Basic Linear Algebra Subprograms

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 ("CBLAS interface") and Fortran ("BLAS interface"). 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	3.11.0 / 11 November 2022
Written in	depends on implementation
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 is in the public domain.[3][4]

Most computing libraries that offer linear algebra routines conform to common BLAS user interface command structures, thus queries to those libraries (and the associated results) are often portable between BLAS library branches, such as cuBLAS (NVIDIA GPU, GPGPU), rocBLAS (AMD GPU, GPGP), and OpenBLAS. This interoperability is then the basis of functioning homogenous code implementations between heterzygous cascades of computing architectures (such as those found in some advanced clustering implementations). Examples of CPU-based BLAS library branches include: OpenBLAS, BLIS (BLAS-like Library Instantiation Software), Arm Performance Libraries,[5] ATLAS, and Intel Math Kernel Library (iMKL). AMD maintains a fork of BLIS that is optimized for the AMD platform, although it is unclear whether integrated ombudsmen resources are present in that particular software-hardware implementation.[6] ATLAS is a portable library that automatically optimizes itself for an arbitrary architecture. iMKL is a freeware[7] and proprietary[8] vendor library optimized for x86 and x86-64 with a performance emphasis on Intel processors.[9] OpenBLAS is an open-source library that is hand-optimized for many of the popular architectures. 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 LAPACK, LINPACK, Armadillo, GNU Octave, Mathematica,[10] MATLAB,[11] NumPy,[12] R, Julia and Lisp-Stat.

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).[13] 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 needed 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).[14] Between 1973 and 1977, several of these kernel operations were identified.[15] 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.[16] 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.[17]

The original BLAS concerned only densely stored vectors and matrices. Further extensions to BLAS, such as for sparse matrices, have been addressed.[18]

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.[19] 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", "a x plus y") and several other operations.

Level 2

This level contains *matrix-vector operations* including, among other things, a *ge*neralized *m*atrix-*v*ector 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.[20] 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."[20]

Level 3

This level, formally published in 1990,[19] 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 computing

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

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,[21] 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.[22] 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.[23] A highly tuned implementation based on these ideas is part of the GotoBLAS, OpenBLAS and BLIS.

A common variation of `gemm` is the `gemm3m`, which calculates a complex product using "three real matrix multiplications and five real matrix additions instead of the conventional four real matrix multiplications and two real matrix additions", an algorithm similar to Strassen algorithm first described by Peter Ungar.[24]

Implementations

Accelerate
: Apple's framework for macOS and iOS, which includes tuned versions of BLAS and LAPACK.[25][26]

Arm Performance Libraries
: Arm Performance Libraries, supporting Arm 64-bit AArch64-based processors, available from Arm.[5]

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. Optimized for most modern CPUs. BLIS is a complete refactoring of the GotoBLAS that reduces the amount of code that must be written for a given platform.[28][29]

C++ AMP BLAS
: The C++ AMP BLAS Library is an open source implementation of BLAS for Microsoft's AMP language extension for Visual C++.[30]

cuBLAS
: Optimized BLAS for NVIDIA based GPU cards, requiring few additional library calls.[31]

NVBLAS
: Optimized BLAS for NVIDIA based GPU cards, providing only Level 3 functions, but as direct drop-in replacement for other BLAS libraries.[32]

clBLAS
: An OpenCL implementation of BLAS by AMD. Part of the AMD Compute Libraries.[33]

clBLAST
: A tuned OpenCL implementation of most of the BLAS api.[34]

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.

ESSL
: IBM's Engineering and Scientific Subroutine Library, supporting the PowerPC architecture under AIX and Linux.[35]

GotoBLAS
: Kazushige Goto's BSD-licensed implementation of BLAS, tuned in particular for Intel Nehalem/Atom, VIA Nanoprocessor, AMD Opteron.[36]

GNU Scientific Library
: Multi-platform implementation of many numerical routines. Contains a CBLAS interface.

