Fr. Feb 20 2009

NSF Workshop: Future Directions in Tensor-Based Computation and Modeling

Software and Language: How Do We Build an Infrastructure that Supports High-Performance, Tensor-Based Computation?

Julien Langou, University of Colorado Denver.
1. Sca/LAPACK infrastructure:
   lesson from the past, reason for the success of Sca/LAPACK, present of Sca/LAPACK

2. Current direction in NLA infrastructure (and motivation for it):
   (2.a) Communication Optimal Algorithm (sequential and parallel distributed)
       i. Motivation
       ii. Design
       iii. Practice
       iv. Theory (Optimality)
   (2.b) Tiled Algorithm (multicore architecture)
       i. Design and Results
       ii. An interesting Middleware

3. Open Questions to the tensor community.
1. Sca/LAPACK infrastructure:
   lesson from the past, reason for the success of Sca/LAPACK, present of
   Sca/LAPACK

2. Current direction in NLA infrastructure (and motivation for it):
   (2.a) Communication Optimal Algorithm (sequential and parallel distributed)
      i. Motivation
      ii. Design
      iii. Practice
      iv. Theory (Optimality)
   (2.b) Tiled Algorithm (multicore architecture)
      i. Design and Results
      ii. An interesting Middleware

3. Open Questions to the tensor community.
Communication Optimal and Tiled Algorithm for “2D” Linear Algebra

Infrastructure of LAPACK

- provide routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems. The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are also provided, as are related computations such as reordering of the Schur factorizations and estimating condition numbers. Dense and banded matrices are handled, but not general sparse matrices.
- support real, complex, support single and double
- interface in Fortran (accessible for C)
- great care for manipulating NaNs, Infs, denorms
- large test suite
- rely on the BLAS for high performance
- standard interfaces
- optimize version from vendors (Intel MKL, IBM ESSL,....)
- huge community support (contribution and maintenance)
- insists on readability of the code, documentation and comments
- no memory allocation (but workspace query mechanism)
- error handling (code tries not to abort whenever possible and returns INFO value)
- tunable software through ILAENV
- open source code
- exceptional longevity! (in particular in the context of ever changing architecture)
History of LAPACK:
1.0 February 29, 1992
1.0a June 30, 1992
1.0b October 31, 1992
1.1 March 31, 1993
2.0 September 30, 1994
3.0 June 30, 1999
3.0 (update) October 31, 1999
3.0 (update) May 31, 2000
3.1 November 12, 2006
3.1.1 February 26, 2007
3.2 November 18, 2008
LAPACK 3.1
This material is based upon work supported by the National Science Foundation under Grant No. NSF-0444486.

1. **Hessenberg QR algorithm with the small bulge multi-shift QR algorithm together with aggressive early deflation.** This is an implementation of the 2003 SIAM SIAG LA Prize winning algorithm of Braman, Byers and Mathias, that significantly speeds up the nonsymmetric eigenproblem.

2. **Improvements of the Hessenberg reduction subroutines.** These accelerate the first phase of the nonsymmetric eigenvalue problem. See the reference by G. Quintana-Orti and van de Geijn below.

3. **New MRRR eigenvalue algorithms that also support subset computations.** These implementations of the 2006 SIAM SIAG LA Prize winning algorithm of Dhillon and Parlett are also significantly more accurate than the version in LAPACK 3.0.

4. **Mixed precision iterative refinement subroutines** for exploiting fast single precision hardware. On platforms like the Cell processor that do single precision much faster than double, linear systems can be solved many times faster. Even on commodity processors there is a factor of 2 in speed between single and double precision. These are prototype routines in the sense that their interfaces might changed based on user feedback.

5. **New partial column norm updating strategy for QR factorization with column pivoting.** This fixes a subtle numerical bug dating back to LINPACK that can give completely wrong results. See the reference by Drmac and Bujanovic below.

6. **Thread safety:** Removed all the SAVE and DATA statements (or provided alternate routines without those statements), increasing reliability on SMPs.

7. Additional support for matrices with **NaN/subnormal elements**, optimization of the balancing subroutines, improving reliability.
Communication Optimal and Tiled Algorithm for “2D” Linear Algebra

LAPACK 3.1
This material is based upon work supported by the National Science Foundation under Grant No. NSF-0444486.
Contributors:

1. Hessenberg QR algorithm with the small bulge multi-shift QR algorithm together with aggressive early deflation. Karen Braman and Ralph Byers, Dept. of Mathematics, University of Kansas, USA

2. Improvements of the Hessenberg reduction subroutines. Daniel Kressner, Dept. of Mathematics, University of Zagreb, Croatia

3. New MRRR eigenvalue algorithms that also support subset computations Inderjit Dhillon, University of Texas at Austin, USA Beresford Parlett, University of California at Berkeley, USA Christof Voemel, Lawrence Berkeley National Laboratory, USA


5. New partial column norm updating strategy for QR factorization with pivoting. Zlatko Drmac and Zvonomir Bujanovic, Dept. of Mathematics, University of Zagreb, Croatia

6. Thread safety: Removed all the SAVE and DATA statements (or provided alternate routines without those statements) Sven Hammarling, NAG Ltd., UK

