[LAWN 290]
2016 Dense Linear Algebra Software Packages Survey

Jack Dongarra[1], Jim Demmel[2], Julien Langou[3], Julie Langou[1]

1. University of Tennessee Knoxville, USA
2. UC Berkeley, USA
3. UC Denver, USA

Summary

About the Survey

The 2016 Dense Linear Algebra Software Packages Survey was administered from January 1st 2016 to April 12 2016. 234 respondents answered the survey. The survey was advertised directly to the Linear Algebra community via our LAPACK/ScaLAPACK forum, NA Digest and we also directly contacted vendors and linear algebra experts. The breakdown of respondents was: 74% researchers or scientists, 25% were Principal Investigators and 25% Software maintainers or System administrators.

The goal of the survey was to get the Linear Algebra community opinion and provide input on dense linear algebra software packages, in particular LAPACK, ScaLAPACK, PLASMA and MAGMA. The ultimate purpose of the survey was to improve these libraries to benefit our user community. The survey would allow the team to prioritize the many possible improvements that could be done. We also asked input from users accessing these libraries via 3rd party interfaces, for example MATLAB, Intel's MKL, Python's NumPy, AMD's ACML, and many others.

The survey was composed of six parts:

1. A general section about user's applications and their needs.
2. Specific questions about your LAPACK and its uses.
3. Specific questions about your ScaLAPACK and its uses.
4. Specific questions about your PLASMA and its uses.
5. Specific questions about your MAGMA and its uses.
6. An open section for any additional comments.

Survey link: https://www.surveymonkey.com/r/2016DenseLinearAlgebra
Survey response: https://www.surveymonkey.com/results/SM-J68KNV8Q/summary

Acknowledgment

The 2016 Dense Linear Algebra Software Packages Survey was supported by the NSF ACI (grant numbers 1339676, 1339797, 1339822).

Linear Algebra section – 252 respondents

Usage

80% of respondents recognize linear algebra as an important/dominant part of their application. The top three domains being Physics (58%), Computer science (45%), and Mathematics (44%); applications vary from Combustion chemistry to Computational Electromagnetics, Aerospace Engineering, Climate modeling, Biology , or Optics, Nanophotonics .

Linear Algebra packages are mostly run on multicore machines (over 80%) and Sequential, Distributed CPU and hybrid are between 45% and 53%. Each user is using on average two different architectures to run their applications. It is worth noting that some respondents are also trying at solving Linear Algebra problems on mobile and/or low-powered architecture.

Linux is the primary choice (95%) when it comes to running a Linear Algebra package. Mac OS X comes second with 32% and Windows third with 22%. Again, some users mentioned the use of mobile OS's like iOS or Android.

The utilization rate of LAPACK is vey high at 87%. Indeed, 99.55% of our respondents stating they know the LAPACK package, and 89% claiming the use of it. ScaLAPACK has an utilization rate of around 40%. It is worth noting that MAGMA is clearly gaining momentum within the community with a utilization rate at 22%.

While a large part of those respondents use a vendor library with INTEL's MKL being their first choice because of its ease of use, its integration with Xeon Phi (MIC) accelerators. The second was Eigen – chosen due to the ease of Matlab-like programming style, and the ease of integration with C++ and NVIDIA cuBLAS for its performance. Ease of use being a major factor, LIBFLAME and ELEMENTAL were also mentioned.

It is worth noting that almost 20% of our respondents still require writing their own linear algebra codes due to their unique needs or due to hand tuning of individual routines for performance. Also interesting to mention that 36% of our respondents need to solve many independent problems at the same time (64% need one at a time).

Need

Our respondents indicate as General Dense, positive definite, symmetric or hermitian the top three dominant structures. More than half of our respondents are using all those three structures. Nonetheless there is also a plethora of other sparsity or other mathematical structures mentioned. It is worth noting that 7% of respondents mentioned an unknown structure. It was also mentioned more than once that dense solvers are sometimes used in place of special solvers since they are faster to implement.

In terms of functions performed, 83% of our respondents mentioned Solving linear systems of equations and more than half solving symmetric eigenvalue problems.

Regarding dimension their problems are for the most part - 61% O(1000). Overall, we can see a wide spectrum of answer from 19% with 0(10) problems and 28% with over O(10000).

Autotuning is a significant need from the linear algebra community – over 50% of respondents would you like an interface that performed "on-line autotuning" when possible, among those almost 20% listed it as a pressing need. Most of our respondents listed 10% overhead or less as a decent overhead for their application while wanting to have a special tool to analyze their matrix and/or an initial run to be able to control autotuning. This question generated the most comments in our survey (90).

Reproducibility, i.e., being able to get bit-wise identical answers from repeated calls to the same routine with the same inputs on the same platform, is important to more than 75% of our respondent. Reproducibility is often listed as more important in the development phase (debugging, testing, validation) than in Production environment. To have a way to enable or disable reproducibility is what most respondent would like to see.

To handle floating-point exceptions is also major need of our respondents. 60% of them are expressing a need to handle floating-point exceptions. Some suggestions include an on/off flag to enable functionality, a overhead between 5% and 10%, and a notification flag if floating point exceptions are found. Like reproducibility handling, floating point exceptions is often listed as more important in the development phase (debugging, testing, validation) than in Production environment.

LAPACK – 186 respondents

Ease of Use

Over three-fourths of respondents mentioned that LAPACK interface is easy to use. Less than 10% of respondents mentioned major roadblocks preventing them from using LAPACK. The main roadblocks are the lack of cross-platform support and a non-native C package.

Install

Most of our respondents are installing LAPACK via a vendor package (66%) or from a platform distribution package (41%) . Still, 41% are using the makefile install and 10% using the CMAKE install from the netlib package.

The respondents emphasize the ease of use of the installation procedure of package with an 88% response rate. The only issues mentioned are related to the Windows platform or the lack of parallel build capability.

