned tradeoffs in more detail. First consider memory usage. Sca/LAPACK currently expects the user to pass in the right amount of workspace, since the (old) F77 standard does not include allocate/malloc. The amount of workspace can be a complicated function of the problem to be solved, so the user can query each Sca/LAPACK driver routine to inquire what the optimal workspace actually is. Thus, a wrapper in F95 or C can query the F77 driver to get the optimal workspace, try to allocate it, and then call the F77 driver again to solve the problem. If the maximum workspace is not available, we could make drivers queryable for the minimum workspace and try again, and eventually return an error message if too little space is available. This approach is reliable as long as we allocate workspace on the heap with allocate/malloc, and not on the stack, since there is generally no graceful way to recover from stack overflow.

The details of the interface can affect the tradeoff between ease of use and performance. For example, there are two F95 interfaces to LAPACK, LAPACK95 [10] and LAPACK3E [2]. LAPACK95 uses assumed-size arrays, so that passing in a submatrix can cause a copy of $O(n^2)$ input data to a new contiguous space (and copying back again on output), costing both time and memory. In contrast LAPACK3E supports the older Fortran 77 style interface that works with submatrices in-place. Based on the larger usage of LAPACK95 over LAPACK3E, we will go with the LAPACK95 interface. We also plan to drop support for the CLAPACK translation of LAPACK to C, and just supply a C wrapper. We do not plan to provide a separate C++ wrapper.

Now we consider ease-of-use issues in a possibly dynamically changing cluster environment, or in one where the user does not know and does not wish to know the computational resources available, but may have other demands like reproducibility. These enhancements will incorporate novel techniques for managing high latencies, low bisection bandwidths, and other characteristics of the modern computing environment, and will also be designed to adapt their behavior in response to information obtained from other system components. Our primary goals are to develop a new generation of software libraries and algorithms needed for the effective and reliable use of capacity cluster and high performance capability computers, to validate the libraries and algorithms on important scientific and engineering applications, and to provide mechanisms for the integration of other software components frequently used in applications.

We will develop parameterizable and annotatable software libraries that will permit performance tuning and dynamic algorithm selection for a broad range of architectures. These extensions will take the form of performance models and assertions made by the user and library writer. In addition, we will promote standards activities for key aspects of execution. The libraries and algorithms will be designed and implemented for complex and dynamic environments and cover a wide range of scales in time and space. Successful implementation along these lines requires:

- predictability and robustness of accuracy and performance,
- run–time resource management and algorithm selection,
- support for a multiplicity of programming environments,
- reproducibility and auditability of the computations
- new algorithmic techniques for latency tolerant applications; and
- scalable algorithms that expose enough concurrency to keep resources busy, and for latency hiding.

Predictability in accuracy and performance is an often expressed requirement whose importance increases in the cluster environment. Robustness in accuracy and performance addresses the issues associated with maintaining an expected level of accuracy or performance for changes in the computing environment, problem size, or other application parameters. The library will acquire, at run–time, cluster and application information necessary to make decisions on which components and algorithms will best solve the underlying problem.

The growing gap between the speed of microprocessors and memory technology, resulting in deep memory hierarchies imply that the memory subsystem is a critical performance factor posing increased challenges in the design and implementation of algorithm and software systems [38]. Novel latency tolerant algorithms that explores a wider range of the latency/bandwidth/memory space are needed. We propose to identify algorithms and applications that would benefit by a latency tolerant approach and construct new algorithms where appropriate. Our initial experiments show, for example, that up to 10x speedups are attainable by using different reduction algorithms and other communication primitives in the Basic Linear Algebra Communication Subroutines (BLACS) underlying ScaLAPACK [69, 85] In a multithreaded execution, when a processor reaches a point where remote memory access is necessary, the request is sent out on the network and a context–switch occurs to a new thread of computation. This effectively masks a long and unpredictable latency due to remote loads, thereby providing tolerance to remote access latency. As a byproduct, we will develop standards to profile the degree of parallelism, granularity, precision, instruction set mix, inter–processor communication, latency etc. These tools will develop and evolve as the cluster environment matures.