7. Additional support for matrices with NaN/subnormal elements, optimization of the balancing subroutines Bobby Cheng, MathWorks, USA
Communication Optimal and Tiled Algorithm for “2D” Linear Algebra

LAPACK 3.1
This material is based upon work supported by the National Science Foundation under Grant No. NSF-0444486.
Thanks for bug-report/patches to

1. Eduardo Anglada (Universidad Autonoma de Madrid, Spain)
2. David Barnes (University of Kent, England)
3. Alberto Garcia (Universidad del Pais Vasco, Spain)
4. Tim Hopkins (University of Kent, England)
5. Javier Junquera (CITIMAC, Universidad de Cantabria, Spain)
6. Mathworks: Penny Anderson, Bobby Cheng, Pat Quillen, Cleve Moler, Duncan Po, Bin Shi, Greg Wolodkin (MathWorks, USA)
7. George McBane (Grand Valley State University, USA)
8. Matyas Sustik (University of Texas at Austin, USA)
9. Michael Wimmer (Universitt Regensburg, Germany)
10. Simon Wood (University of Bath, UK) and in more generally all the R developers
### ZGEEV [n=1500, jobvl=V, jobvr=N, random matrix, uniform [-1.1] entries dist]

<table>
<thead>
<tr>
<th></th>
<th>LAPACK 3.0</th>
<th>LAPACK 3.1</th>
<th>(3.0)/(3.1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>checks</td>
<td>0.00s</td>
<td>0.00s</td>
<td>(0.00%)/(0.00%)</td>
</tr>
<tr>
<td>scaling</td>
<td>0.13s</td>
<td>0.12s</td>
<td>(0.05%)/(0.20%)</td>
</tr>
<tr>
<td>balancing</td>
<td>0.08s</td>
<td>0.08s</td>
<td>(0.04%)/(0.13%)</td>
</tr>
<tr>
<td>Hessenberg reduction</td>
<td>13.39s</td>
<td>12.17s</td>
<td>(5.81%)/(19.31%)</td>
</tr>
<tr>
<td>zunghr</td>
<td>3.93s</td>
<td>3.91s</td>
<td>(1.70%)/(6.20%)</td>
</tr>
<tr>
<td>Hessenberg QR alg.</td>
<td>203.16s</td>
<td>36.81s</td>
<td>(88.10%)/(58.42%)</td>
</tr>
<tr>
<td>compute eigenvectors</td>
<td>9.82s</td>
<td>9.83s</td>
<td>(4.26%)/(15.59%)</td>
</tr>
<tr>
<td>undo balancing</td>
<td>0.09s</td>
<td>0.09s</td>
<td>(0.04%)/(0.15%)</td>
</tr>
<tr>
<td>undo scaling</td>
<td>0.00s</td>
<td>0.00s</td>
<td>(0.00%)/(0.00%)</td>
</tr>
<tr>
<td>total</td>
<td>230.60s</td>
<td>63.02s</td>
<td>(100.00%)/(100.00%)</td>
</tr>
</tbody>
</table>

ARCH: Intel Pentium 4 (3.4 GHz)
F77: GNU Fortran (GCC) 3.4.4
BLAS: libgoto_prescott32p-r1.00.so (one thread)
Communication Optimal and Tiled Algorithm for “2D” Linear Algebra

Available through Matlab (for example) since LAPACK release


There is a classical trick that enables you to access LAPACK-3.1 right now from Matlab. You can force Matlab to use LAPACK-3.1 instead of LAPACK-3.0 through a LD_PRELOAD statement. Since LAPACK-3.1 is backward compatible with LAPACK-3.0, this is safe. [...]

Here are some results for the Matlab command eig(A) where A is randn(n) matrix on Pentium IV 3.00GHz (512KB cache).

<table>
<thead>
<tr>
<th>n</th>
<th>Matlab hacked speedup</th>
<th>n</th>
<th>Matlab hacked speedup</th>
<th>n</th>
<th>Matlab hacked speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.06 0.05 1.20</td>
<td>150</td>
<td>0.17 0.12 1.91</td>
<td>200</td>
<td>0.38 0.21 1.80</td>
</tr>
<tr>
<td>250</td>
<td>0.78 0.36 2.16</td>
<td>300</td>
<td>1.35 0.58 2.32</td>
<td>350</td>
<td>2.15 0.85 2.52</td>
</tr>
<tr>
<td>400</td>
<td>3.09 1.14 2.71</td>
<td>450</td>
<td>4.65 1.51 3.07</td>
<td>500</td>
<td>6.13 1.99 3.08</td>
</tr>
<tr>
<td>550</td>
<td>8.45 2.70 3.12</td>
<td>600</td>
<td>10.43 3.43 3.04</td>
<td>650</td>
<td>13.25 4.09 3.23</td>
</tr>
<tr>
<td>700</td>
<td>16.33 4.73 3.45</td>
<td>750</td>
<td>20.40 5.55 3.67</td>
<td>800</td>
<td>24.55 6.26 3.92</td>
</tr>
<tr>
<td>850</td>
<td>29.57 7.27 4.06</td>
<td>900</td>
<td>34.15 8.25 4.13</td>
<td>950</td>
<td>40.56 9.40 4.31</td>
</tr>
<tr>
<td>1000</td>
<td>47.66 10.66 4.55</td>
<td>1050</td>
<td>55.20 12.11 4.55</td>
<td>1100</td>
<td>61.73 13.50 4.57</td>
</tr>
<tr>
<td>1150</td>
<td>71.57 15.03 4.76</td>
<td>1200</td>
<td>83.23 16.63 5.00</td>
<td>1250</td>
<td>93.10 18.32 5.08</td>
</tr>
<tr>
<td>1300</td>
<td>104.99 20.39 5.14</td>
<td>1350</td>
<td>116.85 22.50 5.19</td>
<td>1400</td>
<td>128.84 24.21 5.32</td>
</tr>
<tr>
<td>1450</td>
<td>146.04 26.95 5.41</td>
<td>1500</td>
<td>192.96 29.46 6.54</td>
<td>1550</td>
<td>24.21 5.32 3.67</td>
</tr>
<tr>
<td>1600</td>
<td>29.46 6.54 3.67</td>
<td>1650</td>
<td>32.28 5.32 3.67</td>
<td>1700</td>
<td>35.38 5.41 3.67</td>
</tr>
<tr>
<td>1750</td>
<td>40.76 6.54 3.67</td>
<td>1800</td>
<td>47.82 5.41 3.67</td>
<td>1850</td>
<td>51.47 5.41 3.67</td>
</tr>
<tr>
<td>1900</td>
<td>55.31 5.41 3.67</td>
<td>1950</td>
<td>60.74 5.41 3.67</td>
<td>2000</td>
<td>63.55 5.41 3.67</td>
</tr>
</tbody>
</table>


