Basic Linear Algebra Subprograms

From Wikipedia, the free encyclopedia

Basic Linear Algebra Subprograms (BLAS) is a specification that prescribes a set of low-level routines for performing common linear algebra operations such as vector addition, scalar multiplication, dot products, linear combinations, and matrix multiplication. They are the *de facto* standard low-level routines for linear algebra libraries; the routines have bindings for both C and Fortran. Although the BLAS specification is general, BLAS implementations are often optimized for speed on a particular machine, so using them can bring substantial performance benefits. BLAS implementations will take advantage of special floating point hardware such as vector registers or SIMD instructions.

BLAS

Stable release	3.7.0 / 24 December 2016
Written in	Fortran
Platform	Cross-platform
Type	Library
Website	www.netlib.org/blas/ (http://www.netlib.org/blas/)

It originated as a Fortran library in 1979[1] and its interface was standardized by the BLAS Technical (BLAST) Forum, whose latest BLAS report can be found on the netlib website.[2] This Fortran library is known as the *reference implementation* (sometimes confusingly referred to as *the* BLAS library) and is not optimized for speed but in the public domain.[3][4]

Most libraries that offer linear algebra routines conform to the BLAS interface, allowing library users to develop programs that are agnostic of the BLAS library being used. Examples of BLAS libraries include: AMD Core Math Library (ACML), ATLAS, Intel Math Kernel Library (MKL), and OpenBLAS. ACML is no longer supported.[5] MKL is a freeware[6] and proprietary[7] vendor library optimized for x86 and x86-64 with a performance emphasis on Intel processors.[8] OpenBLAS is an open-source library that is hand-optimized for many of the popular architectures. ATLAS is a portable library that automatically optimizes itself for an arbitrary architecture. The LINPACK benchmarks rely heavily on the BLAS routine gemm for its performance measurements.

Many numerical software applications use BLAS-compatible libraries to do linear algebra computations, including Armadillo, LAPACK, LINPACK, GNU Octave, Mathematica,[9] MATLAB,[10] NumPy,[11] and R.

Contents

Background

With the advent of numerical programming, sophisticated subroutine libraries became useful. These libraries would contain subroutines for common high-level mathematical operations such as root finding, matrix inversion, and solving systems of equations. The language of choice was FORTRAN. The most prominent numerical programming library was IBM's Scientific Subroutine Package (SSP).[12] These subroutine libraries allowed programmers to concentrate on their specific problems and avoid re-implementing well-known algorithms. The library routines would also be better than average implementations; matrix algorithms, for example, might use full pivoting to get better numerical accuracy. The library routines would also have more efficient routines. For example, a library may include a program to solve a matrix that is upper triangular. The libraries would include single-precision and double-precision versions of some algorithms.

Initially, these subroutines used hard-coded loops for their low-level operations. For example, if a subroutine need to perform a matrix multiplication, then the subroutine would have three nested loops. Linear algebra programs have many common low-level operations (the so-called "kernel" operations, not related to operating systems).[13] Between 1973 and 1977, several of these kernel operations were identified.[14] These kernel operations became defined subroutines that math libraries could call. The kernel calls had advantages over hard-coded loops: the library routine would be more readable, there were fewer chances for bugs, and the kernel implementation could be optimized for speed. A specification for these kernel operations using scalars and vectors, the level-1 Basic Linear Algebra Subroutines (BLAS), was published in 1979.[15] BLAS was used to implement the linear algebra subroutine library LINPACK.

The BLAS abstraction allows customization for high performance. For example, LINPACK is a general purpose library that can be used on many different machines without modification. LINPACK could use a generic version of BLAS. To gain performance, different machines might use tailored versions of BLAS. As computer architectures became more sophisticated, vector machines appeared. BLAS for a vector machine could use the machine's fast vector operations. (While vector processors eventually fell out of favor, vector instructions in modern CPUs are essential for optimal performance in BLAS routines.)

