Overview of Dense Numerical Linear Algebra Libraries

- **BLAS**: kernel for dense linear algebra
- **LAPACK**: sequential dense linear algebra
- **ScaLAPACK**: parallel distributed dense linear algebra
- And Beyond
What is dense linear algebra?

• Not just matmul!
• Linear Systems: $Ax=b$
• Least Squares: choose $x$ to minimize $\|Ax-b\|_2$
  – Overdetermined or underdetermined
  – Unconstrained, constrained, weighted
• Eigenvalues and vectors of Symmetric Matrices
  • Standard ($Ax = \lambda x$), Generalized ($Ax=\lambda Bx$)
• Eigenvalues and vectors of Unsymmetric matrices
  • Eigenvalues, Schur form, eigenvectors, invariant subspaces
  • Standard, Generalized
• Singular Values and vectors (SVD)
  – Standard, Generalized
• Different matrix structures
  – Real, complex; Symmetric, Hermitian, positive definite; dense, triangular, banded …
• Level of detail
  – Simple Driver (“$x=A\backslash b$”)
  – Expert Drivers with error bounds, extra-precision, other options
  – Lower level routines (“apply certain kind of orthogonal transformation”, matmul…)
Dense Linear Algebra

- **Common Operations**

\[
Ax = b; \quad \min_x \| Ax - b \|; \quad Ax = \lambda x
\]

- A major source of large dense linear systems is problems involving the solution of boundary integral equations.
  - The price one pays for replacing three dimensions with two is that what started as a sparse problem in \( O(n^3) \) variables is replaced by a dense problem in \( O(n^2) \).

- Dense systems of linear equations are found in numerous other applications, including:
  - airplane wing design;
  - radar cross-section studies;
  - flow around ships and other off-shore constructions;
  - diffusion of solid bodies in a liquid;
  - noise reduction; and
  - diffusion of light through small particles.
## Existing Math Software - Dense LA

<table>
<thead>
<tr>
<th>DIRECT SOLVERS</th>
<th>License</th>
<th>Support</th>
<th>Type</th>
<th>Language</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Real</td>
<td>Complex</td>
<td>F77</td>
</tr>
<tr>
<td><strong>Eigen</strong></td>
<td>Mozilla</td>
<td>yes</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>Elemental</strong></td>
<td>BSD</td>
<td>yes</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>FLAME</strong></td>
<td>LGPL</td>
<td>yes</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>FLENS</strong></td>
<td>BSD</td>
<td>yes</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td><strong>LAPACK</strong></td>
<td>BSD</td>
<td>yes</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>LAPACK95</strong></td>
<td>BSD</td>
<td>yes</td>
<td>X</td>
<td>X</td>
<td>F95</td>
</tr>
<tr>
<td><strong>MAGMA</strong></td>
<td>BSD</td>
<td>yes</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>NAPACK</strong></td>
<td>BSD</td>
<td>yes</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>PLAPACK</strong></td>
<td>?</td>
<td>no</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>PLASMA</strong></td>
<td>BSD</td>
<td>yes</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>PRISM</strong></td>
<td>?</td>
<td>no</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>rejtrix</strong></td>
<td>by-nc-sa</td>
<td>yes</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ScalAPACK</strong></td>
<td>BSD</td>
<td>yes</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>Trilinos/Pliris</strong></td>
<td>BSD</td>
<td>yes</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>ViennaCL</strong></td>
<td>MIT</td>
<td>yes</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[http://www.netlib.org/utk/people/JackDongarra/la-sw.html](http://www.netlib.org/utk/people/JackDongarra/la-sw.html)

- LINPACK, EISPACK, LAPACK, ScaLAPACK
  - PLASMA, MAGMA
A brief history of (Dense) Linear Algebra software

- In the beginning was the do-loop…
  - Libraries like EISPACK (for eigenvalue problems)
- Then the BLAS (1) were invented (1973-1977)
  - Standard library of 15 operations (mostly) on vectors
    - “AXPY” ( \( y = \alpha \cdot x + y \) ), dot product, scale (\( x = \alpha \cdot x \) ), etc
    - Up to 4 versions of each (S/D/C/Z), 46 routines, 3300 LOC
  - Goals
    - Common “pattern” to ease programming, readability
    - Robustness, via careful coding (avoiding over/underflow)
    - Portability + Efficiency via machine specific implementations
- Why BLAS 1? They do \( O(n^1) \) ops on \( O(n^1) \) data
- Used in libraries like LINPACK (for linear systems)
  - Source of the name “LINPACK Benchmark” (not the code!)
What Is LINPACK?

• LINPACK is a package of mathematical software for solving problems in linear algebra, mainly dense linear systems of linear equations.

• LINPACK: "LINear algebra PACKage"
  ➢ Written in Fortran 66

• The project had its origins in 1974

• The project had four primary contributors: myself when I was at Argonne National Lab, Jim Bunch from the University of California-San Diego, Cleve Moler who was at New Mexico at that time, and Pete Stewart from the University of Maryland.

• LINPACK as a software package has been largely superseded by LAPACK, which has been designed to run efficiently on shared-memory, vector supercomputers.
Computing in 1974

- High Performance Computers:
  - IBM 370/195, CDC 7600, Univac 1110, DEC PDP-10, Honeywell 6030
- Fortran 66
- Trying to achieve software portability
- Run efficiently
- BLAS (Level 1)
  - Vector operations
- Software released in 1979
  - About the time of the Cray 1
The Linpack Benchmark is a measure of a computer’s floating-point rate of execution.

- It is determined by running a computer program that solves a dense system of linear equations.

Over the years the characteristics of the benchmark has changed a bit.

- In fact, there are three benchmarks included in the Linpack Benchmark report.

LINPACKBenchmark

- Dense linear system solve with LU factorization using partial pivoting
- Operation count is: \( \frac{2}{3} n^3 + O(n^2) \)
- Benchmark Measure: MFlop/s
- Original benchmark measures the execution rate for a Fortran program on a matrix of size 100x100.
Accidental Benchmarker

**Appendix B of the Linpack Users’ Guide**
- Designed to help users extrapolate execution time for Linpack software package
- First benchmark report from 1977;
  - Cray 1 to DEC PDP-10

