nag®

CSC / NAG Autumn School on

Core Algorithms in High-Performance Scientific Computing

Libraries III

Ian Bush

Direct Methods in Parallel

ScaLAPACK

Numerical Algorithms Group Ltd, HECToR CSE

HECToR

RESEARCH COUNCILS UK

Contents

- The (Local) Building Blocks of ScaLAPACK
- BLACS
- Data Distribution for ScaLAPACK
- The PBLAS
- Using ScaLAPACK
- Performance Issues
- Useful Tools and Utilities
- Calling routines from C
- Further Reading / References

HECToR

RESEARCH COUNCILS UK

Acknowledgments

- Thanks to Craig Lucas for providing huge amounts of this talk

HECToR

RESEARCH COUNCILS UK

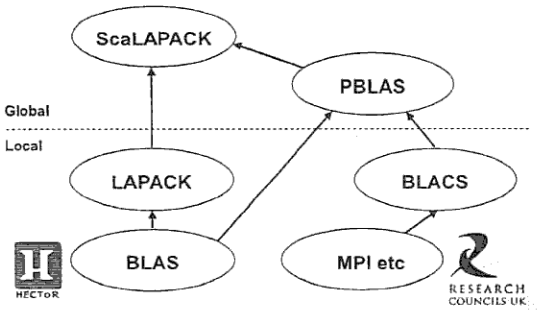
What is ScaLAPACK ?

- ScaLAPACK is the distributed memory version of LAPACK
  - Much of LAPACK implemented
  - Also a number of utilities to help the parallel programmer
- It consists of a number of building blocks:
  - ScaLAPACK itself
  - PBLAS
  - BLACS
  - LAPACK
  - BLAS
- We know all about the last two already!

HECToR

RESEARCH COUNCILS UK

Structure of the Libraries



HECToR

RESEARCH COUNCILS UK

What Do We Have To Do ?

- ScaLAPACK maps matrices onto 2 dimensional processor grids
  - i.e. 2D arrays are mapped onto 2D processor grids
- So first have to create one (or more !) processor grids
  - This uses the BLACS layer
- Then have to map our matrices onto the processor grids we have just created
  - This uses the ScaLAPACK layer

HECToR

RESEARCH COUNCILS UK

BLACS

- Basic Linear Algebra Communication Subroutines
- Frequently occurring operations in linear algebra
- Portable, machine specific versions built on appropriate communications routines, MPI, PVM etc
  - Almost always MPI nowadays
- Vendor optimised versions

BLACS

- We think of processes being arranged (conceptually) on a grid, for example:

	0	1	2
0	0	1	2
1	3	4	5

2 by 3 process grid

Process of rank 3 is at coordinate (1,0)

- Process grid is enclosed in a *context*, like an MPI communicator

Initial set up

```
CALL BLACS_PINFO( IAM, NPROCS )
void Cblacs_pinfo( int *iam, int *nprocs)
```

integer IAM = rank

integer NPROCS = size (number of processes)

```
CALL BLACS_GET( 0, 0, ICTXT )
void Cblacs_get( 0, 0, int *ictxt )
```

integer ICTXT returned, a default contexts for the process grid

Initial set up

```
CALL BLACS_GRIDINIT( ICTXT, ORDER, NPROW, NPCOL )
void Cblacs_gridinit( int *ictxt, char *order,
  int nprow, int npcol)
```

character ORDER orders the grid in row (r) or column (c) major order

integers NPROW/COL number of rows and columns to be used in grid

0	1	2
3	4	5

2 by 3 process grid with row major ordering

Initial set up

- Alternatively to blacs_gridinit:

```
CALL BLACS_GRIDMAP (ictxt, usermap, ldumap,
  nprow, npcol)
void Cblacs_gridmap ( int *ictxt, int *pmap,
  int ldpmap, int nprow, int npcol
```

- More complicated
  - Can be used to set up sub grids
  - Useful if you have multiple linear algebra operations that can be done in parallel
  - Somewhat more painful to use than mpi_comm_split

BLACS Info Routines

- Get number of rows and columns and 'my' coordinate in grid

```
CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW,
  MYCOL )
void Cblacs_gridinfo( int ictxt, int *nprow, int
  *npcol, int *myrow, int *mycol )
```

- integers MYROW/COL is coordinate in grid described by ICTXT context, returned as -1 if not in grid.
- Also `BLACS_PNUM` and `BLACS_PCOORD` for getting coordinate from processes number (rank) or vice versa