Other machine features became available and could also be exploited. Consequently, BLAS was augmented from 1984 to 1986 with

level-2 kernel operations that concerned vector-matrix operations. Memory hierarchy was also recognized as something to exploit. Many computers have cache memory that is much faster than main memory; keeping matrix manipulations localized allows better usage of the cache. In 1987 and 1988, the level 3 BLAS were identified to do matrix-matrix operations. The level 3 BLAS encouraged block-partitioned algorithms. The LAPACK library uses level 3 BLAS.[16]

The original BLAS concerned only densely stored vectors and matrices. Further extensions to BLAS, such as for sparse matrices, have been addressed.[17]

ATLAS

Automatically Tuned Linear Algebra Software (ATLAS) attempts to make a BLAS implementation with higher performance. ATLAS defines many BLAS operations in terms of some core routines and then tries to automatically tailor the core routines to have good performance. A search is performed to choose good block sizes. The block sizes may depend on the computer's cache size and architecture. Tests are also made to see if copying arrays and vectors improves performance. For example, it may be advantageous to copy arguments so that they are cache-line aligned so user-supplied routines can use SIMD instructions.

Functionality

BLAS functionality is categorized into three sets of routines called "levels", which correspond to both the chronological order of definition and publication, as well as the degree of the polynomial in the complexities of algorithms; Level 1 BLAS operations typically take linear time, $O(n)$, Level 2 operations quadratic time and Level 3 operations cubic time.[18] Modern BLAS implementations typically provide all three levels.

Level 1

This level consists of all the routines described in the original presentation of BLAS (1979),[1] which defined only *vector operations* on strided arrays: dot products, vector norms, a generalized vector addition of the form

$$\boldsymbol{y} \leftarrow \alpha \boldsymbol{x} + \boldsymbol{y}$$

(called "axpy") and several other operations.

Level 2

This level contains *matrix-vector operations* including, among other things, a generalized matrix-vector multiplication (gemv):

$$\boldsymbol{y} \leftarrow \alpha \boldsymbol{A} \boldsymbol{x} + \beta \boldsymbol{y}$$

as well as a solver for $\boldsymbol{x}$ in the linear equation

$$\boldsymbol{T} \boldsymbol{x} = \boldsymbol{y}$$

with $\boldsymbol{T}$ being triangular. Design of the Level 2 BLAS started in 1984, with results published in 1988.[19] The Level 2 subroutines are especially intended to improve performance of programs using BLAS on vector processors, where Level 1 BLAS are suboptimal "because they hide the matrix-vector nature of the operations from the compiler."[19]

Level 3

This level, formally published in 1990,[18] contains *matrix-matrix operations*, including a "general matrix multiplication" (gemm), of the form

$$\boldsymbol{C} \leftarrow \alpha \boldsymbol{A} \boldsymbol{B} + \beta \boldsymbol{C}$$

where $\boldsymbol{A}$ and $\boldsymbol{B}$ can optionally be transposed or hermitian-conjugated inside the routine and all three matrices may be strided. The ordinary matrix multiplication $\boldsymbol{A}\ \boldsymbol{B}$ can be performed by setting α to one and $\boldsymbol{C}$ to an all-zeros matrix of the appropriate size.

Also included in Level 3 are routines for solving

$$\boldsymbol{B} \leftarrow \alpha \boldsymbol{T}^{-1} \boldsymbol{B}$$

where $\boldsymbol{T}$ is a triangular matrix, among other functionality.

Due to the ubiquity of matrix multiplications in many scientific applications, including for the implementation of the rest of Level 3 BLAS,[20] and because faster algorithms exist beyond the obvious repetition of matrix-vector multiplication, gemm is a prime target of optimization for BLAS implementers. E.g., by decomposing one or both of $\boldsymbol{A}$, $\boldsymbol{B}$ into block matrices, gemm can be implemented recursively. This is one of the motivations for including the β parameter, so the results of previous blocks can be accumulated. Note that this decomposition requires the special case $\beta = 1$ which many implementations optimize for, thereby eliminating one multiplication for each value of $\boldsymbol{C}$. This decomposition allows for better locality of reference both in space and time of the data used in the product. This, in turn, takes advantage of the cache on the system.[21] For systems with more than one level of cache, the blocking can be applied a second time to the order in which the blocks are used in the computation. Both of these levels of optimization are used in implementations such as ATLAS. More recently, implementations by Kazushige Goto have shown that blocking only for the L2 cache, combined with careful amortizing of copying to contiguous memory to reduce TLB misses, is superior to ATLAS.[22] A highly tuned

implementation based on these ideas is part of the GotoBLAS and OpenBLAS.