<table>
<thead>
<tr>
<th>Facility</th>
<th>TIME (sec)</th>
<th>UNIT</th>
<th>Computer</th>
<th>Type</th>
<th>Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>NCAR</td>
<td>14.0</td>
<td>0.14</td>
<td>CRAY-1</td>
<td>S</td>
<td>CFT, Assembly BLAS</td>
</tr>
<tr>
<td>LASL</td>
<td>3.57</td>
<td>0.36</td>
<td>CDC 7600</td>
<td>S</td>
<td>FTN, Assembly BLAS</td>
</tr>
<tr>
<td>NCAR</td>
<td>3.34</td>
<td>0.61</td>
<td>CDC 7600</td>
<td>S</td>
<td>FTN</td>
</tr>
<tr>
<td>Argonne</td>
<td>2.31</td>
<td>0.86</td>
<td>IBM 370/195</td>
<td>D</td>
<td>H</td>
</tr>
<tr>
<td>NCAR</td>
<td>1.84</td>
<td>1.10</td>
<td>CDC 7600</td>
<td>S</td>
<td>Local</td>
</tr>
<tr>
<td>Argonne</td>
<td>1.77</td>
<td>1.33</td>
<td>IBM 3033</td>
<td>D</td>
<td>H</td>
</tr>
<tr>
<td>NASA Langley</td>
<td>1.96</td>
<td>1.42</td>
<td>CDC Cyber 173</td>
<td>S</td>
<td>FTN</td>
</tr>
<tr>
<td>Montana State</td>
<td>1.40</td>
<td>1.47</td>
<td>CDC Cyber 173</td>
<td>S</td>
<td>Ext. 4.6</td>
</tr>
<tr>
<td>LBL</td>
<td>1.04</td>
<td>1.53</td>
<td>CDC 7600</td>
<td>S</td>
<td>CHT, No optimize</td>
</tr>
<tr>
<td>SLAC</td>
<td>1.84</td>
<td>1.79</td>
<td>IBM 370/168</td>
<td>D</td>
<td>H Ext., Fast mult.</td>
</tr>
<tr>
<td>Michigan</td>
<td>1.89</td>
<td>1.84</td>
<td>Amdahl 470/V4</td>
<td>D</td>
<td>H</td>
</tr>
<tr>
<td>Toronto</td>
<td>1.77</td>
<td>2.19</td>
<td>IBM 370/165</td>
<td>D</td>
<td>H Ext., Fast mult.</td>
</tr>
<tr>
<td>Northwestern</td>
<td>1.81</td>
<td>4.20</td>
<td>CDC 6600</td>
<td>S</td>
<td>FTN</td>
</tr>
<tr>
<td>Texas</td>
<td>1.35</td>
<td>5.63</td>
<td>CDC 6600</td>
<td>S</td>
<td>RUN</td>
</tr>
<tr>
<td>China Lake</td>
<td>1.22</td>
<td>5.69</td>
<td>Univac 1110</td>
<td>S</td>
<td>V</td>
</tr>
<tr>
<td>Yale</td>
<td>1.25</td>
<td>7.53</td>
<td>DEC KL-20</td>
<td>S</td>
<td>F20</td>
</tr>
<tr>
<td>Bell Labs</td>
<td>1.73</td>
<td>10.1</td>
<td>Honeywell 6080</td>
<td>S</td>
<td>Y</td>
</tr>
<tr>
<td>Wisconsin</td>
<td>1.83</td>
<td>10.1</td>
<td>Univac 1110</td>
<td>S</td>
<td>V</td>
</tr>
<tr>
<td>Iowa State</td>
<td>1.54</td>
<td>10.1</td>
<td>Intel 8053 mod3</td>
<td>D</td>
<td>H</td>
</tr>
<tr>
<td>U. Ill. Chicago</td>
<td>1.61</td>
<td>11.9</td>
<td>IBM 370/158</td>
<td>D</td>
<td>61</td>
</tr>
<tr>
<td>Purdue</td>
<td>1.56</td>
<td>16.6</td>
<td>CDC 6500</td>
<td>S</td>
<td>FUN</td>
</tr>
<tr>
<td>U. C. San Diego</td>
<td>1.31</td>
<td>38.2</td>
<td>Burroughs 6700</td>
<td>S</td>
<td>H</td>
</tr>
<tr>
<td>Yale</td>
<td>1.07</td>
<td>49.9</td>
<td>DEC KA-10</td>
<td>S</td>
<td>E40</td>
</tr>
</tbody>
</table>

* TIME(100) = (100/75)**3 * SGEFA(75) + (100/75)**2 * SGESL(75)
## High Performance Linpack (HPL)

<table>
<thead>
<tr>
<th>Benchmark Name</th>
<th>Matrix dimension</th>
<th>Optimizations allowed</th>
<th>Parallel Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linpack 100</td>
<td>100</td>
<td>compiler</td>
<td>—(^a)</td>
</tr>
<tr>
<td>Linpack 1000(^b)</td>
<td>1000</td>
<td>hand, code replacement</td>
<td>—(^c)</td>
</tr>
<tr>
<td>Linpack Parallel</td>
<td>1000</td>
<td>hand, code replacement</td>
<td>Yes</td>
</tr>
<tr>
<td>HPLinpack(^d)</td>
<td>arbitrary</td>
<td>hand, code replacement</td>
<td>Yes</td>
</tr>
</tbody>
</table>

\(^a\) Compiler parallelization possible.

\(^b\) Also known as TPP (Toward Peak Performance) or Best Effort

\(^c\) Multiprocessor implementations allowed.

\(^d\) Highly-Parallel LINPACK Benchmark is also known as NxN Linpack Benchmark or High Parallel Computing (HPC).
Main points
• Factorization column (zero) mostly sequential due to memory bottleneck
• Level 1 BLAS
• Divide pivot row has little parallelism
• Rank -1 Schur complement update is the only easy parallelize task
• Partial pivoting complicates things even further
• Bulk synchronous parallelism (fork-join)
  • Load imbalance
  • Non-trivial Amdahl fraction in the panel
  • Potential workaround (look-ahead) has complicated implementation
A brief history of (Dense) Linear Algebra software

**But the BLAS-1 weren't enough**

- Consider AXPY \( y = \alpha \cdot x + y \): 2n flops on 3n read/writes
- Computational intensity = \( (2n)/(3n) = 2/3 \)
- Too low to run near peak speed (read/write dominates)

**So the BLAS-2 were developed (1984-1986)**

- Standard library of 25 operations (mostly) on matrix/vector pairs
  - "GEMV": \( y = \alpha \cdot A \cdot x + \beta \cdot x \), "GER": \( A = A + \alpha \cdot x \cdot y^T \), \( x = T^{-1} \cdot x \)
  - Up to 4 versions of each (S/D/C/Z), 66 routines, 18K LOC
- Why BLAS 2? They do \( O(n^2) \) ops on \( O(n^2) \) data
- So computational intensity still just \( \sim (2n^2)/(n^2) = 2 \)
  - OK for vector machines, but not for machine with caches
A brief history of (Dense) Linear Algebra software


- Standard library of 9 operations (mostly) on matrix/matrix pairs
  - "GEMM": $C = \alpha \cdot A \cdot B + \beta \cdot C$, $C = \alpha \cdot A \cdot A^T + \beta \cdot C$, $B = T^{-1} \cdot B$
  - Up to 4 versions of each (S/D/C/Z), 30 routines, 10K LOC

Why BLAS 3? They do $O(n^3)$ ops on $O(n^2)$ data

So computational intensity $(2n^3)/(4n^2) = n/2$ - big at last!

Good for machines with caches, other mem. hierarchy levels

How much BLAS1/2/3 code so far (all at www.netlib.org/blas)

- Source: 142 routines, 31K LOC, Testing: 28K LOC
- Reference (unoptimized) implementation only
  - Ex: 3 nested loops for GEMM
Algorithms have two costs:

1. Arithmetic (FLOPS)

2. Communication: moving data between
   - levels of a memory hierarchy (sequential case)
   - processors over a network (parallel case).
Memory Hierarchy

By taking advantage of the principle of locality:

- Present the user with as much memory as is available in the cheapest technology.
- Provide access at the speed offered by the fastest technology.