Grid Destruction Routines

```
CALL BLACS_GRIDEXIT (ictxt)
blacs_gridexit (int ictxt);
```

BLACS Finalization

- Normal termination:

```
Call blacs_exit( idoneflag )
void Cblacs_exit( int doneflag )
```

  - If doneflag is non-zero message passing will continue after the blacs_exit is exited

- Error

```
Call blacs_abort( ictxt, ierr )
void Cblacs_exit( int ictxt, int ierr )
```

Typical code fragment

```
Integer :: bl_ctxt
Integer :: nprow, npcol, myprow, mypcol
... !Work out how to split the processors
Call blacs_get( 0, 0, bl_ctxt )
Call blacs_gridinit( bl_ctxt, 'C', nprow, npcol )
Call blacs_gridinfo( bl_ctxt, nprow, npcol, &
                     myprow, mypcol )
... ! Your calculation here !
Call blacs_gridexit( bl_ctxt )
Call blacs_exit( 0 ) ! No more message passing
```

More BLACS

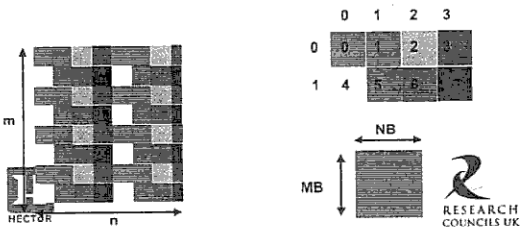
- BLACS can also do lots more
  - Especially message passing
- I prefer to use MPI for this so will not go into here
  - i.e. I only use BLACS for processor grid handling
- More info, including examples, at
  http://www.netlib.org/blacs/

Mapping The Array Onto The Grid

- We now have to decide on how to map the array onto the grid
- ScaLAPACK (and PBLAS) uses a ***block cyclic distribution***

Block Cyclic Distribution

- Data is distributed in a *Block Cyclic* manner, according to the BLACS process grid and blocks of MB by NB
- For a 2 by 4 processor grid



Block Cyclic Distribution

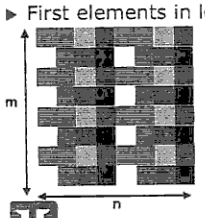
- Block cyclic ensures good load balancing and aids scalability, analysis supports this.
- We need redistribution routines to map our data to this scheme.

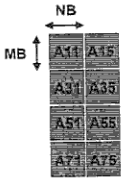
HECToR

RESEARCH COUNCILS UK

Data On A Given Processor

- Process at (0,0) is required to store (4 X MB) X (2 X NB) of the global array.
- Storage is Fortran column major order.
- First elements in local array.





HECToR

RESEARCH COUNCILS UK

Array Descriptors

- How do we tell the program how the array is mapped onto the processor grid ?
- We us an ***Array Descriptor***
- Each data object needs to be assigned an array descriptor.
- It contains all the information required to establish the mapping between global array entries and its local storage
- All global arrays must be distributed and have an array descriptor prior to calling a PBLAS or ScaLAPACK routine.

HECToR

RESEARCH COUNCILS UK

Array Descriptors

- Array descriptor (for dense matrices) is a nine element integer array, DESCA for an array A:

Element:	Name:	Value:	Global/Local
1	DTYPE_A	1 for dense matrices	Global
2	CTXT_A	BLACS Context	Global
3	M_A	Number of rows in global array A	Global
4	N_A	Number of columns in global array A	Global
5	MB_A	Blocking factor for rows	Global
6	NB_A	Blocking factor for columns	Global
7	RSRC	Process row owning first element	Global
8	CSRC	Process column owning first element	Global
9	LLD_A	Leading dimension of local array	Local

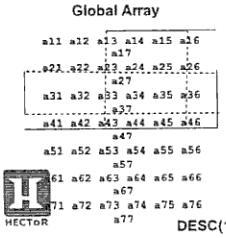
HECToR

RESEARCH COUNCILS UK

Array Descriptor Example

- Array descriptor for process (0,0), of a 2 by 2 process grid for a 7x7 array. Local leading dim 5
  - Note arrays can be rectangular

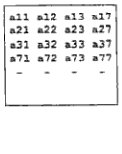
Global Array



Processor grid

0	1
2	3

Local Array on (0,0)



DESC(1,CTXT,7,7,3,3,0,0,5)

