Band Matrix Optimization within LAPACK's xGETRF

Andrew Gearhart

Research supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227)

LAPACK- Linear Algebra PACKage

```
( L   A    P    A    C    K)
( L  -A    P   -A    C   -K)
( L   A    P    A   -C   -K)
( L  -A    P   -A   -C    K)
( L   A   -P   -A    C    K)
( L  -A   -P    A    C   -K)
```

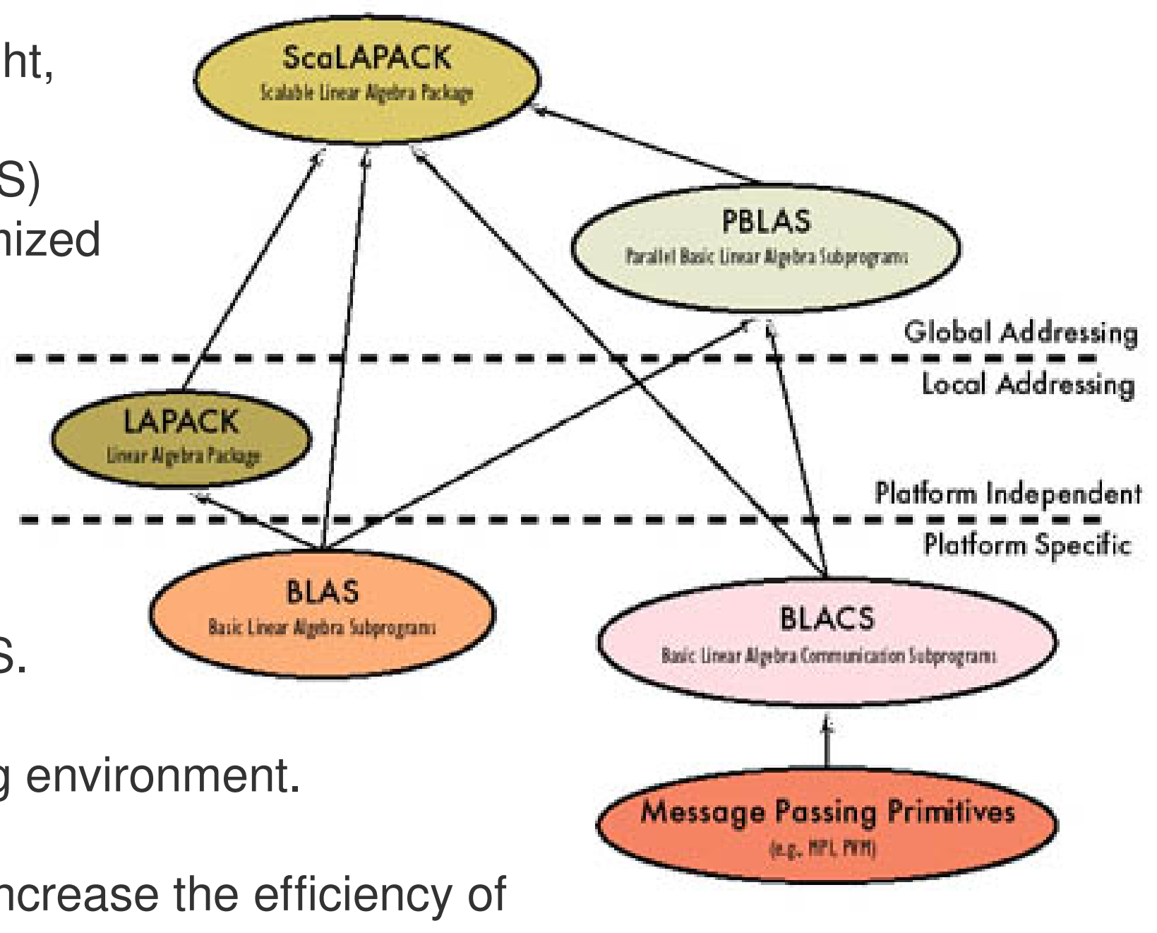
LAPACK (Linear Algebra PACKage) is a library of Fortran 77 subroutines for solving the most commonly occurring problems in numerical linear algebra. It consists of routines for dense and band matrix operations, but is not able to handle sparse operations.