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

Intel MKL
: The Intel Math Kernel Library, supporting x86 32-bits and 64-bits, available free from Intel.[7] Includes optimizations for Intel Pentium, Core and Intel Xeon CPUs and Intel Xeon

Phi; support for Linux, Windows and macOS.[37]

MathKeisan
: NEC's math library, supporting NEC SX architecture under SUPER-UX, and Itanium under Linux[38]

Netlib BLAS
: The official reference implementation on Netlib, written in Fortran 77.[39]

Netlib CBLAS
: Reference C interface to the BLAS. It is also possible (and popular) to call the Fortran BLAS from C.[40]

OpenBLAS
: Optimized BLAS based on GotoBLAS, supporting x86, x86-64, MIPS and ARM processors.[41]

PDLIB/SX
: NEC's Public Domain Mathematical Library for the NEC SX-4 system.[42]

rocBLAS
: Implementation that runs on AMD GPUs via ROCm.[43]

SCSL
: SGI's Scientific Computing Software Library contains BLAS and LAPACK implementations for SGI's Irix workstations.[44]

Sun Performance Library
: Optimized BLAS and LAPACK for SPARC, Core and AMD64 architectures under Solaris 8, 9, and 10 as well as Linux.[45]

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.[46]

Libraries using 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.[47]

LAPACK
: LAPACK is a higher level Linear Algebra library built upon BLAS. Like BLAS, a reference implementation exists, but many alternatives like libFlame and MKL exist.

Mir
: An LLVM-accelerated generic numerical library for science and machine learning written in D. It provides generic linear algebra subprograms (GLAS). It can be built on a CBLAS implementation.[48]

Similar libraries (not compatible with BLAS)

Elemental
: Elemental is an open source software for distributed-memory dense and sparse-direct linear algebra and optimization.[49]

HASEM
: is a C++ template library, being able to solve linear equations and to compute eigenvalues. It is licensed under BSD License.[50]

LAMA

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

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.

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 was finally standardized in 2002.[51]

Batched BLAS

The traditional BLAS functions have been also ported to architectures that support large amounts of parallelism such as GPUs. Here, the traditional BLAS functions provide typically good performance for large matrices. However, when computing e.g., matrix-matrix-products of many small matrices by using the GEMM routine, those architectures show significant performance losses. To address this issue, in 2017 a batched version of the BLAS function has been specified.[52]

Taking the GEMM routine from above as an example, the batched version performs the following computation simultaneously for many matrices:

$$\boldsymbol{C}[k] \leftarrow \alpha \boldsymbol{A}[k]\boldsymbol{B}[k] + \beta \boldsymbol{C}[k] \quad \forall k$$

The index k in square brackets indicates that the operation is performed for all matrices k in a stack. Often, this operation is implemented for a strided batched memory layout where all matrices follow concatenated in the arrays $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$.

Batched BLAS functions can be a versatile tool and allow e.g. a fast implementation of exponential integrators and Magnus integrators that handle long integration periods with many time steps.[53] Here, the matrix exponentiation, the computationally expensive part of the integration, can be implemented in parallel for all time-steps by using Batched BLAS functions.

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** (3): 308–323. doi:10.1145/355841.355847 (https://doi.org/10.1145%2F355841.355847). hdl:2060/19780018835 (https://hdl.handle.net/2060%2F19780018835). S2CID 6585321 (https://api.semanticscholar.org/CorpusID:6585321). Algorithm 539.