Documentation

Our respondents indicate that they are using mostly the HTML Pages (57%) and the LAPACK User Guide (54%) for their documentation need. It is worth noting that 10% are using our LAPACK/ScaLAPACK forum as a source of documentation, and some users are using vendor documentation such as the one from Intel MKL.

Almost all of of our respondents (95%) indicate that the documentation of LAPACK is suitable for their needs.

Improvement

The top 3 interface improvements our respondents would like to see are: an automatic memory allocation (66%), the use of optional arguments to return more detailed information on request (47%), and Quickly explored your input matrix to try to automatically identify the best algorithm to use (e.g. by testing for symmetry or sparsity) 41%.

Our respondents are wishing for LAPACK to add the following dense linear algebra functionality.

Algorithm

- Level 2 AXPY (i.e., a X + b Y for matrices) general congruence update of the form B <- X**T A X or B <- X**H A X skew-symmetric BLAS
- Skew-symmetric linear solvers
- Skew-symmetric eigensolvers
- Symmetric and nonsymmetric matrix
- Exponential computing a subset of singular values
- Symmetric eigensolvers using Jacobi algorithm Sylvester/Lyapunov linear solver
- Level 3 accumulated application of Givens rotation in several subroutines
- Matrix logarithm A general matrix function subroutine
- Eigenvalues for banded matrices, eigenvalues for general matrix.
- QR for mixed precision (DS)
- Symmetric DGEMM
- Compute the first N largest singular values of a matrix, where N << matrix size. Rank revealing QR that terminates once rank N is reached, where N << matrix size.
- Perhaps this is more BLAS than LAPACK, but multiplication of two upper (or lower) triangular matrices.
- What I would really like is a generalized eigensolver that does inverse iteration with pivoting, but there is none in LAPACK as a GSEP in quadruple precision. I have been able to compile LAPACK in qp, but what I really need is scalapack in qp. The C code in ScaLaPack effectively prevents me from using the ifort -r16 flag to compile the version of Scalapack that does exist, pssygvx, with the automatic promotion of real to real*16. Even if I could, it wouldn't be II with pivoting.
- Methods to handle indefinite symmetric generalized eigenvalue problems.
- More routines to handle complex symmetric cases.

- Utility routines to form symmetric quadratic matrix products from nonsymmetric factors (like A'*A, or A'*P*A for Hermitian P).
- Eigenvalue routines that allow specifying custom shift strategies or initial shift guesses.
- Routines for quadratic eigenvalue problems, palindromic problems.
- Improved eigenvector extraction after Schur decomposition.
- Better respect for const-correctness (why does xLARFT need to modify and then unmodify an argument?). Related, better respect for thread-safety (remove giant stack arrays).
- Some functionality from SLICOT like Hamiltonian eigenvalue problems, product eigenvalue problems.
- Harder to get: improved MRRR, and dqds-like algorithms for non-symmetric problems!
- The QR with column pivoting could take a threshold epsilon, then stop the factorization when the epsilon-rank is reached.
- Rank-revealing QR might be nice, as would be the LR factorization without pivoting
- fast randomized algs, svd. matrix compression: ACA (adaptive cross-approx). ID (interpolative decomp).
- Mixed operations in BLAS. E.g. double x complex double.
- Beside having "normal", "transposed" and "conjugate transposed", an additional "conjugate" option for matrix operations would simplify things.
- Sparse matrices?
- Having a restricted threshold LDL^T factorization (similar to HSL_MA64) would be very useful.
- rank revealing QR
- Tall skinny QR, updating/downdating
- I would very much like a matrix transposition routine. I've looked and can't find one. We have several linear solvers. Some use single-threaded LAPACK calls, often called independently by different threads. Others, such as MUMPS, were designed for multithreaded LAPACK calls. This has proven to make software integration difficult. It would be nice if the API for routines like DGEMM would be extended, or perhaps there be a separate name (e.g., DGEMM_OMP), so that we could specify which one we want at any given time.

Performance

- Faster ZGGEV in the form of Bo Kagstrom's multishift QZ.
- Efficient QZ implementation Solvers for product eigenvalue problems

Capabilities:

- Micro kernel tuning
- Introducing (omp) threading and working close together with projects such as openblas.
- Shared memory MPI3 parallelization for small to medium sized problems.
- The ability to work with high precision would be very useful. It lets the user quickly check on a smaller problem how sensitive it is to limited precision. This is a feature in the Julia language already, and I use it heavily.
-

Ease of use

- Windows version of LAPACK

ScaLAPACK – 80 respondents

Usage

95% of respondents mentioned using Fortran, C or C++ or a combination of those. Those three languages are almost used in equal proportion (Fortran ~60%, C ~50%, C++ ~40%).

Ease of Use

52% of respondents mentioned that ScaLAPACK interface is NOT easy to use. One third of respondents mentioned major roadblocks preventing them from using ScaLAPACK. There are multiple roadblocks: lack of performance, difficulty understanding data distribution, build and install issues, lack of functionality, hard to understand documentation, lack of examples. Many mentioned using Elemental as a replacement.

Install

Most of our respondents are installing ScaLAPACK via a vendor package (63%) or from platform distribution package (31%) . Still, 46% are using the makefile install and 11% using the CMAKE install from the Netlib package, and only 5% the Python Installer.

The respondents emphasize the ease of use of the installation procedure of package with an 84% response rate. The only issues mentioned are the lack of parallel build capability and the sometimes hard to get it to work on cross-compiled platform (especially the testings). Respondents are acknowledging the great improvement in the build system with ScaLAPACK 2.0 with the change in the library structure and the addition of the cmake build.

Documentation

Our respondents indicate that they are using mostly the ScaLAPACK User Guide (77%), and routine's comments (51%) for their documentation need. Note that 16% use our LAPACK/ScaLAPACK forum as a source of documentation, and some users are using vendor documentation such as the one from Intel MKL.