LAPACK was originally developed to replace matrix codes such as LINPACK and EISPACK by providing a software structure specifically designed for optimization upon vector, superscalar, and shared memory processors.

```
      (              l    l    l    l )
      (              a   -a    a   -a )
1/4 * ( p    p                 -p   -p )
      ( a   -a                 -a    a )
      ( c    c   -c   -c              )
      ( k   -k   -k    k              )
```

The subroutines within the software package are able to handle real and complex datatypes, for both single and double precision. Routine names are based upon a coded system that indicates the datatype, type of matrix and operation implemented by the given function.

As can be seen within the diagram at right, LAPACK requires access to a set of Basic Linear Algebra Subroutines (BLAS) to run. Often, the BLAS library can optimized for the target machine via an autotuning framework such as ATLAS.


ScaLAPACK
Scalable Linear Algebra Package
PBLAS
Parallel Basic Linear Algebra Subprograms
Global Addressing
Local Addressing
LAPACK
Linear Algebra Package
Platform Independent
Platform Specific
BLAS
Basic Linear Algebra Subprograms
BLACS
Basic Linear Algebra Communication Subprograms
Message Passing Primitives
(e.g., MPI, PVM)

Another key point is that efficient execution of LAPACK routines is critical for the performance of high-level libraries, such as ScaLAPACK or PBLAS. Both of these libraries extend matrix operations into the distributed computing environment.

The primary objective of this work is to increase the efficiency of LAPACK's dense matrix LU factorization routine by taking advantage of any latent bandedness of input matrices.

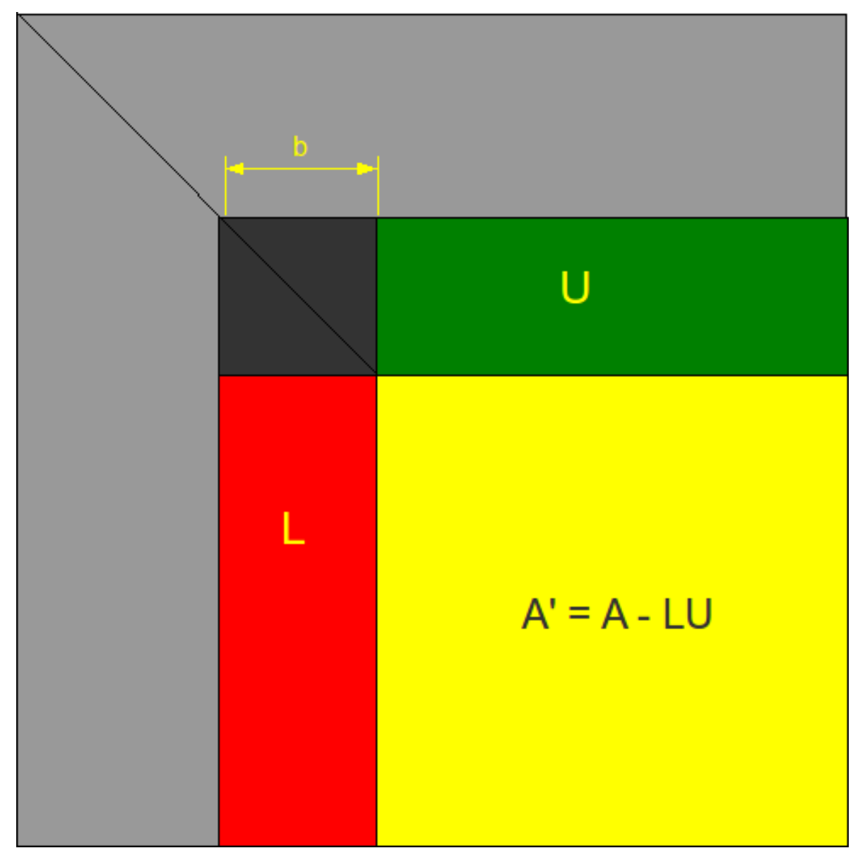
LU with xGETRF

LAPACK's dense LU factorization routines, xGETRF, are a key component of solving linear systems of equations. They implement a right-looking blocked algorithm, such that three major steps are performed:

O Use a Level 2 BLAS operation (xGETF2) to factor the current block panel (creating L in the figure below)

O Perform a triangular solve to create U (xTRSM)

O Update the trailing submatrix (the Schur Complement, in yellow) via A' = A-LU. This is done in one step via a BLAS 3 matrix multiply (xGEMM).



