Packages of Subroutines for Linear Algebra

74
Blas

Jack Dongarra
University of Tennessee

Victor Eijkhout
University of Tennessee

Julien Langou
University of Tennessee

74.1 Introduction

A significant amount of execution time in complicated linear algebraic programs is known to be spent in a few low-level operations. Consequently, reducing the overall execution time of an application program leads to the problem of optimizing these low-level operations. Such low-level optimization are highly machine-dependent and are a matter for specialists. A separation has, therefore, been made by the computational linear algebra community: On the one hand highly efficient, machine-dependent building blocks of linear algebra, the BLAS, (basic linear algebra subprograms) are provided on a given computational platform. On the other hand, linear algebra programs attempt to make the most use of those computational blocks in order to have as good performance as possible across a wide range of platforms.

The first major concerted effort to achieve agreement on the specification of a set of linear algebra kernels resulted in the Level-1 BLAS [LHK79]. The Level-1 BLAS specification and implementation are the result of a collaborative project in 1973 through 1977. The Level-1 BLAS were extensively and successfully exploited by LINPACK [DBM79], a software package for the solution of dense and banded linear equations and linear least squares problems.

With the advent of vector machines, hierarchical memory machines and shared memory parallel machines, specifications for the Level-2 and 3 BLAS [DDD90a], [DDD90b], [DDH88a], [DDH88b], concerned with matrix–vector and matrix–matrix operations respectively, were drawn up in 1984 through 1986 and 1987 to 1988. These specifications made it possible to construct new software to utilize the memory hierarchy of modern computers more effectively. In particular, the Level-3 BLAS allowed the construction of software based upon block-partitioned algorithms, typified by the linear algebra software package LAPACK (see Chapter 75).

In this chapter, we present information about the BLAS. We begin with basic definition, then describe the calling sequences, report a number of facts, and finally conclude with a few examples.

Au: Add #1 head (74.2) here (Astrisk)

*

Definitions:

Two-dimensional column-major format. A two-dimensional array is said to be in column-major format when its entries are stored by column. This is the way the Fortran language stores two-dimensional arrays. For example, the declaration `double precision A(2,2)` stores the entries of the 2-by-2

two-dimensional array in the one-dimensional order: `A(1,1),A(2,1),A(1,2),A(2,2)`.
Row-major format. storage of the entries by row. This is the way C stores two-dimensional arrays.
Packed format. The packed format is relevant for symmetric, Hermitian, or triangular matrices. Half of the matrix (either the upper part or the lower part depending on an `UPLO` parameter) is stored in a one-dimensional array. For example, if `UPLO='L'`, the 3-by-3 symmetric matrix `A` is stored as `A(1,1), A(2,1),A(3,1),A(2,2),A(3,2),A(3,3))`.
Leading dimension. In column-major format, the leading dimension is the first dimension of a two-dimensional array (as opposed to the dimension of the matrix stored in that array). The leading dimension of an array is necessary to access its elements. For example, if `LDA` is the leading dimension of `A`, and `A` is stored in the two-dimensional column-major format, $A_{i,j}$ is stored in position $i + (j - 1) * LDA$. The leading dimension of an m-by-n matrix is often m. It enables convenient abstraction to access submatrices (see examples). An m-by-n matrix should have `LDA` $\geq m$.
Increment. The increment of a vector is the number of storage elements from one element to the next. If the increment of a vector is 1, this means that the vector is stored contiguously in memory, an increment of 2 corresponds to using every other array element. The increment is useful to manipulate rows of a matrix stored in column-major format (see Example 3).
BLAS vector description. A vector description in BLAS is defined by three quantities — a vector length, an array or a starting element within an array, and the increment. This gives `(n,X,1)` for a vector of size `n` starting at index `X` with increment of `1` .
BLAS description of a general matrix. A general matrix is described in BLAS by four quantities: A `TRANS` label: (`'N'`, `'T'` or `'H'`), its dimensions `m` and `n`, an array or a starting element within an array, and a leading dimension. This gives `(TRANS,M,N,A,LDA)` if one wants to operate on the `TRANS` of an `M-` by `-N` matrix starting in `A(1,1)` and stored in an array of leading dimension `LDA`.
Prefixes The first letter of the name of a BLAS routine indicates the Fortran type on which it operates on

S - REAL	C - COMPLEX
D - DOUBLE PRECISION	Z - COMPLEX*16

Prefixes for Level-2 and Level-3 BLAS
Matrix types:

GE - GEneral	GB - General Band	
SY - SYmmetric	SB - Sym. Band	SP - Sym. Packed
HE - HErmitian	HB - Herm. Band	HP - Herm. Packed
TR - TRiangular	TB - Triang. Band	TP - Triang. Packed

