Prospectus for the Next LAPACK and ScaLAPACK Libraries

James Demmel[1], Jack Dongarra[23],
Beresford Parlett[1], William Kahan[1], Ming Gu[1], David Bindel[1],
Yozo Hida[1], Xiaoye Li[1], Osni Marques[1], E. Jason Riedy[1], Christof Voemel[1],
Julien Langou[2], Piotr Luszczek[2], Jakub Kurzak[2],
Alfredo Buttari[2], Julie Langou[2], Stanimire Tomov[2]

[1] University of California, Berkeley CA 94720, USA,
[2] University of Tennessee, Knoxville TN 37996, USA,
[3] Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA,

1 Introduction and Motivation

LAPACK and ScaLAPACK are widely used software libraries for numerical linear algebra. There have been over 68M web hits at www.netlib.org for the associated libraries LAPACK, ScaLAPACK, CLAPACK and LAPACK95. LAPACK and ScaLAPACK are used to solve leading edge science problems and they have been adopted by many vendors and software providers as the basis for their own libraries, including AMD, Apple (under Mac OS X), Cray, Fujitsu, HP, IBM, Intel, NEC, SGI, several Linux distributions (such as Debian), NAG, IMSL, the MathWorks (producers of MATLAB), Interactive Supercomputing, and PGI. Future improvements in these libraries will therefore have a large impact on users.

The ScaLAPACK and LAPACK development is mostly driven by algorithm research, the result of the user/vendor survey, the demands and opportunities of new architectures and programming languages, and the enthusiastic participation of the research community in developing and offering improved versions of existing Sca/LAPACK codes [51].

Brief outline of the paper: Section 2 discusses challenges in making current algorithms run efficiently, scalably, and reliably on future architectures. Section 3 discusses two kinds of improved algorithms: faster ones and more accurate ones. Since it is hard to improve both simultaneously, we choose to include a new faster algorithm if it is about as accurate as previous algorithms, and we include a new more accurate algorithm if it is at least about as fast as the previous algorithms. Section 4 describes new linear algebra functionality that will be included in new Sca/LAPACK releases. Section 5 describes our proposed software structure for Sca/LAPACK. Section 6 describes initial performance results.

2 Challenges of Future Architectures

Parallel computing is becoming ubiquitous at all scales of computation. It is no longer just exemplified by the TOP 500 list of the fastest computers in the

world. In a few years typical laptops are predicted to have 64 cores per multicore processor chip, and up to 256 hardware threads per chip. So unless all algorithms (not just numerical linear algebra!) can exploit this parallelism, they will cease to speed up, and in fact slow down compared to machine peak.

Furthermore, the gap between processor speed and memory speed continues to grow exponentially: processor speeds are improving at 59% per year, main memory bandwidth at only 23%, and main memory latency at a mere 5.5% [39]. This means that an algorithm that is efficient today, because it does enough floating point operations per memory reference to mask slow memory speed, may not be efficient in the near future. The same story holds for parallelism, with communication network bandwidth improving at just 26%, and network latency unimproved since the Cray T3E in 1996 until recently.

The largest scale target architectures of importance for LAPACK and ScaLAPACK include platforms now installed at NSF and DOE sites, as well as near term procurements. Longer term the High Productivity Computing Systems (HPCS) program [45] is supporting the construction of petascale computers by Cray (Cascade) and IBM (PERCS).

LAPACK and ScaLAPACK will have to run efficiently and correctly on a much wider array of platforms than in the past. In addition to the above architecturally diverse set of supercomputers and multicore chips in laptops, some future architectures are expected to be heterogeneous. For example, a cluster purchased over time will consist of some old, slow processors and some new, fast ones. Some processors may have higher loads from multiple users than others. Even single machines will be heterogenous, consisting of a CPU and other, faster, special purposes processors like GPUs, SSE units, etc. These will not just be heterogeneous in performance, but possibly in floating point semantics, with different units treating exceptions differently, or only computing in single precision. In a cluster, if one processor runs fastest handling denormalized numbers according to the IEEE 754 floating point standard [46], and another is fastest when flushing them to zero, then sending a number from one processor to another may change its value or even lead to a trap. Either way, correctness is a challenge, and not just for linear algebra.

