TITLE

Basic Linear Algebra Subprograms

BYLINE

Robert A. van de Geijn
Department of Computer Science
The University of Texas at Austin
Austin, TX
USA
`rvdg@cs.utexas.edu`

Kazushige Goto
Texas Advanced Computing Center
The University of Texas at Austin
Austin, TX
USA
`kgoto@tacc.utexas.edu`

Synonym

BLAS

Definition

The Basic Linear Algebra Subprograms (BLAS) are an interface to commonly used fundamental linear algebra operations.

Discussion

Introduction

The BLAS interface supports portable high-performance implementation of applications that are matrix and vector computation intensive. The library or application developer focuses on casting computation in terms of the operations supported by the BLAS, leaving the architecture-specific optimization of that software layer to an expert.

A Motivating Example

The use of the BLAS interface will be illustrated by considering the Cholesky factorization of an $n \times n$ matrix A. When A is Symmetric Positive Definite (a property that guarantees that the algorithm completes) its Cholesky factorization is given by the lower triangular matrix L such that $A = LL^T$.

An algorithm for this operation can be derived as follows: Partition

$$A \rightarrow \left(\begin{array}{c|c} \alpha_{11} & \star \\ \hline a_{21} & A_{22} \end{array}\right) \quad \text{and} \quad L \rightarrow \left(\begin{array}{c|c} \lambda_{11} & 0 \\ \hline l_{21} & L_{22} \end{array}\right),$$

where α_{11} and λ_{11} are scalars, a_{21} and l_{21} are vectors, A_{22} is symmetric, L_{22} is lower triangular, and the $\star$ indicates the symmetric part of A that is not used. Then

$$\left(\begin{array}{c|c} \alpha_{11} & \star \\ \hline a_{21} & A_{22} \end{array}\right) = \left(\begin{array}{c|c} \lambda_{11} & 0 \\ \hline l_{21} & L_{22} \end{array}\right) \left(\begin{array}{c|c} \lambda_{11} & 0 \\ \hline l_{21} & L_{22} \end{array}\right)^T = \left(\begin{array}{c|c} \lambda_{11}^2 & \star \\ \hline \lambda_{11} l_{21} & l_{21} l_{21}^T + L_{22} L_{22}^T. \end{array}\right)$$

This yields the following algorithm for overwriting A with L:

- $\alpha_{11} \leftarrow \sqrt{\alpha_{11}}$.
- $a_{21} \leftarrow a_{21} / \alpha_{11}$.
- $A_{22} \leftarrow -a_{21} a_{21}^T + A_{22}$, updating only the lower triangular part of A_{22}. (This is called a symmetric rank-1 update.)
- Continue by overwriting A_{22} with L_{22} where $A_{22} = L_{22} L_{22}^T$.

A simple code in Fortran is given by

```
do j=1, n
  A( j,j ) = sqrt( A( j,j ) )
  do i=j+1,n
    A( i,j ) = A( i,j ) / A( j,j )
  enddo
  do k=j+1,n
    do i=k,n
      A( i,k ) = A( i,k ) - A( i,j ) * A( k,j )
    enddo
  enddo
enddo
```

Vector-Vector Operations (Level-1 BLAS)

The first BLAS interface was proposed in the 1970s when vector supercomputers were widely used for computational science. Such computers could achieve near-peak performance as long as the bulk of computation was cast in terms of vector operations and memory was accessed mostly contiguously. This interface is now referred to as the Level-1 BLAS.

Let x and y be vectors of appropriate length and α be scalar. Commonly encountered vector operations are multiplication of a vector by a scalar ($x \leftarrow \alpha x$), inner (dot) product ($\alpha \leftarrow x^T y$), and scaled vector addition ($y \leftarrow \alpha x + y$). This last operation is known as an `axpy`: $\underline{\text{a}}$lpha times $\underline{\text{x}}$ $\underline{\text{p}}$lus $\underline{\text{y}}$.

The Cholesky factorization, coded in terms of such operations, is given by

```
do j=1, n
  A( j,j ) = sqrt( A( j,j ) )
  call dscal( n-j, 1.0d00 / A( j,j ), A( j+1, j ), 1 )
  do k=j+1,n
    call daxpy( n-k+1, -A( k,j ), A(k,j), 1, A( k, k ), 1 );
  enddo
enddo
```

Here

- The first letter in `dscal` and `daxpy` indicates that the computation is with $\underline{\text{d}}$ouble precision numbers.
- The call to `dscal` performs the computation $a_{21} \leftarrow a_{21}/\alpha_{11}$.
- The loop

  ```
  do i=k,n
    A( i,k ) = A( i,k ) - A( i,j ) * A( k,j )
  enddo
  ```

  is replaced by the call

  ```
  call daxpy( n-k+1, -A( k,j ), A( k,j ), 1, A( k, k ), 1 )
  ```

If the operations supported by `dscal` and `daxpy` achieve high performance on a target archecture *then* so will the implementation of the Cholesky factorization, since it casts most computation in terms of those operations.