2. LAPACK 3.2: What's new

1. **Extra Precise Iterative Refinement**: New linear solvers that “guarantee” fully accurate answers (or give a warning that the answer cannot be trusted). The matrix types supported in this release are: GE (general), SY (symmetric), PO (positive definite), HE (Hermitian), and GB (general band) in all the relevant precisions. See reference [3] below.

2. **XBLAS, or portable “extra precise BLAS”**: Our new linear solvers in (1) depend on these to perform iterative refinement. See reference [3] below. The XBLAS will be released in a separate package. See “More Details”.

3. **Non-Negative Diagonals from Householder QR**: The QR factorization routines now guarantee that the diagonal is both real and non-negative. Factoring a uniformly random matrix now correctly generates an orthogonal Q from the Haar distribution. See reference [4] below.

4. **High Performance QR and Householder Reflections on Low-Profile Matrices**: The auxiliary routines to apply Householder reflections (e.g. DLARFB) automatically reduce the cost of QR from $O(n^3)$ to $O(n^2+nb^2)$ for matrices stored in a dense format for band matrices with bandwidth $b$ with no user interface changes. Other users of these routines can see similar benefits in particular on “narrow profile” matrices. See reference [4] below.

5. **New fast and accurate Jacobi SVD**: High accuracy SVD routine for dense matrices, which can compute tiny singular values to many more correct digits than xGESVD when the matrix has columns differing widely in norm, and usually runs faster than xGESVD too. See references [5, 6] below.

6. **Routines for Rectangular Full Packed format**: The RFP format (SF, HF, PF, TF) enables efficient routines with optimal storage for symmetric, Hermitian or triangular matrices. Since these routines utilize the Level 3 BLAS, they are generally much more efficient than the existing packed storage routines (SP, HP, PP, TP). See reference [7] below.


8. **Mixed precision iterative refinement** routines for exploiting fast single precision hardware: On platforms like the Cell processor that do single precision much faster than double, linear systems can be solved many times faster. Even on commodity processors there is a factor of 2 in speed between single and double precision. The matrix types supported in this release are: GE (general), PO (positive definite). See reference [9] below.

9. **Some new variants added for the one sided factorization**: LU gets right-looking, left-looking, Crout and Recursive, QR gets right-looking and left-looking, Cholesky gets left-looking, right-looking and top-looking. Depending on the computer architecture (or speed of the underlying BLAS), one of these variants may be faster than the original LAPACK implementation.

10. **More robust DQDS algorithm**: Fixed some rare convergence failures for the bidiagonal DQDS SVD routine.

4. External Contributors

- Ralph Byers (University of Kansas, USA)
- Zlatko Drmac (University of Zagreb, Croatia)
- Peng Du (University of Tennessee, Knoxville, USA)
- Fred Gustavson (IBM Watson Research Center, NY, US)
- Craig Lucas (University of Manchester / NAG Ltd., UK)
- Kresimir Veselic (Fernuniversitaet Hagen, Hagen, Germany)
- Jerzy Wasniewski (Technical University of Denmark, Lyngby, Copenhagen, Denmark)

5. Thanks

Thanks for bug-report/patches/suggestions to:

Patrick Alken (University of Colorado at Boulder, USA), Penny Anderson, Bobby Cheng, Cleve Moler, Duncan Po, and Pat Quillen (MathWorks, MA, USA), Michael Baudin (Scilab, FR), Michael Chuvelev (Intel, USA), Phil DeMier (IBM, USA), Michel Devel (UTINAM institute, University of Franche-Comte, UMR CNRSA, FR), Alan Edelmann (Massachusetts Institute of Technology, MA, USA), Carlo de Falco and all the Octave developers, Fernando Guevara (University of Utah, UT, USA), Christian Keil, Zbigniew Leyk (Wolfram, USA), Joao Moreira de Sa Coutinho, Lawrence Mulholland and Mick Pont (NAG, UK), Clint Whaley (University of Texas at San Antonio, TX, USA), Mikhail Wolfson (MIT, USA), Vittorio Zecca.
Infrastructure of LAPACK: test suite