2. "BLAS Technical Forum" (http://netlib.org/blas/blast-forum). *netlib.org*. Retrieved 2017-07-07.
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) 2016-10-12 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" (http://www.netlib.org/utk/people/JackDongarra/PAPERS/netlib-history6.pdf) (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. "Arm Performance Libraries" (https://www.arm.com/products/development-tools/server-and-hpc/allinea-studio/performance-libraries). Arm. 2020. Retrieved 2020-12-16.
6. "BLAS Library" (https://developer.amd.com/amd-aocl/blas-library/).
7. "No Cost Options for Intel Math Kernel Library (MKL), Support yourself, Royalty-Free" (http://software.intel.com/articles/free_mkl). Intel. 2015. Retrieved 2015-08-31.
8. "Intel Math Kernel Library (Intel MKL)" (http://software.intel.com/intel-mkl). Intel. 2015. Retrieved 2015-08-25.
9. "Optimization Notice" (http://software.intel.com/articles/optimization-notice). Intel. 2012. Retrieved 2013-04-10.
10. Douglas Quinney (2003). "So what's new in Mathematica 5.0?" (https://web.archive.org/web/20131029204826/http://78.158.56.101/archive/msor/headocs/34mathematica5.pdf) (PDF). *MSOR Connections*. The Higher Education Academy. **3** (4). Archived from the original (http://78.158.56.101/archive/msor/headocs/34mathematica5.pdf) (PDF) on 2013-10-29.
11. Cleve Moler (2000). "MATLAB Incorporates LAPACK" (http://www.mathworks.com/company/newsletters/articles/matlab-incorporates-lapack.html). MathWorks. Retrieved 2013-10-26.
12. Stéfan van der Walt; S. Chris Colbert & Gaël Varoquaux (2011). "The NumPy array: a structure for efficient numerical computation". *Computing in Science and Engineering*. **13** (2): 22–30. arXiv:1102.1523 (https://arxiv.org/abs/1102.1523). Bibcode:2011CSE....13b..22V (https://ui.adsabs.harvard.edu/abs/2011CSE....13b..22V). doi:10.1109/MCSE.2011.37 (https://doi.org/10.1109%2FMCSE.2011.37). S2CID 16907816 (https://api.semanticscholar.org/CorpusID:16907816).
13. Boisvert, Ronald F. (2000). "Mathematical software: past, present, and future". *Mathematics and Computers in Simulation*. **54** (4–5): 227–241. arXiv:cs/0004004 (https://arxiv.org/abs/cs/0004004). Bibcode:2000cs.....4004B (https://ui.adsabs.harvard.edu/abs/2000cs.....4004B). doi:10.1016/S0378-4754(00)00185-3 (https://doi.org/10.1016%2FS0378-4754%2800%2900185-3). S2CID 15157725 (https://api.semanticscholar.org/CorpusID:15157725).
14. 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.
15. BLAST Forum 2001, p. 1.
16. Lawson et al. 1979.
17. BLAST Forum 2001, pp. 1–2.
18. BLAST Forum 2001, p. 2.