A representative calling sequence for a Level-1 BLAS routine is given by

```
_axpy( n, alpha, x, incx, y, incy )
```

which implements the operation $y = \alpha x + y$. Here

- The "_" indicates the data type. The choices for this first letter are

`s`	single precision
`d`	double precision
`c`	single precision complex
`z`	double precision complex

- The operation is identified as `axpy`: alpha times x plus y.
- `n` indicates the number of elements in the vectors x and y.
- `alpha` is the scalar α.
- `x` and `y` indicate the memory locations where the first elements of x and y are stored, respectively.
- `incx` and `incy` equal the increment by which one has to stride through memory to locate the elements of vectors x and y, respectively.

The following are the most frequently used Level-1 BLAS:

routine/ function	operation
`_swap`	$x \leftrightarrow y$
`_scal`	$x \leftarrow \alpha x$
`_copy`	$y \leftarrow x$
`_axpy`	$y \leftarrow \alpha x + y$
`_dot`	$x^T y$
`_nrm2`	$\Vert x \Vert_2$
`_asum`	$\Vert \mathrm{re}(x) \Vert_1 + \Vert \mathrm{im}(x) \Vert_1$
`i_max`	$\min(k) : \vert \mathrm{re}(x_k) \vert + \vert \mathrm{im}(x_k) \vert = \max(\vert \mathrm{re}(x_i) \vert + \vert \mathrm{im}(x_i) \vert)$

Matrix-Vector Operations (Level-2 BLAS)

The next level of BLAS supports operations with matrices and vectors. The simplest example of such an operation is the matrix-vector product: $y \leftarrow Ax$ where x and y are vectors and A is a matrix. Another example is the computation $A_{22} = -a_{21}a_{21}^T + A_{22}$ (symmetric rank-1 update) in the Cholesky factorization. This operation can be recoded as

```
do j=1, n
  A( j,j ) = sqrt( A( j,j ) )
  call dscal( n-j, 1.0d00 / A( j,j ), A( j+1, j ), 1 )
  call dsyr( 'Lower triangular', n-j, -1.0d00,
             A( j+1,j ), 1, A( j+1,j+1 ), lda )
enddo
```

Here `dsyr` is the routine that implements a double precision symmetric rank-1 update. Readability of the code is improved by casting computation in terms of routines that implement the operations that appear in the algorithm: `dscal` for $a_{21} = a_{21}/\alpha_{11}$ and `dsyr` for $A_{22} = -a_{21}a_{21}^T + A_{22}$.

The naming convention for Level-2 BLAS routines is given by

`_XXYY,`

where

- "_" can take on the values `s`, `d`, `c`, `z`.
- `XX` indicates the shape of the matrix:

XX	matrix shape
ge	general (rectangular)
sy	symmetric
he	Hermitian
tr	triangular

  In addition, operations with banded matrices are supported, which we do not discuss here.

- `YY` indicates the operation to be performed:

`YY`	matrix shape
`mv`	matrix vector multiplication
`sv`	solve vector
`r`	rank-1 update
`r2`	rank-2 update

A representative call to a Level-2 BLAS operation is given by

```
dsyr( uplo, n, alpha, x, incx, A, lda )
```

which implements the operation $A = \alpha x x^T + A$, updating the lower or upper triangular part of A by choosing `uplo` as `'Lower triangular'` or `'Upper triangular'`, respectively. The parameter `lda` (the leading dimension of matrix A) indicates the increment by which memory has to be traversed in order to address successive elements in a row of matrix `A`.

The following table gives the most commonly used Level-2 BLAS operations:

routine/ function	operation
`_gemv`	general matrix-vector multiplication
`_symv`	symmetric matrix-vector multiplication
`_trmv`	triangular matrix-vector multiplication
`_trsv`	triangular solve vector
`_ger`	general rank-1 update
`_syr`	symmetric rank-1 update
`_syr2`	symmetric rank-2 update

There are also interfaces for operation with banded matrices stored in packed format as well as for operations with Hermitian matrices.

