Qualcomm Technologies, Inc.

Whitepaper: Linear Algebra Package for Qualcomm® Snapdragon™ Math Libraries

LAPACK for Snapdragon Math Libraries brings advanced linear algebra functions to Android running on Snapdragon processor

Nathan Whitehead
March 2017

Table of Contents

Summary

Math-intensive use cases like robotics, augmented reality and machine learning continue their inexorable migration onto mobile devices. Supported by improvements in cameras, sensors, multicore processors and mobile operating systems, developers continue to find new ways to move the computing heart of their applications from servers and the cloud all the way down to handheld devices.

But some libraries required for advanced computing use cases are not yet accessible to mobile developers. Vehicle navigation, real-time computer vision and deep learning, for example, are made vastly easier by the use of the Linear Algebra PACKage (LAPACK), a library for solving common but thorny math problems. Yet the functions in LAPACK and the related Basic Linear Algebra Subprograms (BLAS) are written in Fortran and go back to the 1970s, predating all of mobile computing and most of desktop computing. They are absent on some mobile platforms and incomplete on others.

This paper introduces LAPACK for Snapdragon Math Libraries, a package that is designed to make the more than 1,700 LAPACK functions accessible to Android applications running on the Snapdragon processor. Developers who have been limited to running applications like deep learning and data analytics on servers or in the cloud can now use Snapdragon Math Libraries to run those and other math-dependent applications with computational accuracy, high performance and easy portability across the Snapdragon family.

What is LAPACK?

Linear Algebra PACKage, or LAPACK, is a math library optimized for solving common problems, for example:

- Simultaneous linear systems of equations
- Least-squares solutions of linear systems of equations
- Eigenvalue problems
- Singular value problems

LAPACK includes commonly used factorizations like Cholesky, LU, QR, SVD and Schur decomposition. LAPACK offers single, double, single-complex and double-complex precision for dense and banded storage formats.

The roots of LAPACK lie in LINPACK, a collection of subroutines for solving linear equations and linear least-squares problems, and EISPACK, a package for finding eigenvalues and eigenvectors of matrices. As shown in Figure 1, they stretch back to the 1970s, were written in Fortran and were intended to run on supercomputers – characteristics that have caused problems for developers trying to implement linear algebra for other environments.

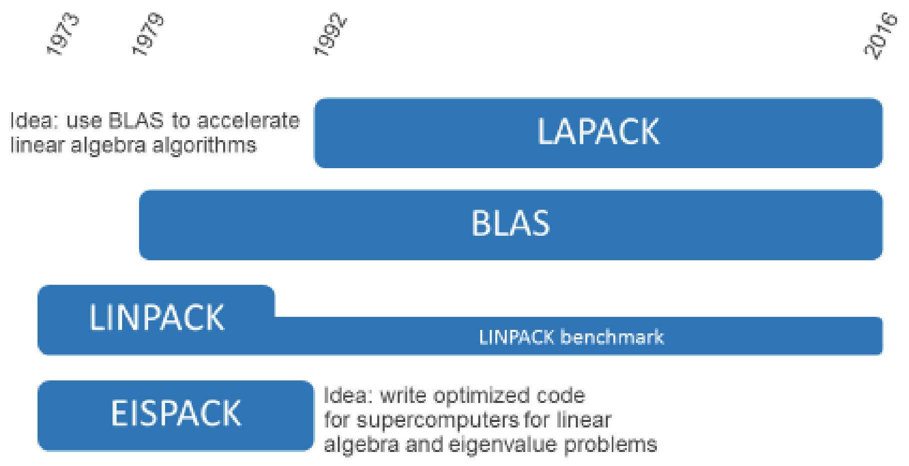
1973
1979
1992
2016
Idea: use BLAS to accelerate linear algebra algorithms
LAPACK
BLAS
LINPACK
LINPACK benchmark
EISPACK
Idea: write optimized code for supercomputers for linear algebra and eigenvalue problems

Figure 1: History of LAPACK

In the late 1970s, Basic Linear Algebra Subprograms (BLAS) emerged as a new way to organize linear algebra libraries.

Relationship between LAPACK and BLAS

Early on, it was common practice to write an entire algorithm like QR factorization as a single function. But as more computing architectures arose, that practice became untenable. Different architectures required customized coding patterns to get maximum performance. Optimizing functions such as QR factorization thus required both low-level machine knowledge as well as mathematical sophistication to understand the algorithm itself. The goal of BLAS was to allow the same linear algebra primitives to work unaltered on any processor with good performance. The idea behind LAPACK was to rewrite LINPACK and EISPACK to take advantage of BLAS packages on different platforms.

BLAS is in effect an interface and LAPACK takes advantage of it. Every vendor can optimize a version of BLAS optimized for its own platform that takes advantage of details in the micro-architecture. Because the higher-level algorithms reside in LAPACK and are implemented on top of BLAS, improvements to BLAS improve LAPACK directly. LAPACK provides functions at various levels of abstraction all the way from solving entire high-level mathematical problems down to simple utility functions that happen to not be included in BLAS.