For numerical libraries there are deeper issues relating to portability that we will address. For example, while it is reasonable to assume IEEE arithmetic, hardware vendors provide variants on the standard resulting in the same computation producing different results when executed on different parts of the cluster [18]. Furthermore, repeatability is an issue for applications that are ill–conditioned or are sensitive to rounding errors. We plan to develop high confidence algorithms and software that will address this issue. The implementation strategies would allow the user to control the application to guarantee repeatability, perhaps at the expense of performance (Java's original goal was exact reproducibility, but backed away from this for floating point applications because of the performance penalties caused by different platforms having different width floating point registers, i.e. Intel and Sun).

To meet the above goals, we will develop

- **Parameterizable libraries**. We propose to design and construct libraries that are parameterized to allow their performance to be optimized over a range of current and future memory hierarchies, including the ones expected in computational clusters.

- **Annotated libraries**. We will identify opportunities where information about algorithms contained in library functions can aid the compilation process and run–time environment. At the library interface level, this includes memory-hierarchy tuning parameters and a performance model that depends on input parameters, such as problem size.

- **Cluster environment**. We will pioneer new methods for packaging, distributing, and accessing algorithms and software technology for cluster applications.

In summary, the new libraries will speed transfer of recent algorithmic technology in linear algebra, hierarchical methods, and other areas to large–scale computer users. A new division of labor between compiler writers, library writers, and algorithm developers and application developers will emerge. The capability of controlling large complex heterogeneous and dynamic applications over a distributed network will enable the assembly of multidisciplinary applications, parts of which are built by independent teams.

6 Performance tuning.

Our experience with automatics tuning of dense linear algebra with ATLAS [90] and sparse linear algebra with Sparsity [58, 87] shows that their value arises from their ability to discover and use performance information by empirical methods that would otherwise be difficult to come by. While many of the techniques used by ATLAS and Sparsity are specific to their particular operations, many others are more generally useful. For instance, information about o the type and number of floating units, the pipeline length, the number TLB entries, etc. are all generally useful in code optimization. During this stage of our work we plan to develop methodologies so we and other developers can query an ATLAS like system for this information in order to aid in developing automatic performance models for algorithm tuning and selection.

There are a large number of tuning parameters that LAPACK and ScaLAPACK use. There are over 1300 calls in LAPACK to the routine ILAENV which takes as input the name of the calling LAPACK subroutine (such as SGETRF which performs LU decomposition with partial pivoting) and parameters describing the size of the problem (such as the matrix dimension n) and returns a tuning parameter (such as a threshold n_2, such that if $n \leq n_2$ SGETRF will use an unblocked (BLAS2) routine, and if $n > n_2$ SGETRF will use a blocked BLAS3 routine. The idea is that for small enough n the overhead of the BLAS3 routine outweighs the speeds and BLAS2 is faster. Exactly where the n_2 threshold should be depends on the platform and BLAS implementation, and is best determined empirically. We propose to develop an automatic tuning system analogous to ATLAS and Sparsity for this purpose.

In the case of ScaLAPACK there are yet more parameters having to do with the number of processors, the dimensions determining the block cyclic layout of the data, and so on. Tuning here (depending on the maximum available number of processors, not all of which we may want to use, and their computational and communication speeds) will require performance modeling to pick the best configuration.