- 360K lines of code in TESTING versus 490K lines of codes in SRC,
- test suite is very important for contributors/vendors to assess the validity of their implementation,
- the new BLAS (see BLAS Technical Forum, 1999) specifies a set of interfaces but not a test suite that implements these interfaces,
**Idea behind LAPACK:** rely on the BLAS for efficiency

**Blocked LU and QR algorithms (LAPACK)**

<table>
<thead>
<tr>
<th>LAPACK block LU (right-looking): dgetrf</th>
<th>LAPACK block QR (right-looking): dgeqrf</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="LU Block Diagram" /></td>
<td><img src="image2" alt="QR Block Diagram" /></td>
</tr>
<tr>
<td>dgetf2</td>
<td>dgeqf2 + dlarft</td>
</tr>
<tr>
<td>lu( )</td>
<td>qr( )</td>
</tr>
<tr>
<td>dtrsm (+ dswap)</td>
<td></td>
</tr>
<tr>
<td><img src="image3" alt="Update Diagram" /></td>
<td></td>
</tr>
<tr>
<td>dgemm</td>
<td>dlarfb</td>
</tr>
<tr>
<td><img src="image4" alt="Remaining Submatrix Diagram" /></td>
<td></td>
</tr>
</tbody>
</table>
Idea behind LAPACK: rely on the BLAS for efficiency

**Linking scenarios**

1. **LAPACK-3.1.1 (unblocked code) + Goto BLAS**

```bash
/usr/bin/gfortran -l/home/langou/opt/include ./timer_dgesv_unblocked.c
   -L/home/langou/opt/lib/
   -llapack_cwrapper -llapack -lcblas -lgoto -lpthread -o gesv_unblocked_goto.exe
```
Idea behind LAPACK: rely on the BLAS for efficiency

AMD opteron 2.2GHz (1 core)
Idea behind LAPACK: rely on the BLAS for efficiency

## Linking scenarios

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. LAPACK-3.1.1 (unblocked code) + Goto BLAS</td>
<td><code>/usr/bin/gfortran -I/home/langou/opt/include ./timer_dgesv_unblocked.c</code></td>
</tr>
<tr>
<td></td>
<td><code>-L/home/langou/opt/lib/</code></td>
</tr>
<tr>
<td></td>
<td><code>-llapack_cwrapper -llapack -lcblas -lgoto -lpthread -o gesv_unblocked_goto.exe</code></td>
</tr>
<tr>
<td>2. LAPACK-3.1.1 + reference BLAS</td>
<td><code>/usr/bin/gfortran -I/home/langou/opt/include ./timer_dgesv.c -L/home/langou/opt/lib/</code></td>
</tr>
<tr>
<td></td>
<td><code>-llapack_cwrapper -llapack -lcblas -lrefblas -o gesv_lapack_refblas.exe</code></td>
</tr>
</tbody>
</table>
Idea behind LAPACK: rely on the BLAS for efficiency

AMD opteron 2.2GHz (1 core)
Idea behind LAPACK: rely on the BLAS for efficiency

Linking scenarios

1. LAPACK-3.1.1 (unblocked code) + Goto BLAS
/usr/bin/gfortran -l/home/langou/opt/include ./timer_dgesv_unblocked.c
   -L/home/langou/opt/lib/
   -llapack_cwrapper -llapack -lcblas -lgoto -lpthread -o gesv_unblocked_goto.exe

2. LAPACK-3.1.1 + reference BLAS
/usr/bin/gfortran -l/home/langou/opt/include ./timer_dgesv.c -L/home/langou/opt/lib/
   -llapack_cwrapper -llapack -lcblas -lrefblas -o gesv_lapack_refblas.exe

3. LAPACK-3.1.1 + Goto BLAS
/usr/bin/gfortran -l/home/langou/opt/include ./timer_dgesv.c -L/home/langou/opt/lib/
   -llapack_cwrapper -llapack -lcblas -lgoto -lpthread -o gesv_lapack_goto.exe
Idea behind LAPACK: rely on the BLAS for efficiency

**AMD opteron 2.2GHz (1 core)**

![Graph showing performance over matrix size](image-url)
Idea behind LAPACK: rely on the BLAS for efficiency

\texttt{export GOTO\_NUM\_THREADS=8}
Idea behind LAPACK: rely on the BLAS for efficiency