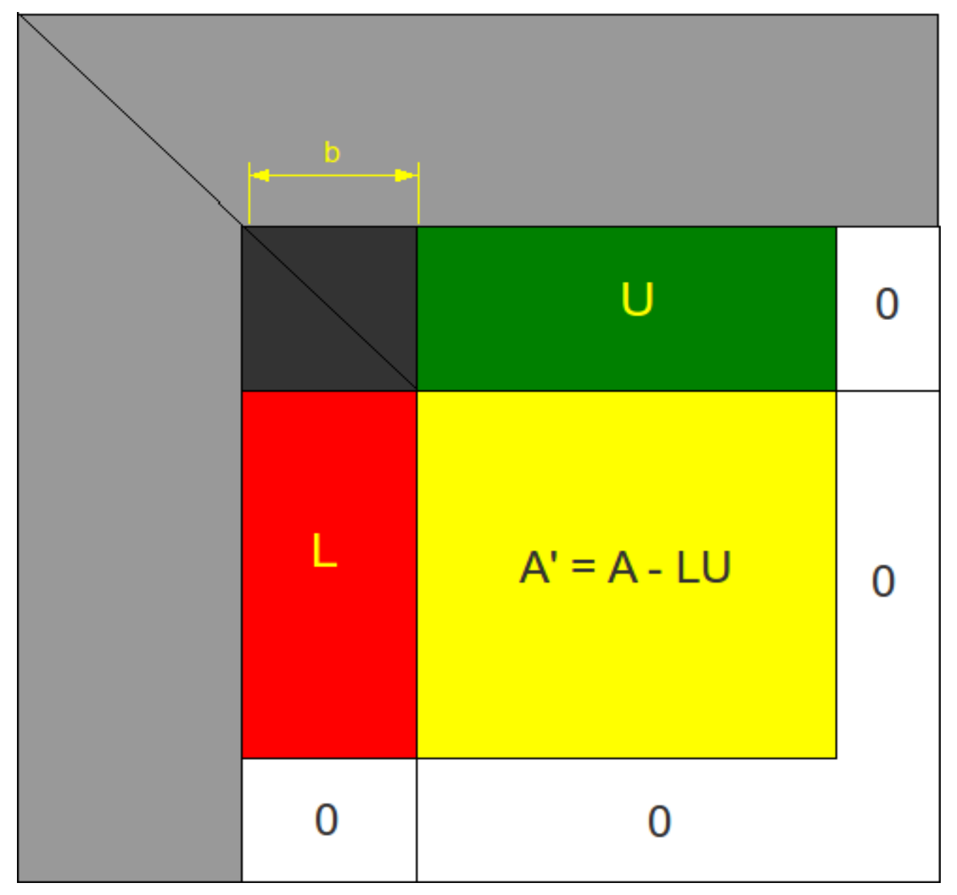
This block-based organization of the LU algorithm allows for the use of xGEMM to perform aggregated updates to the Schur complement. This allows of O(n^3) operations to be performed on O(n^2) data, providing significant potential for optimization and the majority of computation.

It Is this final step of LAPACK's LU algorithm that has been modified for this work, to increase performance on a specific type of input matrix.

Truncated Schur Update

For known band matrices, the LAPACK band LU factorization routines (xGBTRF) are highly efficient. However, if a band matrix is input into xGETRF (the dense factorization) many extraneous flops are performed upon the zero entries. Thus, xGETRF within the current LAPACK release is not able to implicitly obtain performance improvements from input band matrices.

Furthermore, utilizing the band routines first require the input matrix to be copied into a band format which may result in prohibitive memory traffic for very large matrices.



To address the above mentioned problem, this work presents a modified version of LAPACK's xGETRF that is able to implicitly achieve performance benefit from bandedness.

To do this, at each step of the factorization L and U are checked for trailing blocks of 0 entries (see Figure to left). This checking process is started from the end of L and U to minimize the overhead for non-banded matrices.

Looking at the Figure, one can see that the updates effect upon the trailing submatrix is now limited to the smaller yellow region.

Thus, with the realization we can reduce the size of the xGEMM operation and not perform flops upon the 0 entries of L and U. We call this "Truncating the Schur Update".