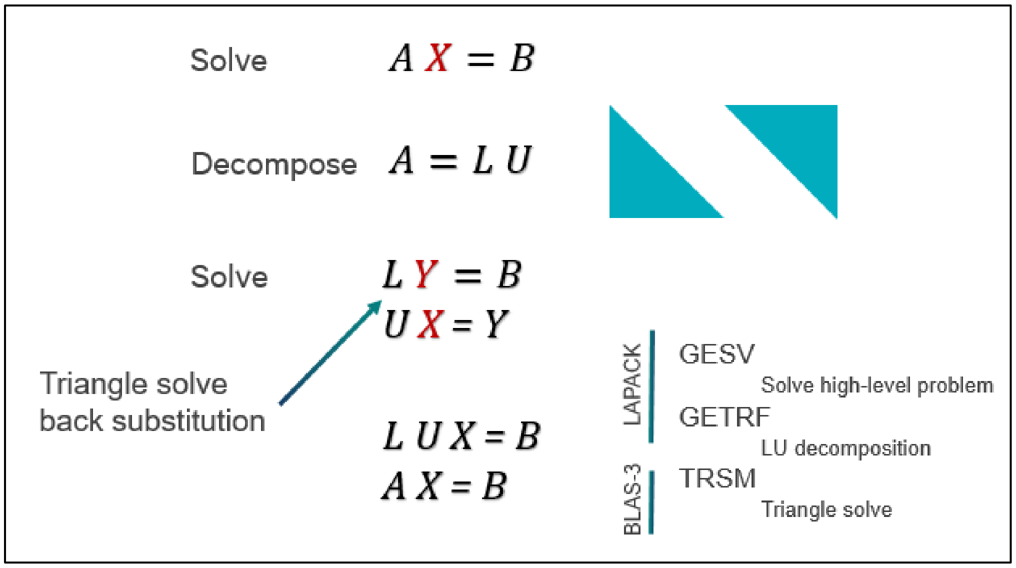

Figure 2: Solving for X in a linear system

Example: Solving a linear system

Figure 2 shows the option of solving a linear system at different levels.

The work to be done is to solve a system of equations using a matrix factorization – in this case, an LU decomposition:

1. Decompose A=LU.
2. Perform the triangle solve LY=B, solve for Y using back-substitution
3. Perform the triangle solve UX = Y, solve for X using back-substitution.

The result is the actual X for AX=B.

With LAPACK and BLAS, a developer can call the necessary functions at different layers:

- GESV is a high-level LAPACK interface that can perform all of the work of simply solving AX=B.
- GETRF runs one level lower in LAPACK to perform the LU decomposition but doesn't perform the triangle solves to reach the final answer. (It is common to reuse the same factorization multiple times or use it for other things.)
- TRSM, the triangle solves, are in the highest level of BLAS-3 and solve in parallel, block by block.

Thus, LAPACK has the power and flexibility needed for performing linear algebra on any platform. But is it ready for mobile?

Obstacles to LAPACK on mobile processors

Application domains such as automotive, data analytics and neural networks are primed for the leap from desktop, server and cloud computing to mobile. As developers attempt to move those math-heavy applications over, several obstacles arise.

Compiler errors on mobile platforms

Figure 3 depicts a small sample of the many far-reaching relationships among libraries, packages, frameworks and applications around LAPACK. These stacks get very tall and tangled.

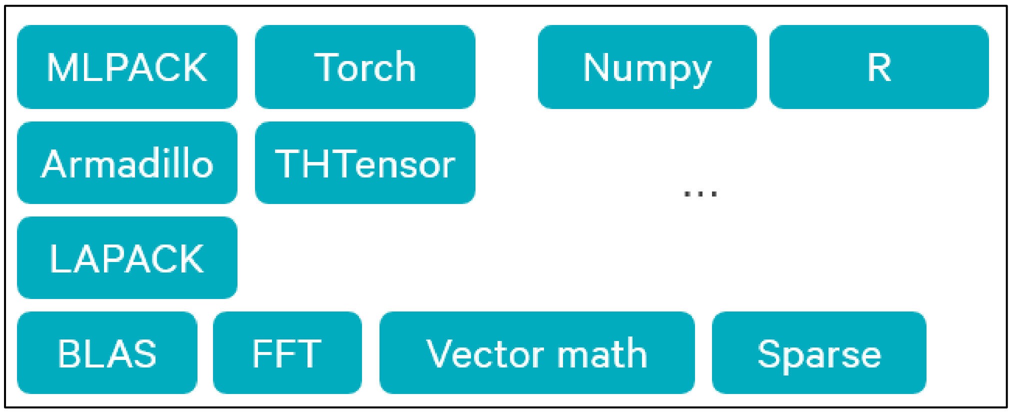

Figure 3: Software relationships

Mathematics software like BLAS and LAPACK sit near the bottom of the stack, especially in application domains like machine learning, so abstractions, interfaces, frameworks and libraries accumulate on top of them. Developers then write applications that depend on all the elements in the stack, often with no idea that the math functions they need are buried at the bottom.

Consider developers of successful machine learning desktop applications based on the Torch framework. They port their applications to Android running on a mobile chipset and attempt to recompile the Torch framework, only to get compiler errors referring to a missing LAPACK library. They've been using LAPACK all along without knowing it because the libraries were available for the desktop. Now they must stop and research which LAPACK to use to take advantage of the same machine learning framework on a different processor.

Open source not the answer

The most commonly used machine learning frameworks, like TensorFlow, Torch, Caffe, MXNet, CNTK, Theano and DSSTNE are constructed on BLAS and LAPACK, so virtually all roads lead to them. To avoid them would require the extraordinarily difficult work of using sparse direct solvers.

Ideally, the platform vendor provides optimized BLAS and LAPACK libraries. Absent these, developers turn to open source versions like OpenBLAS or Eigen running on the CPU, which provide the math support, but less-than-optimal performance.

However, the open source LAPACK libraries do not work on Android because Fortran is not a supported language in Android. Standard tricks such as quickly converting the Fortran code to C for compilation do not work because the version of Fortran used in LAPACK is incompatible with the standard conversion tools. Building the public LAPACK sources would involve compiling a custom toolchain for Fortran which would require long-term maintenance. It would also involve distributing the Fortran runtime binary and ensuring that it behaves correctly in the Android environment.

Decades of evolving knowledge

LAPACK is not simple. It encompasses 1750 functions, and a typical function includes multiple references to journal articles and conference proceedings, representing deep research.