## Linking scenarios

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. LAPACK-3.1.1 (unblocked code) + Goto BLAS</td>
<td>/usr/bin/gfortran -l/home/langou/opt/include./timer_dgesv_unblocked.c -L/home/langou/opt/lib/ -llapack_cwrapper -llapack -lcblas -lgoto -lpthread -o gesv_unblocked_goto.exe</td>
</tr>
<tr>
<td>2. LAPACK-3.1.1 + reference BLAS</td>
<td>/usr/bin/gfortran -l/home/langou/opt/include./timer_dgesv.c -L/home/langou/opt/lib/ -llapack_cwrapper -llapack -lcblas -lrblas -o gesv_lapack_refblas.exe</td>
</tr>
<tr>
<td>3. LAPACK-3.1.1 + Goto BLAS</td>
<td>/usr/bin/gfortran -l/home/langou/opt/include./timer_dgesv.c -L/home/langou/opt/lib/ -llapack_cwrapper -llapack -lcblas -lgoto -lpthread -o gesv_lapack_goto.exe</td>
</tr>
<tr>
<td>4. Goto LAPACK</td>
<td>/usr/bin/gfortran -l/home/langou/opt/include./timer_dgesv.c -L/home/langou/opt/lib/ -llapack_cwrapper -lcblas -lgoto -lpthread -o gesv_goto.exe</td>
</tr>
</tbody>
</table>
Communication Optimal and Tiled Algorithm for “2D” Linear Algebra

Idea behind LAPACK: rely on the BLAS for efficiency

GOTO - LAPACK

![Graph showing MFLOPs/sec vs matrix size (n) for different LAPACK versions](image-url)
Anatomoy of a typical LAPACK routine: DGEEV

<table>
<thead>
<tr>
<th>Category</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>comments line</td>
<td>221</td>
</tr>
<tr>
<td>do sthg useful</td>
<td>103</td>
</tr>
<tr>
<td>workspace query</td>
<td>46</td>
</tr>
<tr>
<td>declaration</td>
<td>23</td>
</tr>
<tr>
<td>error handling</td>
<td>13</td>
</tr>
<tr>
<td>total</td>
<td>423</td>
</tr>
</tbody>
</table>
Communication Optimal and Tiled Algorithm for “2D” Linear Algebra

Infrastructure of LAPACK

- provide routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems. The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are also provided, as are related computations such as reordering of the Schur factorizations and estimating condition numbers. Dense and banded matrices are handled, but not general sparse matrices.

- support real, complex, support single and double

- interface in Fortran (accessible for C)

- great care for manipulating NaNs, Infs, denorms

- large test suite

- rely on the BLAS for high performance

- standard interfaces

- optimize version from vendors (Intel MKL, IBM ESSL,...)

- huge community support (contribution and maintenance)

- insists on readability of the code, documentation and comments

- no memory allocation (but workspace query mechanism)

- error handling (code tries not to abort whenever possible and returns INFO value)

- tunable software through ILAENV

- exceptional longevity! (in particular in the context of ever changing architecture)

Infrastructure of ScaLAPACK

- The data layout is very very important and is indeed the key to the (weak) scalability of ScaLAPACK’s algorithm
1. Sca/LAPACK infrastructure: 
   lesson from the past, reason for the success of Sca/LAPACK, present of Sca/LAPACK

2. Current direction in NLA infrastructure (and motivation for it):
   (2.a) Communication Optimal Algorithm (sequential and parallel distributed)
      i. Motivation
      ii. Design
      iii. Practice
      iv. Theory (Optimality)
   (2.b) Tiled Algorithm (multicore architecture)
      i. Design and Results
      ii. An interesting Middleware

3. Open Questions to the tensor community.
### Block LU and QR algorithms (LAPACK)

<table>
<thead>
<tr>
<th>LAPACK block LU (right-looking): dgetrf</th>
<th>LAPACK block QR (right-looking): dgeqrf</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="U" alt="LU factorization" /></td>
<td><img src="R" alt="QR factorization" /></td>
</tr>
<tr>
<td><img src="L" alt="Panel factorization" /></td>
<td><img src="V" alt="Panel factorization" /></td>
</tr>
<tr>
<td><img src="U%5ET" alt="Update of the remaining submatrix" /></td>
<td><img src="V" alt="Update of the remaining submatrix" /></td>
</tr>
<tr>
<td><img src="L" alt="Update of the remaining submatrix" /></td>
<td><img src="R" alt="Update of the remaining submatrix" /></td>
</tr>
<tr>
<td><img src="A%5E%7B(1)%7D" alt="Update of the remaining submatrix" /></td>
<td><img src="A%5E%7B(1)%7D" alt="Update of the remaining submatrix" /></td>
</tr>
<tr>
<td><img src="A%5E%7B(2)%7D" alt="Update of the remaining submatrix" /></td>
<td><img src="A%5E%7B(2)%7D" alt="Update of the remaining submatrix" /></td>
</tr>
</tbody>
</table>
## Blocked LU and QR algorithms (LAPACK)

<table>
<thead>
<tr>
<th>LAPACK block LU (right-looking): dgetrf</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Diagram of blocked LU factorization" /></td>
</tr>
</tbody>
</table>

- **Panel factorization**
  - Panel factorization uses `dgetf2` and updates the remaining submatrix with `lu()`.
  - **Latency bounded**: more than `nb` AllReduce for $n*nb^2$ ops.

- **Update of the remaining submatrix**
  - Uses `dtrsm (+ dsow)` for updates.
  - **CPU - bandwidth bounded**: the bulk of the computation: $n*n*nb$ ops, highly parallelizable, efficient and scalable.

- **Final Steps**
  - Further updates with `dgemm`.
  - The algorithm is highly parallelizable, efficient and scalable.
Parallelization of LU and QR.