These are illustrations of the basic problem of designing numerical library software that addresses both computational time and space complexity issues on the user's behalf and in a manner as transparent to the user as possible. The software intends to allow users to either link against an archived library of executable routines or benefit from the convenience of prebuilt executable programs without the hassle of properly having to resolve linker dependencies. The user is assumed to call one routine from a serial environment while working on a single processor of the cluster. The software executes the application. If it is possible to finish executing the problem faster by mapping the problem into a parallel environment, then this is the thread of execution taken. Otherwise, the application is executed locally with the best choice of a serial algorithm. The details for parallelizing the user's problem such as resource discovery, selection, and allocation, mapping the data onto (and off of) the working cluster of processors, executing the user's application in parallel, freeing the allocated resources, and returning control to the user's process in the serial environment from which the procedure began are all handled by the software. Whether the application was executed in a parallel or serial environment is presumed not to be of interest to the user but may be explicitly queried. All the user knows is that the application executed successfully and, hopefully, in a timely manner. Target systems are intended to be "Beowulf like". There are essentially three components to the software: data collection routines, data movement routines, and application routines.

Our software will dynamically make decisions on how to solve the user's problem by coupling the cluster state information with knowledge of the particular application. Specifically, a decision is based upon the scheduler's ability to successfully predict that a particular subset of the available processors on the cluster will enable a reduction of the total time to solution when compared to serial expectations for the specific application and user parameters. The relevant times are the time that is spent handling the user's data before and after the parallel application plus the amount of time required to execute the parallel application. If the decision is to solve the user's problem locally (sequentially) then the relevant LAPACK routine is executed. On the contrary, if the decision is to solve the user's problem in parallel then a process is forked that will be responsible for spawning the parallel job and the parent process waits for its return in the sequential environment. The selected processors are allocated (in MPI), the user's data is mapped (block cyclically decomposed) onto the processors (the data may be in memory or on disk), the parallel application is executed (e.g. ScaLAPACK), the data is reverse mapped, the parallel process group is freed, and the solution and control are returned to the user's process.

At runtime our software makes choices at the software and hardware levels for obtaining a best parameter set for the selected algorithm by applying expertise from the literature and empirical investigations of the core kernels on the target system. The algorithm selection depends on the size of the input data and empirical results from previous runs for the particular operation on the cluster. The overheads associated with this dynamic adaptation of the user's problem to the hardware and software systems available can be minimal.

7 Reliability and support.

Specifically, we will develop tools, libraries, networking protocols, and methodologies to facilitate:

- Building, tuning, and testing of computational software libraries on multiple computing platforms and operating environments.
- Automatic cataloging of software libraries according to the target platforms and operating environments for which they were built.
- Accurate and precise identification of the characteristics of a computing platform and operating environment which affect the selection of computational software libraries.
- Automatically locating and incorporating appropriate versions of desired software components, into a program - where this binding may occur at either build time or run time.
- Distribution and location of software components in a way which preserves the integrity of those components and which utilizes existing distribution channels (e.g. mirrored web and ftp servers) with minimal disruption.
- Easy and nearly-automatic configuration of target platforms to support diverse applications for different computing environments.

In order to facilitate development of software which is portable across a variety of computing platforms, we will provide a system for automatic compilation and testing of new or updated libraries. This will consist of:

- Facilities for submitting, over the Internet, new or updated libraries for compilation and testing,
- An access control mechanism to prevent the use of this system by unauthorized users,
- A set of computing platforms to which libraries can be submitted for compilation and testing,
- A queuing system for submitting compilation and testing tasks to each platform and monitoring the progress of these operations,
- A "sandbox" environment on each platform to allow for safe compilation and testing of untrusted codes without compromising the host environment,
- Reporting of test results to the software author or maintainer, and
- If compilation and tests are successful, automatic cataloging and filing of the compiled libraries.

Our goals in establishing this system include: encouraging development of portable software libraries, encouraging development of testing methodologies as part of the development of software libraries, providing a mechanism by which libraries for diverse platforms can be automatically produced, and automatically cataloging libraries with sufficient precision to allow effective matching of libraries to target platforms.

8 External Collaboration and Management.

Both PIs have a long and successful history of collaboration on LAPACK and ScaLAPACK, and will continue their past practices of periodic face-to-face meetings, design reviews, presentations of draft designs at conferences to get user inputs, and so on. The difference between managing this effort and prior ones is our attempt to enlist the help of a number of outside people and organizations. The ones listed below have agreed to help, and many of them have sent us letters of support which are attached to the proposal. Support ranges from offering their software and ideas to help in programming and development (see the attached letters for details).