It will be a challenge to map LAPACK's and ScaLAPACK's current software hierarchy of BLAS/BLACS/PBLAS/LAPACK/ScaLAPACK efficiently to all these platforms. For example, on a platform with multiple levels of parallelism (multicores, SMPs, distributed memory) would it be better to treat each SMP node as a ScaLAPACK process, calling parallelized BLAS or should each processor within the SMP be mapped to a process, or something else?

A more radical departure from current practice would be to make our algorithms asynchronous. Currently our algorithms are block synchronous, with phases of computation followed by communication with (implicit) barriers. But on networks that can overlap communication and computation, or on multithreaded shared memory machines, block synchrony can leave a significant fraction of the platform idle at any time. For example, the LINPACK benchmark

version of LU decomposition exploits such asynchrony and can run 2x faster than its block synchronous ScaLAPACK counterpart (see Section 5).

3 Better Algorithms

Three categories of routines are going to be addressed in Sca/LAPACK: (1) faster and/or more accurate algorithms for functions in LAPACK, which also need to be put in ScaLAPACK (discussed here) (2) functions now in LAPACK but not in ScaLAPACK (discussed in Section 4), and (3) functions in neither LAPACK nor ScaLAPACK (also discussed in Section 4). The following are lists of planned improvements.

Linear systems and least squares problems. Possible improvements include (1) iterative refinement using portable extra precision BLAS [52, 5, 14, 13] to get guaranteed accuracy in linear systems [24] and least squares problems; (2) iterative refinement where the LU factorization is computed in single precision even though all the data is in double precision, in order to exploit the fact that single can be from 2x faster than double (on an SSE unit) to 10x faster (on an IBM Cell) – refinement is used to try to make the answer as accurate as standard double precision LU factorization; (3) recursive data structures of Gustavson, Kågström et al [35] to improve memory locality, in particular for symmetric packed matrix factorizations, but keeping the usual columnwise matrix interface; (4) a more stable pivoting scheme for symmetric indefinite matrices proposed by Ashcraft, Grimes and Lewis [4], that keeps the L factor more bounded than the current Bunch-Kaufman factorization; (5) Cholesky factorization with diagonal pivoting [44] that avoids a breakdown if the matrix is nearly indefinite/rank-deficient, which is useful both for optimization problems and computing high accuracy symmetric positive definite eigenvalue decompositions (EVD); (6) improved condition estimators for tridiagonal [25, 43] or triangular [34] matrices; and (7) "latency avoiding" variations on the LU and QR decompositions that reduce the number of messages sent by factor equal to the block size in the 2D block-cyclic layout, which may be advantageous when the latency is large.

Eigenvalue problems. Possible improvements include (1) the 2003 SIAM Linear Algebra Prize winning work of Braman, Byers, and Mathias [16, 17] for solving the nonsymmetric eigenvalue problem up to 10x faster, as well as extensions to QZ in collaboration with Kågström and Kressner [50]; (2) a more complete implementation of the 2006 SIAM Linear Algebra Prize winning work of Dhillon and Parlett [57] on the Multiple Relatively Robust Representations (MRRR) algorithm for the symmetric eigenvalue problem, including work by Parlet and Vömel [58] to deal with tight clusters of eigenvalues and by Bientensi, Dhillon and can de Geihn on load balancing in the parallel version [9]; (3) extensions of the MRRR algorithm to the SVD [41], though some potential obstacles to guaranteed stability remain [70]; (4) a faster reduction to bidiagonal form for the SVD by Howell, Fulton, et al [37] that uses new BLAS [13] to achieve better memory locality; (5) a different faster bidiagonal reduction suitable for the case when only left or only right singular vectors are desired, but with possi-

ble less numerical stability [7, 59]; (6) when only a few eigen- or singular vectors are desired, the successive band reduction approach of Bischof and Lang [11] can move most flops from level 2 to level 3 BLAS; (7) an efficent algorithm by Drmač and Veselić for computing the SVD with high relative accuracy [33]; and (8) analogous high accuracy algorithms for the symmetric indefinite EVD by Slapničar [61] and by Dopico, Molera and Moro [32].

4 Added Functionality

Putting more of LAPACK into ScaLAPACK. Numerous matrix data types supported by LAPACK are not in ScaLAPACK. The most important omissions 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. 3 [35, 2]. Alternatively the packed storage may be partially expanded 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 [12]. (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; see further below.

The table below compares the available functions in LAPACK and ScaLAPACK. The relevant user interfaces ('drivers') are listed by subject and acronyms are used for the software in the respective libraries. The table also shows that in the ScaLAPACK library the implementation of some driver routines and their specialized computational routines are currently missing. The highest priority ones to include are marked "add". We also want expert drivers that compute error bounds.