84% of our respondents indicate that the documentation of LAPACK is suitable for their needs.

Improvement

The top 3 interface improvements our respondents would like to see are: an automatic memory allocation (64%), automatic conversion of input distributed matrix layout to a more efficient one, if that would speedup your code (52%), and the use of optional arguments to return more detailed information on request (39%). Many respondents mentioned the BLACS layer as being hard to understand and would like ScaLAPACK to use MPI communicators explicitly.

Our respondents wish for ScaLAPACK to add the following dense linear algebra functionality.

Algorithm

- Eigenvector computation for nonsymmetric eigenvalue problems.
- Quadruple precision generalized eigensolver in quadruple precision which does inverse iteration with pivoting.
- Sparse cholesky with a BSD-friendly license that I could use as an alternative to cholmod/suitesparse
- pdgeev and pzgeev
- QR with column pivoting could use a threshold to stop as soon as the rank is revealed. QR with column pivoting is slow, uses only BLAS1/2 in contrast to the LAPACK version dgeqp3. Here a randomized algorithm might be better. We are considering implementing one based on recent work by Duersch/Gu or Martinsson.
- Support for symmetric packed format. Not only positive definite matrices.
- Rank revealing QR
- Small matrix optimization

Performance

Good performance for pdgemm (pdsyrk) for a broader range of matrix sizes and nodes, in particular tall-and-skinny. Furthermore, it would be valuable if the research done on 2.5D multiplication would become available in the form of a better scaling pdgemm.

Testing

- ScaLAPACK subroutines are much less tested compared to those in LAPACK.

Documentation

- As the cyclic distribution is complicated, there are a lot of restrictions for almost all subroutines. But it is not quite easy to figure out all the restrictions.
- Better samples programs would be quite helpful.

Installation

- When I tried this last (maybe 5 years ago), ScaLAPACK was a complete disaster to install as it required shuffling around Makefile fragments. This may have been fixed in the meantime, but it was definitely not appropriate for the 2000s any more.

Code Improvement

- The quality of comments. From time to time I see typos in the comments; some of them can be misleading. But I don't want to contact the maintainer merely for fixing typos. The ScaLAPACK team should provide an easy and efficient way to encourage users to report typos. This should be something less official than bug reports/feature requests. Fixing typos is also much simpler than fixing bugs, and should be done in a timely manner.

Capabilities

- Possibility to specify a MPI communicator in which ScaLAPACK executes. Currently some strange hacks are required.
- A robust communicator for processors. (In current BLACS, we easily get in trouble if we create multiple levels of communicators, i.e., recursively creating new communicators using subset of procs).
- The ability to work easily with matrices defined on different communicators. Not having to deal with BLACS communicators. We use ScaLAPACK from C++, but do not use the C++ API (is it really supported ?). We write our own C++ Matrix classes encapsulating ScaLAPACK functions, which is clearly suboptimal (as we are not experts to do that).

Data layout

- 2D block-cyclic data layout fits some algorithms, but not all of them. If, e.g., one wants to develop a distributed-memory algorithm for which 1D block-column data distribution is a natural choice, it would fail to co-operate with the rest of ScaLAPACK.
- Routines to help automate the migration of data to the block structures needed
- Suppose, for example, I have a matrix of order > 100,000 written to a file. It is trivial to read this in an use an LAPACK routine to get eigenvalues, but if I wanted to avail myself of the much superior parallelization available in ScaLAPACK I have to put in considerable thought about how to distribute the data. Why can't some of this work be better automated, even if it means one would not get optimal performance?

PLASMA – 19 respondents

Usage

90% of respondents mentioned using Fortran, C or C++. C++ being the most widely used language (45%), followed by C ~36%, and Fortran ~27%.

Ease of Use

80% of respondents mentioned that PLASMA interface is easy to use. A majority of respondents mentioned major roadblocks preventing them from using PLASMA. The roadblocks are multiple: the huge number of dependencies, build and install process, the lack of community adoption, the lack of interoperability, lack of functionality, lack of reliability (examples crashing) , lack of performance. Many respondents are also stating they do not need it for their applications.

Install

Most of our respondents are installing PLASMA via Makefile (53%) and 24% are using the CMAKE install from the Netlib package, while 19% use the Python Installer.

The respondents emphasize the ease of use of the installation procedure of package with a 72% response rate

Documentation

Our respondents indicate that they are using mostly the PLASMA User Guide (69%), and routine's comments (31%), and webpages: Doxygen documentation (25%) for their documentation need. Note that 13% are using our PLASMA User forum as a source of documentation.

75% of our respondents indicate that the documentation of PLASMA is suitable for their needs.

Improvement

The top 3 interface improvements our respondents would like to see are: an automatic memory allocation (80%), Quickly explored your input matrix to try to automatically identify the best algorithm to use (e.g. by testing for symmetry or sparsity) 60%, and allowed user-defined data types (e.g. very high precision numbers) (40%)

Our respondents wished that PLASMA added the following dense linear algebra functionality.

- There is a HUGE effort needed to make the use of Plasma as easy and transparent to the end user as current LAPACK libraries. What end users want to know are things like: 1. Do I need to modify my application code in any way? If yes, how? Sidenote: I don't care about an example code and I don't want to become a world expert in GPU architectures, I only want to know about my application. 2. How do I compile the code? 3. How do I run the code? I suppose most of these complaints should probably be addressed to OLCF, but in spite of countless annual surveys, their documentation remains as useless as ever.
- Some kind of distributed parallel alternative would be useful, though, I figure this would be a completely different task.
- Automatic parameter tuning and complete mixed precision implementation

MAGMA – 42 respondents

Usage

81% of respondents mentioned using Fortran, C or C++ or a combination of those. C++ being the most widely used language (50%), followed by C ~25%, and CUDA ~15%.