1. Cleve Moler, Founder, The Mathworks, www.mathworks.com (email only)
2. Bruce Greer, Principal Engineer, Math Kernel Library, Intel, www.intel.com
3. Greg Henry, Senior SW Engineer, Math Kernel Library, Intel, www.intel.com
4. Richard Altmeier, VP of Storage and SW Engineering, SGI, www.sgi.com
5. John Levesque, Senior Technologist, Cray, www.cray.com (email only)
6. Prof. Froilan Dopico, Univ. Carlos III de Madrid, Spain `www.uc3m.es/uc3m/dpto/MATEM/indice.html`
7. Prof. Kresimir Veselič, FernUniversität Hagen, Germany, `www.fernuni-hagen.de/MATHPHYS/veselic/welcome.html`
8. Prof. Zlatko Drmač, University of Zagreb, Croatia, `www.math.hr/~drmac`
9. Prof. Sven Hammarling, Principal Consultant, Numerical Algorithms Group Ltd (NAG), www.nag.co.uk
10. Prof. Volker Mehrmann, Technische Universität Berlin, `www.math.tu-berlin.de/~mehrmann`
11. Dr. Robert Schreiber, Principal Scientist, HP Laboratories, www.hp.com

We will of course include these collaborators in our meetings as appropriate. In addition, since our past experience is that testing is a major bottleneck, as described in Sec. 7, we plan to develop a web-based service for testing and timing so that our collaborators can remotely submit routines for extensive testing.

References

[1] B. S. Andersen, J. Wazniewski, and Gustavson. F. G. A recursive formulation of Cholesky factorization of a matrix in packed storage. *ACM Trans. Math. Software*, 27(2):214–244, 2001.

[2] E. Anderson. Lapack3e. www.netlib.org/lapack3e, 2003.

[3] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Blackford, and D. Sorensen. *LAPACK Users' Guide (third edition)*. SIAM, Philadelphia, 1999.

[4] T. Apel, V. Mehrmann, and D. Watkins. Structured eigenvalue methods for the computation of corner singularities in 3D anisotropic elastic structures. Preprint 01–25, SFB393, Sonderforschungsbereich 393, Fakultät für Mathematik, TU Chemnitz, 2001.

[5] T. Apel, V. Mehrmann, and D. Watkins. Structured eigenvalue methods for the computation of corner singularities in 3D anisotropic elastic structures. *Comp. Meth. App. Mech. Eng.*, 191:4459–4473, 2002.

[6] T. Apel, V. Mehrmann, and D. Watkins. Numerical solution of large scale structured polynomial eigenvalue problems. In *Foundations of Computational Mathematics*. Springer Verlag, 2003.

[7] C. Ashcraft, R. G. Grimes, and J. G. Lewis. Accurate symmetric indefinite linear equation solvers. *SIAM J. Matrix Anal. Appl.*, 20(2):513–561, 1998.

[8] Z. Bai, J. Demmel, and M. Gu. Inverse free parallel spectral divide and conquer algorithms for nonsymmetric eigenproblems. *Num. Math.*, 76:279–308, 1997. UC Berkeley CS Division Report UCB//CSD-94-793, Feb 94.

[9] D. Bailey, J. Demmel, G. Henry, Y. Hida, J. Iskandar, W. Kahan, S. Kang, A. Kapur, X. Li, M. Martin, B. Thompson, T. Tung, and D. Yoo. Design, implementation and testing of extended and mixed precision BLAS. *ACM Trans. Math. Soft.*, 28(2):152–205, June 2002.

[10] V. Barker, S. Blackford, J. Dongarra, J. Du Croz, S. Hammarling, M. Marinova, J. Wasniewski, and P. Yalamov. *LAPACK95 Users' Guie*. SIAM, 2001. www.netlib.org/lapack95.