Level-2 and Level-3 BLAS Options

TRANS	= '**N**o transpose', '**T**ranspose', '**C**onjugate transpose'
UPLO	= '**U**pper triangular', '**L**ower triangular'
DIAG	= '**N**on-unit triangular', '**U**nit triangular'
SIDE	= '**L**eft', '**R**ight' on the right)

Calls:

1. Below is the calling sequence for the `GEMV`matrix-vector multiply subroutine. The `GEMV` routine performs the mathematical operation

$$y \longleftarrow \alpha Ax + \beta y, \quad \text{or} \quad y \longleftarrow \alpha A^T x + \beta y.$$

For double precision, the calling sequence looks like

```
      SUBROUTINE DGEMV ( TRANS, M, N, ALPHA, A, LDA, X, INCX,
     $                   BETA, Y, INCY ).
```

The types of the variables are as follows:

```
      DOUBLE PRECISION   ALPHA, BETA
      INTEGER            INCX, INCY, LDA, M, N
      CHARACTER*1        TRANS
      DOUBLE PRECISION   A( LDA, * ), X( * ), Y( * )
```

The meaning of the variables is as follows:

`TRANS` : Specifies the operation to be performed as
`TRANS='N'` $y \leftarrow \alpha Ax + \beta y$,
`TRANS='T'` $y \leftarrow \alpha A^T x + \beta y$.

`M` : Specifies the number of rows of the matrix A.

`N` : Specifies the number of columns of the matrix A.

`ALPHA`: Specifies the scalar α.

`A` : Points to the first entry of the two-dimensional array that stores the elements of A in column major format.

`LDA` : Specifies the leading dimension of A.

`X` : Points to the incremented array that contains the vector x.

`INCX` : Specifies the increment for the elements of `X`.

`BETA` : Specifies the scalar β.

`Y` : Points to the incremented array that contains the vector y.

`INCY` : Specifies the increment for the elements of `Y`.

All variables are left unchanged after completion of the routines, except `Y`.

2. The following table is the quick reference to the BLAS.

Facts:

1. The BLAS represent fairly simple linear algebra kernels that are easily coded in a few lines. One could also download reference source codes and compile them (`http://www.netlib.org/blas`). However, this strategy is unlikely to give good performance, no matter the level of sophistication of the compiler. The recommended way to obtain an efficient BLAS is to use vendor BLAS libraries, or to install the ATLAS [WPD01] package. Figure 74.1 illustrates the fact that ATLAS BLAS clearly outperform the reference implementation for the matrix–matrix multiply by a factor of five on a modern architecture.
2. To a great extent, the user community has embraced the BLAS, not only for performance reasons, but also because developing software around a core of common routines like the BLAS is good software engineering practice. Highly efficient, machine-specific implementations of the BLAS are available for most modern high-performance computers. To obtain an up-to-date list of available optimized BLAS, see the BLAS FAQ at `http://www.netlib.org/blas`.
3. Level-1 BLAS operates on $\mathcal{O}(n)$ data and performs $\mathcal{O}(n)$ operations. Level-2 BLAS operates on $\mathcal{O}(n^2)$ data and performs $\mathcal{O}(n^2)$ operations. Level-3 BLAS operates on $\mathcal{O}(n^2)$ data and performs $\mathcal{O}(n^3)$ operations. Therefore, the ratio operation/data is $\mathcal{O}(1)$ for Level-1 BLAS and Level-2 BLAS, but $\mathcal{O}(n)$ for Level-3 BLAS. On modern architecture, where memory access is particularly slow compared to computation time, Level-3 BLAS exploit this $\mathcal{O}(n)$ ratio to mask the memory access bottleneck (latency and bandwidth). Figure 74.1 illustrates the fact that Level-3 BLAS on a modern