Schur update truncation promises several advantages over the traditional xGETRF routine:

O The user does not have to be aware of matrix structure to obtain performance benefit

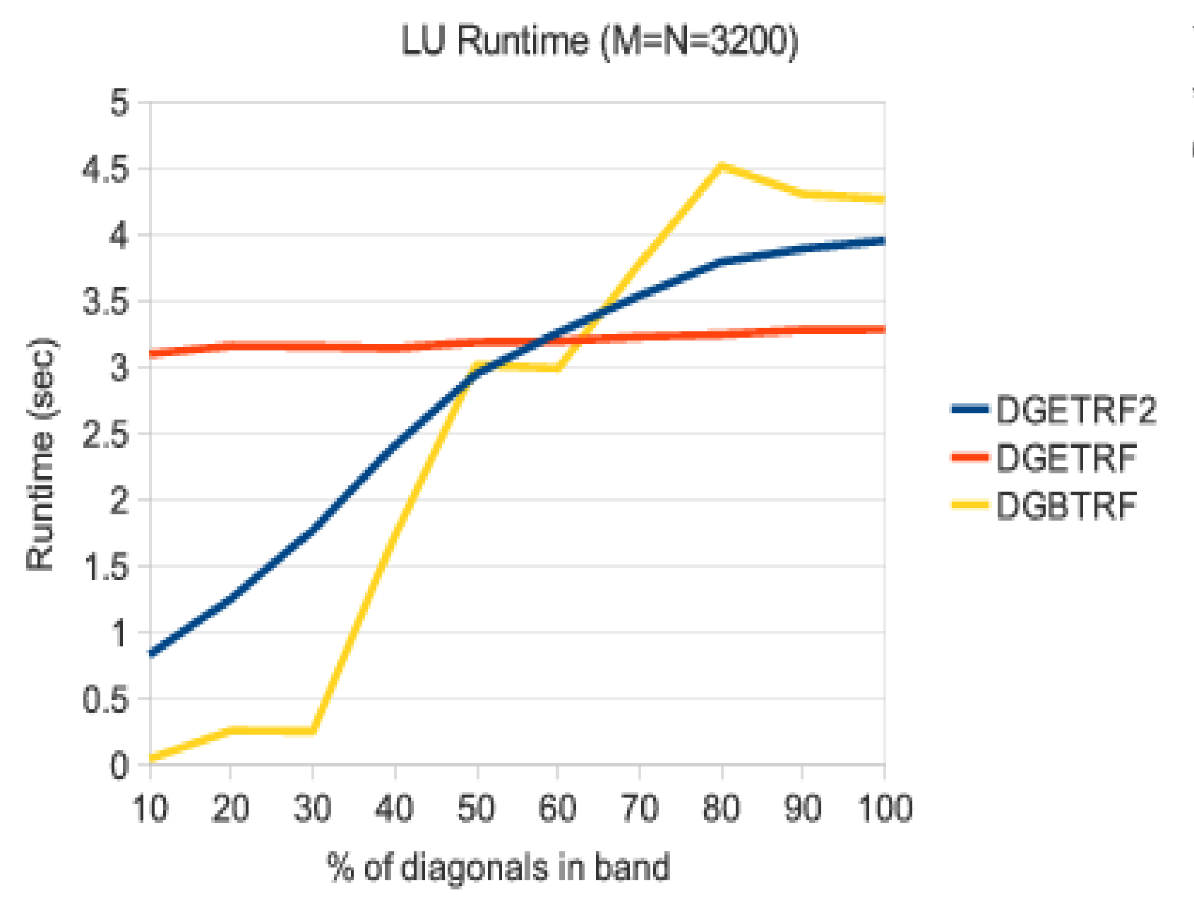
O Schur update truncation incurs little overhead, and does not significantly penalize dense matrix factorizations

O Performance benefits are obtained without copying the input matrix into a band format