[11] J. Barlow, N. Bosner, and Z. Drmač. A new stable bidiagonal reduction algorithm. `www.cse.psu.edu/~barlow/fastbidiag3.ps`, 2004.

[12] P. Benner, R. Byers, V. Mehrmann, and H. Xu. Numerical computation of deflating subspaces of embedded Hamiltonian pencils. Technical Report SFB393/99-15, Fakultät für Mathematik, TU Chemnitz, 09107 Chemnitz, FRG, 1999.

[13] P. Benner, V. Mehrmann, V. Sima, S. Van Huffel, and A. Varga. SLICOT - a subroutine library in systems and control theory. *Applied and Computational Control, Signals, and Circuits*, 1:499–539, 1999.

[14] P. Bientinisi, I. S. Dhillon, and R. van de Geijn. A parallel eigensolver for dense symmetric matrices based on multiple relatively robust representations. Technial Report TR-03-26, Computer Science Dept., University of Texas, 2003.

[15] C. H. Bischof, B. Lang, and X. Sun. A framework for symmetric band reduction. *ACM Trans. Math. Software*, 26(4):581–601, 2000.

[16] 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, Philadelphia, 1997.

[17] L. S. Blackford, J. Choi, A. Cleary, J. Demmel, I. Dhillon, J. J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. W. Walker, and R. C. Whaley. Scalapack prototype software. Netlib, Oak Ridge National Laboratory, 1997.

[18] L. S. Blackford, A. Cleary, J. Demmel, I. Dhillon, J. J. Dongarra, S. Hammarling, A. Petitet, H. Ren, K. Stanley, and R. C. Whaley. Practical experience in the dangers of heterogeneous computing. LAPACK Working Note No.112. Technical Report CS-96-330, Department of Computer Science, University of Tennessee, 107 Ayres Hall, Knoxville, TN 37996-1301, USA, 1996.

[19] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley. An updated set of Basic Linear Algebra Subroutines (BLAS). *ACM Trans. Math. Soft.*, 28(2), June 2002.

[20] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, R. C. Whaley, Z. Maany, F. Krough, G. Corliss, C. Hu, B. Keafott, W. Walster, and J. Wolff v. Gudenberg. Basic Linear Algebra Subprograms Techical (BLAST) Forum Standard. *Intern. J. High Performance Comput.*, 15(3-4), 2001.

[21] K. Braman, R. Byers, and R. Mathias. The multishift QR algorithm. Part I: Maintaining well-focused shifts and Level 3 performance. *SIAM J. Matrix Anal. Appl.*, 23(4):929–947, 2001.

[22] K. Braman, R. Byers, and R. Mathias. The multishift QR algorithm. Part II: Aggressive early deflation. *SIAM J. Matrix Anal. Appl.*, 23(4):948–973, 2001.

[23] R. Byers. Numerical stability and instability in matrix sign function based algorithms. In C. Byrnes and A. Lindquist, editors, *Computational and Combinatorial Methods in Systems Theory*, pages 185–200. North-Holland, 1986.

[24] R. Byers. Solving the algebraic Riccati equation with the matrix sign function. *Lin. Alg. Appl.*, 85:267–279, 1987.

[25] R. Byers, C. He, and V. Mehrmann. The matrix sign function method and the computation of invariant subspaces. preprint, 1994.

[26] S. Chandrasekaran and M. Gu. Fast and stable algorithms for banded plus semiseparable systems of linear equations. *SIAM J. Matrix Anal. Appl.*, 25(2):373–384, 2003.

[27] S. H. Cheng and N. Higham. A modified Cholesky algorithm based on a symmetric indefinite factorization. *SIAM J. Mat. Anal. Appl.*, 19(4):1097–1110, 1998.

[28] P. Davies and N. J. Higham. A Schur-Parlett algorithm for computing matrix functions. *SIAM J. Matrix Anal. Appl.*, 25(2):464–485, 2003.