Level-1 BLAS

`dim scalar vector vector scalars 5-element array`		prefixes
`SUBROUTINE xROTG ( A, B, C, S )`	Generate plane rotation	S, D
`SUBROUTINE xROTMG( D1, D2, A, B, PARAM )`	Generate modified plane rotation	S, D
`SUBROUTINE xROT ( N, X, INCX, Y, INCY, C, S )`	Apply plane rotation	S, D
`SUBROUTINE xROTM ( N, X, INCX, Y, INCY, PARAM )`	Apply modified plane rotation	S, D
`SUBROUTINE xSWAP ( N, X, INCX, Y, INCY )`	$x \leftrightarrow y$	S, D, C, Z
`SUBROUTINE xSCAL ( N, ALPHA, X, INCX )`	$x \leftarrow \alpha x$	S, D, C, Z, CS, ZD
`SUBROUTINE xCOPY ( N, X, INCX, Y, INCY )`	$y \leftarrow x$	S, D, C, Z
`SUBROUTINE xAXPY ( N, ALPHA, X, INCX, Y, INCY )`	$y \leftarrow \alpha x + y$	S, D, C, Z
`FUNCTION xDOT ( N, X, INCX, Y, INCY )`	$dot \leftarrow x^T y$	S, D, DS
`FUNCTION xDOTU ( N, X, INCX, Y, INCY )`	$dot \leftarrow x^T y$	C, Z
`FUNCTION xDOTC ( N, X, INCX, Y, INCY )`	$dot \leftarrow x^H y$	C, Z
`FUNCTION xxDOT ( N, X, INCX, Y, INCY )`	$dot \leftarrow \alpha + x^T y$	SDS
`FUNCTION xNRM2 ( N, X, INCX )`	$nrm2 \leftarrow \|\|x\|\|_2$	S, D, SC, DZ
`FUNCTION xASUM ( N, X, INCX )`	$asum \leftarrow \|\|re(x)\|\|_1 + \|\|im(x)\|\|_1$	S, D, SC, DZ
`FUNCTION IxAMAX( N, X, INCX )`	$amax \leftarrow 1^{st} k \|re(x_k)\| + \|im(x_k)\| = max(\|re(x_i)\| + \|im(x_i)\|)$	S, D, C, Z

Level-2 BLAS

`options dim b-width scalar matrix vector scalar vector`		
`xGEMV ( TRANS, M, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )`	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$	S, D, C, Z
`xGBMV ( TRANS, M, N, KL, KU, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )`	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$	S, D, C, Z
`xHEMV ( UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )`	$y \leftarrow \alpha Ax + \beta y$	C, Z
`xHBMV ( UPLO, N, K, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )`	$y \leftarrow \alpha Ax + \beta y$	C, Z
`xHPMV ( UPLO, N, ALPHA, AP, X, INCX, BETA, Y, INCY )`	$y \leftarrow \alpha Ax + \beta y$	C, Z
`xSYMV ( UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )`	$y \leftarrow \alpha Ax + \beta y$	S, D
`xSBMV ( UPLO, N, K, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )`	$y \leftarrow \alpha Ax + \beta y$	S, D
`xSPMV ( UPLO, N, ALPHA, AP, X, INCX, BETA, Y, INCY )`	$y \leftarrow \alpha Ax + \beta y$	S, D
`xTRMV ( UPLO, TRANS, DIAG, N, A, LDA, X, INCX )`	$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
`xTBMV ( UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX )`	$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
`xTPMV ( UPLO, TRANS, DIAG, N, AP, X, INCX )`	$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
`xTRSV ( UPLO, TRANS, DIAG, N, A, LDA, X, INCX )`	$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$	S, D, C, Z
`xTBSV ( UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX )`	$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$	S, D, C, Z
`xTPSV ( UPLO, TRANS, DIAG, N, AP, X, INCX )`	$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$	S, D, C, Z
`options dim scalar vector vector matrix`		

xGER	(M, N, ALPHA, X, INCX, Y, INCY, A, LDA)	$A \leftarrow \alpha x y^T + A, A - m \times n$	S, D
xGERU	(M, N, ALPHA, X, INCX, Y, INCY, A, LDA)	$A \leftarrow \alpha x y^T + A, A - m \times n$	C, Z
xGERC	(M, N, ALPHA, X, INCX, Y, INCY, A, LDA)	$A \leftarrow \alpha x y^H + A, A - m \times n$	C, Z
xHER	(UPLO, N, ALPHA, X, INCX, A, LDA)	$A \leftarrow \alpha x x^H + A$	C, Z
xHPR	(UPLO, N, ALPHA, X, INCX, AP)	$A \leftarrow \alpha x x^H + A$	C, Z
xHER2	(UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA)	$A \leftarrow \alpha x y^H + y(\alpha x)^H + A$	C, Z
xHPR2	(UPLO, N, ALPHA, X, INCX, Y, INCY, AP)	$A \leftarrow \alpha x y^H + y(\alpha x)^H + A$	C, Z
xSYR	(UPLO, N, ALPHA, X, INCX, A, LDA)	$A \leftarrow \alpha x x^T + A$	S, D
xSPR	(UPLO, N, ALPHA, X, INCX, AP)	$A \leftarrow \alpha x x^T + A$	S, D
xSYR2	(UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA)	$A \leftarrow \alpha x y^T + \alpha y x^T + A$	S, D
xSPR2	(UPLO, N, ALPHA, X, INCX, Y, INCY, AP)	$A \leftarrow \alpha x y^T + \alpha y x^T + A$	S, D