LAPACK code has been tested for decades. It is under active development, with contributors fixing bugs all the time. Ambitious developers could try to do it themselves and re-implement LAPACK, but the chances of their getting it right and keeping it right over time are slim.

There are many corner cases to consider to avoid losing numerical precision with linear algebra. The code that embodies the textbook version of the algorithms seems reasonably concise, but any mistake can lead to underflow, overflow and other inaccuracies.

Correctly implementing the algorithms in LAPACK requires mathematical analysis. Many functions are written by teams of mathematicians whose expertise it would be difficult to duplicate.

Fortran as a legacy language

LAPACK consists entirely of Fortran code, so the lack of vendor-optimized Fortran compilers makes it hard to use LAPACK in almost any new environment.

Few vendors want to deal with a legacy language like Fortran. Android, for example, has no out-of-the-box support for LAPACK, so there is no easy way to compile the official, non-optimized reference version. Although the commonly used GNU C Compiler (GCC) supports Fortran, the Android NDK toolchain turns it off and makes it all but inaccessible. A determined developer could rebuild GCC from scratch with the goal of enabling languages such as Fortran and getting a toolchain that works, but that would require patching several configuration files, and the resulting binaries might interact with Android in unexpected, undesirable ways.

Thus, LAPACK depends on a very old programming language created with mainframes and supercomputers in mind. Over the years, the vendors of server architectures, where high-performance computing is common and expected, have always provided optimized BLAS and usually LAPACK.

In short, it is difficult to get Fortran to work on mobile platforms. The practical absence of support for Fortran on Android has kept LAPACK out of reach and posed an obstacle to developing mobile applications for deep learning and machine learning. Because LAPACK includes math functions that can be applied in many different ways, it is hard to estimate all the use cases its absence affects.

Introducing LAPACK for Snapdragon Math Libraries

When developers consider using LAPACK on a new platform, they don't want to have to reinvent everything. In fact, there is no reason to reinvent it all, because it already works.

The Snapdragon Math Libraries are engineered to make it easy to port frameworks and applications to Qualcomm Technologies silicon. Qualcomm Technologies, Inc. (QTI), has added high-performance implementations of BLAS and LAPACK primitives to Snapdragon Math Libraries, optimized for all generations of Snapdragon processors.

LAPACK for Snapdragon Math Libraries includes almost 2000 primitives (BLAS levels 1, 2 and 3; LAPACK 3.6.0), including all precisions (single, double, single-complex, and double-complex). The libraries and frameworks offer a standardized, predictable interface that is designed to allow developers to focus at the application level instead of trying to compile old un-optimized Fortran code for ARM Android and ARM Linux. Snapdragon Math Libraries is designed to bring all of LAPACK to a mobile industry-leading processor used in hundreds of millions of mobile devices and pave the way for it in the coming generation of ARM-powered servers as well.

LAPACK and BLAS in Snapdragon Math Libraries are parallel at the core level, taking advantage of vector instructions on all CPU cores.

Performance improvements with Snapdragon Math Libraries

Snapdragon Math Libraries 0.15.1 has been extensively optimized for Qualcomm Technologies processors and compares favorably with the Eigen library. On the Snapdragon 820 processor, a 64-bit ARM architecture, Snapdragon Math Libraries is several times faster than Eigen 3.3.0 at various problem sizes and precisions when doing BLAS-3 GEMM, dense matrix multiply (Figure 3).

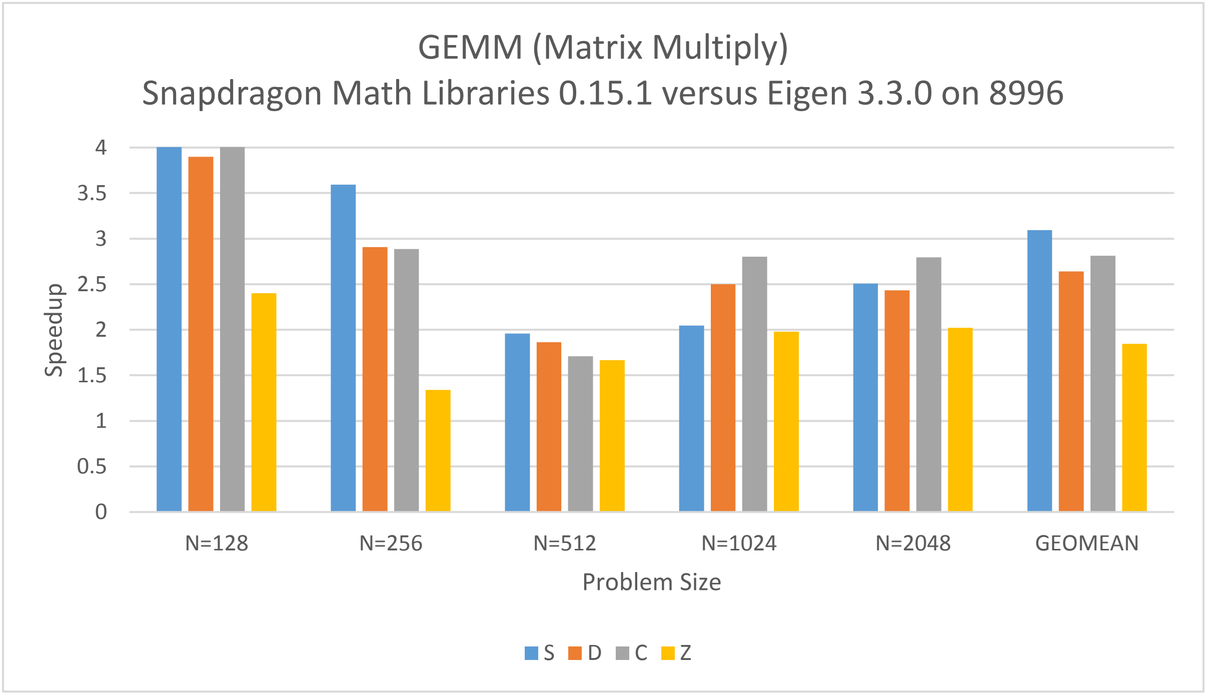

Figure 4: Snapdragon Math Libraries 0.15.1 is several times faster than Eigen 3.3.0 for dense matrix multiplication.