Parallelize the update:

• Easy and done in any reasonable software.
• This is the 2/3n^3 term in the FLOPs count.
• Can be done efficiently with LAPACK+multithreaded BLAS.

\[
L \quad U
\]

\[
L \quad U \\
A^{(1)}
\]

\[
dgetf2
\]

\[
\text{lu( } \)
\]

\[
dtrsm (+ dswp)
\]

\[
dgemm
\]

\[
A^{(2)}
\]

\[
L \quad U \\
A^{(2)}
\]
**Parallelization of LU and QR.**

<table>
<thead>
<tr>
<th>Parallelize the update:</th>
<th>dgemm</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Easy and done in any reasonable software.</td>
<td></td>
</tr>
<tr>
<td>• This is the $2/3n^3$ term in the FLOPs count.</td>
<td></td>
</tr>
<tr>
<td>• Can be done efficiently with LAPACK+multithreaded BLAS</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parallelize the panel factorization:</th>
<th>dgetf2</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Not an option in multicore context ($p &lt; 16$)</td>
<td></td>
</tr>
<tr>
<td>• See e.g. ScALAPACK or HPL but still by far the slowest and the bottleneck of the computation.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hide the panel factorization:</th>
<th>dgetf2</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Lookahead (see e.g. High Performance LINPACK)</td>
<td></td>
</tr>
<tr>
<td>• Dynamic Scheduling</td>
<td></td>
</tr>
</tbody>
</table>
Hiding the panel factorization with dynamic scheduling.

Courtesy from Alfredo Buttari, UTennessee
What about strong scalability?
What about strong scalability?

N = 1536
NB = 64
procs = 16

Courtesy from Jakub Kurzak, UTennessee
What about strong scalability?

N = 1536
NB = 64
procs = 16

We can not hide the panel factorization in the MM, actually it is the MMs that are hidden by the panel factorizations!

Courtesy from Jakub Kurzak, UTennessee
What about strong scalability?

N = 1536
NB = 64
procs = 16

We can not hide the panel factorization \( (n^2) \) with the \( \text{MM}(n^3) \), actually it is the MMs that are hidden by the panel factorizations!

NEED FOR NEW MATHEMATICAL ALGORITHMS

Courtesy from Jakub Kurzak, UTennessee
A new generation of algorithms?

<table>
<thead>
<tr>
<th>Algorithms follow hardware evolution along time.</th>
<th>LINPACK (80’s) (Vector operations)</th>
<th>LAPACK (90’s) (Blocking, cache friendly)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rely on - Level-1 BLAS operations</td>
<td>Rely on - Level-3 BLAS operations</td>
<td></td>
</tr>
</tbody>
</table>
A new generation of algorithms?

Algorithms follow hardware evolution along time.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LINPACK (80’s)</td>
<td>Rely on Level-1 BLAS</td>
<td></td>
</tr>
<tr>
<td>(Vector operations)</td>
<td>operations</td>
<td></td>
</tr>
<tr>
<td>LAPACK (90’s)</td>
<td>Rely on Level-3 BLAS</td>
<td></td>
</tr>
<tr>
<td>(Blocking, cache friendly)</td>
<td>operations</td>
<td></td>
</tr>
<tr>
<td>New Algorithms (00’s)</td>
<td>Rely on DAG/scheduler, block data layout, some extra kernels</td>
<td></td>
</tr>
<tr>
<td>(multicore friendly)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Those new algorithms
- have a very low granularity, they scale very well (multicore, petascale computing, ... )
- removes a lots of dependencies among the tasks, (multicore, distributed computing)
- avoid latency (distributed computing, out-of-core)
- rely on fast kernels

Those new algorithms need new kernels and rely on efficient scheduling algorithms.
2005-2007: New algorithms based on 2D partitionning:

- UTexas (van de Geijn): SYRK, CHOL (multicore), LU, QR (out-of-core)
- UTennessee (Dongarra): CHOL (multicore)
- HPC2N (Kågström)/IBM (Gustavson): Chol (Distributed)
- UCBerkeley (Demmel)/INRIA(Grigori): LU/QR (distributed)
- UCDenver (Langou): LU/QR (distributed)

A 3rd revolution for dense linear algebra?
Performance of the tile QR factorization in single precision on a 3.2 GHz CELL processor with eight SPEs. Square matrices were used. Solid horizontal line marks performance of the SSSRFB kernel times the number of SPEs ($22.16 \times 8 = 177$ [Gflop/s]).

“The presented implementation of tile QR factorization on the CELL processor allows for factorization of a 4000–by–4000 dense matrix in single precision in exactly half of a second. To the author’s knowledge, at present, it is the fastest reported time of solving such problem by any semiconductor device implemented on a single semiconductor die.”

Jakub Kurzak and Jack Dongarra, LAWN 201 – QR Factorization for the CELL Processor, May 2008.
Q and R: Strong scalability

In this experiment, we fix the problem: \( m=1,000,000 \) and \( n=50 \). Then we increase the number of processors.

Blue Gene L
frost.ncar.edu
Communication Optimal and Tiled Algorithm for “2D” Linear Algebra
Strategy:

1. obtain some lower bounds for the cost (latency, bandwidth, # of operations) of LU, QR and Cholesky in sequential and parallel distributed

2. compute the costs of our algorithms and compare with the lower bound.