HECToR

RESEARCH COUNCILS UK

How To Create An Array Descriptor

- You should use the routine `descinit` to create an array descriptor

```
SUBROUTINE DESCINIT( DESC, M, N, MB, NB, IRSRC, ICSRC, ICTXT,
     $                     LLD, INFO )
INTEGER :: ICSRC, ICTXT, INFO, IRSRC, LLD, M, MB, N, NB
INTEGER, Dimension( * ) :: DESC( * )
DESC    The array descriptor of a distributed matrix to be set.
M       The number of rows in the distributed matrix. M >= 0.
N       The number of columns in the distributed matrix. N >= 0.
MB      The blocking factor used to distribute the rows of the matrix.
NB      The blocking factor used to distribute the columns of the matrix.
IRSRC   The process row where the first row of the matrix is distributed.
ICSRC   The process col where the first col of the matrix is distributed.
ICTXT   The BLACS context handle
LLD     The leading dimension of the local array
```

HECToR

RESEARCH COUNCILS UK

Some Tips For Creating Array Descriptors

- When you create a descriptor
  - You are free to choose the processor grid, so what values would be good ?
  - You are free to choose the blocking factors, so what values would be good ?
  - The local dimensions of the distributed matrix a bit of a pain to calculate, is there an easy way ?

HECToR RESEARCH COUNCILS UK

Choosing Processor Grids

- General experience is that for best performance you should choose as square a processor grid as possible
- So for 16 procs use a 4x4 grid, for a 20 processor grid use 4x5 or 5x4
  - In general seems to be no great difference which way around you put the grid
- Don't use a 1xn grid !
  - Might be easier to program but performance is usually horrible !

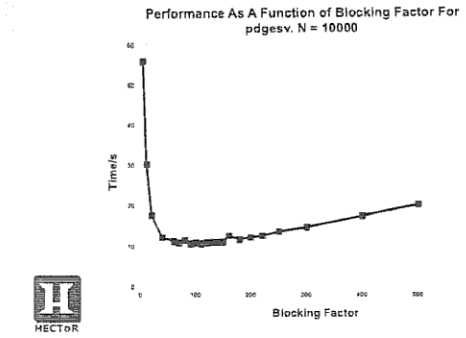
HECToR RESEARCH COUNCILS UK

Choosing blocking factors

- I strongly suggest you choose MB=NB
  - i.e. Square blocks
  - Not all implementations of ScaLAPACK support rectangular blocks
- Blocking factor affects performance
  - For same reasons as cache blocking affects BLAS/ LAPACK
- Good values are usually in the range 30-200
  - I tend to default to using 96
  - Best to test, if you can

HECToR RESEARCH COUNCILS UK

Choosing Blocking Factors



HECToR RESEARCH COUNCILS UK

Calculating The Leading Dimension

- The ScaLAPACK utility `numroc` is very useful here
  - NUMROC computes the NUMber of Rows Or Columns of a distributed matrix owned by the process indicated by IPROC.

```
INTEGER FUNCTION NUMROC( N, NB, IPROC, ISRCPROC, NPROCS )
N           The dimension
NB          The blocking factor
IPROC       In practice the coordinate of this processor
ISRCPROC    The coordinate of the process that possesses the
            first row or column of the distributed matrix.
NPROCS      The number of processors along this dimension of the
            processor grid
```

HECToR RESEARCH COUNCILS UK

A Code Fragment

```
Real, Dimension( :, : ), Allocatable :: a
Integer, Dimension( 1:9 ) :: desc_a
Integer :: n, nb, np, nq, bl_ctxt, error
Integer :: nprow, npcol, myprow, mypcol
Call blacs_get( 0, 0, bl_ctxt )
Call blacs_gridinit( bl_ctxt, 'C', nprow, npcol )
Call blacs_gridinfo( bl_ctxt, nprow, npcol, &
                     myprow, mypcol )
nb = 96
np = numroc( n, nb, myprow, 0, nprow )
nq = numroc( n, nb, mypcol, 0, npcol )
Allocate( a( 1:np, 1:nq ) )
Call descinit( desc_a, n, n, nb, nb, 0, 0, bl_ctxt, np, error )
```

HECToR RESEARCH COUNCILS UK

Almost there ...

Phew ... A lot of stuff but we can now use these things !

Technically two libraries to use,

- PBLAS
- ScaLAPACK

In practice use is now very similar. Talk about PBLAS first.