**Speed (ns):** 1s, 10s, 100s, 100s, 10,000,000s (10s ms), 10,000,000,000s (10s sec)

**Size (bytes):** 100s, Ks, Ms, Gs, Ts
Why Higher Level BLAS?

- Can only do arithmetic on data at the top of the hierarchy
- Higher level BLAS lets us do this

<table>
<thead>
<tr>
<th>BLAS</th>
<th>Memory Refs</th>
<th>Flops</th>
<th>Flops/ Memory Refs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1 ( y = y + \alpha x )</td>
<td>3n</td>
<td>2n</td>
<td>2/3</td>
</tr>
<tr>
<td>Level 2 ( y = y + Ax )</td>
<td>( n^2 )</td>
<td>2n(^2)</td>
<td>2</td>
</tr>
<tr>
<td>Level 3 ( C = C + AB )</td>
<td>4n(^2)</td>
<td>2n(^3)</td>
<td>n/2</td>
</tr>
</tbody>
</table>
Level 1, 2 and 3 BLAS

- Level 1 BLAS
  Vector-Vector operations

- Level 2 BLAS
  Matrix-Vector operations

- Level 3 BLAS
  Matrix-Matrix operations
Level 1, 2 and 3 BLAS

1 core Intel Xeon E5-2670 (Sandy Bridge); 2.6 GHz; Peak = 20.8 Gflop/s

1 core Intel Xeon E5-2670 (Sandy Bridge); 2.6 GHz.
24 MB shared L3 cache, and each core has a private 256 KB L2 and 64 KB L1.
The theoretical peak per core DP is 8 flop/cycle * 2.6 GHz = 20.8 Gflop/s per core.
Compiled with gcc 4.4.6 and using MKL_composer_xe_2013.3.163
### Level 1 BLAS

<table>
<thead>
<tr>
<th>Routine</th>
<th>dim</th>
<th>scalar/vector</th>
<th>vector/scalar</th>
<th>5-element array</th>
</tr>
</thead>
<tbody>
<tr>
<td>xGEMV</td>
<td>TRANS, N, M, ALPHA, X, Y, INCX, BETA, Y, INCY</td>
<td>y := αAx + βy = αATx + βy, y := αAHx + βy, A - m x n</td>
<td>S, D</td>
<td>C, Z</td>
</tr>
<tr>
<td>xGEMV</td>
<td>TRANS, N, M, ALPHA, A, X, INCX, Y, INCY</td>
<td>y := αAx + βy = αATx + βy, y := αAHx + βy, A - m x n</td>
<td>S, D</td>
<td>C, Z</td>
</tr>
<tr>
<td>xHERV</td>
<td>TRANS, N, M, ALPHA, A, X, INCX, Y, INCY</td>
<td>y := αAx + βy = αATx + βy, y := αAHx + βy, A - m x n</td>
<td>S, D</td>
<td>C, Z</td>
</tr>
<tr>
<td>xHERV</td>
<td>TRANS, N, M, ALPHA, A, X, INCY, Y, INCY</td>
<td>y := αAx + βy = αATx + βy, y := αAHx + βy, A - m x n</td>
<td>S, D</td>
<td>C, Z</td>
</tr>
<tr>
<td>xHEPP</td>
<td>TRANS, N, M, ALPHA, A, X, INCX, AP</td>
<td>y := αAx + βy = αATx + βy, y := αAHx + βy, A - m x n</td>
<td>S, D</td>
<td>C, Z</td>
</tr>
<tr>
<td>xHEP2</td>
<td>TRANS, N, M, ALPHA, X, INCY, Y, INCY</td>
<td>A := αAxT + A, A := αATx + A</td>
<td>S, D</td>
<td>C, Z</td>
</tr>
<tr>
<td>xHEP2</td>
<td>TRANS, N, M, ALPHA, X, INCX, AP</td>
<td>A := αAxT + A, A := αATx + A</td>
<td>S, D</td>
<td>C, Z</td>
</tr>
<tr>
<td>xHEP2</td>
<td>TRANS, N, M, ALPHA, X, INCX, Y, INCY</td>
<td>A := αAxT + A, A := αATx + A</td>
<td>S, D</td>
<td>C, Z</td>
</tr>
<tr>
<td>xHEP2</td>
<td>TRANS, N, M, ALPHA, X, INCY, Y, INCX</td>
<td>A := αAxT + A, A := αATx + A</td>
<td>S, D</td>
<td>C, Z</td>
</tr>
</tbody>
</table>

### Level 2 BLAS

<table>
<thead>
<tr>
<th>Routine</th>
<th>dim</th>
<th>b-vector</th>
<th>scalar/scalar</th>
<th>vector/scalar</th>
</tr>
</thead>
<tbody>
<tr>
<td>xGEMV</td>
<td>TRANS, N, M, ALPHA, X, INCX, Y, INCY</td>
<td>y := αAx + βy = αATx + βy, y := αAHx + βy, A - m x n</td>
<td>S, D</td>
<td>C, Z</td>
</tr>
<tr>
<td>xHERV</td>
<td>TRANS, N, M, ALPHA, A, X, INCX, Y, INCY</td>
<td>y := αAx + βy = αATx + βy, y := αAHx + βy, A - m x n</td>
<td>S, D</td>
<td>C, Z</td>
</tr>
<tr>
<td>xHEP2</td>
<td>TRANS, N, M, ALPHA, X, INCX, AP</td>
<td>A := αAxT + A, A := αATx + A</td>
<td>S, D</td>
<td>C, Z</td>
</tr>
<tr>
<td>xHEP2</td>
<td>TRANS, N, M, ALPHA, X, INCY, Y, INCY</td>
<td>A := αAxT + A, A := αATx + A</td>
<td>S, D</td>
<td>C, Z</td>
</tr>
<tr>
<td>xHEP2</td>
<td>TRANS, N, M, ALPHA, X, INCY, Y, INCX</td>
<td>A := αAxT + A, A := αATx + A</td>
<td>S, D</td>
<td>C, Z</td>
</tr>
<tr>
<td>xHEP2</td>
<td>TRANS, N, M, ALPHA, X, INCX, Y, INCY</td>
<td>A := αAxT + A, A := αATx + A</td>
<td>S, D</td>
<td>C, Z</td>
</tr>
</tbody>
</table>

### Level 3 BLAS

<table>
<thead>
<tr>
<th>Routine</th>
<th>dim</th>
<th>scalar/scalar</th>
<th>matrix/scalar</th>
<th>matrix/scalar</th>
</tr>
</thead>
<tbody>
<tr>
<td>xGEMV</td>
<td>TRANS, N, M, ALPHA, X, INCX, Y, INCY</td>
<td>C := αop(A)op(B) + C, C := op(A)X, X := ATx, A := A, A := A</td>
<td>S, D</td>
<td>C, Z</td>
</tr>
<tr>
<td>xHERV</td>
<td>TRANS, N, M, ALPHA, A, X, INCX, Y, INCY</td>
<td>C := αop(A)op(B) + C, C := op(A)X, X := ATx, A := A, A := A</td>
<td>S, D</td>
<td>C, Z</td>
</tr>
<tr>
<td>xHEP2</td>
<td>TRANS, N, M, ALPHA, X, INCX, AP</td>
<td>A := αAxT + A, A := αATx + A</td>
<td>S, D</td>
<td>C, Z</td>
</tr>
<tr>
<td>xHEP2</td>
<td>TRANS, N, M, ALPHA, X, INCY, Y, INCY</td>
<td>A := αAxT + A, A := αATx + A</td>
<td>S, D</td>
<td>C, Z</td>
</tr>
<tr>
<td>xHEP2</td>
<td>TRANS, N, M, ALPHA, X, INCY, Y, INCX</td>
<td>A := αAxT + A, A := αATx + A</td>
<td>S, D</td>
<td>C, Z</td>
</tr>
<tr>
<td>xHEP2</td>
<td>TRANS, N, M, ALPHA, X, INCX, Y, INCY</td>
<td>A := αAxT + A, A := αATx + A</td>
<td>S, D</td>
<td>C, Z</td>
</tr>
</tbody>
</table>
A brief history of (Dense) Linear Algebra software