Lower bounds:

1. For LU, observe that:

\[
\begin{pmatrix}
I & 0 & -B \\
A & I & 0 \\
0 & 0 & I
\end{pmatrix} = \begin{pmatrix}
I & 0 & -B \\
A & I & I \cdot A \\
0 & 0 & I
\end{pmatrix}
\]

therefore lower bound for matrix-matrix multiply (latency, bandwidth and operations) also holds for LU.

2. For Cholesky, observe that:

\[
\begin{pmatrix}
I & A^T & -B \\
A & I + AA^T & 0 \\
-B^T & 0 & D
\end{pmatrix} = \begin{pmatrix}
I & 0 & -B \\
A & I & I \cdot A \\
-B^T & (A \cdot B)^T & X
\end{pmatrix}
\]


Performance models of parallel CAQR and ScaLAPACK’s parallel QR factorization PDGEQRF on a square $n \times n$ matrix with $P$ processors, along with lower bounds on the number of flops, words, and messages. The matrix is stored in a 2-D $P_r \times P_c$ block cyclic layout with square $b \times b$ blocks. We choose $b$, $P_r$, and $P_c$ optimally and independently for each algorithm. Everything (messages, words, and flops) is counted along the critical path.

<table>
<thead>
<tr>
<th></th>
<th>Par. CAQR</th>
<th>PDGEQRF</th>
<th>Lower bound</th>
</tr>
</thead>
<tbody>
<tr>
<td># flops</td>
<td>$\frac{4n^3}{3P}$</td>
<td>$\frac{4n^3}{3P}$</td>
<td>$O\left(\frac{n^3}{P}\right)$</td>
</tr>
<tr>
<td># words</td>
<td>$\frac{3n^2}{4\sqrt{P}} \log P$</td>
<td>$\frac{3n^2}{4\sqrt{P}} \log P$</td>
<td>$O\left(\frac{n^2}{\sqrt{P}}\right)$</td>
</tr>
<tr>
<td># messages</td>
<td>$\frac{3}{8} \sqrt{P} \log^3 P$</td>
<td>$\frac{5n}{4} \log^2 P$</td>
<td>$O\left(\sqrt{P}\right)$</td>
</tr>
</tbody>
</table>
### Performance models of parallel CAQR and ScaLAPACK’s parallel QR factorization PDGEQRF on a square $n \times n$ matrix with $P$ processors, along with lower bounds on the number of flops, words, and messages. The matrix is stored in a 2-D $P_r \times P_c$ block cyclic layout with square $b \times b$ blocks. We choose $b$, $P_r$, and $P_c$ optimally and independently for each algorithm. Everything (messages, words, and flops) is counted along the critical path.

<table>
<thead>
<tr>
<th></th>
<th>Par. CAQR</th>
<th>PDGEQRF</th>
<th>Lower bound</th>
</tr>
</thead>
<tbody>
<tr>
<td># flops</td>
<td>$\frac{4n^3}{3P}$</td>
<td>$\frac{4n^3}{3P}$</td>
<td>$O\left(\frac{n^3}{P}\right)$</td>
</tr>
<tr>
<td># words</td>
<td>$\frac{3n^2}{4\sqrt{P}} \log P$</td>
<td>$\frac{3n^2}{4\sqrt{P}} \log P$</td>
<td>$O\left(\frac{n^2}{\sqrt{P}}\right)$</td>
</tr>
<tr>
<td># messages</td>
<td>$\frac{3}{8} \sqrt{P} \log^3 P$</td>
<td>$\frac{5n}{4} \log^2 P$</td>
<td>$O\left(\sqrt{P}\right)$</td>
</tr>
</tbody>
</table>

David Mahoney (this morning): **NLA: everything is $O(n^3)$.**
<table>
<thead>
<tr>
<th>Seq. CAQR</th>
<th>Householder QR</th>
<th>Lower bound</th>
</tr>
</thead>
<tbody>
<tr>
<td># flops</td>
<td>$\frac{4}{3} n^3$</td>
<td>$\frac{4}{3} n^3$</td>
</tr>
<tr>
<td># words</td>
<td>$3 \frac{n^3}{\sqrt{W}}$</td>
<td>$\frac{1}{3} n^4$</td>
</tr>
<tr>
<td># messages</td>
<td>$12 \frac{n^3}{W^{3/2}}$</td>
<td>$\frac{1}{2} n^3$</td>
</tr>
</tbody>
</table>

Performance models of sequential CAQR and blocked sequential Householder QR on a square $n \times n$ matrix with fast memory size $W$, along with lower bounds on the number of flops, words, and messages.
An interesting middleware: SMPSs

```c
#pragma css task
    inout(RV1[NB][NB]) output(T[NB][NB])
void dgeqrt(double *RV1, double *T);

#pragma css task
    inout(R[NB][NB], V2[NB][NB]) output(T[NB][NB])
void dtsqrt(double *R, double *V2, double *T);

#pragma css task
    input(V1[NB][NB], T[NB][NB]) inout(C1[NB][NB])
void dlarfb(double *V1, double *T, double *C1);

#pragma css task
    input(V2[NB][NB], T[NB][NB]) inout(C1[NB][NB], C2[NB][NB])
void dssrfb(double *V2, double *T, double *C1, double *C2);

#pragma css start
for (k = 0; k < TILES; k++) {
    dgeqrt(A[k][k], T[k][k]);
    for (m = k+1; m < TILES; m++)
        dtsqrt(A[k][k], A[m][k], T[m][k]);
    for (n = k+1; n < TILES; n++)
        dlarfb(A[k][k], T[k][k], A[k][n]);
    for (m = k+1; m < TILES; m++)
        dssrfb(A[m][k], T[m][k], A[k][n], A[m][n]);
}
#pragma css finish
```

