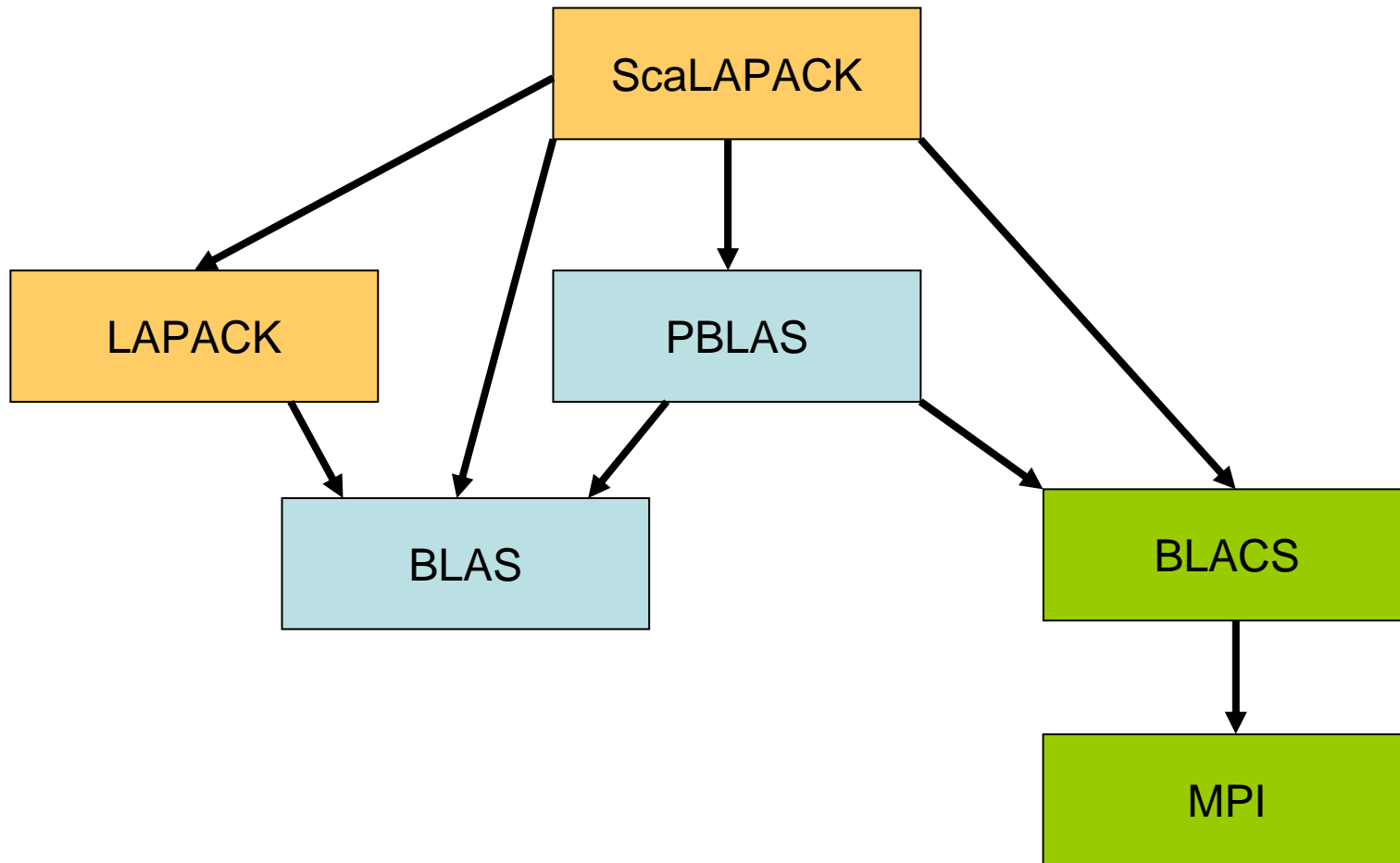# BLAS, LAPACK
BLACS, PBLAS, ScaLAPACK

High Performance Portable Libraries for
Dense Linear Algebra

Introduction

- In this lecture we will cover the following topic:
  - High performance portable dense linear algebra libraries
- The following libraries will be introduced:
  - BLAS
  - LAPACK
  - BLACS
  - PBLAS
  - ScaLAPACK
- This will also be covered:
  - FORTRAN