Implementations

Accelerate
: Apple's framework for macOS and iOS, which includes tuned versions of BLAS and LAPACK.[23][24]

ACML
: The AMD Core Math Library, supporting the AMD Athlon and Opteron CPUs under Linux and Windows.[25]

C++ AMP BLAS
: The C++ AMP BLAS Library is an open source implementation of BLAS for Microsoft's AMP language extension for Visual C++.[26]

ATLAS
: Automatically Tuned Linear Algebra Software, an open source implementation of BLAS APIs for C and Fortran 77.[27]

BLIS
: BLAS-like Library Instantiation Software framework for rapid instantiation.[28]

cuBLAS
: Optimized BLAS for NVIDIA based GPU cards.[29]

clBLAS
: An OpenCL implementation of BLAS.[30]

clBLAST
: A tuned OpenCL implementation of BLAS. [31]

Eigen BLAS
: A Fortran 77 and C BLAS library implemented on top of the MPL-licensed Eigen library, supporting x86, x86 64, ARM (NEON), and PowerPC architectures.[1] (http://eigen.tuxfamily.org) (Note: as of Eigen 3.0.3, the BLAS interface is not built by default and the documentation refers to it as "a work in progress which is far to be ready for use".)

ESSL
: IBM's Engineering and Scientific Subroutine Library, supporting the PowerPC architecture under AIX and Linux.[32]

GotoBLAS
: Kazushige Goto's BSD-licensed implementation of BLAS, tuned in particular for Intel Nehalem/Atom, VIA Nanoprocessor, AMD Opteron.[33]

HP MLIB
: HP's Math library supporting IA-64, PA-RISC, x86 and Opteron architecture under HPUX and Linux.

Intel MKL
: The Intel Math Kernel Library, supporting x86 32-bits and 64-bits, available free from Intel.[6] Includes optimizations for Intel Pentium, Core and Intel Xeon CPUs and Intel Xeon Phi; support for Linux, Windows and macOS.[34]

MathKeisan
: NEC's math library, supporting NEC SX architecture under SUPER-UX, and Itanium under Linux[35]

Netlib BLAS
: The official reference implementation on Netlib, written in Fortran 77.[36]

Netlib CBLAS
: Reference C interface to the BLAS. It is also possible (and popular) to call the Fortran BLAS from C.[37]

OpenBLAS
: Optimized BLAS based on GotoBLAS, supporting x86, x86-64, MIPS and ARM processors.[38]

PDLIB/SX
: NEC's Public Domain Mathematical Library for the NEC SX-4 system.[39]

SCSL
: SGI's Scientific Computing Software Library contains BLAS and LAPACK implementations for SGI's Irix workstations.[40]

Sun Performance Library
: Optimized BLAS and LAPACK for SPARC, Core and AMD64 architectures under Solaris 8, 9, and 10 as well as Linux.[41]

Similar libraries but not compatible with BLAS

Armadillo
: Armadillo is a C++ linear algebra library aiming towards a good balance between speed and ease of use. It employs template classes, and has optional links to BLAS/ATLAS and LAPACK. It is sponsored by NICTA (in Australia) and is licensed under a free license.[42]

ACL
: AMD Compute Libraries [43]

- clBLAS: complete set of BLAS level 1, 2 & 3 routines [30]
- clSparse:[44] Routines for thin Matrix
- clFFT:[45] fast FFT
- clRNG:[46] Random Generators MRG31k3p, MRG32k3a, LFSR113 und Philox-4×32-10

CUDA SDK
: The NVIDIA CUDA SDK includes BLAS functionality for writing C programs that runs on GeForce 8 Series (Tesla-Architecture) or newer graphics cards. The library cuBLAS has been designed with the purpose of implementing the BLAS capabilities using the CUDA SDK.