Eigen also includes higher-level functions corresponding to various LAPACK routines. Snapdragon Math Libraries 0.15.1 is much faster than Eigen 3.3.0 for the popular functions GETRF (LU decomposition), Figure 3, and POTRF (Cholesky decomposition), Figure 3

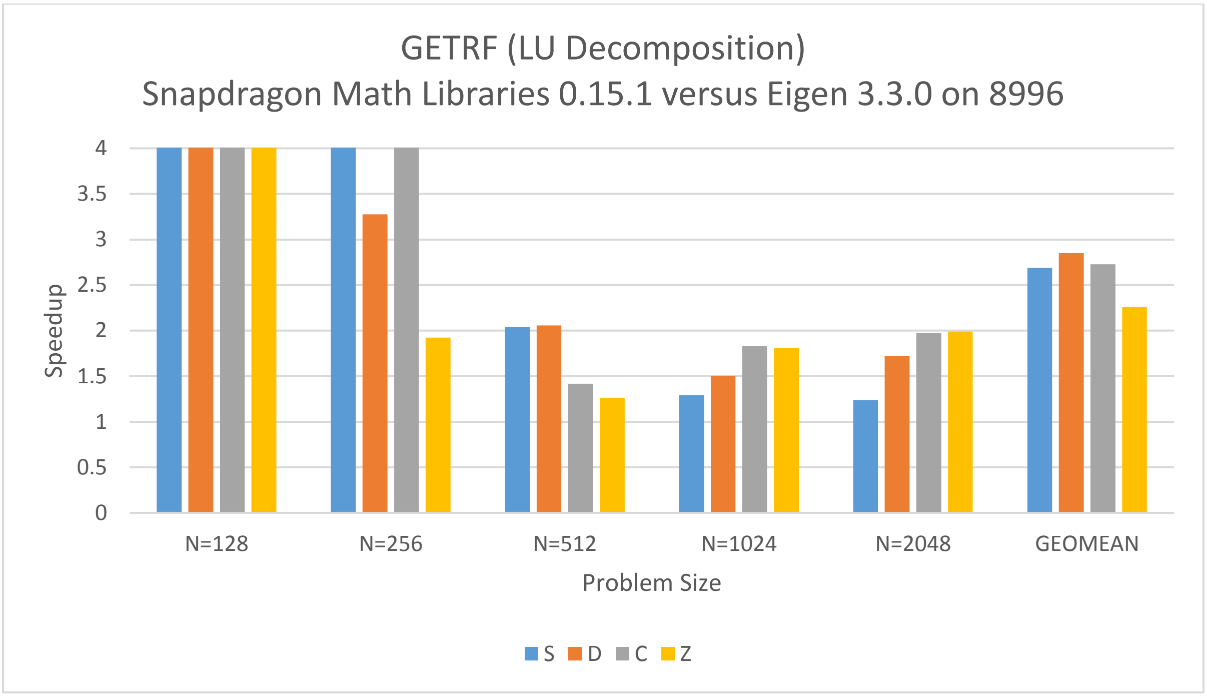

Figure 5: Snapdragon Math Libraries 0.15.1 is faster than Eigen 3.3.0 on 8996 for LU decomposition, especially for smaller problem sizes.

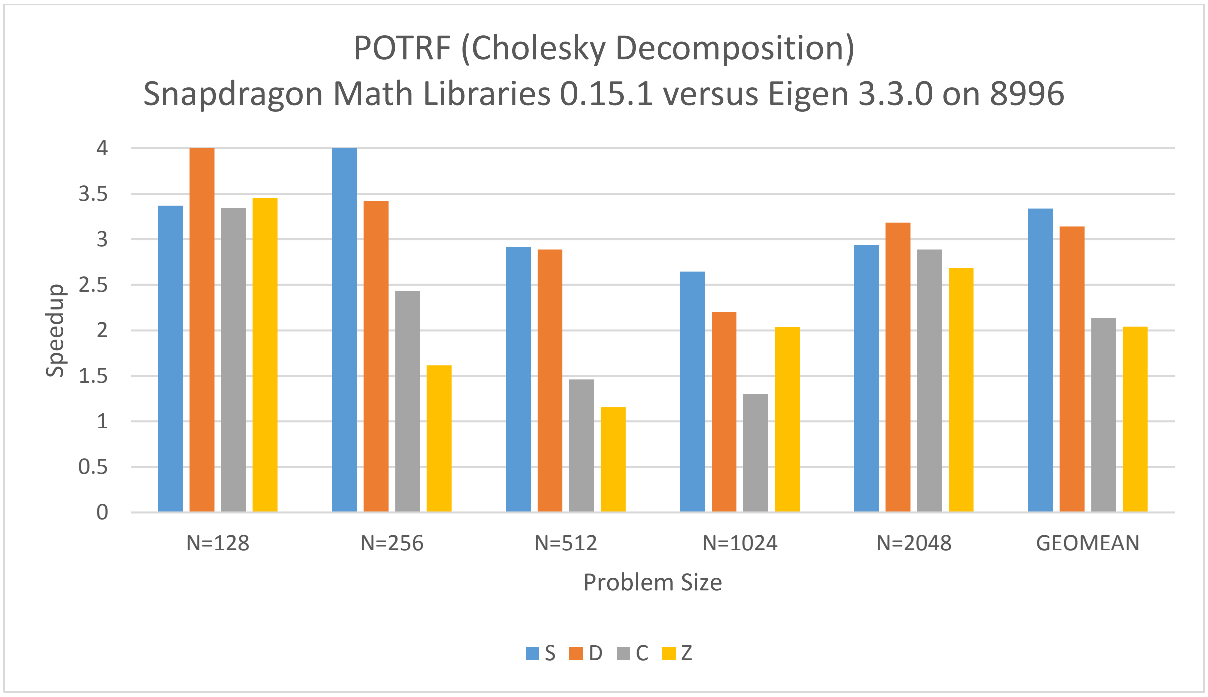

Figure 6: Snapdragon Math Libraries 0.15.1 is faster than Eigen 3.3.0 on 8996 for Cholesky factorization across diverse problem sizes and precisions.