[29] J. Demmel. *Applied Numerical Linear Algebra*. SIAM, 1997.

[30] J. Demmel. CS 267 Course Notes: Applications of Parallel Processing. Computer Science Division, University of California, 1999. `http://www.cs.berkeley.edu/~demmel/cs267_Spr99`.

[31] J. Demmel, M. Gu, S. Eisenstat, I. Slapničar, K. Veselić, and Z. Drmač. Computing the singular value decomposition with high relative accuracy. *Lin. Alg. Appl.*, 299(1–3):21–80, 1999.

[32] J. Demmel, Y. Hida, W. Kahan, X. S. Li, S. Mokherjee, and E. J. Riedy. Error bounds from extra precise iterative refinement. 50 pages, in progress, 2004.

[33] J. Demmel and K. Veselić. Jacobi's method is more accurate than QR. *SIAM J. Mat. Anal. Appl.*, 13(4):1204–1246, 1992.

[34] I. S. Dhillon. Reliable computation of the condition number of a tridiagonal matrix in O(n) time. *SIAM J. Matrix Anal. Appl.*, 19(3):776–796, 1998.

[35] J. Dongarra, J. Bunch, C. Moler, and G. W. Stewart. *LINPACK User's Guide*. SIAM, Philadelphia, PA, 1979.

[36] J. Dongarra and E. D'Azevedo. The design and implmentation of the parallel out-of-core ScaLAPACK LU, QR, and Cholesky factorization routines. Computer Science Dept. Technical Report CS-97-347, University of Tennessee, Knoxville, TN, January 1997. www.netlib.org/lapack/lawns/lawn118.ps.

[37] J. Dongarra, S. Hammarling, and D. Walker. Key concepts for parallel out-of-core LU factorization. Computer Science Dept. Technical Report CS-96-324, University of Tennessee, Knoxville, TN, April 1996. www.netlib.org/lapack/lawns/lawn110.ps.

[38] J. Dongarra, R. van de Geijn, and D. Walker. A look at scalable dense linear algebra libraries. In *Scalable High-Performance Computing Conference*. IEEE Computer Society Press, April 1992.

[39] F. M. Dopico, J. M. Molera, and J. Moro. An orthogonal high relative accuracy algorithm for the symmetric eigenproblem. *SIAM. J. Mat. Anal. Appl.*, 25(2):301–351, 2003.

[40] Z. Drmač and K. Veselić. New fast and accurate Jacobi SVD algorithm. Technical report, Dept. of Mathematics, University of Zagreb, 2004.

[41] I. S. Duff and C. Vömel. Incremental Norm Estimation for Dense and Sparse Matrices. *BIT*, 42(2):300–322, 2002.

[42] E. Elmroth, F. Gustavson, I. Jonsson, and B. Kå gström. Recursive blocked algorithms and hybrid data structures for dense matrix library software. *SIAM Review*, 46(1):3–45, 2004.

[43] L. Arge et al. Cache-oblivious and cache-aware algorithms. Schloss Dagstuhl International Conference, www.dagstuhl.de/04301, 2004.

[44] C. Fulton, G. Howell, J. Demmel, and S. Hammarling. Cache-efficient bidiagonalization using BLAS 2.5 operators. 28 pages, in progress, 2004.

[45] I. Gohberg, P. Lancaster, and L. Rodman. *Matrix Polynomials*. Academic Press, New York, 1982.

[46] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, 3rd edition, 1996.

[47] S. Graham and M. Snir et al. The Future of Supercomputing - An Interim Report. National Research Council of the National Academies, 2003.

[48] J. Grosh, A. Laub, D. Nelson, and et al S. Szykman. Federal Plan for High-End Computing: Report of the High-End Computing Revitalization Task Force (HECRTF). OSTP report, www.hpcc.gov/hecrtf-outreach, 10 May 2004.