HECToR

RESEARCH COUNCILS UK

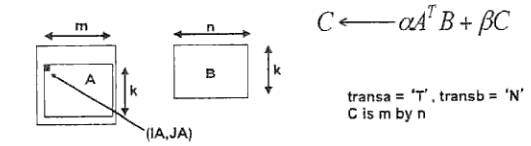
PBLAS

- Parallel Basic Linear Algebra Subprograms
- Extremely Similar to BLAS in functionality
- Same three "levels"
  - Level 1 – Vector Operations
  - Level 2 – Vector-Matrix Operations
  - Level 3 – Matrix-Matrix Operations
- But now acting on distributed arrays.
- Built on top of the BLAS and BLACS
- Same advantages as BLAS – portable, vendor optimised etc
- Same naming system except a P on the front

HECToR

RESEARCH COUNCILS UK

Parallel Matrix Matrix Multiply



$$C \longleftarrow \alpha A^T B + \beta C$$

transa = 'T', transb = 'N'
C is m by n

```
CALL DGEMM (TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
```

↓

```
CALL PDGEMM (TRANSA, TRANSB, M, N, K, ALPHA, A, IA, IJ, DESCA, B, IB, JB,
             DESCB, BETA, C, IC, JC, DESCC)
```

HECToR

RESEARCH COUNCILS UK

A Note On Vectors

- Consider

```
CALL PDGEMV( transa, m, n, alpha, a, ia, ja,
  desc_a, x, ix, jx, desc_x, incx, beta, y, iy,
  jy, desc_y, incy)
```

which performs parallel matrix vector multiply. Note the vectors are distributed – they have a descriptor

- Distributed vectors are treated just as if they were matrices with one of their dimensions set to unity

HECToR

RESEARCH COUNCILS UK

ScaLAPACK

- Scalable Linear Algebra PACKage
- Efficient – uses optimised computation and communication engines and LAPACK/BLAS for good serial performance on individual processes.
- Scalable
- Portable – machine dependencies contained within BLAS and BLACS.
- Easy to Use – calling interface similar to LAPACK.

HECToR

RESEARCH COUNCILS UK

Contents of ScaLAPACK

- Only a subset of LAPACK routines are provided. (This is being addressed.)
- There are three types of routine
  - Driver routines – solve a complete problem like a solving linear system or computing the eigenvalues of matrix. Use these if they do what you need. Global and local input error checking.
  - Computation routines – solve specific computational tasks like an LU factorization or the reduction of matrix to tridiagonal form. Global and local input error checking.
  - Auxiliary routines – low level computation routines such as scaling a matrix or computing a matrix norm.

HECToR

RESEARCH COUNCILS UK

Naming Convention

- Similar to LAPACK
- Usually of the form PXYYZZ
- Sometimes an extra letter on the end
  - E.g. X means expert direver

X	data type	S, D, C, Z
YY	matrix type	GB, GE, PO, SY
ZZ(Z)	computation performed	e.g.SV – linear equation solve EV – Eigenvalue problem

HECToR

RESEARCH COUNCILS UK

Contents of ScaLAPACK

- Driver routines
  - Linear Equation SolVers for GEeneral, General Band, symmetric POsitive definite matrices
  - Linear Least Square Solvers for a GEneral matrix
  - Eigenvalue (EV) and Generalized Eigenvalue (GV) Solvers
  - SVD solvers for General matrices.

HECToR

RESEARCH COUNCILS UK

Contents of ScaLAPACK

- Computational routines
  - LU factorization
  - Cholesky factorisation (full rank)
  - LDL^T factorisation
  - QR factorisations (and with column pivoting)
  - Symmetric Tridiagonal Reduction
  - Eigenvalue / Vector computations
  - SVD factorisation
  - Bidiagonal / Hessenberg Reductions
  - Condition number estimation / Error bound computation

HECToR

RESEARCH COUNCILS UK

Example Scalapack Routine

- Linear Solver Simple Driver Routine
  - Solves **Ax=b**

```
CALL PDGESV( N, NRHS, A, IA, JA, DESCA, IPIV, B, IB, JB, DESCB, INFO )
void pdgesv( int n, int nrhs, double *a, int ia, int ja, int desca,
             int *ipiv, double *b, int ib, int jb, int descb, int *info )
```