From:


See also:


1. Sca/LAPACK infrastructure:
   lesson from the past, reason for the success of Sca/LAPACK, present of Sca/LAPACK

2. Current direction in NLA infrastructure (and motivation for it):
   (2.a) Communication Optimal Algorithm (sequential and parallel distributed)
      i. Motivation
      ii. Design
      iii. Practice
      iv. Theory (Optimality)
   (2.b) Tiled Algorithm (multicore architecture)
      i. Design and Results
      ii. An interesting Middleware

3. Open Questions to the tensor community.
Advices.
DO.

DON’T
Communication Optimal and Tiled Algorithm for “2D” Linear Algebra

Infrastructure of LAPACK

- provide routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems. The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are also provided, as are related computations such as reordering of the Schur factorizations and estimating condition numbers. Dense and banded matrices are handled, but not general sparse matrices.

- support real, complex, support single and double

- interface in Fortran (accessible for C)

- great care for manipulating NaNs, Infs, denorms

- large test suite

- rely on the BLAS for high performance

- standard interfaces

- optimize version from vendors (Intel MKL, IBM ESSL,...)

- huge community support (contribution and maintenance)

- insists on readability of the code, documentation and comments

- no memory allocation (but workspace query mechanism)

- error handling (code tries not to abort whenever possible and returns INFO value)

- tunable software through ILAENV

- exceptional longevity! (in particular in the context of ever changing architecture)
Advices.

**DO.**

1. use high level languages, abstraction, and software layers
Advices.

**DO.**

1. use high level languages, abstraction, and software layers
2. use auto-tuning (at least make your software tunable)
Advices.

DO.

1. use high level languages, abstraction, and software layers
2. use auto-tuning (at least make your software tunable)
3. support both C and Fortran, complex and real
Advices.

**DO.**

1. use high level languages, abstraction, and software layers
2. use auto-tuning (at least make your software tunable)
3. support both C and Fortran, complex and real
4. makes the software easily maintainable
Advices.

**DO.**

1. use high level languages, abstraction, and software layers
2. use auto-tuning (at least make your software tunable)
3. support both C and Fortran, complex and real
4. makes the software easily maintainable
5. think your interface twice and think them as a community (involvement)
Advices.

**DO.**

1. use high level languages, abstraction, and software layers
2. use auto-tuning (at least make your software tunable)
3. support both C and Fortran, complex and real
4. makes the sofware easily maintanable
5. think your interface twice and think them as a community (involvment)
6. test suite!
Advices.

**DO.**

1. use high level languages, abstraction, and software layers
2. use auto-tuning (at least make your software tunable)
3. support both C and Fortran, complex and real
4. makes the software easily maintainable
5. think your interface twice and think them as a community (involvement)
6. test suite!

**DON’T**

1. disagree on interfaces!
   - PETSc vs Trilinos
   - ScaLAPACK vs PLAPACK

Most of the disagreement boils down to data structure...
Advices.

**DO.**

1. use high level languages, abstraction, and software layers
2. use auto-tuning (at least make your software tunable)
3. support both C and Fortran, complex and real
4. makes the sofware easily maintainable
5. think your interface twice and think them as a community (involvment)
6. test suite!

**DON’T**

1. disagree on interfaces!
   - PETSc vs Trilinos
   - ScaLAPACK vs PLAPACK
   Most of the disagreement boils down to data structure ...
2. change your interfaces (LAPACK-2.0 and ARPACK)
Two open questions to the tensor community and NSF:
Two open questions to the tensor community and NSF:

1. **Funding model will favor**
   
   (1.a) *One software unit them all:*
   
   - LAPACK
   - ScaLAPACK
   
   (1.b) *healthy competition:*
   
   - MPI (LA-MPI, LAM-MPI, MPICH, FT-MPI),
   - Symmetric Eigenvalue Sparse (BLOPEX, JADAMILU, PRIMME),
   - Sparse Direct (MUMPS, SuperLu, UMFPACK),
   - Sparse Iter. (Petsc, Sparskit, Trilinos)
Two open questions to the tensor community and NSF:

1. **Funding model will favor**
   
   (1.a) *One software unit them all:*
   
   - LAPACK
   - ScaLAPACK

   (1.b) *Healthy competition:*
   
   - MPI (LA-MPI, LAM-MPI, MPICH, FT-MPI),
   - Symmetric Eigenvalue Sparse (BLOPEX, JADAMILU, PRIMME),
   - Sparse Direct (MUMPS, SuperLu, UMFPACK),
   - Sparse Iter. (Petsc, Sparskit, Trilinos)

2. **Sustainability of the software.**
   
   If you spent two years developing a software, you would like it to last at least ??? Through NSF Infrastructure grant?