Because of the standard nature of the BLAS and LAPACK interfaces, it is also possible to use Eigen in conjunction with Snapdragon Math Libraries. By setting a compile-time definition Eigen can be configured to call Snapdragon Math Libraries to accelerate matrix operations. Figure 7 shows the speedup of Eigen 3.3.0 when linked with Snapdragon Math Libraries 0.15.1 for LU decomposition. Figure 8 shows the speedup for Cholesky decomposition.

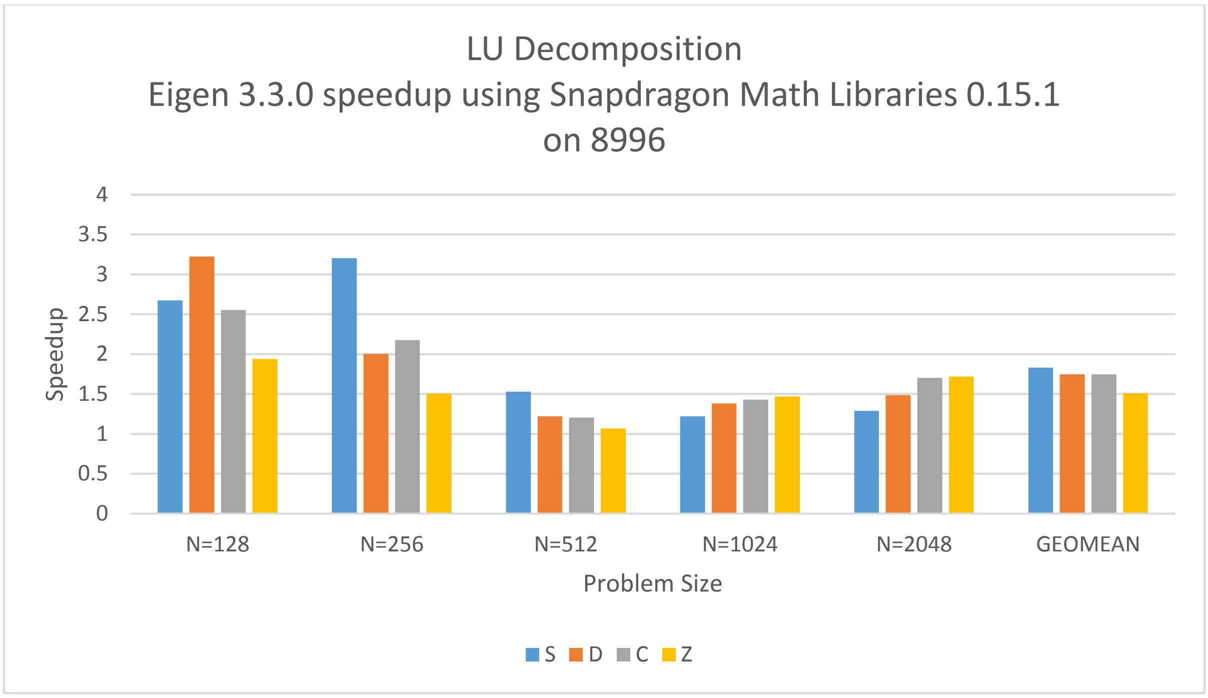

Figure 7: LU decomposition is sped up when Eigen is configured to useSnapdragon Math Libraries for matrix operations.