• LAPACK – “Linear Algebra PACKage” - uses BLAS-3 (1989 – now)
  – Ex: Obvious way to express Gaussian Elimination (GE) is adding multiples of one row to other rows – BLAS-1
    • How do we reorganize GE to use BLAS-3? (details later)
  – Contents of LAPACK (summary)
    • Algorithms we can turn into (nearly) 100% BLAS 3
      – Linear Systems: solve Ax=b for x
      – Least Squares: choose x to minimize ||Ax - b||_2
    • Algorithms that are only 50% BLAS 3 (so far)
      – “Eigenproblems”: Find \( \lambda \) and x where Ax = \( \lambda \) x
      – Singular Value Decomposition (SVD): \((A^T A)x=\sigma^2 x\)
    • Generalized problems (eg Ax = \( \lambda \) Bx)
    • Error bounds for everything
    • Lots of variants depending on A’s structure (banded, A=A^T, etc)
  – How much code? (Release 3.5.0, Nov 2013) (www.netlib.org/lapack)
    • Source: 1674 routines, 490K LOC, Testing: 448K LOC
A brief history of (Dense) Linear Algebra software

• Is LAPACK parallel?
  – Only if the BLAS are parallel (possible in shared memory)

• ScaLAPACK – “Scalable LAPACK” (1995 – now)
  – For distributed memory – uses MPI
  – More complex data structures, algorithms than LAPACK
    • Only (small) subset of LAPACK’s functionality available
  – All at www.netlib.org/scalapack
LAPACK

• http://www.netlib.org/lapack/

• LAPACK (Linear Algebra Package) provides routines for
  – solving systems of simultaneous linear equations,
  – least-squares solutions of linear systems of equations,
  – eigenvalue problems,
  – and singular value problems.

• LAPACK relies on BLAS

• 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. In all areas, similar functionality is provided for real and complex matrices, in both single and double precision.
Main points:
- Factorization column (zero) mostly sequential due to memory bottleneck
- Level 1 BLAS
- Divide pivot row has little parallelism
- Rank -1 Schur complement update is the only easy parallelize task
- Partial pivoting complicates things even further
- Bulk synchronous parallelism (fork-join)
  - Load imbalance
  - Non-trivial Amdahl fraction in the panel
  - Potential workaround (look-ahead) has complicated implementation
Main points
• Panel factorization mostly sequential due to memory bottleneck
• Triangular solve has little parallelism
• Schur complement update is the only easy parallelize task
• Partial pivoting complicates things even further
• Bulk synchronous parallelism (fork-join)
  • Load imbalance
  • Non-trivial Amdahl fraction in the panel
  • Potential workaround (look-ahead) has complicated implementation
Example with GESV

Solve a system of linear equations using a LU factorization

\[
\text{subroutine } \text{dgesv}( n, \text{nrhs}, A, \text{lda}, \text{ipiv}, b, \text{ldb}, \text{info } )
\]

**input:**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>ipiv</th>
<th>info</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n \times n )</td>
<td>( n \times \text{nrhs} )</td>
<td>( n )</td>
<td>( n )</td>
</tr>
</tbody>
</table>

**output:**

<table>
<thead>
<tr>
<th>L</th>
<th>U</th>
<th>X</th>
<th>ipiv</th>
<th>info</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n \times n )</td>
<td>( n \times \text{nrhs} )</td>
<td>( n )</td>
<td>( n )</td>
<td></td>
</tr>
</tbody>
</table>
# Functionalities in LAPACK

<table>
<thead>
<tr>
<th>Type of Problem</th>
<th>Acronyms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear system of equations</td>
<td>SV</td>
</tr>
<tr>
<td>Linear least squares problems</td>
<td>LLS</td>
</tr>
<tr>
<td>Linear equality-constrained least squares problem</td>
<td>LSE</td>
</tr>
<tr>
<td>General linear model problem</td>
<td>GLM</td>
</tr>
<tr>
<td>Symmetric eigenproblems</td>
<td>SEP</td>
</tr>
<tr>
<td>Nonsymmetric eigenproblems</td>
<td>NEP</td>
</tr>
<tr>
<td>Singular value decomposition</td>
<td>SVD</td>
</tr>
<tr>
<td>Generalized symmetric definite eigenproblems</td>
<td>GSEPNP</td>
</tr>
<tr>
<td>Generalized nonsymmetric eigenproblems</td>
<td>GNEP</td>
</tr>
<tr>
<td>Generalized (or quotient) singular value decomposition</td>
<td>GSVD (QSVD)</td>
</tr>
</tbody>
</table>
LAPACK Software

• First release in February 1992
• Version 3.5.0 released in November 2013
• LICENSE: Mod-BSD, freely-available software package - Thus, it can be included in commercial software packages (and has been). We only ask that proper credit be given to the authors.
• Open SVN repository
• Multi-OS
  – *nix, Mac OS/X, Windows
• Multi-build support (cmake)
  – make, xcode, nmake, VS studio, Eclipse, etc..
• LAPACKE: Standard C language APIs for LAPACK (In collaboration with INTEL)
  – 2 layers of interface
    • High-Level Interface : Workspace allocation and NAN Check
    • Low-Level Interface
• Prebuilt Libraries for Windows
• Extensive test suite
• Forum and User support: http://icl.cs.utk.edu/lapack-forum/
Latest Algorithms

Since release 3.0 of LAPACK

- Hessenberg QR algorithm with the small bulge multi-shift QR algorithm together with aggressive early deflation. [2003 SIAM SIAG LA Prize winning algorithm of Braman, Byers and Mathias]
- Improvements of the Hessenberg reduction subroutines. [G. Quintana-Ortí and van de Geijn]
- New MRRR eigenvalue algorithms [2006 SIAM SIAG LA Prize winning algorithm of Dhillon and Parlett]
- New partial column norm updating strategy for QR factorization with column pivoting. [Drmač and Bujanovic]
- Mixed Precision Iterative Refinement for exploiting fast single precision hardware for GE, PO [Langou’s]

3.1

- Variants of various factorization (LU, QR, Chol) [Du]
- RFP (Rectangular Full Packed) format [Gustavson, Langou]
- XBLAS and Extra precise iterative refinement for GESV [Demmel et al.]
- New fast and accurate Jacobi SVD [2009 SIAM SIAG LA Prize, Drmač and Veselić]
- Pivoted Cholesky [Lucas]
- Better multishift Hessenberg QR algorithm with early aggressive deflation [Byers]