The Overall Picture

NETLIB

- All software discussed in this lecture can be downloaded free of charge from NETLIB (repository for freely distributable numerical software)
  - http://www.netlib.org/
- Collection of all NETLIB links mentioned in the notes:
  - http://www.netlib.org/blas/
  - http://www.netlib.org/atlas/
  - http://www.netlib.org/blas/gemm_based/
  - http://www.netlib.org/lapack/
  - http://www.netlib.org/blacs/
 - http://www.netlib.org/scalapack/ (includes PBLAS)

Crash Course: FORTRAN

- FORTRAN is used in these notes to refer to a subset of FORTRAN 90/95 whic is used by LAPACK and ScaLAPACK developers.

- FORTRAN was the dominant language in numerical computing for a long time and it
  - has an extensive compiler support for multi-dimensional arrays
  - has restrictions in the language to allow aggressive compiler optimizations
  - has language support for dynamic memory management, derived types, object orientation, operator overloading, generic interfaces, array expressions, distributed arrays (co-arrays), etc
  - unlike many languages, multidimensional arrays (matrices) are supported and has many features such as automatic allocation and slicing

Fixed Source Format

- FORTRAN 77 has a strict source format known as the "fixed source format"

- Columns are used for different things:
  - 1: Comment column
  - 2-5: Label columns
  - 6: Continuation column
  - 7-72: Statement columns
  - 73-: Truncated (silently)

Set your editor to expand tabs to spaces.
Use 3 as tabstop (two tabs takes you to colunm 7)

Label

Statements

Truncated

Comment

Continuation
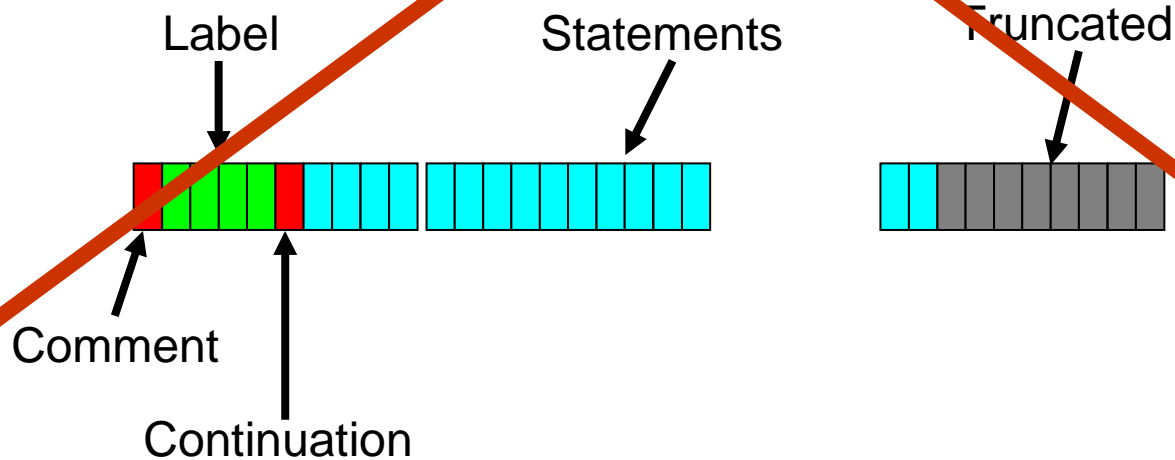
IF-THEN-ELSE

- ## IF-statement:
  - `IF(` <logical expression> `)` <statement>

- ## IF-construct:
  - `IF(` <logical expression> `)` `THEN`
    <block>
    [`ELSE IF(` <logical expression> `)` `THEN`
    <block>]
    [`ELSE`
    <block>]
    `END IF`

GO TO, CONTINUE and Labels

- Labels
  - Integers from `1` to `9999`
  - Used in GO TO statements

- GO TO-statement:
  - `GO TO <label>`
  - Transfers control to statement labeled with <label>

- CONTINUE-statement:
  - `CONTINUE`
  - A do-nothing statement often used together with GO TO

DO

- DO-construct:
  - `DO` <var> `=` <low>`,` <high>[`,` <step>]
    <block>
    `END DO`