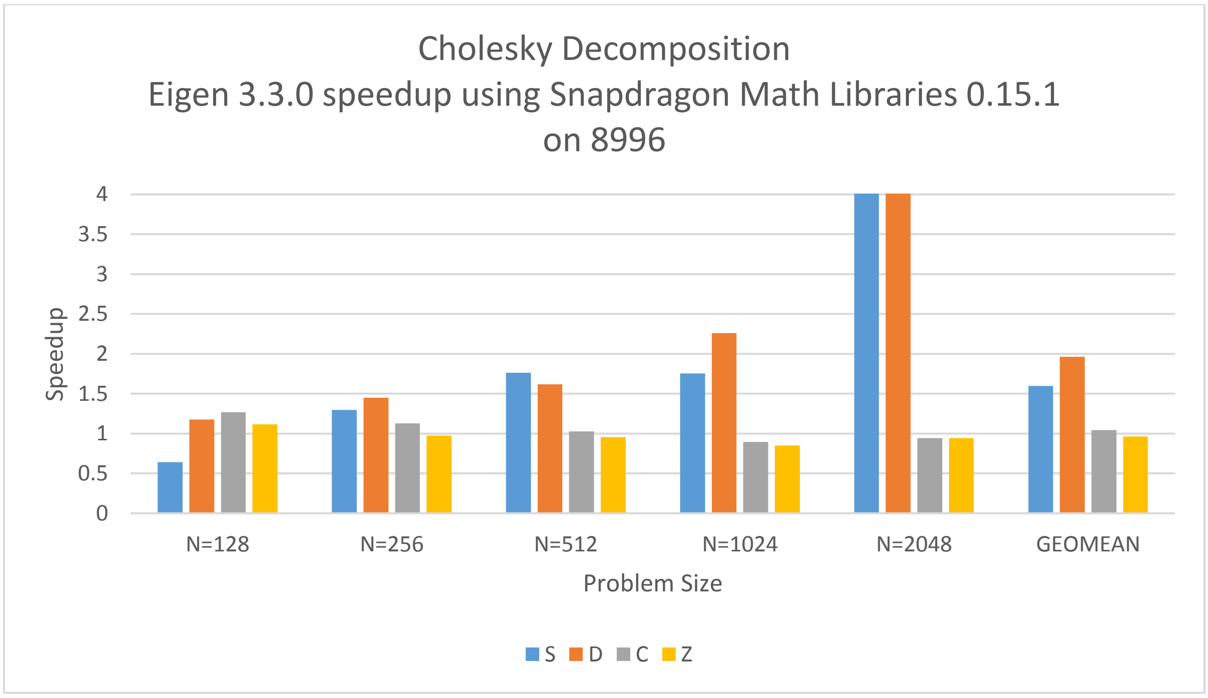

Figure 8: When using Snapdragon Math LibrariesL, Cholesky decomposition in Eigen is sped up for single and double precision but not for complex precisions.

How QTI optimized LAPACK for Snapdragon

To optimize LAPACK for the Snapdragon architecture, QTI engineers ran the Fortran Netlib LAPACK code through a highly customized version of Fable, a tool developed by crystallography scientists at the Lawrence Berkeley National Laboratory (Berkeley Lab). The customized version translates the LAPACK Fortran source code into C++, then uses QSML's own primitives for all basic operations. Snapdragon Math Libraries is engineered to use its own optimized BLAS functions for every BLAS call in LAPACK, provide optimized blocking sizes for blocked algorithms, and choose the best algorithmic variant when LAPACK provides a choice of algorithms.

Figure 3 shows some of the improvements made between the initial un-optimized release of LAPACK functions in Snapdragon Math Libraries 0.15.0 and the latest release, Snapdragon Math Libraries 0.15.1 for LU decomposition.

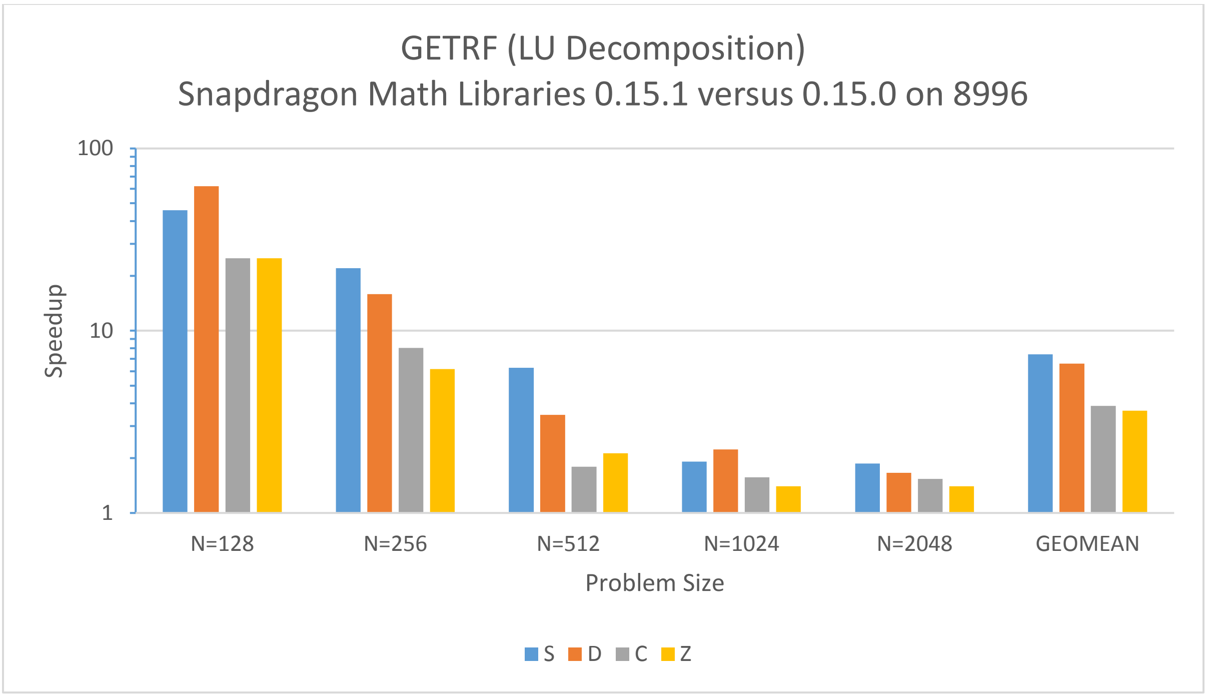

Figure 9: QSML 0.15.1 is much faster than the previous release of Snapdragon Math Libraries 0.15.0, especially for smaller problem sizes. Shown are performance results for LU decomposition on a log plot.

Application domains

Developers can now unlock yet more potential from mobile devices and move a big step closer to going mobile with any application that runs on a server or in the cloud. QSML support for LAPACK and BLAS contains the versions of functions optimized for Snapdragon required in the sample application domains described below.

Machine learning

Machine learning frameworks use linear algebra in neural networks, support vector machines (SVM) and classifiers. Most machine learning for deep neural networks is essentially convolution, or matrix-matrix multiplication in BLAS.

For example, the algorithm behind DeepMind AlphaGo (see **Error! Reference source not found.**) is a randomized Monte Carlo tree search, guided by an evaluation function learned using a deep neural network. Torch, the machine learning framework used to train the evaluation function, solves polynomial equations by calling eigenvalue decomposition (GEEV) in LAPACK.