19. Dongarra, Jack J.; Du Croz, Jeremy; Hammarling, Sven; Duff, Iain S. (1990). "A set of level 3 basic linear algebra subprograms" (https://doi.org/10.1145%2F77626.79170). *ACM Transactions on Mathematical Software*. **16** (1): 1–17. doi:10.1145/77626.79170 (https://doi.org/10.1145%2F77626.79170). ISSN 0098-3500 (https://www.worldcat.org/issn/0098-3500). S2CID 52873593 (https://api.semanticscholar.org/CorpusID:52873593).
20. 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. CiteSeerX 10.1.1.17.5421 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.17.5421). doi:10.1145/42288.42291 (https://doi.org/10.1145%2F42288.42291). S2CID 3579623 (https://api.semanticscholar.org/CorpusID:3579623).
21. Goto, Kazushige; van de Geijn, Robert A. (2008). "High-performance implementation of the level-3 BLAS" (ftp://ftp.cs.utexas.edu/pub/techreports/tr06-23.pdf) (PDF). *ACM Transactions on Mathematical Software*. **35** (1): 1–14. doi:10.1145/1377603.1377607 (https://doi.org/10.1145%2F1377603.1377607). S2CID 14722514 (https://api.semanticscholar.org/CorpusID:14722514).
22. Golub, Gene H.; Van Loan, Charles F. (1996), *Matrix Computations* (3rd ed.), Johns Hopkins, ISBN 978-0-8018-5414-9
23. Goto, Kazushige; van de Geijn, Robert A. (2008). "Anatomy of High-Performance Matrix Multiplication". *ACM Transactions on Mathematical Software*. **34** (3): 12:1–12:25. CiteSeerX 10.1.1.111.3873 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.111.3873). doi:10.1145/1356052.1356053 (https://doi.org/10.1145%2F1356052.1356053). ISSN 0098-3500 (https://www.worldcat.org/issn/0098-3500). S2CID 9359223 (https://api.semanticscholar.org/CorpusID:9359223). (25 pages) [1] (http://www.cs.utexas.edu/~flame/web/FLAMEPublications.html#Goto)
24. Van Zee, Field G.; Smith, Tyler M. (2017-07-24). "Implementing High-performance Complex Matrix Multiplication via the 3m and 4m Methods". *ACM Transactions on Mathematical Software*. **44** (1): 1–36. doi:10.1145/3086466 (https://doi.org/10.1145%2F3086466). S2CID 25580883 (https://api.semanticscholar.org/CorpusID:25580883).
25. "Guides and Sample Code" (https://developer.apple.com/library/mac/#releasenotes/Performance/RN-vecLib/). *developer.apple.com*. Retrieved 2017-07-07.
26. "Guides and Sample Code" (https://developer.apple.com/library/ios/#documentation/Accelerate/Reference/AccelerateFWRef/). *developer.apple.com*. Retrieved 2017-07-07.
27. "Automatically Tuned Linear Algebra Software (ATLAS)" (http://math-atlas.sourceforge.net/). *math-atlas.sourceforge.net*. Retrieved 2017-07-07.
28. *blis: BLAS-like Library Instantiation Software Framework* (https://github.com/flame/blis), flame, 2017-06-30, retrieved 2017-07-07
29. *BLIS GitHub Repository* (https://github.com/flame/blis), 2021-10-15
30. "C++ AMP BLAS Library" (https://web.archive.org/web/20170708151515/http://ampblas.codeplex.com/). *CodePlex*. Archived from the original (http://ampblas.codeplex.com/) on 2017-07-08. Retrieved 2017-07-07.
31. "cuBLAS" (http://developer.nvidia.com/cublas). *NVIDIA Developer*. 2013-07-29. Retrieved 2017-07-07.
32. "NVBLAS" (https://docs.nvidia.com/cuda/nvblas/index.htmls). *NVIDIA Developer*. 2018-05-15. Retrieved 2018-05-15.
33. *clBLAS: a software library containing BLAS functions written in OpenCL* (https://github.com/clMathLibraries/clBLAS), clMathLibraries, 2017-07-03, retrieved 2017-07-07
34. Nugteren, Cedric (2017-07-05), *CLBlast: Tuned OpenCL BLAS* (https://github.com/CNugteren/CLBlast), retrieved 2017-07-07
35. *IBM Knowledge Centre: Engineering and Scientific Subroutine Library* (https://www.ibm.com/support/knowledgecenter/en/SSFHY8/essl_welcome.html)