3.2

- Complete CS decomposition [Sutton]
- Level-3 BLAS symmetric indefinite solve and symmetric indefinite inversion [Langou’s]
- Since LAPACK 3.3, all routines in are now thread-safe

3.3
Latest Algorithms

Since release 3.0 of LAPACK

- xGEQRT: QR factorization (improved interface). [James (UC Denver)]
- xGEQRT3: recursive QR factorization. [James (UC Denver)]. The recursive QR factorization enables cache-oblivious and enables high performance on **tall and skinny matrices**.
- xTPQRT: **Communication-Avoiding** QR sequential kernels. [James (UC Denver)]
- CMAKE build system. **Building under Windows has never been easier.** This also allows us to release dll for Windows, so users no longer need a Fortran compiler to use LAPACK under Windows.
- Doxygen documentation. **LAPACK routine documentation has never been more accessible.** See [http://www.netlib.org/lapack/explore-html/](http://www.netlib.org/lapack/explore-html/).
- New website allowing for easier navigation.
- LAPACKE - Standard C language APIs for LAPACK.

- Symmetric/Hermitian LDLT factorization routines with rook pivoting algorithm [Kozachenko]
- 2-by-1 CSD to be used for tall and skinny matrix with orthonormal columns [Brian Sutton]
- New stopping criteria for balancing [Rodney James]
- New complex division algorithm [Victor Liu]
LAPACKE – the C Standard Interface to LAPACK

• Two Interfaces for two kinds of users
  – High Level Interface: Ease of use Users
    • No more WORK parameter, allocation done for the user
    • Optional NaN check on all vector/matrix inputs before calling any LAPACK FORTRAN routine
    • Support Row and Column major formats
  – Low Level Interface: Expert users and libraries
    • Low overhead and user control are the two main priorities here

• Other features
  – Include and configuration files
  – INFO parameter dropped in the C Interface to LAPACK, and instead routine returns an integer
  – Handle ANSI C99 (default), C structure option, CPP complex type and custom types for complex types

Need Fortran and LAPACK at linking time
Resources

Reference Code:
- Reference code: (current version 3.4.2)
  
  http://www.netlib.org/lapack/lapack.tgz
- LAPACK build for windows (current version 3.4.2)
  
  http://icl.cs.utk.edu/lapack-for-windows/lapack
- LAPACKE: Standard C language APIs for LAPACK (in collaboration with INTEL):
  
  http://www.netlib.org/lapack/#_standard_c_language/apis_for_lapack
- Remi’s wrappers (wrapper for Matlab users):
  
  http://icl.cs.utk.edu/~delmas/lapwrapmw.htm

Vendor Libraries:
more or less same as the BLAS: MKL, ACML, VECLIB, ESSL, etc...

(WARNING: some implementations are just a subset of LAPACK)

Documentation:
- LAPACK Users’ guide:
  
  http://www.netlib.org/lapack/lug/
- LAPACK Working notes (in particular LAWN 41)
  
  http://www.netlib.org/lapack/lawns/downloads/
- LAPACK release notes
  
  http://www.netlib.org/lapack/lapack-3.4.2.html
- LAPACK NAG example and auxiliary routines
  
  

Support:
- LAPACK forum: (more than 1000 topics)
  
  http://icl.cs.utk.edu/lapack-forum/
- LAPACK mailing-list:
  
  lapack@cs.utk.edu
- LAPACK mailing-list archive:
  
  http://icl.cs.utk.edu/lapack-forum/archives/
Organizing Linear Algebra – in books

www.netlib.org/lapack

www.netlib.org/scalapack

www.netlib.org/templates

www.cs.utk.edu/~dongarra/etemplates
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

\[
\begin{align*}
  & d\text{getrf}^2 \\
  & d\text{trsm (+ dswp)} \\
  & d\text{gemm} \\
  & d\text{getrf}^2
\end{align*}
\]
Overview of Dense Numerical Linear Algebra Libraries

- **BLAS**: kernel for dense linear algebra
- **LAPACK**: sequential dense linear algebra
- **ScalAPACK**: parallel distributed dense linear algebra
ScaLAPACK

- Library of software dealing with dense & banded routines
- Distributed Memory - Message Passing
- MIMD Computers and Networks of Workstations
- Clusters of SMPs
ScaLAPACK

- [http://www.netlib.org/scalapack/](http://www.netlib.org/scalapack/)

- ScaLAPACK (Scalable Linear Algebra Package) provides routines for
  - solving systems of simultaneous linear equations,
  - least-squares solutions of linear systems of equations,
  - eigenvalue problems,
  - and singular value problems.

- Relies on LAPACK / BLAS and BLACS / MPI

- Includes PBLAS (Parallel BLAS)

ScaLAPACK is in FORTRAN and C

ScaLAPACK is for PARALLEL DISTRIBUTED

ScaLAPACK is a REFERENCE implementation
Programming Style

- SPMD Fortran 77 with object based design
- Built on various modules
  - PBLAS Interprocessor communication
  - BLACS
    - PVM, MPI, IBM SP, CRI T3, Intel, TMC
    - Provides right level of notation.
  - BLAS
- LAPACK software expertise/quality
  - Software approach
  - Numerical methods
Overall Structure of Software

- **Object based - Array descriptor**
  - Contains information required to establish mapping between a global array entry and its corresponding process and memory location.
  - Provides a flexible framework to easily specify additional data distributions or matrix types.
  - Currently dense, banded, & out-of-core

- Using the concept of context
PBLAS

Similar to the BLAS in functionality and naming.

Built on the BLAS and BLACS

Provide global view of matrix

CALL DGEXXX ( M, N, A( IA, JA ), LDA,... )

CALL PDGEXXX( M, N, A, IA, JA, DESCA,... )
Choosing a Data Distribution

Main issues are:

- Load balancing
- Use of the Level 3 BLAS
Possible Data Layouts

- **1D block and cyclic column distributions**

- **1D block-cycle column and 2D block-cyclic distribution**

- **2D block-cyclic used in ScaLAPACK for dense matrices**
From LAPACK to ScaLAPACK

[LAPACK] subroutine dgesv( n, nrhs, a(ia,ja), lda, ipiv, b(ib,jb), ldb, info )

input:

- A: n x n matrix
- B: n x nrhs matrix
- ipiv: n x 1 matrix
- info: 1 x 1 matrix

output:

- L: n x n lower triangular matrix
- U: n x n upper triangular matrix
- X: n x nrhs matrix
- ipiv: n x 1 matrix
- info: 1 x 1 matrix

LAPACK Data layout
From LAPACK to ScaLAPACK

[LAPACK] subroutine `dgesv`( n, nrhs, a(ia,ja), lda, ipiv, b(ib,jb), ldb, info )