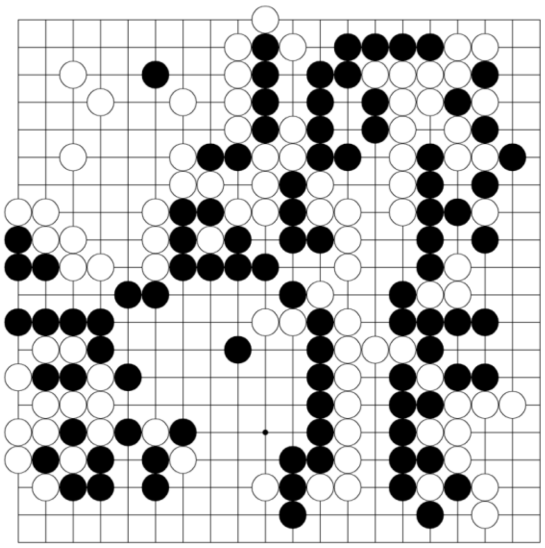

Figure 10: DeepMind AlphaGo defeated Lee Sedol, the highest ranked human player, at Go. Show is the final position for game 1 of the match, won by AlphaGo.

Developers of machine learning applications on the desktop use a given framework such as Torch, TensorFLow, MXNet, CNTK, Caffe, Theano or others for both prediction and training in their models. When they turn to mobile, where the prediction code and training code are the same as on the desktop, even their preliminary investigations into using the framework get nowhere without platform-specific versions of the LAPACK libraries on which their apps depend.

The use case for mobile machine learning is particularly compelling for three reasons:

1. Training and prediction need to use exactly the same framework.
2. It is important to continue the extensive model training already done on the desktop.
3. Being able to run the learning framework on mobile represents a completely new realm of machine learning prediction.

Robotics

Many robotics applications use Kalman filters, as shown in Figure 11, for controlling the motion of a robot.

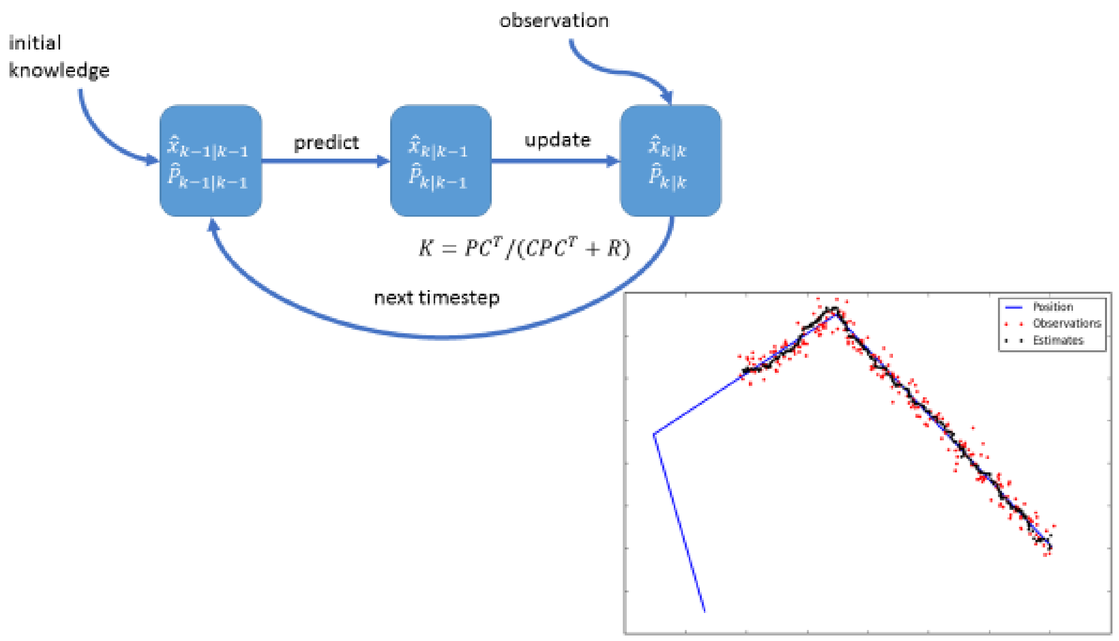
initial knowledge
observation
$\hat{x}_{k-1|k-1}$ $\hat{P}_{k-1|k-1}$
predict
$\hat{x}_{k|k-1}$ $\hat{P}_{k|k-1}$
update
$\hat{x}_{k|k}$ $\hat{P}_{k|k}$
$K = PC^T/(CPC^T + R)$
next timestep
Position
Observations
Estimates

Figure 11: Kalman filter

Kalman filtering, or linear quadratic estimation, can be used to track estimated position and orientation. It applies equations in the prediction step to estimate position in the next timestep based on the current prediction, then uses observations from multiple sensors to update its prediction based on the accuracy of its predictions. The algorithm continuously updates the estimate of the robot's current position in a feedback loop.

One way of performing Kalman filtering uses Cholesky factorization (POTRF) or LDLT decomposition (SYTRF) in LAPACK to compute the Kalman gain matrix K at each timestep.

Augmented reality/Virtual reality

It's not difficult to impose an artificial image against a real background through a camera, but making the image look realistic with correct perspective, as depicted in Figure 12 and Figure 13, takes more work, more math and more computation.