Ease of Use

75% of respondents mentioned that MAGMA interface is easy to use. Two thirds of respondents mentioned major roadblocks preventing them from using MAGMA. The roadblocks mentioned are: lack of performance, build and install issues, lack of multi-process support, lack of examples . Many respondents are also stating they do not need it for their applications, but when they need it, they look at using CUBLAS instead.

Install

Most of our respondents install MAGMA via Makefile (77%) and 21% use the CMAKE install from the Netlib package.

The respondents emphasize the ease of use of the installation procedure of package with an 78% response rate.

Documentation

Our respondents indicate that they are using mostly the MAGMA User Guide (64%), and routine's comments (54%) for their documentation need. Note that 31% use our MAGMA User forum as a source of documentation.

84% of our respondents indicate that the documentation of MAGMA is suitable for their needs.

Improvement

The top 3 interface improvements our respondents would like to see are: an automatic memory allocation (60%), automatic conversion of input distributed matrix layout to a more efficient one, if that would speedup your code (20%), and the use of optional arguments to return more detailed information on request (20%). Interestingly, 32% of the answers were "others", citing interface with Eigen, and GPU aware interface.

Our respondents wish MAGMA to add the following dense linear algebra functionality.

Algorithm

- An option which can choose saving location of a pivot vector either in host or device memory space.
- Complete mixed precision implementation and facilities for row major order
- some more convenient stuff, e.g. a sprint_gpu.
- having direct sparse linear solver for GPU based on LU decomposition (CUDA has such a solver for GPU but only using QR decomposition which is probably slower).
- zheevd is limited by the amount of GPU memory available, currently limiting matrix orders to n~19,000 for 6GB of RAM, there must be a way to exceed this using a combination of CPU RAM and GPU RAM? Also for Hermitian matrices, space saving methods are apparently used for only storing the upper triangle, and yet the amount of memory used is the same (I could be wrong about this, but it seems so).

Installation

- Could not get MAGMA to run on Intel Xeon Phi.
- The installation process for MAGMA sets the number of cores automatically. It would be nice, for simulation, if there was an input argument to vary this number.
- LP64 version having problems with work area creation, due to the use of 32-bit int's (e.g. dsyevdx_mgpu).
- I have used MAGMA library from version 1.4.1. It takes too much time to build it on Windows operation system. I hope you to support pre-built binary files in each operation systems.
-

Code Improvement

- Header files having the correct const attributes

Capabilities

- I wish it is simple to hack the code and sometimes capture dependencies - A more flexible interface that also tells where to put the output (GPU or CPU or both at the same time) will be awesome. - An open-source distributed version of it (not like the CRAY accelerated ScaLAPACK that uses libsci_acc but something really portable within certain limits).
- Support for running on distributed memory systems, e.g., running over MPI. This is especially a problem on nodes that only have a single GPU on them, e.g., Titan, Piz Daint.
- Routines using accelerators only, basically a GPU native implementation.
- More operations running on GPU (even if not efficient) to avoid communication (e.g. SVD).

DETAILED SURVEY RESULTS

 # responses = 252

General Questions (at most 251 answers per question)

Q1- Are dense linear algebra operations important/dominant in your application? (251)		
Answer	Response Percent	Response Count
Yes	80.9%	203
No	8.0%	20
Please specify	11.2%	28

Q2 - Dominant Applications: (232)	
FEM	8
Quantum mechanics	8
optimizatio n	10
ML/data analysis	10
DFT/electronic structure	46
combustion chem	4
electromagnetics	7
protein	1
optics	2
nuclear structure	1
CFD	11
molecular dynamics	2
stat+econ	1
batched	1
total	112

Q3 - Architectures (see Q5 for details) (250)	
Sequential	46%
Multicore	82%
Distributed- CPU-only	50%
Distributed-Hybrid-CPU+accelerator	48%
Cloud (spark etc)	6%
Self hosted accelerator	15%
Other	8%

Q4 - OS used (248)	
Linux	95%
Other Unix-like	15%
MacOS	31%
Windows	23%
Others	4%

Q6 - Libraries known about (248)	
LAPACK	100%
ScaLAPACK	83%
PLASMA	47%
MAGMA	64%

Q7 - Libaries used (via 3rd party too), and why (249)		
LAPACK	89%	lack of Fortran compiler on embedded systems
ScaLAPACK	39%	
PLASMA	6%	
MAGMA	22%	
Write Own	20%	

Q8 - Details about other libraries (number using it, why) (65)		
CUBLAS	8	easier to use, beats MAGMA for SGETRF, SGETRS, batched versions
OpenBLAS/ATLAS	1	
SLICOT	1	some beat LAPACK
CULA	3	better error handling than Magma
Boost, GSL	2	intervals
Chameleon	1	use on various runtime systems
MKL	13	performance, more accurate
numpy	2	ease of use
Eigen	9	ease of use (like Matlab), Cmake support, easy with C++, generic types, heterogeneous environments (ios, android, …)
ViennaCL	1	
Flame	2	easy with C++
ELPA	3	beats ScaLAPACK
elemental	4	C++, has optimization, extended precision, don't like BLACS
ATLAS	2	
FlexiBLAS	1	generic interface, more routines.
SLICOT	1	
Harwell	2	
Blaze	1	modern interface
total	57	
fraction	25%	

Q9 - If "Write own", why? (52)

my operations don't map to BLAS/LAPACK
small rank updates for Cholesky, LU, QR
skew symmetric eigensolvers, matrix functions
hard to distribute LAPACK/MAGMA with Visual Studio
dynamic task submission & scheduling; iterative solvers
pdgemm + GPU. Need QR, SVD on multiple node/GPU
numpy, scipy not parallel
easy and clear bindings in C++
batched Cholesky in MAGMA slower than LU in cublas
need parallel SVD
need quadruple precision for electronic structure
tensors
faster tri- and pentadiagonal solvers
expensive ??
performance on small matrices
block sparse gemm
Gauss-Jordan inversion
scalable band solver, based on SPIKE
simultaneous Jacobi diagonalization of multiple A=A^T
exploit additional symmetry
many small gemv's or gemm's
inconvenient ScaLAPACK interface