Matrix-Matrix Operations (Level-3 BLAS)

The problem with vector operations and matrix-vector operations is that they perform $O(n)$ computations with $O(n)$ data and $O(n^2)$ computations with $O(n^2)$ data, respectively. This makes it hard, if not impossible, to leverage cache memory now that processing speeds greatly outperform memory speeds, unless the problem size is relatively small (fits in cache memory).

The solution is to cast computation in terms of matrix-matrix operations like matrix-matrix multiplication. Consider again Cholesky factorization. Partition

$$A \rightarrow \left(\begin{array}{c|c} A_{11} & \star \\ \hline A_{21} & A_{22} \end{array} \right) \quad \text{and} \quad L \rightarrow \left(\begin{array}{c|c} L_{11} & 0 \\ \hline L_{21} & L_{22} \end{array} \right),$$

where A_{11} and L_{11} are $n_b \times n_b$ submatrices. Then

$$\left(\begin{array}{c|c} A_{11} & \star \\ \hline A_{21} & A_{22} \end{array} \right) = \left(\begin{array}{c|c} L_{11} & 0 \\ \hline L_{21} & L_{22} \end{array} \right) \left(\begin{array}{c|c} L_{11} & 0 \\ \hline L_{21} & L_{22} \end{array} \right)^T = \left(\begin{array}{c|c} L_{11}L_{11}^T & \star \\ \hline L_{21}L_{11}^T & L_{21}L_{21}^T + L_{22}L_{22}^T \end{array} \right)$$

This yields the algorithm

- $A_{11} = L_{11}$ where $A_{11} = L_{11}L_{11}^T$ (Cholesky factorization of a smaller matrix).
- $A_{21} = L_{21}$ where $L_{21}L_{11}^T = A_{21}$ (triangular solve with multiple right-hand sides).
- $A_{22} = -L_{21}L_{21}^T + A_{22}$, updating only the lower triangular part of A_{22} (symmetric rank-k update).
- Continue by overwriting A_{22} with L_{22} where $A_{22} = L_{22}L_{22}^T$.

A representative code in Fortran is given by

```
do j=1, n, nb
  jb = min( nb, n-j+1 )
  call chol( jb, A( j, j ), lda )

  call dtrsm( ‘Right’, ‘Lower triangular’, ‘Transpose’, ‘Nonunit diag’,
              J-JB+1, JB, 1.0d00, A( j, j ), lda, A( j+jb, j ), lda )

  call dsyrk( ‘Lower triangular’, ‘No transpose’, J-JB+1, JB,
              -1.0d00, A( j+jb, j ), lda, 1.0d00, A( j+jb, j+jb ), lda )
enddo
```

Here subroutine `chol` performs a Cholesky factorization; `dtrsm` and `dsyrk` are level-3 BLAS routines:

- The call to `dtrsm` implements $A_{21} \leftarrow L_{21}$ where $L_{21}L_{11}^T = A_{21}$.
- The call to `dsyrk` implements $A_{22} \leftarrow -L_{21}L_{21}^T + A_{22}$.

The bulk of the computation is now cast in terms of matrix-matrix operations which can achieve high performance.

The naming convention for Level-3 BLAS routines are similar to those for the Level-2 BLAS. A representative call to a Level-3 BLAS operation is given by

```
dsyrk( uplo, trans, n, k, alpha, A, lda, beta, C, ldc )
```

which implements the operation $C \leftarrow \alpha A A^T + \beta C$ or $C \leftarrow \alpha A^T A + \beta C$ depending on whether `trans` is chosen as `'No transpose'` or `'Transpose'`, respectively. It updates the lower or upper triangular part of C depending on whether `uplo` equal `'Lower triangular'` or `'Upper triangular'`, respectively. The parameters `lda` and `ldc` are the leading dimensions of arrays `A` and `C`, respectively.

The following table gives the most commonly used Level-3 BLAS operationsx

routine/ function	operation
`_gemm`	general matrix-matrix multiplication
`_symm`	symmetric matrix-matrix multiplication
`_trmm`	triangular matrix-matrix multiplication
`_trsm`	triangular solve with multiple right-hand sides
`_syrk`	symmetric rank-k update
`_syr2k`	symmetric rank-2k update

Impact on performance

Figure 1 illustrates the performance benefits that come from using the different levels of BLAS on a typical architecture.