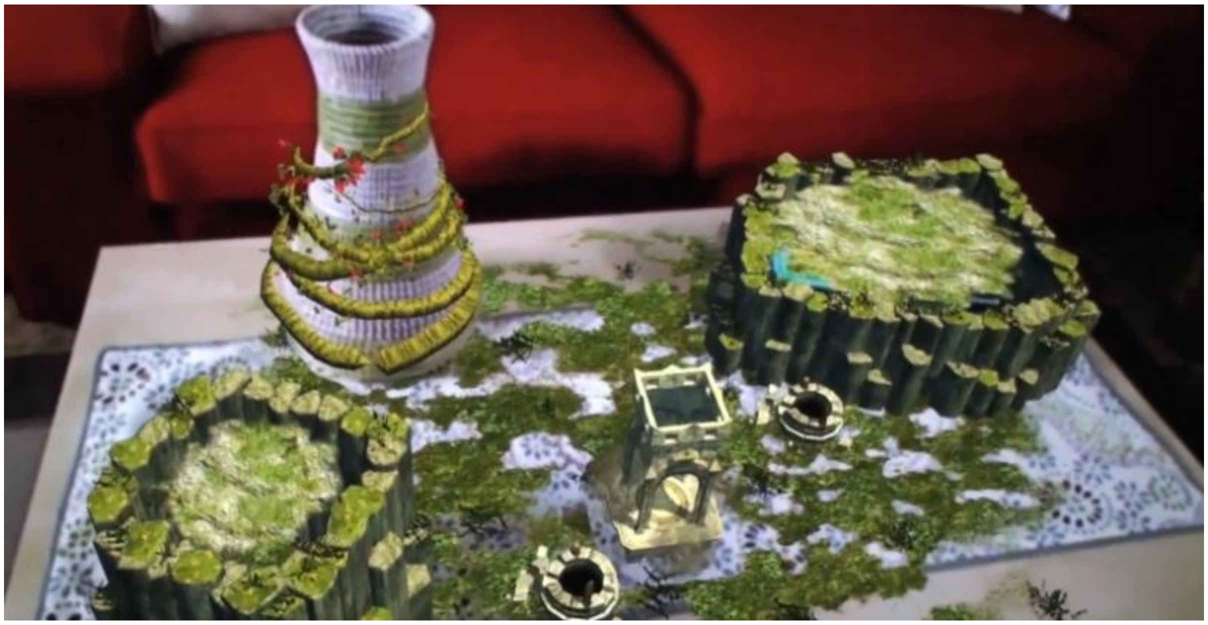

Figure 12: Augmented reality in Vuforia Smart Terrain showing a virtual fantasy game environment layered on top of objects placed on a coffee table.

Figure 13: Virtual reality example

AR and VR depend on algorithms for facial feature detection, sensor integration such as GPS, real-time camera processing of 2D images, object recognition and insertion of objects into a scene.

One example is a pose estimation algorithm for going from 2D to 3D. Once the image is processed in 2D and all the points and edges of the object have been extracted, the next step is to determine the 3D orientation of the object. One way to do this is to minimize the reprojection error using non-linear least squares, an iterative algorithm comprising repeated Cholesky factorization (POTRF), QR factorization (GEQRF) or SVD (GESVD) decomposition.

It used to take several minutes to run CV-related algorithms on a single frame; desktop computers can now run the algorithms in real time. The main barrier to real-time CV processing on mobile devices is making libraries like LAPACK available and getting them to compile.

Data analytics

The statistical analysis required to find patterns in big data and make them useful relies on algorithms, frameworks and libraries.

Consider the recommendation engine of an ecommerce site or online marketplace. One of the underlying techniques is principal component analysis on large data sets of sales. Expressed as a matrix with customers as rows and products as columns, a numeric value indicates the likelihood of the customer's interest in the product based on his/her history of browsing and purchasing. The goal is to reduce the dimensionality of the data to a cluster, then group items together into components – "people who like product A also like product B."

One way to reduce dimensionality is to use truncated SVD (GESVD) from LAPACK and arrive at a "best" prediction matrix for a given dimensionality. By reducing dimensionality from 10,000 users times 10,000 products to 1,000 times 1,000, the cluster can group all users and products into certain types of each. Even with the errors introduced by dimension reduction, the cluster captures sufficient information to predict how different types of customers will like different types of products.

* * *

In all of the foregoing application domains, Snapdragon Math Librariesprovides versions of LAPACK and BLAS optimized for the Snapdragon processor. Developers can enjoy the benefits of high performance, computational accuracy and portability across generations of Snapdragon.

Implementing LAPACK for Snapdragon Math Libraries

To implement LAPACK, developers download and link Snapdragon Math Libraries to their favorite machine learning framework, as they would any other library.

The advantage of software libraries is that developers can modify their builds to link to them and enjoy the benefits of LAPACK without needing to modify the callers of the libraries. There is no need to switch to another framework or learn a new API.

Conclusion

Snapdragon Math Libraries brings comprehensive versions of LAPACK and BLAS optimized for the worldwide installed base of devices running Snapdragon processors. Snapdragon Math Libraries is designed to encompass all LAPACK functions and all their precisions, making them available, accelerated and numerically accurate. Developers working with frameworks built on standard math libraries like BLAS can implement Snapdragon Math Libraries, which uses the same standard interfaces.

Without compiling their own Fortran compiler or figuring out how to get LAPACK libraries to work, developers can easily move frameworks like Caffe, Theano, Torch, Eigen and Armadillo to Android on Snapdragon and run the applications that depend on them. LAPACK for Snapdragon Math Libraries gives developers more choices and possibilities in how they approach the algebra-intensive tasks that will define computing on mobile and ARM-powered servers for the foreseeable future.

Next Steps

LAPACK for Snapdragon Math Libraries is available on the Qualcomm Developer Network: Registered developers may download the Snapdragon Math Libraries for development on Windows, OS X and Linux.

Find out more about Snapdragon Virtual Reality, including the Snapdragon VR SDK.

Snapdragon Math Libraries is also well suited to drone applications like Qualcomm Snapdragon Flight™, a development board for robotics and drones.