  - Example:
    - `DO J = 1, M, 2`
          `...`
      `END DO`

PROGRAM

- In FORTRAN you do not have a special function called MAIN, instead you have the PROGRAM construct:
  - **PROGRAM [name]**

    <declarations>

    <statements>

    END [PROGRAM name]

SUBROUTINEs and FUNCTIONs

- SUBROUTINEs (think of C functions returning void)
  - `CALL` mysub(<arglist>)
- FUNCTIONs (think of C functions returning non-void)
  - <lval> = myfunc(<arglist>)

- Declaring a SUBROUTINE:
  - `SUBROUTINE` <name>`(`<dummy arglist>`)`
    `END [SUBROUTINE name]`
- Declaring a FUNCTION:
  - <type> `FUNCTION` <name>`(``)`
    `END [FUNCTION name]`
  - Example:
    - `INTEGER FUNCTION MAX(a, b)`
      `INTEGER a, b`
      `MAX = a`
      `IF( b .GT. a ) MAX = b`
      `END`

Arithmetic Operators

FORTRAN	C/Java
+	+
–	–
*	*
/	/
**	N/A
N/A	+=
N/A	++

Logical Operators

FORTRAN	C/Java
.GT. >	>
.LT. <	<
.LE. <=	<=
.GE. >=	>=
.EQ. == .EQV. (logical)	==
.NE. /= .NEQV. (logical)	!= (exclusive or)
.AND.	&&
.OR.	||
.NOT.	!

Data Types

- **INTEGER**
  - Signed 32-bit (usually) integer
- **LOGICAL**
  - **.TRUE.** or **.FALSE.**
- **CHARACTER(**<length>**)** or **CHARACTER** (just one character)
  - **'**string**'** or **"**string**"**
- **REAL**
  - Single precision IEEE (usually) floating point **f = 5E+0**
- **DOUBLE PRECISION**
  - Double precision IEEE (usually) floating point **d = 5D+0**
- **COMPLEX**
  - Single precision IEEE (usually) complex number **c = (r, i)**
- **COMPLEX*16**
  - Double precision IEEE (usually) complex number **c = (r, i)**

Arrays (Matrices and Vectors)

- Declaring a vector of 50 INTEGERs
  - `INTEGER vec(50)`

- Declaring a 25x47 matrix of 50 INTEGERs
  - `INTEGER mtx(25, 47)`

- Indexing starts from 1 (unless explicitly stated in the declaration)

- Indexing top left element in matrix:
  - `mtx(1, 1)`

- Indexing bottom right index in matrix:
  - `mtx(25, 47)`

Automatic Arrays

- Size of array is either known at compile time or determined by dummy arguments and the array is not a dummy argument itself.

- Storage will be allocated (think of it as being allocated on the stack) at runtime and deallocated automatically when variable falls out of scope.

  - Example:

    - ```
      SUBROUTINE auto(N)
          INTEGER A(N)
      END
      ```

Assumed Shape Array

- The shape (extent of all dimensions) need not be known at compile time.
- An array where the extent of one or more dimension is determined by dummy arguments is referred to as an assumed shape array.
  - Useful for passing arrays as arguments to subroutines.
  - Example:

```
SUBROUTINE mysub(A, LDA, M, N)
    INTEGER LDA, M, N
    REAL A(LDA, N)
END
```

Assumed Size Arrays

- Extent of last dimension in FORTRAN arrays need not be known at compile time (or at runtime for that matter) to generate indexing code (first dimension in the case of C).

- An array declared with unknown last dimension extent is referred to as an assumed size array.
  - **REAL A(LDA, *)**
    - Indexing code:
      A(i, j) → A + (i-1) + (j-1)*LDA

Comments

- Comments begin with ! and continue til the end of the line.
  - Example:
    - `IF( 1 > 2 ) THEN ! This will never happen`

Continuation (long statements)

- Long statements can be broken into several lines by placing ending the line with an &
    - Example:
        - `A(1, 2) = longvariablename + &`
          `anotherlongvariable`

Storage Formats used by the Libraries

- General matrices:
  - Column Major

- Symmetric and triangular matrices
  - Column Major Column Packed

- Band matrices
  - Diagonal Storage

- Tridiagonal matrices
  - Diagonal Storage

Full Storage Format

Matrix Indices

11	12	13	14	15	16	17	18	19
21	22	23	24	25	26	27	28	29
31	32	33	34	35	36	37	38	39
41	42	43	44	45	46	47	48	49
51	52	53	54	55	56	57	58	59
61	62	63	64	65	66	67	68	69
71	72	73	74	75	76	77	78	79
81	82	83	84	85	86	87	88	89
91	92	93	94	95	96	97	98	99