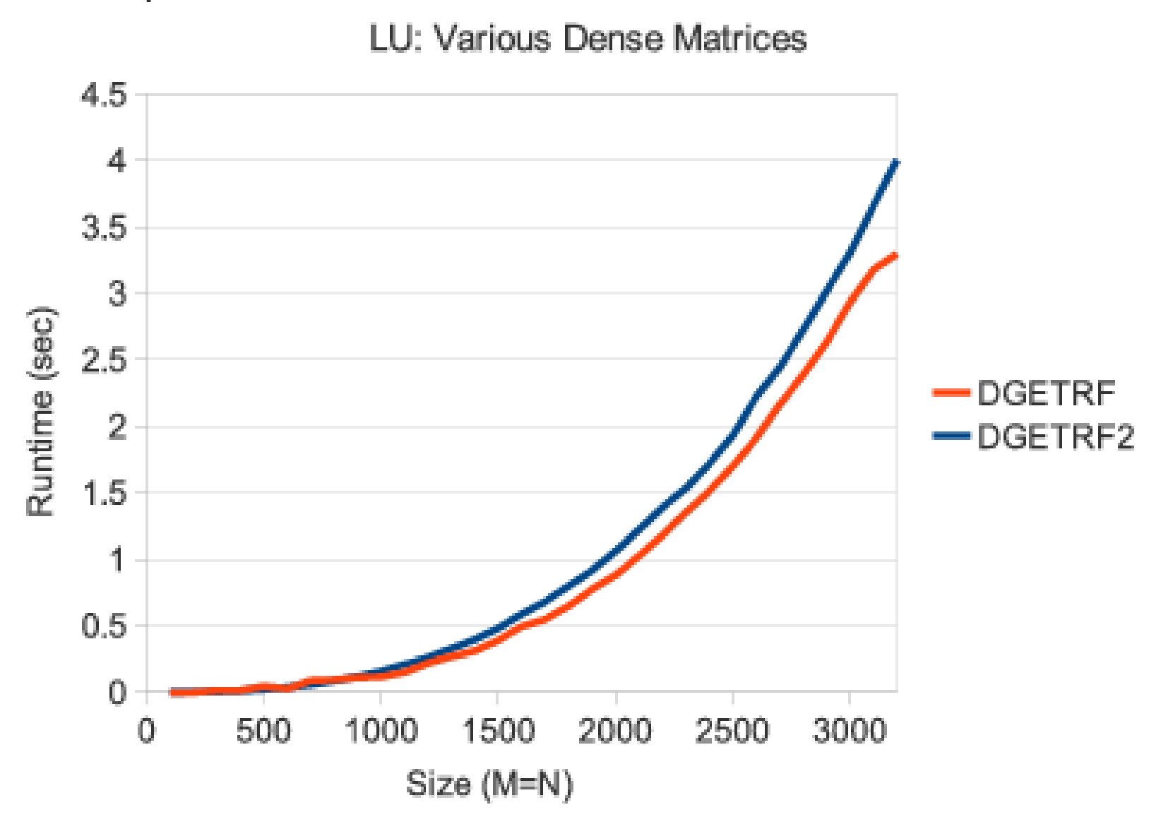
Results

To the right are performance results comparing the vanilla DGETRF ("D" for double precision), the band solver DGBTRF and the Truncated Schur update implementation (DGETRF2).

The x-axis of the plot represents the percentage of diagonals that are occupied by nonzeros. In other words, the value of 100 represents a fully dense matrix. The y-axis represents runtime on an Intel Core2Duo laptop running at 2.4Ghz. DGETRF and DGBTRF results were obtained using Intel's MKL 10.2.3.029 and DGETRF2 utilized an ATLAS-tuned BLAS library (ALTAS 3.9.17).



From the figure, it can be seen that for band matrices DGETRF2 scales in a similar manner to DGBTRF and achieves significant performance improvement over DGETRF.