BLAS-like interfaces

CBLAS A C interface for the BLAS, CBLAS, has also been defined to simplify the use of the BLAS from C and C++. The CBLAS support matrices stored in row and column major format.

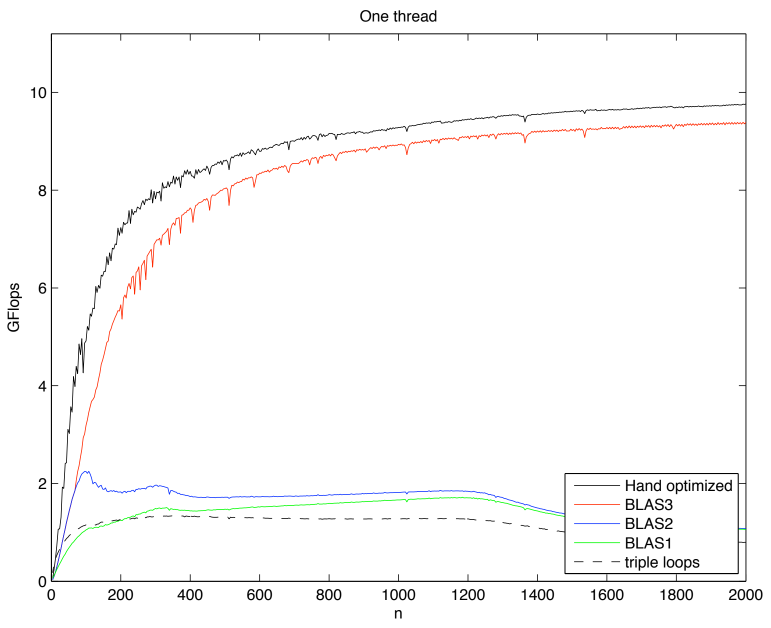

Figure 1: Performance of the different implementations of Cholesky factorization that use different levels of BLAS. The target processor has a peak of 11.2 Gflops (billions of floating point operations per second). BLAS1, BLAS2, and BLAS3 indicate that the bulk of computation was cast in terms of Level-1, -2, or -3 BLAS, respectively.

libflame The `libflame` library that has resulted from the FLAME project encompasses the functionality of the BLAS as well as higher level linear algebra operations. It uses an object-based interface so that a call to a BLAS routine like `_syrk` becomes

```
FLA_Syrk( uplo, trans, alpha, A, beta, C )
```

thus hiding many of the dimension and indexing details.

Sparse BLAS Several efforts were made to define interfaces for BLAS-like operations with sparse matrices. These do not seem to have caught on, possibly because the storage of sparse matrices is much more complex.

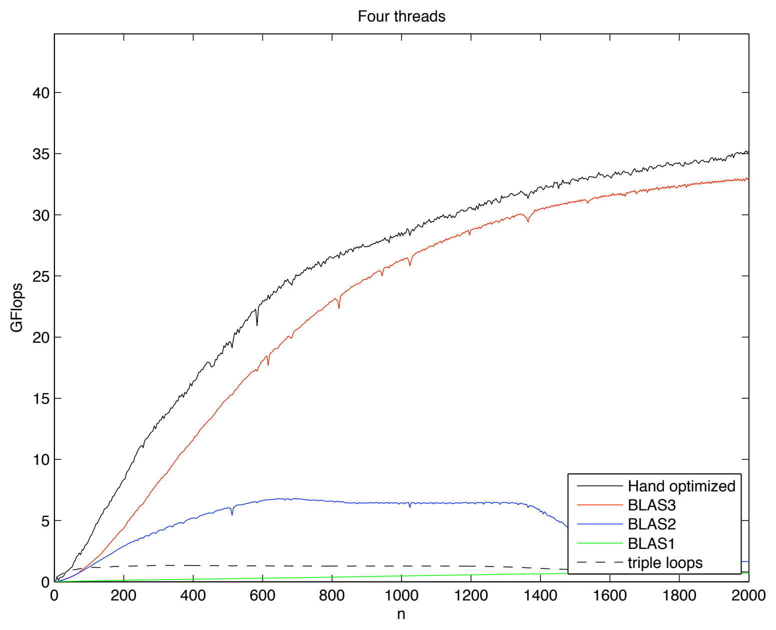

Figure 2: Performance of the different implementations of Cholesky factorization that use different levels of BLAS, using four threads on a architectures with four cores and a peak of 44.8 Gflops.

Parallel BLAS

Parallelism with BLAS operations can be achieved in a number of ways.