Level-3 BLAS

	options; dim; scalar matrix matrix scalar matrix		
xGEMM	(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)	$C \leftarrow \alpha op(A) op(B) + \beta C,$ $op(X) = X, X^T, X^H, C - m \times n$	S, D, C, Z
xSYMM	(SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC)	$C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha BA + \beta C,$ $C - m \times n, A = A^T$	S, D, C, Z
xHEMM	(SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC)	$C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha BA + \beta C,$ $C - m \times n, A = A^H$	C, Z
xSYRK	(UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC)	$C \leftarrow \alpha AA^T + \beta C, C \leftarrow \alpha A^T A + \beta C,$ $C - n \times n$	S, D, C, Z
xHERK	(UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC)	$C \leftarrow \alpha AA^H + \beta C, C \leftarrow \alpha A^H A + \beta C,$ $C - n \times n$	C, Z
xSYR2K	(UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)	$C \leftarrow \alpha AB^T + \bar{\alpha} BA^T + \beta C,$ $C \leftarrow \alpha A^T B + \bar{\alpha} B^T A + \beta C,$ $C - n \times n$	S, D, C, Z
xHER2K	(UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)	$C \leftarrow \alpha AB^H + \bar{\alpha} BA^H + \beta C,$ $C \leftarrow \alpha A^H B + \bar{\alpha} B^H A + \beta C,$ $C - n \times n$	C, Z
xTRMM	(SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB)	$B \leftarrow \alpha op(A) B, B \leftarrow \alpha B op(A),$ $op(A) = A, A^T, A^H, B - m \times n$	S, D, C, Z
xTRSM	(SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB)	$B \leftarrow \alpha op(A^{-1}) B, B \leftarrow \alpha B op(A^{-1}),$ $op(A) = A, A^T, A^H, B - m \times n$	S, D, C, Z

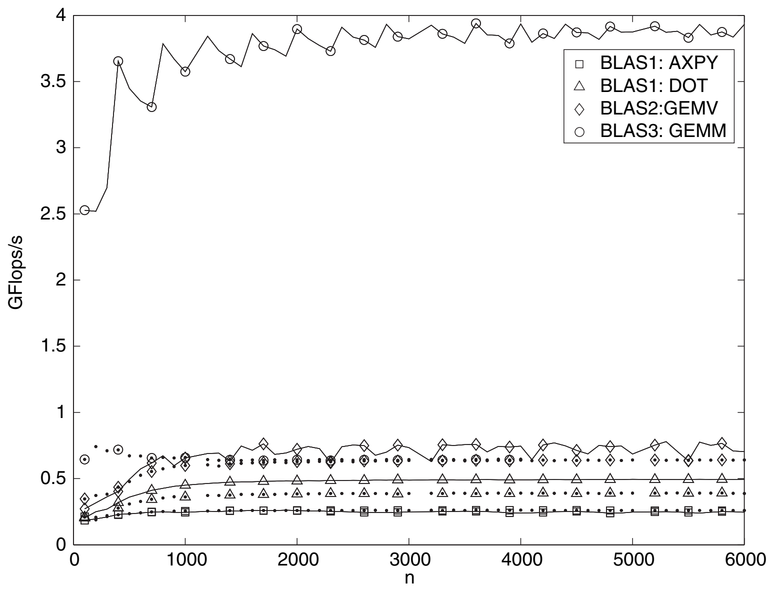

FIGURE 74.1 Performance in GFlops of four BLAS routines for two different BLAS libraries on an Intel Xeon CPU running at 3.20GHz. The first BLAS library represents an optimized BLAS library (here ATLAS v3.7.8), its three performance curves are given with a solid line on the graph. The second library represents straightforward implementation (here reference BLAS from netlib), its three performance curves are given with a dotted line on the graph. This graph illustrates two facts: An optimized Level-3 BLAS is roughly five times faster than Level-1 or Level-2 BLAS and it is also roughly five times faster than a reference Level-3 BLAS implementation. To give an idea of the actual time, multiplying two 6000-by-6000 matrices on this machine will take about 2 minutes using the ATLAS BLAS while it will take about 10 minutes using the reference BLAS implementation.

architecture performs 5 times more floating-point operations per second than a Level-2 or Level-1 BLAS routine. Most of the linear algebra libraries try to make as much as possible use of Level-3 BLAS.