Extending current functionality. We outline possible extensions of Sca/ LAPACK functionality, motivated by users and research progress: (1) efficient updating of factorizations like Cholesky, LDL^T, LU and QR, either using known unblocked techniques [38, 26] or more recent blocked ones; (2) an $O(n^2)$ eigenvalue routine for companion matrices, i.e. to find roots of polynomials, which would replace the `roots()` function in Matlab, based on recent work of Gu, Bini and others on semiseparable matrices [20, 66, 10]; (3) recent structure-preserving algorithms for matrix polynomial eigenvalue problems, especially quadratic eigenvalue problems [63]; (4) new algorithm for matrix functions like the square root, exponential and sign function [23]; (5) algorithms for various Sylvester-type matrix equations (recursive, RECSY; parallel, SCASY) [48, 49, 40]. (6) product and quotient eigenvalue algorithms now in SLICOT [8] are being considered, using the improved underlying EVD algorithms; and (7) out-of-core algorithms [12, 28, 27].

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

5 Software

Improving ease of use. "Ease of use" can be classified as follows: ease of programming (which includes easy conversion from serial to parallel, from LAPACK to ScaLAPACK and the possiblity to use high level interfaces), ease of obtaining predictable results in dynamic environments (for debugging and performance), and ease of installation (including performance tuning).

There are tradeoffs involved in each of these subgoals. In particular, ultimate ease of programming, exemplified by typing $x = A\backslash b$ in order to solve $Ax = b$ (paying no attention to the data type, data structure, memory management or algorithm choice) requires an infrastructure and user interface best left to the builders of systems like MATLAB and may come at a significant performance and reliability penalty. In particular, many users now exercise, and want to continue to exercise, detailed control over data types, data structures, memory management and algorithm choice, to attain both peak performance and reliability (e.g., not running out of memory unexpectedly). But some users also would like Sca/LAPACK to handle workspace allocation automatically, make it possible to call Sca/LAPACK on a greater variety of user-defined data structures, and pick the best algorithm when there is a choice.

To accomodate these "ease of programming" requests as well as requests to make the Sca/LAPACK code accessible from other languages than Fortran, the following steps are considered: (1) Produce new F95 modules for the LAPACK

drivers, for workspace allocation and algorithm selection. (2) Produce new F95 modules for the ScaLAPACK drivers, which convert, if necessary, the user input format (e.g., a simple block row layout across processors) to the optimal one for ScaLAPACK (which may be a 2D block cyclic layout with block sizes that depend on the matrix size, algorithm and architecture). Allocate memory as needed. (3) Produce LAPACK and ScaLAPACK wrappers in other languages. Based on current user surveys, these languages will tentatively be C, C++, Python and MATLAB. See below for software engineering details.

Ease of conversion from serial code (LAPACK) to parallel code (ScaLAPACK) is done by making the interfaces (at least at the driver level) as similar as possible. This includes expanding ScaLAPACK's functionality to include as much of LAPACK as possible (see Section 4).

Obtaining predictable results in a dynamic environment is important for debugging (to get the same answer when the code is rerun), for reproducibility, auditability (for scientific or legal purposes), and for performance (so that runtimes do not vary widely and unpredictably). Reproducibility in the face of asynchronous algorithms and heterogeneous systems will come with a performance penalty but is important for debugging and when auditability is critical.

To ease installation, we will use tools like autoconf and automatic performance tuning, supporting users from those who want to download and use one routine as quickly and simply as possible, to those who want an entire library, and to test and performance tune it carefully.

Improved software engineering. We describe our software engineering (SWE) approach. The main goals are to keep the substantial code base maintainable, testable and evolvable into the future as architectures and languages change. Maintaining compatibility with other software efforts and encouraging 3rd party contributions to the efforts of the Sca/LAPACK team are also goals [51].