36. Milfeld, Kent. "GotoBLAS2" (http://www.tacc.utexas.edu/tacc-software/gotoblas2). Texas Advanced Computing Center. Archived (https://web.archive.org/web/20200323172521/https://www.tacc.utexas.edu/research-development/tacc-software/gotoblas2) from the original on 2020-03-23. Retrieved 2013-08-28.
37. "Intel Math Kernel Library (Intel MKL) | Intel Software" (http://software.intel.com/en-us/intel-mkl/). *software.intel.com*. Retrieved 2017-07-07.
38. Mathkeisan, NEC. "MathKeisan" (http://www.mathkeisan.com/). *www.mathkeisan.com*. Retrieved 2017-07-07.
39. "BLAS (Basic Linear Algebra Subprograms)" (http://www.netlib.org/blas/). *www.netlib.org*. Retrieved 2017-07-07.
40. "BLAS (Basic Linear Algebra Subprograms)" (http://www.netlib.org/blas). *www.netlib.org*. Retrieved 2017-07-07.
41. "OpenBLAS : An optimized BLAS library" (http://www.openblas.net/). *www.openblas.net*. Retrieved 2017-07-07.
42. "PDLIB/SX: Business Solution | NEC" (https://web.archive.org/web/20070222154031/http://www.nec.co.jp/hpc/mediator/sxm_e/software/61.html). Archived from the original (http://www.nec.co.jp/hpc/mediator/sxm_e/software/61.html) on 2007-02-22. Retrieved 2007-05-20.
43. "rocBLAS" (https://rocmdocs.amd.com/en/latest/ROCm_Tools/rocblas.html). *rocmdocs.amd.com*. Retrieved 2021-05-21.
44. "SGI - SCSL Scientific Library: Home Page" (https://web.archive.org/web/20070513173030/http://www.sgi.com/products/software/scsl.html). Archived from the original (http://www.sgi.com/products/software/scsl.html) on 2007-05-13. Retrieved 2007-05-20.
45. "Oracle Developer Studio" (http://www.oracle.com/technetwork/server-storage/solarisstudio/overview/index.html). *www.oracle.com*. Retrieved 2017-07-07.
46. "Boost Basic Linear Algebra - 1.60.0" (http://www.boost.org/doc/libs/1_60_0/libs/numeric/ublas/doc/index.html). *www.boost.org*. Retrieved 2017-07-07.
47. "Armadillo: C++ linear algebra library" (http://arma.sourceforge.net/). *arma.sourceforge.net*. Retrieved 2017-07-07.
48. "Dlang Numerical and System Libraries" (https://github.com/libmir). *GitHub*.
49. "Elemental: distributed-memory dense and sparse-direct linear algebra and optimization — Elemental" (http://libelemental.org/). *libelemental.org*. Retrieved 2017-07-07.
50. "HASEM" (http://sourceforge.net/projects/hasem/). *SourceForge*. 2015-08-17. Retrieved 2017-07-07.
51. 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". *ACM Transactions on Mathematical Software*. **28** (2): 239–267. doi:10.1145/567806.567810 (https://doi.org/10.1145%2F567806.567810). S2CID 9411006 (https://api.semanticscholar.org/CorpusID:9411006).
52. Dongarra, Jack; Hammarling, Sven; Higham, Nicholas J.; Relton, Samuel D.; Valero-Lara, Pedro; Zounon, Mawussi (2017). "The Design and Performance of Batched BLAS on Modern High-Performance Computing Systems" (https://doi.org/10.1016%2Fj.procs.2017.05.138). *Procedia Computer Science*. **108**: 495–504. doi:10.1016/j.procs.2017.05.138 (https://doi.org/10.1016%2Fj.procs.2017.05.138). hdl:2117/106913 (https://hdl.handle.net/2117%2F106913).
53. Herb, Konstantin; Welter, Pol (2022). "Parallel time integration using Batched BLAS (Basic Linear Algebra Subprograms) routines". *Computer Physics Communications*. **270**: 108181. arXiv:2108.07126 (https://arxiv.org/abs/2108.07126). doi:10.1016/j.cpc.2021.108181 (https://doi.org/10.1016%2Fj.cpc.2021.108181). S2CID 237091802 (https://api.semanticscholar.org/CorpusID:237091802).

Further reading

- BLAST Forum (2001-08-21), *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** (4): 403–404, doi:10.1145/356012.356020 (https://doi.org/10.1145%2F356012.356020), S2CID 43081631 (https://api.semanticscholar.org/CorpusID:43081631)
- 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 (https://doi.org/10.1145%2F356022.356032), S2CID 22163977 (https://api.semanticscholar.org/CorpusID:22163977)
- 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 (https://web.archive.org/web/20061009230904/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://doi.org/10.1145%2F567806.567810)