LAPACK multithreading issues, warm-start nonsymeig QR
cula slow on my matrix sizes
multithreading issues (oversubscription)
need LU, LDL^T without pivoting
need more general storage format, for (sub)tensors
handle row-major order, to avoid copying
allow conjugation of input matrices (not just with transpose)
LDL^T with threshold pivoting on square submatrix of larger matrix
matching data structure of other legacy codes (unfortunately)
very high precision
uniformity of interface with sparse codes
too much overhead on small matrices

Q10 - Dominant matrix structures (240)	
General	66%
Pos Def	49%
Sym/Hermitian	62%
Complex Sym	25%
Band	26%
Other sparse	18%
Other math	13%
None/unknown	7%

Comments:
irregular sparsity
block structured
one pos eval, rest neg
J-Hermitian
block sparse/banded (3)
diagonally dominant tridiagonal
Toeplitz
Hamiltonian, sym wrt indef inner prods
hierarchical, semiseparable, block low rank
blockwise low rank
block Toeplitz
FMM-type

Q11 - Dominant functions (details in Q19) (234)	
Ax=b	83%
Least squares	35%
Symeig	52%
Nonsymeig	26%
SVD	42%
Gen Symeig	32%
Gen Nonsymeig	16%
Gen SVD	14%
Other low rank (QR, Chol w/pivot)	32%
updating/downdating	12%
Other factorizations	6%

ILU, ILUt
matrix completion
partial LDL^T
polar decomp, Takagi factor
URV
Gen Symeig with semidef matrices

Singular pencils (Bokg's code)
Interpolating Decomp (CX, CUR)
PARAFAC tensor decomp

Other functions 8%

exp(A) (7)
determinant
sqrt(A) (2)
sign(A)
log(A)

Q12 - Dominant data types (239)	
less than 32 bit	1.3%
32 bit real	18%

Q13 - Accuracy needs (234)	
Standard (small back error)	77%
Standard + error bounds	26%

32 bit complex	11%
64 bit real	86%
64 bit complex	48%
128 bit real	4%
128 bit complex	4%
more than 128 bit	2.1%

Higher accuracy	24%
Lower accuracy, if faster	17%

Comments

Need everything from high to 3 digits
Want low accuracy at start of iteration, then higher (2)
Standard + detect ill-conditioning
Sometimes need high precision
Quad (2)
Want deterministic ScaLAPACK eigensolver
Would like to trade speed/accuracy

Q14 - Problem dimensions (235)

O(10)	19%
O(100)	37%
O(1000)	61%
O(1000x10)	28%

Other

O(10^1)	1
O(10^2)	2
O(10^3)	6
O(10^4)	18
O(10^5)	22
O(10^6)	11
O(10^7)	3
O(10^8)	5
O(10^4x10^3)	1
O(10^5x10^2)	1
O(10^7x10^2)	1

Q15 - solve one problem at a time or many? (233)

One	64%
Many	36%

Q16 - Want autotuning? (230)

Very very much	18%
Yes	36%
Why not	25%
No	20%

Comments:

Many opinions: willing to pay 5%-10% for autotuning, as low as 1%. Want control, be able to choose not to autotune. If tune, save results for later calls. Suggestion that tuning be handled by python, matlab etc at high level. Don't autotune for structure (eg symmetry) since this is known by user. Can let autotuner make suggestion, let user choose.

Q17 - How important is reproducibility? (235)

Critical	15%
Very important	30%
Important	33%
Not important	23%

Comments

They reflect the range of opinions above, and the discussions at recent workshops (important for debugging, sometimes contractually required, some would settle for variations just in trailing digits)

Q18 - How important is exception handling? (237)

Very important	19%
Important	47%
Not important	30%
N/A	4%

Comments

They reflect the range of opinions above.
Many responders said how much slowdown they would tolerate for exception handling:

0%	3
<= 1%	2
5%	13
10%	21
15%	1
20-25%	3

50%	2
100%	3

Many said to make it optional.

LAPACK Specific Questions (at most 186 answers per question)

Q19 - Which routines do you mostly use? (162)

long list, reflects Q11

Q20 - from which language do you call LAPACK?(169)

Fortran (77/90/95/03/08)	85
C	59
C++	77
Python/Numpy	19
Julia	6
R	4
Matlab	10
Octave	2
Cuda	1
PETSc	1
Haskell	1
Eigen	1

Q21 - If considered LAPACK but decided not, why? (21)

Hard in Visual Studio

Eigen/ViennaCL has better interface

Use CLAPACK from LAPACK 3.2, since our platforms do not have Fortran compilers

Want gensymeig that does invit with pivoting

Difficulty of cross-platform support

Use MAGMA

MKL licensing

C and C++ interfaces not easy to use

Need pentadiagonal solver with precomputed LU

Hard to call from C++

Hard with Windows, threading problems (Errata)

Employer didn't want added dependency

Dealing with 32/64 bit ints between C/Fortran

Want LDL^T with threshold pivoting

Q22 - LAPACK Interface easy to use? (186)

Yes	77%
No	23%

Q23 - Using LAPACKE? (179)

Yes	16%	14%
No	84%	86%

Q24: If you prefer a simpler interface, what?(136)

Allocate workspace automatically	68%
Optional args. to return info on request	45%
Allow user data types (eg high precision)	30%
Autotune algorithm	41%
Other (please specify)	17%
Prefer full control for myself	
Hard to install	
Object oriented interface	

Q25 - Which LAPACK documentation do you use?(182)