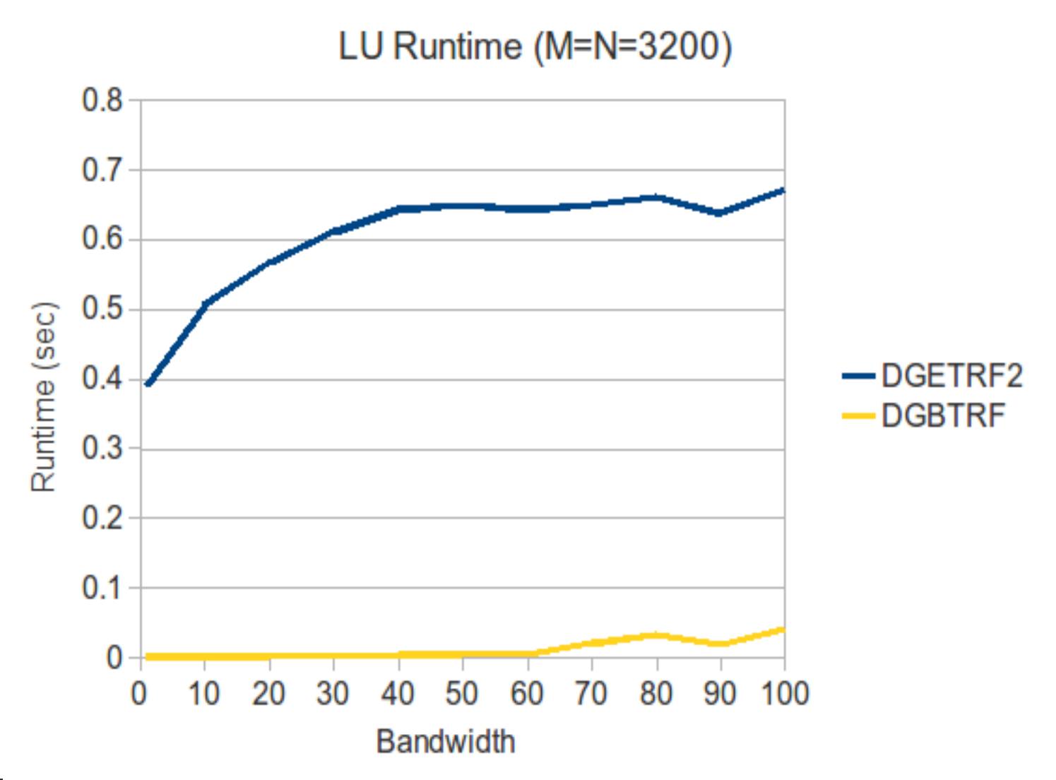


The figure on the left attempts to address possible concern about a performance penalty caused by checking for a possible Schur update truncation.

The performance results on the left compare DGETRF/DGETRF2 computation times for dense matrix factorizations. As one can see, DGETRF2 runtime suffers a small performance penalty over MKL's DGETRF for dense matrices. So, either the MKL BLAS outperforms the ATLAS BLAS, or MKL's DGETRF code includes additional optimization.

This presents a strong argument for the truncated Schur operation to be integrated into the xGETRF subroutines (with perhaps some additional optimizations for closer competition with MKL).

Results (cont)



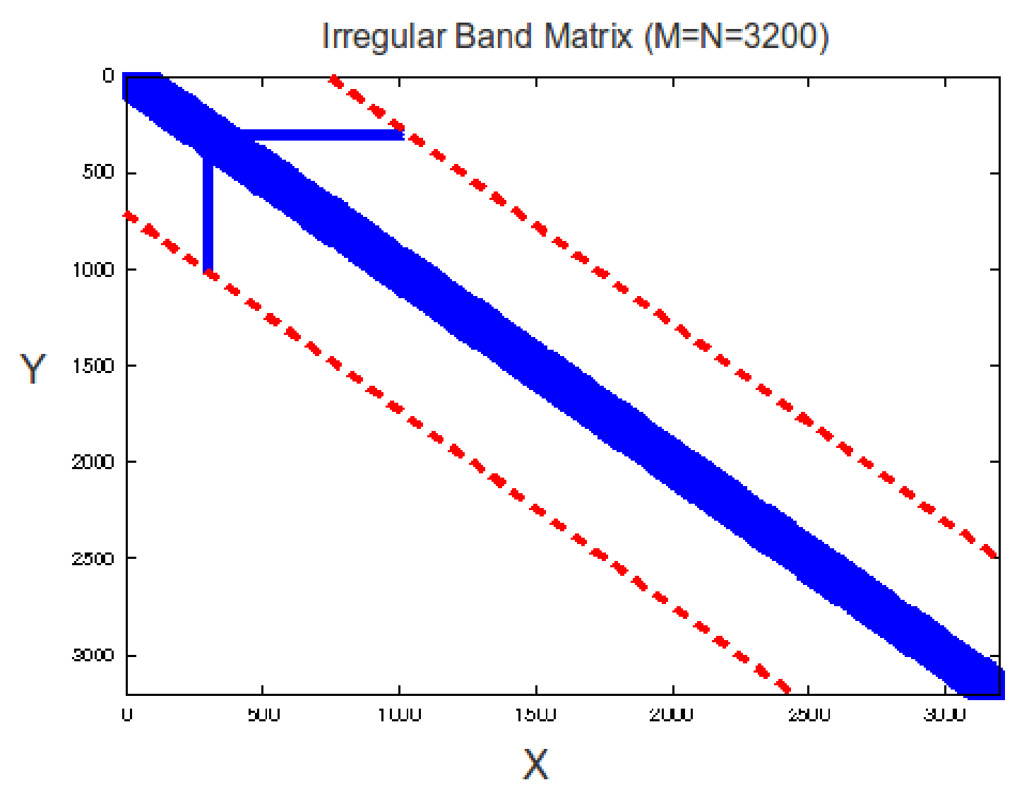
While DGETRF2 results in similar performance to the band LU (DGBTRF) for bandwidths around 10% of the available width, it suffers significantly upon matrices with very narrow bandwidth.