These goals involve tradeoffs. One could explore starting "from scratch", using higher level ways to express the algorithms from which specific implementations could be generated. This approach yields high flexibility allowing the generation of code that is optimized for future computing platforms with different layers of parallelism, different memory hierarchies, different ratios of computation rate to bandwidth to latency, different programming languages and compilers, etc. Indeed, one can think of the problem as implementing the following *meta-program*:

```
(1) for all linear algebra problems
    (linear systems, eigenproblems, ...)
(2)   for all matrix types (general, symmetric, banded, ...)
(3)     for all data types (real/complex, different precisions)
(4)       for all machine architectures, communication topologies
(5)         for all programming interfaces
(6)           provide the best algorithm(s) available in terms of
              performance and accuracy (‘‘algorithms’’ is plural
              because sometimes no single one is always best)
```

This potential scope is quite large, requiring a judicious mixture of prioritization and automation. Indeed, there is prior work in automation [42], but so far this work has addressed only part of the range of algorithmic techniques Sca/LAPACK needs (e.g., not eigenproblems), it may not easily extend to more asynchronous algorithms and still needs to be coupled to automatic performance tuning techniques. Still, some phases of the meta-program are at least partly automatable now, namely steps (3) through (5) (see below).

Note that line (5) of the meta-program is "programming interfaces" not "programming languages," because the question of the best implementation language is separate from providing ways to call it (from multiple languages). Currently Sca/LAPACK is written in F77. Over the years, the Sca/LAPACK team and others have built on this to provide interfaces or versions in other languages: LAPACK95 [6] and LAPACK3E [3] for F95 (LAPACK3E providing a straightforward wrapper, and LAPACK95 using F95 arrays to simplify the interfaces at some memory and performance costs), CLAPACK in C [21] (translated, mostly automatically, using f2c [36]), LAPACK++ [29], TNT [64] in C++, and JLAPACK in Java [47] (translated using f2j).

First we summarize the SWE development plan and then the SWE research plan. The *development plan* includes (1) maintaining the core in Fortan, adopting those features of F95 that most improve ease-of-use and ease-of-development (recursion, modules, environmental enquiries) but do not prevent the most demanding users from attaining the highest performance and reliable control over the run-time environment (so not automatic memory allocation). Keeping Fortran is justified for cost and continuity reasons, as well as the fact that the most effective optimizing compilers still work best on Fortran, even when they share "back ends" with the C compiler, because of the added difficulty of discerning the absence of aliasing in C [22]; (2) F95 wrappers for the drivers to improve ease of use, via automatic workspace allocation and automatic algorithm selection; (3) F95 wrappers for the drivers that use performance models to determine the best layout for the user's matrix (which may be 2D block-cyclic and/or recursive instead of the 1D blocked layouts most natural to users) and which convert to that layout and back invisibly to the user; (4) wrappers for the drivers in other languages, like C, Python and Matlab; (5) using the new BLAS standard [15, 13, 5], for new Sca/LAPACK routines, which also provides new high precision functionality needed for iterative refinement [24], and systematically ensuring thread-safety; (6) using tools like autoconf, bugzilla, svn, automatic overnight build and test, etc. to streamline installation and development and encourage third party contributions, and using modules to provide extra precise versions based on one source; and (7) doing systematic performance tuning, not just the BLAS [69, 68] and the BLACS [54, 65], but the 1300 calls to the ILAENV routine in LAPACK that provides various blocking parameters, and exploiting modeling techniques that let the user choose how much tuning effort to expend [67, 56].

The following are *SWE research tasks*: (1) exploring the best mappings of the Sca/LAPACK software layers (BLAS, BLACS, PBLAS, LAPACK, ScaLAPACK) to the hardware layers of emerging architectures, including deciding that

different layers entirely are needed; (2) exploring the use of new parallel programming languages being funded by NSF, DOE and the DARPA HPCS program, namely UPC [19], Titanium [71], CAF [55], Fortress [1], X10 [60] and Cascade [18]; (3) exploring further automation of the production of library software (existing work such as [42] still needs to address two-sided factorizations, iterative algorithms for the EVD and SVD, more asynchronous algorithms, novel data layouts, how multiple levels of parallelism map to multiple levels of hardware, and so on); (4) using statistical models to accelerate performance tuning at installation time [67]; and (5) choosing the right subset of a cluster to run on at run-time, depending on the dynamically changing load.

Multicore and multithreading. Message passing as used in ScaLAPACK introduces memory overhead unnecessary on multicore platforms, degrading performance and making it harder to schedule potentially parallel operations. Limiting shared memory parallelism to fork-join parallelism (e.g., OpenMP) or the BLAS is inadequate. Shared memory would let us replace data partitioning by work partitioning, although cache locality requirements mean we still need either two dimensional block cyclic (or perhaps recursive) layouts.