LAPACK User Guide	55%
LAPACK Working Notes	13%
HTML page	56%
Man page	15%
Sca/LAPACK User Forum	10%
Routine's comments	42%
Other	13%
old html better than doxygen	

more readable API
Optional automatic memory allocation
hard to remember names in API
C++14 interface
Examples of use in documentation
Interface for shared memory MPI3
Templates for different types
Integers should have size_t as in C/C++
Workspace queries returned at int, overflow in single
Arbitrary precision
If allocate memory internally, optional callback to
use my own memory allocator
Use Structs/union to handle matrix metadata
Optional args good, but might encourage spaghetti
Thread-safety, multi-precision, no logical args
Simpler/encapsulation specification of matrices (2)
Not optional arguments, in C or Fortran

Intel MKL documentation (5)
NAG
google (source code comes up) (6)
internet
cuBLAS site
Stack Overflow
LAPACK source (2)
Python

Q26 - Documentation good enough?(180)

Yes	95%
No	5%

Comments

Visual Studio hard to use
not interactive, look at cppreference.com

more examples (working, on-line)
doxygen pages slow
more comments, references to papers in code
LWORK in eigensolvers tricky

Q27 - How do you install LAPACK?(190)

Vendor package	64%
Platform distribution system (Debian etc)	42%
Makefile	41%
CMAKE	10%
Other	11%

own build system (2)
MKL (3)
cluster installation
Python
OpenBLAS (2)
FEniCS
Matlab
CMAKE is garbage
Bake

Q28 - LAPACK installation ok, or to be improved? (172)

Yes	90%
No	10%

Comments
Testing routines can crash during installation
Visual Studio hard
Too long
Makefile good, avoid scripting languages
Windows hard
What about android?
configure; make; make install would be better
parallel build (make -j32) should work
On Stampede, want "module add" to work
Version for Cygwin/mingw Windows

Q29 - Missing functionality, other comments? (41)

mixed real-complex functions (2)
small rank updates
rank-revealing Chol and LU

Avoid modifying inputs if possible, for multithreading
Level 2 axpy: a*X+b*Y
B = X^T*A*X
skew symmetric solvers
exp(A)
sylvester/Lyapunov solver (but dtrsyl? Just triangular)
symeig using Jacobi (have SVD)
Level 3 accumulated Givens rotations
efficient QZ(A*B)
log(A)
func(A) (presumably given pointer to scalar func(A))
micro kernel tuning a la ATLAS
subset of singular value (have it as of v3.6)
multiply two triangular matrices
better Windows version (not old f2c'd version)
symmetric DGEMM (not SYMM?)
mixed precision QR (?)
quad precision
Gennonsymeig using invit with pivoting
eig(band)
more of LAPACK in PLASMA/MAGMA
shared mem MPI3 parallelization for small/medium
faster ggev using Kagstrom's multishift QZ
gen eig of indef symmetric
A^T*A or A^T*B^A where B = B^T, Hermitian too
quadratic & palidromic eigenproblems
faster evec(Schur form)
better thread safety (LARFT modifies/restored input)
QRCP with threshold to stop
RRQR (2)
Handle row-major order (LAPACKE copies, too expen.)
high precision
fast randomized algs
Interpolative decomp (CX, CUR)
ACA (Adaptive cross-approximation)
allow conjugating inputs, not just conj-transpose
threshold LDL^T a la MC64
LU without pivoting
gemm(A*B*C*…) choosing best order
TSQR (got it!)
updating/downdating

matrix transpose

ScaLAPACK Specific Questions (at most 80 answers per question)

Q30 - Which routines do you mostly use? (66)

long list, reflects Q11

Q31 - from what do you call ScaLAPACK?(73)

Fortran (77/90/95/03/08)	41
C	19
C++	25
Python/Numpy	1
Julia	0
R	1
Matlab	0
Octave	0
Cuda	0
PETSc	1
Haskell	0
Eigen	0

Q32 - If considered ScaLAPACK but didn't use it, why? (18)

EigenExa, ELPA might be faster (2)
too slow on shared mem
DLA on one core, parallelization around it
data distribution not obvious (3)
No QP (? Quadratic programming?)
hard to build in heterogeneous environment
cryptic documentation, mor examples (2)
pdgeev and pzgeev missing
poor performance vs LAPACK on one core
Elemental better (2)
pain to setup
awful interface (2)
Use MPI collectives instead of your own
poor treatment of C/Fortran integer interoperability

Q33 - ScaLAPACK Interface easy to use? (80)

Yes	48%
No	52%

Q34: If you prefer a simpler interface, what? (67)

Allocate workspace automatically	64%
Optional args. to return info on request	37%
Allow user data types (eg high precision)	27%
Autotune algorithm	31%
Autoconvert data structure to faster one	55%

Other (please specify)

idk (? I don't know?)
easier to setup when multiple MPI levels of parallelization (contexts vs communicators)
vendors don't optimize PILAENV
BLACS too cryptic, use only MPI communicators(3)
setting up matrix descriptors hard (2)
autodistribute matrix, given size,MPI communicator

Q35 - Which ScaLAPACK documentation do you use? (73)

ScaLAPACK User guide	77%
LAPACK Working notes	12%
Sca/LAPACK User Forum	16%
Routine's comments	51%
Other	14%

Online documents from Intel IBM etc
Google
Blogs
Stack Overflow
Intel MKL manual (2)
contact developers

Q36 - Documentation good enough? (69)

Yes	84%
No	16%

Comments

Incomplete compared to LAPACK docs (2)

Often hard to find detailed specs
More on C/C++
How to collect result onto single process
More examples (3)

Q37 - How do you install ScaLAPACK? (81)	
Vendor package	63%
Platform distribution system (Debian etc)	31%
Python installer	5%
Makefile	46%
CMAKE	11%
Other	
macports	
Bake	