On the left, it is shown that DGBTRF computes much faster upon small bandwidth matrices than DGETRF2 (sometimes over an order of magnitude). Thus, the truncated Schur optimization does not make DGBTRF obsolete.

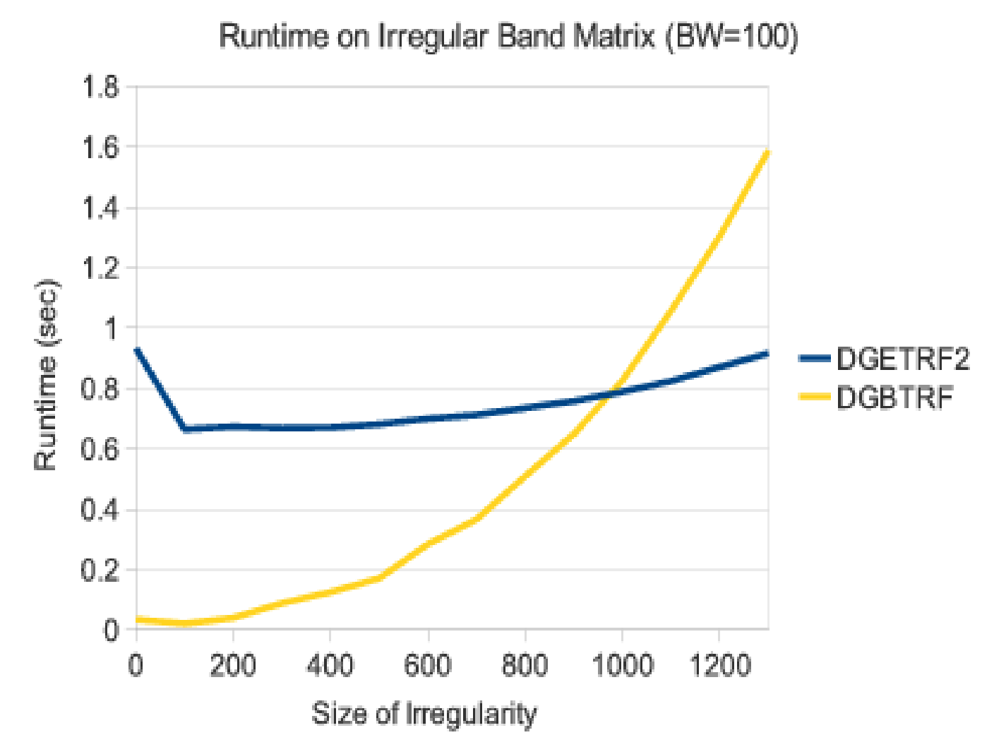
It is also worth noting that the band matrix format results in a much smaller memory footprint for low-bandwidth matrices.

Irregular Band Matrices

One limitation of LAPACK's band solvers is that they can only optimize to the extent of the largest bandwidths in the matrix. If a matrix has identical bandwidths for each row and column, this approach is optimal. However, for a matrix similar to right, the band solver must compute as if the bandwidth of the matrix is equal to the larger red lines. This results in the computation of a large number of extraneous flops.



We call matrices such as those on the right "irregular banded", as the bandwidth varies across rows and columns.



On the other hand, as DGETRF2 computes truncated Schur updates at every step of the factorization, the number of extraneous flops is significantly reduced.

In the figure to left, one can see that DGETRF2 is able to outperform DGBTRF on an irregular matrix of size 3200x3200 when the maximum bandwidth is greater than 900. The matrices used for this test were of similar format to the above figure.

Future Work

Several future goals have been set for this work:

O Complete verification of code running upon complex data, which is currently not working due to link errors

O Survey literature and experts for examples of real-world irregular band matrices that could obtain significant speedup using DGETRF2

O Perform significant testing in preparation for the integration of xGETRF2 into the LAPACK distribution

O Explore further performance improvements to LAPACK that reduce the amount of knowledge needed by the user about input matrices