**input:**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>n</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>A_{11}</code></td>
<td><code>A_{12}</code></td>
<td><code>A_{13}</code></td>
<td><code>B_{11}</code></td>
<td><code>ip_1</code></td>
<td></td>
</tr>
<tr>
<td><code>A_{21}</code></td>
<td><code>A_{22}</code></td>
<td><code>A_{23}</code></td>
<td><code>B_{21}</code></td>
<td><code>ip_2</code></td>
<td></td>
</tr>
<tr>
<td><code>A_{31}</code></td>
<td><code>A_{32}</code></td>
<td><code>A_{33}</code></td>
<td><code>B_{31}</code></td>
<td><code>ip_3</code></td>
<td></td>
</tr>
</tbody>
</table>

**output:**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>n</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>L_{11}</code></td>
<td><code>U_{11}</code></td>
<td><code>U_{12}</code></td>
<td><code>U_{13}</code></td>
<td><code>ip_1</code></td>
<td></td>
</tr>
<tr>
<td><code>L_{21}</code></td>
<td><code>U_{21}</code></td>
<td><code>U_{22}</code></td>
<td><code>U_{23}</code></td>
<td><code>ip_2</code></td>
<td></td>
</tr>
<tr>
<td><code>L_{31}</code></td>
<td><code>U_{31}</code></td>
<td><code>U_{32}</code></td>
<td><code>U_{33}</code></td>
<td><code>ip_3</code></td>
<td></td>
</tr>
</tbody>
</table>

ScaLAPACK Data layout

info
From LAPACK to ScaLAPACK

[LAPACK] subroutine dgesv( n, nrhs, a(ia,ja), lda, ipiv, b(ib,jb), ldb, info )
[ScaLAPACK] subroutine pdgesv( n, nrhs, a, ia, ja, desca, ipiv, b, ib, jb, descb, info )

input:

output:
Distribution and Storage

Matrix is block-partitioned & maps blocks

Distributed 2-D block-cyclic scheme

5x5 matrix partitioned in 2x2 blocks
2x2 process grid point of view

Routines available to distribute/redistribute data.
2D Block Cyclic Layout

Matrix point of view

Processor point of view

Matrix is MxN
Process grid is PxQ, P=2, Q=3
Blocks are MBxNB
### 2D Block Cyclic Layout

<table>
<thead>
<tr>
<th>Matrix point of view</th>
<th>Processor point of view</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Matrix point of view:

- Row 1: 0 0 0
- Row 2: 2 2 2
- Row 3: 4 4 4
- Row 4: 6 6 6
- Row 5: 8 8 8

Processor point of view:

- Column 1: 1 1 1
- Column 2: 3 3 3
- Column 3: 5 5 5
2D Block Cyclic Layout

Matrix point of view

Processor point of view
## 2D Block Cyclic Layout

### Matrix point of view

<table>
<thead>
<tr>
<th>0</th>
<th>2</th>
<th>4</th>
<th>0</th>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

### Processor point of view

<table>
<thead>
<tr>
<th>0 0 0</th>
<th>2 2 2</th>
<th>4 4 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>2 2 2</td>
<td>4 4 4</td>
</tr>
<tr>
<td>0 0 0</td>
<td>2 2 2</td>
<td>4 4 4</td>
</tr>
<tr>
<td>0 0 0</td>
<td>2 2 2</td>
<td>4 4 4</td>
</tr>
<tr>
<td>0 0 0</td>
<td>2 2 2</td>
<td>4 4 4</td>
</tr>
<tr>
<td>0 0 0</td>
<td>2 2 2</td>
<td>4 4 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1 1 1</th>
<th>3 3 3</th>
<th>5 5 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1</td>
<td>3 3 3</td>
<td>5 5 5</td>
</tr>
<tr>
<td>1 1 1</td>
<td>3 3 3</td>
<td>5 5 5</td>
</tr>
<tr>
<td>1 1 1</td>
<td>3 3 3</td>
<td>5 5 5</td>
</tr>
<tr>
<td>1 1 1</td>
<td>3 3 3</td>
<td>5 5 5</td>
</tr>
<tr>
<td>1 1 1</td>
<td>3 3 3</td>
<td>5 5 5</td>
</tr>
</tbody>
</table>
### 2D Block Cyclic Layout

<table>
<thead>
<tr>
<th>Matrix point of view</th>
<th>Processor point of view</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 2 4 0 2 4 0 2 4</td>
<td>0 0 0 2 2 2 4 4 4</td>
</tr>
<tr>
<td>1 3 5 1 3 5 1 3 5</td>
<td>0 0 0 2 2 2 4 4 4</td>
</tr>
<tr>
<td>0 2 4 0 2 4 0 2 4</td>
<td>0 0 0 2 2 2 4 4 4</td>
</tr>
<tr>
<td>1 3 5 1 3 5 1 3 5</td>
<td>0 0 0 2 2 2 4 4 4</td>
</tr>
<tr>
<td>0 2 4 0 2 4 0 2 4</td>
<td>1 1 1 3 3 3 5 5 5</td>
</tr>
<tr>
<td>1 3 5 1 3 5 1 3 5</td>
<td>1 1 1 3 3 3 5 5 5</td>
</tr>
<tr>
<td>0 2 4 0 2 4 0 2 4</td>
<td>1 1 1 3 3 3 5 5 5</td>
</tr>
<tr>
<td>1 3 5 1 3 5 1 3 5</td>
<td>1 1 1 3 3 3 5 5 5</td>
</tr>
<tr>
<td>0 2 4 0 2 4 0 2 4</td>
<td>1 1 1 3 3 3 5 5 5</td>
</tr>
</tbody>
</table>

Matrix point of view vs. Processor point of view.
### 2D Block Cyclic Layout

**Matrix point of view**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

**Processor point of view**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

**Matrix point of view**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

**Processor point of view**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
2D Block Cyclic Layout

Matrix point of view

Processor point of view
2D Block Cyclic Layout

Matrix point of view

Processor point of view

Matrix point of view

Processor point of view
# 2D Block Cyclic Layout

<table>
<thead>
<tr>
<th>Matrix point of view</th>
<th>Processor point of view</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Matrix layout" /></td>
<td><img src="image2" alt="Processor layout" /></td>
</tr>
</tbody>
</table>

**Matrix layout**

- 2D block cyclic layout visualization.

**Processor layout**

- 2D block cyclic layout visualization from a processor perspective.
2D Block Cyclic Layout

Matrix point of view

Processor point of view
2D Block Cyclic Layout

Matrix point of view

Processor point of view
Parallelism in ScaLAPACK

- **Level 3 BLAS block operations**
  - All the reduction routines

- **Pipelining**
  - QR Algorithm, Triangular Solvers, classic factorizations

- **Redundant computations**
  - Condition estimators

- **Static work assignment**
  - Bisection

- **Task parallelism**
  - Sign function eigenvalue computations

- **Divide and Conquer**
  - Tridiagonal and band solvers, symmetric eigenvalue problem and Sign function