Shared memory systems have used client/server, work-crew, and pipelining for parallelism, but because of data dependencies pipelining is most appropriate, as well as being a good match for streaming hardware like the IBM Cell. For efficiency we must avoid pipeline stalls when data dependencies block execution.

We illustrate this for LU factorization. LU has left-looking and right-looking formulations [30]. Transition between the two can be done by automatic code transformations [53], although more than simple dependency analysis is needed. Lookahead can improve performance by performing panel factorizations in parallel with updates to the trailing matrix from the previous algorithm steps [62]. The lookahead can be of arbitrary depth (as exploited by the LINPACK benchmark [31]).

In fact lookahead provides a spectrum of implementations from right-looking (no lookahead) to left-looking (maximum lookahead). Less lookahead avoids pipeline stalls at the beginning of the factorization, but may introduce them at the end; more lookahead provides more work at the end of the factorization but may stall at the beginning.

Recent experiments show that pipeline stalls can be greatly reduced if unlimited lookahead is allowed and the lookahead panel factorizations are dynamically scheduled, which is much easier in shared memory than distributed memory, in part because there is no storage overhead.

6 Performance

We give a few recent performance results for ScaLAPACK driver routines on recent architectures. We discuss strong scalability, i.e. we keep the problem size constant while increasing the number of processors.

Figure 1(left) gives the time to solve $Ax = b$ for $n = 8,000$ on a cluster of dual processor 64 bit AMD Opterons with a Gigabit ethernet. As the number of

processors increases from 1 to 64, the time decreases from 110 sec (3.1 GFlops) to 9 sec (37.0 GFlops) (thanks to Emmanuel Jeannot for sharing the result.)

Figure 1(right) gives the time for the symmetric eigendecomposition with $n = 12,000$ on a 64 processor cluster of dual processor 64 bit Intel Xeon EMTs with a Myrinet MX interconnect. The matrices are generated randomly using the same generator as in the Linpack Benchmark, so there are no tight eigenvalue clusters.

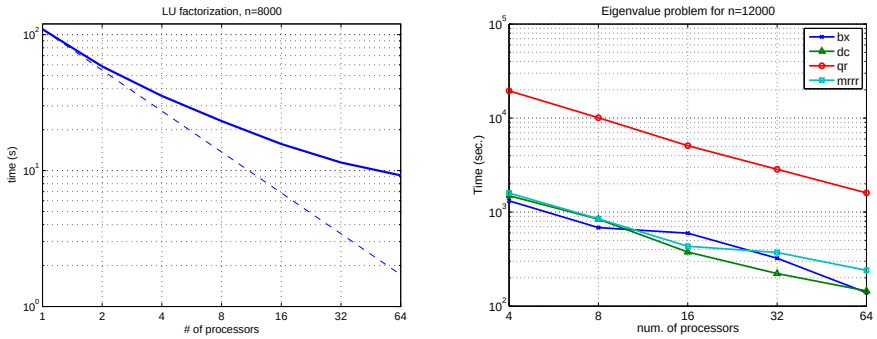

Fig. 1. Left: Scalability of ScaLAPACK's LU (`pdgetrf`) for $n = 8,000$. Right: Scalability of the ScaLAPACK's symmetric eigensolvers with $n = 12,000$. Four eigensolvers are shown: BX (`pdsyevx`), QR (`pdsyev`), DC (`pdsyevd`) and MRRR (`pdsyevr`).

References

1. Allen Steele et al . The Fortress language specification, version 0.707. research.sun.com/projects/plrg/fortress0707.pdf.
2. B. S. Andersen, J. Wazniewski, and Gustavson. F. G. A recursive formulation of Cholesky factorization of a matrix in packed storage. *ACM Trans. Math. Soft.*, 27(2):214–244, 2001.
3. E. Anderson. LAPACK3E. www.netlib.org/lapack3e, 2003.
4. 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.
5. 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.
6. V. Barker, S. Blackford, J. Dongarra, J. Du Croz, S. Hammarling, M. Marinova, J. Wasniewski, and P. Yalamov. *LAPACK95 Users' Guide*. SIAM, 2001. www.netlib.org/lapack95.
7. J. Barlow, N. Bosner, and Z. Drmač. A new stable bidiagonal reduction algorithm. `www.cse.psu.edu/~barlow/fastbidiag3.ps`, 2004.
8. 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.