Q38 - ScaLAPACK installation ok, or to be improved? (74)	
Yes	84%
No	16%
Comments	
v2.0 better than v1.0	
If same level of testing as for LAPACK, not easy	
Decided not to bother after reading guidelines	
How to specify BLACS options not clear	
consistency across architectures	
want parallel build (make -j)	
should use configure; make; make install	
should be easy to build without running binaries	
want module add	
C interface	

Q39 - Missing functionality, other comments (17)

Lots missing vs LAPACK
Robust communicator; BLACS trouble if we create
multiple levels of communicators (3)
Evecs for nonsymeig: cyclic distribution causes
restrictions that are hard to understand
Less complete testing than for LAPACK
Poorer comments than for LAPACK (eg typos)
Encourage users to report typos, should be easier
than a formal bug report
Allow more layouts, eg 1D block column
Quad prec gen eig with inverse iteration with pivoting
Automatic data structure change
More example programs
PDGEEV and PZGEEV
Better performance of PDGEMM/PDSYRK for
various matrix sizes (tall-skinny)
C++ interface
QRCP with threshold, randomized
Symmetric packed format

RRQR
Make installation easier
Better performance on small matrices

PLASMA Specific Questions (at most 19 answers per question)

Q40 - Which routines do you mostly use? (18)

GELS, SYSV, QR(2), GEMM(2), ORGQR

solvers
via HiPLARx R package

Q41 - from which language do you call PLASMA? (15)

Fortran (77/90/95/03/08)	3
C	4
C++	5
Python/Numpy	0
Julia	0
R	1
Matlab	0
Octave	0
Cuda	0
PETSc	0
Haskell	0
Eigen	0

Q42 - If considered PLASMA but didn't use it, why? (17)

too many dependencies
either use distributed memory code, or vendor LAPACK
examples codes crashed on our multicore systems
ZGEEV missing
PLASMA has own runtime, so interacts poorly with MPI+{OpernMP, Pthreads,TBB}
Didn't want to install it myself
Sub-optimal results, community adoption not high

Q43 - Is PLASMA interface easy to use? (19)

Yes	68%
No	32%

Q44: If you prefered a simpler interface, what? (6)

Allocate workspace automatically	63%
Optional args. to return info on request	0%
Allow user data types (eg high precision)	38%
Autotune algorithm	50%
Other (please specify)	0

Q45 - Which PLASMA documentation do you use? (16)

PLASMA User guide	69%
PLASMA User Forum	13%
webpages: doxygen	25%
Routine's comments	31%
Other	**25%**

Q46 - Documentation good enough? (16)

Yes	75%
No	25%

Comments (3)

webbased documentation hard to navigate due to all the different alternatives (something else suggested)

Q47 - How do you install PLASMA? (18)

Q48 - PLASMA installation ok, or to be improved? (21)

Vendor package	6%
Platform distribution system (Debian etc)	12%
Python installer	18%
Makefile	53%
CMAKE	24%
Other (2)	12%
not installed (2)	

Yes 76%
No 24%

Comments

hard to install (3), make available in fedora/ubuntu or preinstall on large DOE machines.

Q49 - Missing functionality, other comments (6)

autotuning
mixed precision
too hard to use on Titan (lots of details)
want distributed parallel alternative

MAGMA Specific Questions (at most 42 answers per question)

Q50 - Which routines do you mostly use? (42)

solvers, QR, eigensolvers

Q51 - from which language do you call MAGMA? (40)

Fortran (77/90/95/03/08)	5
C	8
C++	17
Python/Numpy	2
Julia	0
R	2
Matlab	0
Octave	0
Cuda	5
PETSc	0
Haskell	0
Eigen	0
Java	1
OpenCL	1

Q52 - If considered MAGMA but didn't use it, why? (24)

don't need accelerators
too hard to install, because of external libraries, Windows
hard to install
low performance on small matrices
need multiprocessor support, we use Sca/LAPACK
cuBLAS and cula good enough
vendor performance better
cuBLAS already installed with NVCC
poor documentation
zgetri missing
need distributed memory parallelism
needed ZGEEV, may try again
wasn't efficient drop-in replacement for LAPACK in legacy codes

Q53 - Is MAGMA interface easy to use? (40)

Yes	75%
No	25%

Q54: If you prefer a simpler interface, what? (25)

Allocate workspace automatically	60%
Optional args. to return info on request	20%
Allow user data types (eg high precision)	20%

Q55 - Which MAGMA documentation do you use? (36)

MAGMA User guide	64%
MAGMA User Forum	31%
Routine's comments	54%
Other (6)	17%
google, code studies, examples	

Q56 - Documentation good enough? (35)

Autotune algorithm	20%
Other (please specify) (7)	32%
Incomplete documentation (details given)	
CMAKE build	
Interface with Eigen	
Reuse allocated space on GPU	
interfaces that accept and return data on GPU	

Yes 84%
No 16%
Comments (6)
not as comprehensive as cublas
need to read source code, need to read mpgpu.pdf from 2013
bugs in doxygen pages
more details about data location, algorithms

Q57 - How do you install MAGMA? (39)

Vendor package	3%
Platform distribution system (Debian etc)	6%
Makefile	77%
CMAKE	21%
Other (4)	10%
wrote own CMAKE for clMAGMA	
AUR - arch user repository	

Q58 - MAGMA installation ok, or to be improved? (40)
Yes 78%
No 22%
Comments (7)
Visual Studio, better CMAKE (2), clMAGMA has issues, what to download confusing, need git
custom make.inc less than ideal

Q59 - Missing functionality, other comments (21)
better error handling
github
MRRR
save pivot vector to host or GPU
CMAKE
mixed precision
row major order
spring_gpu
header files with correct const attributes
be able to have Input/output on host or GPU (2)
open-source distributed version (2)
zheevd that uses GPU and CPU RAM for big problems
32 bit ints too small for workspace size
MAGMA on XeonPhi
solvers without pivoting

Anything else you want to tell us (at most 201 answers per question)

Q60 - I am a (201)