Eigen
: The Eigen template library provides an easy to use highly generic C++98 template interface to matrix/vector operations and related algorithms like solving algorithms, decompositions etc. It uses vector capabilities and is optimized for both fixed size and dynamic sized and sparse matrices.[47]

Elemental
: Elemental is an open source software for distributed-memory dense and sparse-direct linear algebra and optimization.[48]

GSL
: The GNU Scientific Library Contains a multi-platform implementation in C which is distributed under the GNU General Public License.

HASEM
: is a C++ template library, being able to solve linear equations and to compute eigenvalues. It is licensed under BSD License.[49]

LAMA
: The Library for Accelerated Math Applications (LAMA) is a C++ template library for writing numerical solvers targeting various hardwares (e.g. GPUs through CUDA or OpenCL) on distributed memory systems, hiding the hardware specific programming from the program developer

Libflame
: FLAME project implementation of dense linear algebra library[50]

MAGMA
: Matrix Algebra on GPU and Multicore Architectures (MAGMA) project develops a dense linear algebra library similar to LAPACK but for heterogeneous and hybrid architectures including multicore systems accelerated with general-purpose computing on graphics processing units.[51]

Mir
: An LLVM-accelerated generic numerical library for science and machine learning written in D. It provides generic linear algebra subprograms (GLAS).[52]

MTL4
: The Matrix Template Library version 4 is a generic C++ template library providing sparse and dense BLAS functionality. MTL4 establishes an intuitive interface (similar to MATLAB) and broad applicability thanks to generic programming.

PLASMA
: The Parallel Linear Algebra for Scalable Multi-core Architectures (PLASMA) project is a modern replacement of LAPACK for multi-core architectures. PLASMA is a software framework for development of asynchronous operations and features out of order scheduling with a runtime scheduler called QUARK that may be used for any code that expresses its dependencies with a directed acyclic graph.[53]

uBLAS
: A generic C++ template class library providing BLAS functionality. Part of the Boost library. It provides bindings to many hardware-accelerated libraries in a unifying notation. Moreover, uBLAS focuses on correctness of the algorithms using advanced C++ features.[54]

Sparse BLAS

Several extensions to BLAS for handling sparse matrices have been suggested over the course of the library's history; a small set of sparse matrix kernel routines were finally standardized in 2002.[55]

See also

- List of numerical libraries
- Math Kernel Library, math library optimized for the Intel architecture; includes BLAS, LAPACK
- Numerical linear algebra, the type of problem BLAS solves

References

1. *Lawson, C. L.; Hanson, R. J.; Kincaid, D.; Krogh, F. T. (1979). "Basic Linear Algebra Subprograms for FORTRAN usage". *ACM Trans. Math. Softw.* 5: 308–323. doi:10.1145/355841.355847. Algorithm 539.
2. Netlib website (http://netlib.org/blas/blast-forum)
3. blaseman (http://www.lahey.com/docs/blaseman_lin62.pdf) Archived (https://web.archive.org/web/20161012014431/http://www.lahey.com/docs/blaseman_lin62.pdf) October 12, 2016, at the Wayback Machine. *"The products are the implementations of the public domain BLAS (Basic Linear Algebra Subprograms) and LAPACK (Linear Algebra PACKage), which have been developed by groups of people such as Prof. Jack Dongarra, University of Tennessee, USA and all published on the WWW (URL: http://www.netlib.org/)."*