[49] B. Grosser. *Ein paralleler und hochgenauer O(n^2) Algorithmus für die bidiagonale Singulärwertzerlegung*. PhD thesis, University of Wuppertal, Wuppertal, Germany, 2001.

[50] M. Gu and S. C. Eisenstat. A stable and fast algorithm for updating the singular value decomposition. in preparation.

[51] J. A. Gunnels and F. G. Gustavson. Representing a symmetric or triangular matrix as two rectangular matrices. Poster presentation. SIAM Conference on Parallel Processing, San Francisco, 2004.

[52] F. Gustavson. Recursion leads to automatic variable blocking for dense linear-algebra algorithms. *IBM J. of Res. and Dev.*, 41(6), 1997. www.research.ibm.com/journal/rd/416/gustavson.html.

[53] G. I. Hargreaves. Computing the condition number of tridiagonal and diagonal-plus-semiseparable matrices in linear time. Technical Report submitted, Department of Mathematics, University of Manchester, Manchester, England, 2004.

[54] N. J. Higham. Analysis of the cholesky decomposition of a semi-definite matrix. In M. G. Cox and S. Hammarling, editors, *Reliable Numerical Computation*, chapter 9, pages 161–186. Clarendon Press, Oxford, 1990.

[55] N. J. Higham. *Accuracy and Stability of Numerical Algorithms.* SIAM, Philadelphia, PA, 2nd edition, 2002.

[56] N. J. Higham and M. I. Smith. Computing the matrix cosine. *Numerical Algorithms*, 34:13–26, 2003.

[57] C. Holland, W. Reed, and D. et al Wolf. Report on High Performance Computing for the National Security Community. US Dept. of Defense, July 1, 2002.

[58] E.-J. Im, K. Yelick, and R. Vuduc. SPARSITY: An optimization framework for sparse matrix kernels. *Intern. J. High Perf. Comp. Appl.*, 18(1):135–158, 2004.

[59] Fifth International Workshop on Accurate Solution of Eigenvalue Problems. www.fernuni-hagen.de/MATHPHYS/iwasep5, 2004.

[60] C. Kenney and A. Laub. Rational iteration methods for the matrix sign function. *SIAM J. Mat. Anal. Appl.*, 21:487–494, 1991.

[61] C. Kenney and A. Laub. On scaling Newton's method for polar decomposition and the matrix sign function. *SIAM J. Mat. Anal. Appl.*, 13(3):688–706, 1992.

[62] D. et al Keyes. A Science-Based Case for Large-Scale Simulation. DOE Office of Science, www.pnl.gov/scales, July 30, 2004.

[63] X. S. Li, J. W. Demmel, D. H. Bailey, G. Henry, Y. Hida, J. Iskandar, W. Kahan, S. Y. Kang, A. Kapur, M. C. Martin, B. J. Thompson, T. Tung, and D. J. Yoo. Design, implementation and testing of extended and mixed precision BLAS. *ACM Trans. Math. Software*, 28(2):152–205, 2002.

[64] O. E. Livne and G. H. Golub. Scaling by binormalization. Computer Science Dept. Report SCCM-03-12, SCCM, Stanford, 2003.

[65] C. W. et al McCurdy. Creating Science-Driven Computer Architecture: A New Path to Scientific Leadership. LBNL Report PUB-5483, Oct 2002.

[66] K. Meerbergen. Locking and restarting quadratic eigenvalue solvers. Report RAL-TR-1999-011, CLRC, Rutherford Appleton Laboratory, Dept. of Comp. and Inf., Atlas Centre, Oxon OX11 0QX, GB, 1999.

[67] V. Mehrmann and D. Watkins. Structure-preserving methods for computing eigenpairs of large sparse skew-Hamiltoninan/Hamiltonian pencils. *SIAM J. Sci. Comput.*, 22:1905–1925, 2001.

[68] V. Mehrmann and D. Watkins. Polynomial eigenvalue problems with hamiltonian structure. *Elec. Trans. Num. Anal.*, 13:106–113, 2002.