Computer Scientist (libraries/tools)	39%
Computer Scientist (middleware)	12%
Computational Scientist	74%
Researcher	73%
Software maintainer	21%
Sys Admin	6%
Manager	6%

Principal Investigator	25%
Other (10)	6%
student, mathematician (2), teacher (2), software developer, physicist (~3), engineer	

Q61 - What domains are you involved in (201)	
Computer science	46%
Physics	59%
Chemistry	25%
Climate modeling/material science	10%
Biology	10%
Math	46%
Geology	4%
Econ/Finance	3%
Other (33)	17%
speech/language processing, mechanics, engineering (2), statistics(5), software engineering, signal processing, engineering (14), comp. mechanics earth science, applied physics, combustion, aerodynamics (3), continuum mechanics, optics	

Q62 - contact info

Q63 - Additional comments and suggestions (32)

Topics not mentioned before:

Make ScaLAPACK scale beyond 5K to 10K cores

Persuade Mathworks to use MAGMA/PLASMA, if performance better than LAPACK

Too many versions of BLAS and LAPACK in circulation, not all compatible, makes installation/portability hard

SURVEY COMMENTS

- Thank you for the amazing work of scalapack, and if you ever create some scalable eigenvector solver please keep me in touch !
- "Just want a good Python wrapper. Also an easy way to install, if possible..."
- Examples of magma working with OpenACC and OpenMP target.
- Thank you for being free of charge!
- "I appreciate these nice software packages.
- I would appreciate more if these packages are better."
- "I have used magma library from version 1.4.1.
- It takes too much time to build it on Windows operation system.
- I hope you to support pre-built binary files in each operation systems."
- nice
- Surprised not to see a DPLASMA section. That's where we are heading.
- Please do something about the error handling in BLAS.
- This survey was an excellent idea
- It would be nice to have a BLAS routine that initializes arrays to certain values (zero in most cases)
- Great software, lets get it to Exascale!
- XSEDE needs to provide more software that can be easily installed via "module add", ESPECIALLY for common dependencies like LAPACK and ScaLAPACK. I've spent hours at a time fighting with Stampede and searching the internet for help on getting libraries like LAPACK and ScaLAPACK installed and running on my Stampede account. Please include development headers (including the *.a and *.so files) so that software can be compiled against them.
- Our main application is described at http://qboxcode.org LAPACK and ScaLAPACK are _extremely_ useful for our applications (thanks for developing and providing them). Scaling of ScaLAPACK beyond 5k to 10k cores (and beyond) is currently a serious issue limiting our progress. We would benefit considerably from research in that area.
- Keep up the good work, the tools you develop are useful and used by many.
- Continued updating and development of LAPACK is an excellent idea.
- Thanks for considering improving the packages
- Good survey! Thanks for the interest!
- I am pleased to see the Scalapack now has a simple CMake install procedure; getting Scalapack and BLACS set up together used to be a pain, so this is a huge improvement.
- Thank you for the wonderful library. I look forward to any enhancements that may come in the future.
- persuade Mathworks to use MAGMA / PLASMA since every computer is multicore, if better than LAPACK (the only one I use)
- I miss being part of the computational software development group. I'm a Ph.D. mechanical engineer with lots of experience in computational dynamics modeling and simulation. I've used the ADAMS software (out of Ann Arbor, MI originally from Mechanical Dynamics, Inc. but now MSC) for34 years. I was instrumental in getting mathematicians into MDI and ADAMS. I also knew Bill Gear, we used his solver for DAE's. I've spent many years trying to get automotive, military and academia more interest in computational aspects of engineering science analysis but I saw where the math was a big challenge. Its still is for too many engineers making things. But I've been a big Prof. Dongarra fan and have watched his influence grow in the computational sciences. I would love to contribute more to linear algebra through my engineering research in dynamical system instabilities. I've found a very clever way to evaluate a mechanical system's instabilities using quasi-static time dependent methods in ADAMS. I'm thinking better code could deliver improved response for critical computational issues. Good stuff here. Best to you Al
- Thanks !
- If you do nothing else, please get rid of BLACS and use MPI properly in ScaLAPACK.

- Great Survey.
- Many thanks for decades of excellence! Looking forward to using your more advanced tools in future.
- The problem with BLAS, LAPACK, and everything that builds on it is that there are a million variations, all of which 1/ provide a set of functions that purport to be compatible 2/ that, without an exception, use entirely incompatible, quirky, unportable, and impossible to use link interfaces. A consequence of this is that if you want to build portable software, the only real option you have is to build only on the most basic variant such as either the vendor (or distribution) provided libblas, or just not do it at all. It is not possible to overstate the amount of time and energy wasted when installing widely used software on a new cluster just to find that, as on every cluster before, the set of BLAS libraries is yet different, comes with different names, is only installed as static libraries, and has undocumented dependencies on other static libraries that one has to discover anew. While I see the value in having competing BLAS libraries and things that build on it, collectively our community has surely spend tens of man years of work in making this work in practice. What this means is that in reality, the proliferation of BLAS libraries, the fact that they build on each other in unclear ways, and that projects seem to come and go every few years has therefore be a real detriment to our community, leading to a rather negative general attitude in the community. My take on this is that the dense linear algebra community really needs to get together and do some soul searching on whether they want to continue with this model. The current approach allows for rapid experimentation and development of new approaches for new platforms. At the same time, it makes it incredibly difficult to use for many projects, and consequently to far less uptake of these new ideas than could be possible if there were only one or two, well supported, stable, projects that had a predictable development path on which one could build software that we know will still work in 5 or 8 years.
- Thanks for the good work! I rarely use LAPACK directly, but rather use wrappers because it makes my code shorter, more concise, easier to read and easier to use. Wrappers will also handle memory allocation etc for me. But I really appreciate the work that is going on "underneath".
- Thanks to Ichitaro Yamazaki for all his help and input!