- **Cyclic reduction**
  - Reduced system in the band solver

- **Data parallelism**
  - Sign function
# Functionalities in LAPACK

<table>
<thead>
<tr>
<th>Type of Problem</th>
<th>Acronyms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear system of equations</td>
<td>SV</td>
</tr>
<tr>
<td>Linear least squares problems</td>
<td>LLS</td>
</tr>
<tr>
<td>Linear equality-constrained least squares problem</td>
<td>LSE</td>
</tr>
<tr>
<td>General linear model problem</td>
<td>GLM</td>
</tr>
<tr>
<td>Symmetric eigenproblems</td>
<td>SEP</td>
</tr>
<tr>
<td>Nonsymmetric eigenproblems</td>
<td>NEP</td>
</tr>
<tr>
<td>Singular value decomposition</td>
<td>SVD</td>
</tr>
<tr>
<td>Generalized symmetric definite eigenproblems</td>
<td>GSEPNP</td>
</tr>
<tr>
<td>Generalized nonsymmetric eigenproblems</td>
<td>GNEP</td>
</tr>
<tr>
<td>Generalized (or quotient) singular value decomposition</td>
<td>GSVD (QSVD)</td>
</tr>
</tbody>
</table>
# Functionnalities in ScaLAPACK

<table>
<thead>
<tr>
<th>Type of Problem</th>
<th>Acronyms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear system of equations</td>
<td>SV</td>
</tr>
<tr>
<td>Linear least squares problems</td>
<td>LLS</td>
</tr>
<tr>
<td>Linear equality-constrained least squares problem</td>
<td>LSE</td>
</tr>
<tr>
<td>General linear model problem</td>
<td>GLM</td>
</tr>
<tr>
<td>Symmetric eigenproblems</td>
<td>SEP</td>
</tr>
<tr>
<td>Nonsymmetric eigenproblems</td>
<td>NEP</td>
</tr>
<tr>
<td>Singular value decomposition</td>
<td>SVD</td>
</tr>
<tr>
<td>Generalized symmetric definite eigenproblems</td>
<td>GSEPN</td>
</tr>
<tr>
<td>Generalized nonsymmetric eigenproblems</td>
<td>GNENP</td>
</tr>
<tr>
<td>Generalized (or quotient) singular value decomposition</td>
<td>GSVD (QSVD)</td>
</tr>
</tbody>
</table>
Major Changes to Software

• **Must rethink the design of our software**
  - Another disruptive technology
    - Similar to what happened with cluster computing and message passing
  - Rethink and rewrite the applications, algorithms, and software

• **Numerical libraries for example will change**
  - For example, both LAPACK and ScaLAPACK will undergo major changes to accommodate this
Moore’s Law Reinterpreted

- Number of cores per chip doubles every 2 years, while clock speed decreases (not increases).
  - Need to deal with systems with millions of concurrent threads
  - Future generation will have billions of threads!
  - Need to be able to easily replace inter-chip parallelism with intro-chip parallelism

- Number of threads of execution doubles every 2 years
Key Challenges at Exascale

- Levels of parallelism
  - $O(100M \text{ and beyond})$
- Hybrid architectures
  - Node composed of multiple multicore sockets + accelerators
- Bandwidth vs Arithmetic rate
  - Most approaches assume flops expensive
- Storage Capacity
  - Issue of weak scalability in future systems
- Fault occurrence; shared responsibility
  - Process failure recovery

- Power Management
  - API for fine grain management
- Language constraints
  - Fortran, C & MPI, Open-MP
- Autotuning
  - Systems complex and changing
- Bulk Sync Processing
  - Break fork join parallelism
- Lack of reproducibility; unnecessarily expensive (most of the time)
  - Can’t guarantee bitwise results
- Need for effective scheduling of tasks
# A New Generation of DLA Software

<table>
<thead>
<tr>
<th>Software/Algorithms follow hardware evolution in time</th>
<th>LINPACK (70’s) (Vector operations)</th>
<th>LAPACK (80’s) (Blocking, cache friendly)</th>
<th>ScaLAPACK (90’s) (Distributed Memory)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rely on</td>
<td>Rely on</td>
<td>Rely on</td>
<td></td>
</tr>
<tr>
<td>- Level-1 BLAS operations</td>
<td>- Level-3 BLAS operations</td>
<td>- PBLAS Mess Passing</td>
<td></td>
</tr>
</tbody>
</table>

## 2D Block Cyclic Layout

<table>
<thead>
<tr>
<th>Matrix point of view</th>
<th>Processor point of view</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 2 4 0 2 4 0 2 4</td>
<td>0 0 2 2 2 4 4 4 4</td>
</tr>
<tr>
<td>1 3 5 1 3 5 1 3 5</td>
<td>0 0 2 2 2 4 4 4 4</td>
</tr>
<tr>
<td>0 2 4 0 2 4 0 2 4</td>
<td>0 0 2 2 2 4 4 4 4</td>
</tr>
<tr>
<td>1 3 5 1 3 5 1 3 5</td>
<td>0 0 2 2 2 4 4 4 4</td>
</tr>
<tr>
<td>0 2 4 0 2 4 0 2 4</td>
<td>0 0 2 2 2 4 4 4 4</td>
</tr>
<tr>
<td>1 3 5 1 3 5 1 3 5</td>
<td>0 0 2 2 2 4 4 4 4</td>
</tr>
<tr>
<td>0 2 4 0 2 4 0 2 4</td>
<td>0 0 2 2 2 4 4 4 4</td>
</tr>
<tr>
<td>1 3 5 1 3 5 1 3 5</td>
<td>0 0 2 2 2 4 4 4 4</td>
</tr>
<tr>
<td>0 2 4 0 2 4 0 2 4</td>
<td>0 0 2 2 2 4 4 4 4</td>
</tr>
</tbody>
</table>
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

Fork - Join parallelism
Bulk Sync Processing
Synchronization (in LAPACK LU)

Fork-join, bulk synchronous processing

Allowing for delayed update, out of order, asynchronous, dataflow execution
PLASMA LU Factorization

Dataflow Driven
• Tile data layout where each data tile is contiguous in memory
• Decomposed into several fine-grained tasks, which better fit the memory of the small core caches
PLASMA LU: Tile Algorithm and Nested Parallelism

• Operates on one, two, or three matrix tiles at a time using a single core
  – This is called a kernel; executed independently of other kernels
  – Mostly Level 3 BLAS are used

• Data flows between kernels as prescribed by the programmer

• Coordination is done transparently via runtime scheduler (QUARK)
  – Parallelism level adjusted at runtime
  – Look-ahead adjusted at runtime

• Uses single-threaded BLAS with all the optimization benefits

• Panel is done on multiple cores
  – Recursive formulation of LU for better BLAS use
  – Level 1 BLAS are faster because they work on combined cache size
FOR $k = 0..TILES-1$
  $A[k][k] \leftarrow \text{DPOTRF}(A[k][k])$
  FOR $m = k+1..TILES-1$
    $A[m][k] \leftarrow \text{DTRSM}(A[k][k], A[m][k])$
  FOR $m = k+1..TILES-1$
    $A[m][m] \leftarrow \text{DSYRK}(A[m][k], A[m][m])$
    FOR $n = k+1..m-1$
      $A[m][n] \leftarrow \text{DGEMM}(A[m][k], A[n][k], A[m][n])$
Parallel Linear Algebra s/w for Multicore/Hybrid Architectures

● Objectives
  ▪ High utilization of each core
  ▪ Scaling to large number of cores
  ▪ Synchronization reducing algorithms

● Methodology
  ▪ Dynamic DAG scheduling (QUARK)
  ▪ Explicit parallelism
  ▪ Implicit communication
  ▪ Fine granularity / block data layout