4. Jack Dongarra; Gene Golub; Eric Grosse; Cleve Moler; Keith Moore. "Netlib and NA-Net: building a scientific computing community" (PDF). netlib.org. Retrieved 2016-02-13. "*The Netlib software repository was created in 1984 to facilitate quick distribution of public domain software routines for use in scientific computation.*"
5. "ACML – AMD Core Math Library". AMD. 2013. Retrieved 26 August 2015.
6. "No Cost Options for Intel Math Kernel Library (MKL), Support yourself, Royalty-Free". Intel. 2015. Retrieved 31 August 2015.
7. "Intel® Math Kernel Library (Intel® MKL)". Intel. 2015. Retrieved 25 August 2015.
8. "Optimization Notice". Intel. 2012. Retrieved 10 April 2013.
9. Douglas Quinney (2003). "So what's new in Mathematica 5.0?" (PDF). *MSOR Connections*. The Higher Education Academy. 3 (4). Archived from the original (PDF) on 2013-10-29.
10. Cleve Moler (2000). "MATLAB Incorporates LAPACK". MathWorks. Retrieved 26 October 2013.
11. Stéfan van der Walt; S. Chris Colbert & Gaël Varoquaux (2011). "The NumPy array: a structure for efficient numerical computation". *Computing in Science and Engineering*. IEEE. arXiv:1102.1523.
12. Boisvert, Ronald F. (2000). "Mathematical software: past, present, and future". *Mathematics and Computers in Simulation*. 54 (4–5): 227–241. arXiv:cs/0004004. doi:10.1016/S0378-4754(00)00185-3.
13. Even the SSP (which appeared around 1966) had some basic routines such as RADD (add rows), CADD (add columns), SRMA (scale row and add to another row), and RINT (row interchange). These routines apparently were not used as kernel operations to implement other routines such as matrix inversion. See IBM (1970), *System/360 Scientific Subroutine Package, Version III, Programmer's Manual* (5th ed.), International Business Machines, GH20-0205-4.
14. BLAST Forum 2001, p. 1.
15. Lawson et al. 1979.
16. BLAST Forum 2001, pp. 1–2.
17. BLAST Forum 2001, p. 2.
18. Dongarra, Jack J.; Du Croz, Jeremy; Hammarling, Sven; Duff, Iain S. (1990). "A set of level 3 basic linear algebra subprograms". *ACM Transactions on Mathematical Software*. 16 (1): 1–17. doi:10.1145/77626.79170. ISSN 0098-3500.
19. Dongarra, Jack J.; Du Croz, Jeremy; Hammarling, Sven; Hanson, Richard J. (1988). "An extended set of FORTRAN Basic Linear Algebra Subprograms". *ACM Trans. Math. Softw.* 14: 1–17. doi:10.1145/42288.42291.
20. Goto, Kazushige; van de Geijn, Robert (2008). "High-performance implementation of the level-3 BLAS" (PDF). *ACM Transactions on Mathematical Software*. 35 (1).
21. Golub, Gene H.; Van Loan, Charles F. (1996), *Matrix Computations* (3rd ed.), Johns Hopkins, ISBN 978-0-8018-5414-9
22. Goto, Kazushige; van de Geijn, Robert A. (2008). "Anatomy of High-performance Matrix Multiplication". *ACM Trans. Math. Softw.* 34 (3): 12:1–12:25. doi:10.1145/1356052.1356053. ISSN 0098-3500.
23. http://developer.apple.com/library/mac/#releasenotes/Performance/RN-vecLib/
24. http://developer.apple.com/library/ios/#documentation/Accelerate/Reference/AccelerateFWRef/
25. http://developer.amd.com/acml.aspx
26. http://ampblas.codeplex.com/
27. http://math-atlas.sourceforge.net/
28. https://github.com/flame/blis
29. http://developer.nvidia.com/cublas
30. https://github.com/clMathLibraries/clBLAS
31. https://github.com/CNugteren/CLBlast
32. http://publib.boulder.ibm.com/infocenter/clresctr/index.jsp?topic=/com.ibm.cluster.essl.doc/esslbooks.html
33. "Archived copy". Archived from the original on 2012-05-17. Retrieved 2012-05-24.
34. http://software.intel.com/en-us/intel-mkl/
35. http://www.mathkeisan.com/
36. http://www.netlib.org/blas/
37. http://www.netlib.org/blas
38. OpenBLAS : An optimized BLAS library (http://www.openblas.net/)
39. "Archived copy". Archived from the original on 2007-02-22. Retrieved 2007-05-20.
40. "Archived copy". Archived from the original on 2007-05-13. Retrieved 2007-05-20.
41. http://www.oracle.com/technetwork/server-storage/solarisstudio/overview/index.html
42. http://arma.sourceforge.net/
43. http://developer.amd.com/tools-and-sdks/opencl-zone/acl-amd-compute-libraries/
44. https://github.com/clMathLibraries/clSPARSE
45. https://github.com/clMathLibraries/clFFT
46. https://github.com/clMathLibraries/clRNG
47. http://eigen.tuxfamily.org
48. Elemental: distributed-memory dense and sparse-direct linear algebra and optimization — Elemental (http://libelemental.org/)
49. http://sourceforge.net/projects/hasem/
50. "Archived copy". Archived from the original on 2010-08-03. Retrieved 2011-02-21.
51. http://icl.eecs.utk.edu/magma/
52. "Dlang Numerical and System Libraries".
53. http://icl.eecs.utk.edu/
54. http://www.boost.org/doc/libs/1_60_0/libs/numeric/ublas/doc/index.html
55. Duff, Iain S.; Heroux, Michael A.; Pozo, Roldan (2002). "An Overview of the Sparse Basic Linear Algebra Subprograms: The New Standard from the BLAS Technical Forum". *TOMS*. 28 (2): 239–267. doi:10.1145/567806.567810.