4. Most of shared memory computers have a multithreaded BLAS library. By programming a sequential code and linking with the multithreaded BLAS library, the application will use all the computing units of the shared memory system without any explicit parallelism in the user application code.
5. Although here we present only calling sequences from Fortran, it is possible to call the BLAS from C. The major problem for C users is the Fortran interface used by the BLAS. The data layout (the BLAS interface assumes column-major format) and the passage of parameters by reference (as opposed to values) has to be done carefully. See Example 2 for an illustration. Nowadays most BLAS distributions provide a C-interface to the BLAS. This solves these two issues and, thus, we highly recommend its use.
6. The Level-1, Level-2, and Level-3 BLAS are now extended by a new standard with more functionality and better software standard see [BDD02], [Don02]. For example, the C-interface to the BLAS (see Fact 5) is included in this new BLAS.

Examples:

In all of these examples, A is an m-by-n matrix and it is stored in an `M`-by-`N` array starting at position `A`, B is an n-by-n matrix and it is stored in an `N`-by-`N` array starting at position `B`, C is an m-by-m matrix and it is stored in an `M`-by-`M` array starting at position `C`, X is a vector of size n and it is stored in an `N` array starting at position `X`, Y is a vector of size m and it is stored in an `M` array starting at position `Y`. All the two-dimensional arrays are in column-major format, all the one-dimensional arrays have increment one. We assume that m and n are both greater than 11 in Example 3.

1. To perform the operation $y \longleftarrow Ax$ the BLAS calling sequence is

   ```
   CALL DGEMV ( 'N', M, N, 1.0D0, A, M, X, 1, 0.0D0, Y, 1 ).
   ```

2. To perform the operation $x \longleftarrow \alpha A^T y + \beta x$ the BLAS calling sequence is

   ```
   CALL DGEMV ( 'T', M, N, ALPHA, A, M, Y, 1, BETA, X, 1 ).
   ```

   From C, this would give

   ```
   IONE = 1;
   dgemv ( "T", &M, &N, &ALPHA, A, &M, Y, &IONE, &BETA, X, &IONE );
   ```

3. To perform the operation
   $y(2:10) \longleftarrow 2A(3:11, 4:11) * B(4, 3:10)^T - 3y(2:10)$;
   the BLAS calling sequence is

   ```
   CALL DGEMV ( 'N', 9, 8, 2.0D0, A(3,4), M, B(4,3), N, -3.0D0,
   Y(2), 1 ).
   ```

   (Note the use of `LDA` to operate on the submatrix of A and the use of `INCX` to operate on a row of B.)
4. LAPACK (see Chapter 75) is a library based on the BLAS. Opening its Fortran files enables one to gain a good understanding of how to use BLAS routines. For example, a good way to start is to have a look at `dgetrf.f`, which performs a right-looking LU factorization by making calls to `DTRSM` and `DGEMM`.

References

[BDD02] S. Blackford, J. Demmel, J.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 subprograms (BLAS). *ACM Trans. Math. Softw.*, 28(2):135–151, 2002.

[Don02] J.J. Dongarra. Basic linear algebra subprograms technical forum standard. *Int. J. High Perform. Appl. Supercomp.*, 16(1–2):1–199, 2002.

[DBM79] J.J. Dongarra, J.R. Bunch, C.B. Moler, and G.W. Stewart. *LINPACK Users' Guide*. SIAM, Philadelphia, 1979.

[DDD90a] J.J. Dongarra, J. Du Croz, I.S. Duff, and S. Hammarling. A set of Level 3 basic linear algebra subprograms. *ACM Trans. Math. Softw.*, 16:1–17, 1990.

[DDD90b] J.J. Dongarra, J. Du Croz, I.S. Duff, and S. Hammarling. Algorithm 679: a set of Level 3 basic linear algebra subprograms. *ACM Trans. Math. Softw.*, 16:18–28, 1990.

[DDH88a] J.J. Dongarra, J. Du Croz, S. Hammarling, and R.J. Hanson. An extended set of FORTRAN basic linear algebra subprograms. *ACM Trans. Math. Softw.*, 14:1–17, 1988.

[DDH88b] J.J. Dongarra, J. Du Croz, S. Hammarling, and R.J. Hanson. Algorithm 656: an extended set of FORTRAN basic linear algebra subprograms. *ACM Trans. Math. Softw.*, 14:18–32, 1988.

[LHK79] C.L. Lawson, R.J. Hanson, D. Kincaid, and F. Krogh. Basic linear algebra subprograms for FORTRAN usage. *ACM Trans. Math. Softw.*, 5:308–323, 1979.

[WPD01] R.C. Whaley, A. Petitet, and J.J. Dongarra. Automated empirical optimizations of software and the ATLAS project. *Parallel Comp.*, 27:3–35, 2001.