9. 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.
10. D. Bini, Y. Eidelman, L. Gemignani, and I. Gohberg. Fast QR algorithms for Hessenberg matrices which are rank-1 perturbations of unitary matrices. Dept. of Mathematics report 1587, University of Pisa, Italy, 2005. `www.dm.unipi.it/~gemignani/papers/begg.ps`.
11. C. H. Bischof, B. Lang, and X. Sun. A framework for symmetric band reduction. *ACM Trans. Math. Soft.*, 26(4):581–601, 2000.
12. 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.
13. 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.
14. 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.
15. S. Blackford, G. Corliss, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, C. Hu, W. Kahan, L. Kaufman, B. Kearfott, F. Krogh, X. Li, Z. Maany, A. Petitet, R. Pozo, K. Remington, W. Walster, C. Whaley, J. Wolff v. Gudenberg, and A. Lumsdaine. Basic Linear Algebra Subprograms Technical (BLAST) Forum Standard. *Intern. J. High Performance Comput.*, 15(3–4), 2001. 305 pages, also available at `www.netlib.org/blas/blast-forum/`.
16. 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.
17. 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.
18. D. Callahan, B. Chamberlain, and H. Zima. The Cascade high-productivity language. In *9th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS 2004)*, pages 52–60. IEEE Computer Society, April 2004. www.gwu.edu/ upc/publications/productivity.pdf.
19. F. Cantonnet, Y. Yao, M. Zahran, and T. El-Ghazawi. Productivity analysis of the UPC language. In *IPDPS 2004 PMEO workshop*, 2004. www.gwu.edu/ upc/publications/productivity.pdf.
20. 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.
21. CLAPACK: LAPACK in C. http://www.netlib.org/clapack/.
22. C. Coarfa, Y. Dotsenko, J. Mellor-Crummey, D. Chavarria-Miranda, F. Contonnet, T. El-Ghazawi, A. Mohanti, and Y. Yao. An evaluation of global address space languages: Co-Array Fortran and Unified Parallel C. In *Proc. 10th ACM SIGPLAN Symp. on Principles and Practice and Parallel Programming (PPoPP 2005)*, 2005. www.hipersoft.rice.edu/caf/publications/index.html.
23. P. Davies and N. J. Higham. A Schur-Parlett algorithm for computing matrix functions. *SIAM J. Matrix Anal. Appl.*, 25(2):464–485, 2003.

24. J. Demmel, Y. Hida, W. Kahan, X. S. Li, S. Mukherjee, and E. J. Riedy. Error bounds from extra precise iterative refinement. *ACM TOMS*, 32(2):325–351, 2006.
25. 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.
26. J. Dongarra, J. Bunch, C. Moler, and G. W. Stewart. *LINPACK User's Guide*. SIAM, Philadelphia, PA, 1979.
27. J. Dongarra and E. D'Azevedo. The design and implementation 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.
28. 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.
29. J. Dongarra, R. Pozo, and D. Walker. Lapack++: A design overview of ovject-oriented extensions for high performance linear algebra. In *Supercomputing 93*. IEEE, 1993. math.nist.gov/lapack++.
30. J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst. *Numerical Linear Algebra for High-Performance Computers*. SIAM, Philadelphia, PA, 1998.
31. J. J. Dongarra, P. Luszczek, and A. Petitet. The LINPACK Benchmark: past, present and future. *Concurrency Computat.: Pract. Exper.*, 15:803–820, 2003.
32. F. M. Dopico, J. M. Molera, and J. Moro. An orthogonal high relative accuracy algorithm for the symmetric eigenproblem. *SIAM. J. Matrix Anal. Appl.*, 25(2):301–351, 2003.
33. Z. Drmač and K. Veselić. New fast and accurate Jacobi SVD algorithm. Technical report, Dept. of Mathematics, University of Zagreb, 2004.
34. I. S. Duff and C. Vömel. Incremental Norm Estimation for Dense and Sparse Matrices. *BIT*, 42(2):300–322, 2002.
35. E. Elmroth, F. Gustavson, I. Jonsson, and B. Kågström. Recursive blocked algorithms and hybrid data structures for dense matrix library software. *SIAM Review*, 46(1):3–45, 2004.
36. f2c: Fortran-to-C translator. http://www.netlib.org/f2c.
37. C. Fulton, G. Howell, J. Demmel, and S. Hammarling. Cache-efficient bidiagonalization using BLAS 2.5 operators. 28 pages, in progress, 2004.
38. G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, 3rd edition, 1996.
39. S. Graham, M. Snir, and eds. C. Patterson. *Getting up to Speed: The Future of Supercomputing*. National Research Council, 2005.
40. R. Granat, I. Jonsson, and B. Kågström. Combining Explicit and Recursive Blocking for Solving Triangular Sylvester-Type Matrix Equations in Distrubuted Memory Platforms. In M. Vanneschi M. Danelutto, D. Laforenza, editor, *Euro-Par 2004*, volume 3149, pages 742–750. Lecture Notes in Computer Science, Springer, 2004.
41. B. Grosser. *Ein paralleler und hochgenauer $O(n^2)$ Algorithmus für die bidiagonale Singulärwertzerlegung*. PhD thesis, University of Wuppertal, Wuppertal, Germany, 2001.
42. J. A. Gunnels, F. G. Gustavson, G. M. Henry, and R. A. van de Geijn. FLAME: Formal Linear Algebra Methods Environment. *ACM Trans. Math. Soft.*, 27(4):422–455, 2001.