- BLAST Forum (21 August 2001), *Basic Linear Algebra Subprograms Technical (BLAST) Forum Standard*, Knoxville, TN: University of Tennessee
- Dodson, D. S.; Grimes, R. G. (1982), "Remark on algorithm 539: Basic Linear Algebra Subprograms for Fortran usage", *ACM Trans. Math. Softw.*, 8: 403–404, doi:10.1145/356012.356020
- Dodson, D. S. (1983), "Corrigendum: Remark on "Algorithm 539: Basic Linear Algebra Subroutines for FORTRAN usage" ", *ACM Trans. Math. Softw.*, 9: 140, doi:10.1145/356022.356032
- 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 (1988), pp. 18–32.
- 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 (1990), pp. 1–17.
- 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 (1990), pp. 18–28.

New BLAS

- 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, An Updated Set of Basic Linear Algebra Subprograms (BLAS), ACM Trans. Math. Softw., 28-2 (2002), pp. 135–151.
- J. Dongarra, Basic Linear Algebra Subprograms Technical Forum Standard, International Journal of High Performance Applications

and Supercomputing, 16(1) (2002), pp. 1–111, and International Journal of High Performance Applications and Supercomputing, 16(2) (2002), pp. 115–199.

External links

- BLAS homepage (http://www.netlib.org/blas/) on Netlib.org
- BLAS FAQ (http://www.netlib.org/blas/faq.html)
- BLAS Quick Reference Guide (http://www.netlib.org/lapack/lug/node145.html) from LAPACK Users' Guide
- Lawson Oral History (https://web.archive.org/web/20061009230911/http://history.siam.org/oralhistories/lawson.htm) One of the original authors of the BLAS discusses its creation in an oral history interview. Charles L. Lawson Oral history interview by Thomas Haigh, 6 and 7 November 2004, San Clemente, California. Society for Industrial and Applied Mathematics, Philadelphia, PA.
- Dongarra Oral History (http://history.siam.org/oralhistories/dongarra.htm) In an oral history interview, Jack Dongarra explores the early relationship of BLAS to LINPACK, the creation of higher level BLAS versions for new architectures, and his later work on the ATLAS system to automatically optimize BLAS for particular machines. Jack Dongarra, Oral history interview by Thomas Haigh, 26 April 2005, University of Tennessee, Knoxville TN. Society for Industrial and Applied Mathematics, Philadelphia, PA
- How does BLAS get such extreme performance? (https://stackoverflow.com/questions/1303182/how-does-blas-get-such-extreme-performance) Ten naive 1000×1000 matrix multiplications (10^{10} floating point multiply-adds) takes 15.77 seconds on 2.6 GHz processor; BLAS implementation takes 1.32 seconds.
- An Overview of the Sparse Basic Linear Algebra Subprograms: The New Standard from the BLAS Technical Forum [2] (https://dx.doi.org/10.1145/567806.567810)