Multithreaded BLAS On shared-memory architectures multithreaded BLAS are often available. Such implementations achieve parallelism within each BLAS call without need for changing code that is written in terms of the interface. In Figure 2 shows the performance of the Cholesky factorization codes when multithreaded BLAS are used on a multicore architecture.

PBLAS As part of the ScaLAPACK project, an interface for distributed memory parallel BLAS was proposed, the PBLAS. The goal was to make this interface closely resemble the traditional BLAS. A call to `dsyrk` becomes

```
pdsyrk(uplo, trans, n, k, alpha, A, iA, jA, descA,
                                beta, C, iC, jC, descC)
```

where the new parameters `iA`, `jA`, `descA`, etc., encapsulate information about the submatrix with which to multiply and the distribution to a logical two-dimensional mesh of processing nodes.

PLAPACK The PLAPACK project provides an alternative to ScaLAPACK. It also provides BLAS for distributed memory architectures, but (like `libflame`) goes one step further towards encapsulation. The call for parallel symmetric rank-k update becomes

```
PLA_Syrk( uplo, trans, alpha, A, beta, C )
```

where all information about the matrices, their distribution, and the storage of local submatrices is encapsulated in the parameters `A` and `C`.

Available Implementations

Many of the software and hardware vendors market high-performance implementations of the BLAS. Examples include IBM's ESSL, Intel's MKL, AMD's ACML, NEC's MathKeisan, and HP's MLIB libraries. Widely used open source implementations include ATLAS and the GotoBLAS. Comparisons of performance of some of these implementations are given in Figures 3 and 4.

The details about the platform on which the performance data was gathered nor the versions of the libraries that were used are given because architectures and libraries continuously change and therefore which is faster or slower can easily change with the next release of a processor or library.

Related Entries

ATLAS

FLAME

LAPACK

PBLAS

PLAPACK

ScaLAPACK

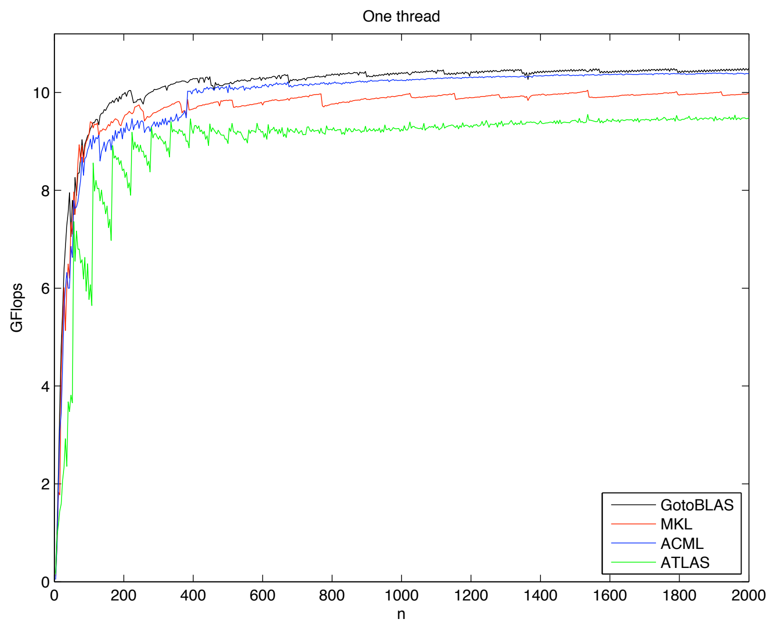

Figure 3: Performance of different BLAS libraries for matrix-matrix multiplication (dgemm).

Bibliographic Notes and Further Reading

What came to be called the Level-1 BLAS were first published in 1979, followed by the Level-2 BLAS in 1988 and Level-3 BLAS in 1990 [10, 5, 4].

Matrix-matrix multiplication (_gemm) is considered the most important operation, since high-performance implementations of the other Level-3 BLAS can be coded in terms of it [9]. Many implementations of _gemm are now based on the techniques developed by Kazushige Goto [8]. These techniques extend to the high-performance implementation of other Level-3 BLAS [7] and multithreaded architectures [11]. Practical algorithms for the distributed memory parallel implementation of matrix-matrix multiplication, used by ScaLAPACK and PLAPACK, were first discussed in [12, 1] and for other Level-3 BLAS in [3].