43. 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.
44. 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.
45. High productivity computing systems (hpcs). www.highproductivity.org.
46. IEEE Standard for Binary Floating Point Arithmetic Revision. `grouper.ieee.org/groups/754`, 2002.
47. JLAPACK: LAPACK in Java. http://icl.cs.utk.edu/f2j.
48. I. Jonsson and B. Kågström. Recursive blocked algorithms for solving triangular systems. I. one-sided and coupled Sylvester-type matrix equations. *ACM Trans. Math. Software*, 28(4):392–415, 2002.
49. I. Jonsson and B. Kågström. Recursive blocked algorithms for solving triangular systems. II. Two-sided and generalized Sylvester and Lyapunov matrix equations. *ACM Trans. Math. Software*, 28(4):416–435, 2002.
50. B. Kågström and D. Kressner. Multishift Variants of the QZ Algorithm with Aggressive Early Deflation. *SIAM J. Matrix Anal. Appl.*, 29(1):199–227, 2006.
51. LAPACK Contributor Webpage. http://www.netlib.org/lapack-dev/contributions.html.
52. 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. Soft.*, 28(2):152–205, 2002.
53. V. Menon and K. Pingali. Look left, look right, look left again: An application of fractal symbolic analysis to linear algebra code restructuring. *Int. J. Parallel Comput.*, 32(6):501–523, 2004.
54. 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.
55. R. Numrich and J. Reid. Co-array Fortran for parallel programming. Fortran Forum, 17, 1998.
56. OSKI: Optimized Sparse Kernel Interface. http://bebop.cs.berkeley.edu/oski/.
57. B. N. Parlett and I. S. Dhillon. Orthogonal eigenvectors and relative gaps. *SIAM J. Matrix Anal. Appl.*, 25(3):858–899, 2004.
58. 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.
59. R. Ralha. One-sided reduction to bidiagonal form. *Lin. Alg. Appl.*, 358:219–238, 2003.
60. V. Saraswat. Report on the experimental language X10, v0.41. IBM Research technical report, 2005.
61. I. Slapničar. Highly accurate symmetric eigenvalue decomposition and hyperbolic SVD. *Lin. Alg. Appl.*, 358:387–424, 2002.
62. P. E. Strazdins. A comparison of lookahead and algorithmic blocking techniques for parallel matrix factorization. *Int. J. Parallel Distrib. Systems Networks*, 4(1):26–35, 2001.
63. F. Tisseur and K. Meerbergen. A survey of the quadratic eigenvalue problem. *SIAM Review*, 43:234–286, 2001.

64. TNT: Template Numerical Toolkit. http://math.nist.gov/tnt.
65. 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.
66. 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.
67. R. Vuduc, J. Demmel, and J. Bilmes. Statistical models for automatic performance tuning. In *Intern. Conf. Comput. Science*, May 2001.
68. R. C. Whaley and J. Dongarra. The ATLAS WWW home page. `http://www.netlib.org/atlas/`.
69. 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.
70. P. Willems. personal communication, 2006.
71. K. Yelick, L. Semenzato, G. Pike, C. Miyamoto, B. Liblit, A. Krishnamurthy, P. Hilfinger, S. Graham, D. Gay, P. Colella, and A. Aiken. Titanium: A high-performnace Java dialect. *Concurrency: Practice and Experience*, 10:825–836, 1998.