HECToR

RESEARCH COUNCILS UK

Example Program

```
Program parallel_symm_diag

  Use mpi

  Implicit None

  Integer, Parameter :: wp = Selected_real_kind( 13, 70 )

  Real( wp ), Dimension( :, : ), Allocatable :: a, z
  Real( wp ), Dimension( :    ), Allocatable :: b, work
  Real( wp ), Dimension( 1:1 ) :: tmp
  Integer, External :: numroc
  Integer, Dimension( : ), Allocatable :: rput
  Integer, Dimension( 1:9 ) :: desc_a
  Integer :: n, nb, np, nq, bl_ctxt, nproc, me
  Integer :: nprow, npcol, myprow, mypcol
  Integer :: log_p
  Integer :: nput
  Integer :: error
  Integer :: t1, t2, rate
  Integer :: i
```

HECToR

RESEARCH COUNCILS UK

Example Program

```
! Initialise MPI
 Call mpi_init( error )

 Call mpi_comm_rank( mpi_comm_world, me   , error )
 Call mpi_comm_size( mpi_comm_world, nproc, error )
```

HECToR

RESEARCH COUNCILS UK

Example Program

```
! Split the processors into as square a gird as possible
! For simplicit assume we are running on a power of 2 number
! of procs
log_P = 0
Do
   If( 2 ** log_P == nproc ) Then
      Exit
   End If
   log_P = log_P + 1
End Do
nprow = log_P / 2
npcol = log_P - nprow
nprow = 2 ** nprow
npcol = 2 ** npcol
If( me == 0 ) Then
   Write( *, * ) 'Procs: ', nproc
   Write( *, * ) 'Grid: ', nprow, npcol
End If
! Initialise blacs
Call blacs_get( -1, 0, bl_ctxt )
! Create the processor grid
Call blacs_gridinit( bl_ctxt, 'R', nprow, npcol )
Call blacs_gridinfo( bl_ctxt, nprow, npcol, myprow, mypcol )
```

HECToR

RESEARCH COUNCILS UK

Example Program

```
! Read in the size of Problem
If( me == 0 ) Then
   Write( *, * ) 'Order and blocking factor'
   Read( *, * ) n, nb
   Write( *, * ) 'N = ', n, ' block = ', nb
End If
! Set up the array descriptors
Call mpi_bcast( n , 1, mpi_integer, 0, mpi_comm_world, error )
Call mpi_bcast( nb, 1, mpi_integer, 0, mpi_comm_world, error )
np = numroc( n, nb, myprow, 0, nprow )
nq = numroc( n, nb, mypcol, 0, npcol )
Call descinit( desc_a, n, n, nb, nb, 0, 0, bl_ctxt, np, error )
! Space for matrix, evecs and evals. Yes I know I should check status.
Allocate( a( 1:np, 1:nq ) )
Allocate( z( 1:np, 1:nq ) )
Allocate( b( 1:n ) )
```

HECToR

RESEARCH COUNCILS UK

Example Program

```
! Initialize a with random numbers on the off diags,
 ! unity on the diag
 Call Random_seed( size = nput )
 Allocate( rput( 1:nput ) )
 rput = ( me + 1 ) ** 3
 Call Random_seed( put = rput )
 Deallocate( rput )
 Call Random_number( a )
 a = 0.5_wp * ( a + Transpose( a ) ) ! Make sure a symmetric
 a = a / 10.0_wp ! In real problems typically off diags smaller
 Do i = 1, n
    Call pdelset( a, i, i, desc_a, 1.0_wp ) ! Utility
 End Do
```

HECToR

RESEARCH COUNCILS UK

Example Program

```
! Call the diag
 Call mpi_barrier( mpi_comm_world, error )
 Call system_clock( t1 )
 ! First call enquires about the memory size
 Call pdsyev( 'V', 'U', n, a, 1, 1, desc_a, b, &
      z, 1, 1, desc_a, tmp, -1, error )
 Allocate( work( 1:Nint( tmp( 1 ) ) ) )
 Call pdsyev( 'V', 'U', n, a, 1, 1, desc_a, b, &
      z, 1, 1, desc_a, work, Size( work ), error )
 Call system_clock( t2, rate )
 Call mpi_barrier( mpi_comm_world, error )
```