● Arbitrary DAG with dynamic scheduling

Diagram:
- Fork-join parallelism
- DAG scheduled parallelism
- Time
- Cores
PLASMA Local Scheduling

Dynamic Scheduling: Sliding Window

- DAGs get very big, very fast
  - So windows of active tasks are used; this means no global critical path
  - Matrix of NBxNB tiles; \( NB^3 \) operation
    - \( NB=100 \) gives 1 million tasks
Dynamic Scheduling: Sliding Window

- DAGs get very big, very fast
  - So windows of active tasks are used; this means no global critical path
- Matrix of $NB \times NB$ tiles; $NB^3$ operation
  - $NB=100$ gives 1 million tasks
PLASMA Local Scheduling

Dynamic Scheduling: Sliding Window

• DAGs get very big, very fast
  • So windows of active tasks are used; this means no global critical path
• Matrix of NBxNB tiles; NB^3 operation
  • NB=100 gives 1 million tasks
PLASMA Local Scheduling
Dynamic Scheduling: Sliding Window

• DAGs get very big, very fast
  • So windows of active tasks are used; this means no global critical path
• Matrix of NBxNB tiles; NB^3 operation
  • NB=100 gives 1 million tasks
Example: QR Factorization

FOR \( k = 0 \) .. \( \text{SIZE} - 1 \)

\[
A[k][k], T[k][k] \leftarrow \text{GEQRT}( A[k][k] )
\]

FOR \( m = k+1 \) .. \( \text{SIZE} - 1 \)

\[
A[k][k]|\text{Up}, A[m][k], T[m][k] \leftarrow \text{TSQRT}( A[k][k]|\text{Up}, A[m][k], T[m][k] )
\]

FOR \( n = k+1 \) .. \( \text{SIZE} - 1 \)

\[
A[k][n] \leftarrow \text{UNMQR}( A[k][k]|\text{Low}, T[k][k], A[k][n] )
\]

FOR \( m = k+1 \) .. \( \text{SIZE} - 1 \)

\[
A[k][n], A[m][n] \leftarrow \text{TSMQR}( A[m][k], T[m][k], A[k][n], A[m][n] )
\]
for (k = 0; k < A.mt; k++) {
    Insert_Task( zgeqrt, A[k][k], INOUT,
                T[k][k], OUTPUT);
    for (m = k + 1; m < A.mt; m++) {
        Insert_Task( ztsqrt, A[k][k], INOUT | REGION_D | REGION_U,
                     A[m][k], INOUT | LOCALITY,
                     T[m][k], OUTPUT);
    }
    for (n = k + 1; n < A.nt; n++) {
        Insert_Task( zunmqr, A[k][k], INPUT | REGION_L,
                     T[k][k], INPUT,
                     A[k][m], INOUT);
        for (m = k + 1; m < A.mt; m++) {
            Insert_Task( ztsmqr, A[k][n], INOUT,
                         A[m][n], INOUT | LOCALITY,
                         A[m][k], INPUT,
                         T[m][k], INPUT);
        }
    }
}
**Algorithms**

Cholesky

- **Algorithm**
  - equivalent to LAPACK

- **Numerics**
  - same as LAPACK

- **Performance**
  - comparable to vendor on few cores
  - much better than vendor on many cores
Algorithm
- equivalent to LAPACK
- same pivot vector
- same L and U factors
- same forward substitution procedure

Numerics
- same as LAPACK

Performance
- comparable to vendor on few cores
- much better than vendor on many cores
**Algorithm**
- the same R factor as LAPACK (absolute values)
- different set of Householder reflectors
- different Q matrix
- different Q generation / application procedure

**Numerics**
- same as LAPACK

**Performance**
- comparable to vendor on few cores
- much better than vendor on many cores
**Algorithm**
- two-stage tridiagonal reduction + QR Algorithm
- fast eigenvalues, slower eigenvectors (possibility to calculate a subset)

**Numerics**
- same as LAPACK

**Performance**
- comparable to MKL for very small problems
- absolutely superior for larger problems
PLASMA_{scdz}gesvd[Tile][Async]()

### Algorithm
- two-stage bidiagonal reduction + QR iteration
- fast singular values, slower singular vectors (possibility of calculating a subset)

### Numerics
- same as LAPACK

### Performance
- comparable with MKL for very small problems
- absolutely superior for larger problems
Pipelining: Cholesky Inversion
3 Steps: Factor, Invert L, Multiply L’s

48 cores
POTRF, TRTRI and LAUUM.
The matrix is 4000 x 4000, tile size is 200 x 200

POTRF+TRTRI+LAUUM: 25 (7t-3)
Cholesky Factorization alone: 3t-2

Pipelined: 18 (3t+6)
**Algorithm**
- the same R factor as LAPACK (absolute values)
- different set of Householder reflectors
- different Q matrix
- different Q generation / application procedure

**Numerics**
- same as LAPACK

**Performance**
- absolutely superior for tall matrices
Quad-socket, quad-core machine Intel Xeon EMT64 E7340 at 2.39 GHz. Theoretical peak is 153.2 Gflop/s with 16 cores. Matrix size 51200 by 3200
Avoiding Synchronization

• “Responsibly Reckless” Algorithms
  ▪ Try fast algorithm (unstable algorithm) that might fail (but rarely)
  ▪ Check for failure, recompute, if needed, with a stable algorithm
Avoid Pivoting; Minimize Synchronization

- To solve $Ax = b$:
  - Compute $A_r = U^TAV$, with $U$ and $V$ random matrices
  - Factorize $A_r$ without pivoting (GENP)
  - Solve $A_r y = U^T b$ and then Solve $x = Vy$

- $U$ and $V$ are Recursive Butterfly Matrices
  - Randomization is cheap ($O(n^2)$ operations)
  - GENP is fast (“Cholesky” speed, take advantage of the GPU)
  - Accuracy is in practice similar to GEPP (with iterative refinement), but...

Think of this as a preconditioner step.

Goal: Transform $A$ into a matrix that would be sufficiently “random” so that, with a probability close to 1, pivoting is not needed.

A butterfly matrix is defined as any $n$-by-$n$ matrix of the form:

$$B = \frac{1}{\sqrt{2}} \begin{pmatrix} R & S \\ R & -S \end{pmatrix}$$

where $R$ and $S$ are random diagonal matrices.
Partial randomization (i.e. gray) is inexpensive.
Factorization without pivoting is scalable without synchronizations.
Summary

• **Major Challenges are ahead for extreme computing**
  - **Parallelism** $O(10^9)$
    - Programming issues
  - **Hybrid**
    - Peak and HPL may be very misleading
    - No where near close to peak for most apps
  - **Fault Tolerance**
    - Today Sequoia BG/Q node failure rate is 1.25 failures/day
  - **Power**
    - 50 Gflops/w (today at 2 Gflops/w)

• **We will need completely new approaches and technologies to reach the Exascale level**
Collaborators / Software / Support

- **PLASMA**

- **MAGMA**

- **Quark (RT for Shared Memory)**

- **PaRSEC** *(Parallel Runtime Scheduling and Execution Control)*

- Collaborating partners
  - University of Tennessee, Knoxville
  - University of California, Berkeley
  - University of Colorado, Denver
  - INRIA, France
  - KAUST, Saudi Arabia