As part of the BLAS Technical Forum an effort was made in the late 1990s to extend the BLAS interfaces to include additional functionality [2]. Outcomes included the CBLAS interface, which is now widely supported,

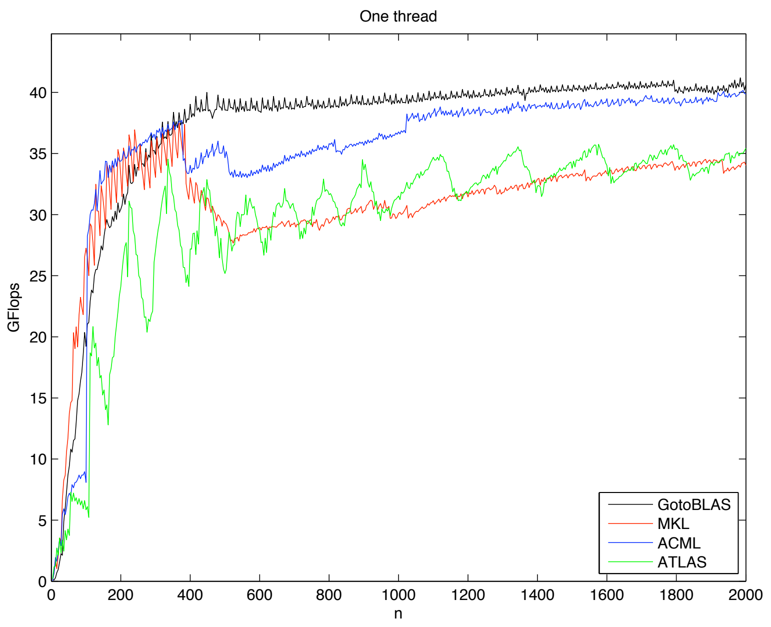

Figure 4: Parallel performance of different BLAS libraries for matrix-matrix multiplication (`dgemm`).

and an interface for Sparse BLAS [6].

References

[1] R. C. Agarwal, F. Gustavson, and M. Zubair. A high-performance matrix multiplication algorithm on a distributed memory parallel computer using overlapped communication. *IBM Journal of Research and Development*, 38(6), 1994.

[2] L. Susan Blackford, James Demmel, Jack Dongarra, Iain Duff, Sven Hammarling, Greg Henry, Michael Heroux, Linda Kaufman, Andrew Lumsdaine, Antoine Petitet, Roldan Pozo, Karin Remington, and R. Clint Whaley. An updated set of Basic Linear Algebra Subprograms (BLAS). *ACM Transactions on Mathematical Software*, 28(2):135–151, June 2002.

[3] Almadena Chtchelkanova, John Gunnels, Greg Morrow, James Overfelt, and Robert A. van de Geijn. Parallel implementation of BLAS: General techniques for level 3 BLAS. *Concurrency: Practice and Experience*, 9(9):837–857, Sept. 1997.

[4] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain Duff. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Soft.*, 16(1):1–17, March 1990.

[5] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J. Hanson. An extended set of FORTRAN basic linear algebra subprograms. *ACM Trans. Math. Soft.*, 14(1):1–17, March 1988.

[6] Iain S. Duff, Michael A. Heroux, and Roldan Pozo. An overview of the Sparse Basic Linear Algebra Subprograms: The new standard from the BLAS Technical Forum. *ACM Transactions on Mathematical Software*, 28(2):239–267, June 2002.

[7] Kazushige Goto and Robert van de Geijn. High-performance implementation of the level-3 BLAS. *ACM Trans. Math. Softw.*, 35(1):1–14, 2008.

[8] Kazushige Goto and Robert A. van de Geijn. Anatomy of high-performance matrix multiplication. *ACM Trans. Math. Softw.*, 34(3):1–25, 2008.

[9] B. Kågström, P. Ling, and C. Van Loan. GEMM-based level 3 BLAS: High performance model implementations and performance evaluation benchmark. *ACM Trans. Math. Soft.*, 24(3):268–302, 1998.

[10] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Soft.*, 5(3):308–323, Sept. 1979.

[11] Bryan Marker, Field G. Van Zee, Kazushige Goto, Gregorio Quintana-Ortí, and Robert A. van de Geijn. Toward scalable matrix multiply on multithreaded architectures. In A.-M. Kermarrec, L. Bougé, and T. Priol, editors, *Euro-Par*, LNCS 4641, pages 748–757, 2007.

[12] Robert van de Geijn and Jerrell Watts. SUMMA: Scalable universal matrix multiplication algorithm. *Concurrency: Practice and Experience*, 9(4):255–274, April 1997.