[69] R. Nishtala, K. Chakrabarti, N. Patel, K. Sanghavi, J. Demmel, K. Yelick, and E. Brewer. Automatic tuning of collective communications in MPI. poster at SIAM Conf. on Parallel Proc., San Francisco, `www.cs.berkeley.edu/~rajeshn/poster_draft_6.ppt`, 2004.

[70] M. Parks. *A study of algorithms to compute the matrix exponential.* PhD thesis, University of California, Berkeley, California, 1994.

[71] B. N. Parlett and I. S. Dhillon. Fernando's solution to Wilkinson's problem: an application of double factorization. *Linear Algebra and Appl.*, 267:247–279, 1997.

[72] B. N. Parlett and I. S. Dhillon. Relatively robust representations of symmetric tridiagonals. *Linear Algebra and Appl.*, 309(1-3):121–151, 2000.

[73] B. N. Parlett and I. S. Dhillon. Multiple representations to compute orthogonal eigenvectors of symmetric tridiagonal matrices, 2004.

[74] B. N. Parlett and I. S. Dhillon. Orthogonal eigenvectors and relative gaps. *SIAM J. Matrix Anal. Appl.*, 25(3):858–899, 2004.

[75] B. N. Parlett and C. Vömel. Tight clusters of glued matrices and the shortcomings of computing orthogonal eigenvectors by multiple relatively robust representations. University of California, Berkeley, 2004. In preparation.

[76] R. Ralha. One-sided reduction to bidiagonal form. *Lin. Alg. Appl.*, 358:219–238, 2003.

[77] D. Rotman and P. Harding. DOE Greenbook - Needs and Directions in High-Performance Computing for the Office of Science. www.doe.gov/bridge, April 2002.

[78] I. Slapničar. Highly accurate symmetric eigenvalue decomposition and hyperbolic SVD. *Linear Algebra and Appl.*, 358:387–424, 2002.

[79] I. Slapničar and N. Truhar. Relative perturbation theory for hyperbolic singular value problem. *Linear Algebra and Appl.*, 358:367–386, 2002.

[80] M. I. Smith. A Schur algorithm for computing matrix pth roots. *SIAM J. Matrix Anal. Appl.*, 24(4):971–989, 2003.

[81] F. Tisseur. Backward error analysis of polynomial eigenvalue problems. *Lin. Alg. App.*, 309:339–361, 2000.

[82] F. Tisseur and K. Meerbergen. A survey of the quadratic eigenvalue problem. *SIAM Review*, 43:234–286, 2001.

[83] L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM, 1997.

[84] F. Triebsch. *Eigenwertalgorithmen für symmetrische λ-Matrizen*. PhD thesis, Fakultät für Mathematik, Technische Universität Chemnitz, 1995.

[85] S. S. Vadhiyar, G. E. Fagg, and J. Dongarra. Towards an accurate model for collective communications. *Intern. J. High Perf. Comp. Appl., special issue on Performance Tuning*, 18(1):159–167, 2004.

[86] R. Vandebril, M. Van Barel, and M. Mastronardi. An implicit QR algorithm for semiseparable matrices to compute the eigendecomposition of symmetric matrices. Report TW 367, Department of Computer Science, K.U.Leuven, Leuven, Belgium, 2003.

[87] R. Vuduc. *Automatic Performance Tuning of Sparse Matrix Kernels*. PhD thesis, University of California, Berkeley, CA, 2003.

[88] D. Watkins. *Fundamentals of Matrix Computations*. Wiley, 2nd edition, 2002.

[89] R. C. Whaley and J. Dongarra. The ATLAS WWW home page. `http://www.netlib.org/atlas/`.

[90] R. C. Whaley, A. Petitet, and J. Dongarra. Automated empirical optimization of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–25, 2001.

[91] J. H. Wilkinson. *Rounding Errors in Algebraic Processes*. Prentice Hall, Englewood Cliffs, 1963.