HECToR

RESEARCH COUNCILS UK

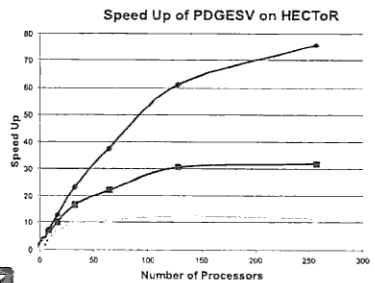
Example Program

```
 If( me == 0 ) Then
    Write( *, * ) 'diag'
    Write( *, * ) 'Error: ', n - Sum( b )
    Write( *, * ) 'Time : ', Real( t2 - t1 ) / rate
 End If

 ! Finalize blacs
 Call blacs_gridexit( bl_ctxt )
 Call blacs_exit( 1 ) ! More message passing to occur

 ! Finalize mpi
 Call mpi_finalize( error )

End Program parallel_symm_diag
```

HECToR

RESEARCH COUNCILS UK

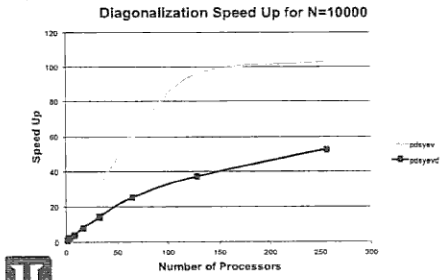
Speed Up for Linear Equation Solve



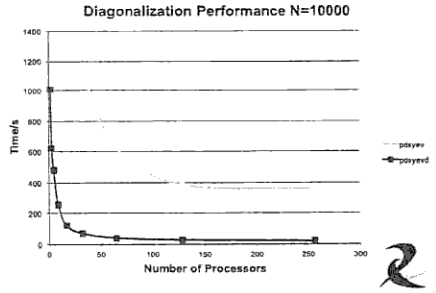
HECToR

RESEARCH COUNCILS UK

Diagonalization Speed Up – 2 Different Routines



Performance is Not Speed Up !



Tools

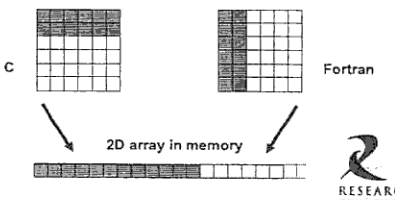
- ScaLAPACK provides lots of utility functions
  - Accessing Distributed objects can be a bit tricky
  - Seen `descinit` and `numroc`
- Useful ones include
  - `p*elset( a, i, j, desc_a, val )`
    - Sets the global value a(i, j) to val
  - `p*elget( scope, top, val, a, i, j, desc_a )`
    - Gets the global value a(i, j) in val
  - `indxg2l( indxglob, nb, iproc, isrcproc, nprocs )`
    - Given a global index return the local index
  - `indxl2g( indxloc, nb, iproc, isrcproc, nprocs )`
    - Given a local index return the global index
- See http://www.netlib.org/scalapack/tools/

Calling From C

- The routines are designed to be called from Fortran
- In particular the arrays are assumed stored in column major, i.e. Fortran, order
- However it is possible to call them from C, provided you are aware of the array storage issue

Array Storage in C

- Fortran stores data in column major order
- C stores data in row major order
- So for 2d data it as if the matrix has been transposed



Cheating with the transpose

- So the simplest way to solve this problem is in C to "lie" about the transpose
- Matrix Multiply
  A*B in row major order is A^T*B^T in column major order. Call PDGEMM with transa=transb='T' (Returns C is row major, and therefore C^T in column major)
- Solving Linear System
  In ScaLAPACK, if we call PDGETRF to find A=LU then we actually have A^T=LU in, so now call PDGETRS with transa='T'. (Which solves A^TX=B)
- No overhead for transpose

Material Not Covered

- Trapezoidal matrix operation in the BLACS
- BLACS message passing
- BLACS topologies
- Narrow band storage in ScaLAPACK
- Out-of-Core Linear System solvers in ScaLAPACK
- Accuracy and stability of algorithms

- Can be found in the following references:

Quick Reference Guides

- http://www.netlib.org/blas/blasqr.ps
- http://www.netlib.org/lapack/lapackqref.ps
- http://www.netlib.org/scalapack/scalapackqref.ps
- http://www.netlib.org/scalapack/pblasqref.ps
- http://www.netlib.org/blacs/cblacsqref.ps
- http://www.netlib.org/blacs/f77blacsqref.ps

FAQs and User Guides

- http://www.netlib.org/blas/faq.html
- http://www.netlib.org/lapack/faq.html
- http://www.netlib.org/scalapack/faq.html

- http://www.netlib.org/lapack/lug/index.html
- http://www.netlib.org/scalapack/slug/index.html
- http://www.netlib.org/blacs/lawn94.ps