Memory Placement

0	9	18	27	36	45	54	63	72
1	10	19	28	37	46	55	64	73
2	11	20	29	38	47	56	65	74
3	12	21	30	39	48	57	66	75
4	13	22	31	40	49	58	67	76
5	14	23	32	41	50	59	68	77
6	15	24	33	42	51	60	69	78
7	16	25	34	43	52	61	70	79
8	17	26	35	44	53	62	71	80

Standard Packed Storage Format

Matrix Indices

```
11   *    *    *    *    *    *    *    *
21  22    *    *    *    *    *    *    *
31  32   33    *    *    *    *    *    *
41  42   43   44    *    *    *    *    *
51  52   53   54   55    *    *    *    *
61  62   63   64   65   66    *    *    *
71  72   73   74   75   76   77    *    *
81  82   83   84   85   86   87   88    *
91  92   93   94   95   96   97   98   99
```

Memory Placement

```
0   *    *    *    *    *    *    *    *
1   9    *    *    *    *    *    *    *
2  10   17    *    *    *    *    *    *
3  11   18   24    *    *    *    *    *
4  12   19   25   30    *    *    *    *
5  13   20   26   31   35    *    *    *
6  14   21   27   32   36   39    *    *
7  15   22   28   33   37   40   42    *
8  16   23   29   34   38   41   43   44
```

Rectangular Full Packed Storage Format

Matrix Indices

```
11 | 66   76   86   96
21   22 | 77   87   97
31   32   33 | 88   98
41   42   43   44 | 99
51   52   53   54   55
61   62   63   64   65
71   72   73   74   75
81   82   83   84   85
91   92   93   94   95
```

Memory Placement

```
0    9   18   27   36
1   10   19   28   37
2   11   20   29   38
3   12   21   30   39
4   13   22   31   40
5   14   23   32   41
6   15   24   33   42
7   16   25   34   43
8   17   26   35   44
```

Banded Storage Format

Full Matrix Indices

```
11   12    *    *    *    *    *    *    *
21   22   23    *    *    *    *    *    *
31   32   33   34    *    *    *    *    *
 *   42   43   44   45    *    *    *    *
 *    *   53   54   55   56    *    *    *
 *    *    *   64   65   66   67    *    *
 *    *    *    *   75   76   77   78    *
 *    *    *    *    *   86   87   88   89
 *    *    *    *    *    *   97   98   99
```

Matrix Indices

```
 *   12   23   34   45   56   67   78   89
11   22   33   44   55   66   77   88   99
21   32   43   54   65   76   87   98    *
31   42   53   64   75   86   97    *    *
```

Memory Placement

```
0    4    8   12   16   20   24   28   32
1    5    9   13   17   21   25   29   33
2    6   10   14   18   22   26   30   34
3    7   11   15   19   23   27   31   35
```

BLAS

- Basic Linear Algebra Subroutines (BLAS)
  - http://www.netlib.org/blas/   Reference implementation
  - http://www.netlib.org/atlas/   Auto-tuning HPC impl.
  - http://www.netlib.org/blas/gemm_based/

    GEMM-based BLAS by
    Kågström et. al.

  - http://www.tacc.utexas.edu/resources/software/

    GotoBLAS

- Interfaces:
  - FORTRAN (official)
  - C, C++, Java, ... (unofficial)
- Language:
  - C, assembler, FORTRAN, ... (depends on vendor)

BLAS - Content

- BLAS contains subroutines and functions for a number of basic linear algebra operations.
  - Dot product
  - Givens rotation generation and application
  - Vector updates
  - Matrix-vector product update
  - Triangular system solve (with single or multiple right hand sides)
  - Matrix-matrix product update
  - ...
- The routines operate on various storage formats and on four data types (single, double, complex, double complex).

Coding Conventions

- _XXYY
  - _: Data type
    - **S**, **D**, **C**, or **Z**
  - XX: Type of matrix
    - **GE**, **GB**: GEneral, General Banded
    - **HE**, **HB**, **HP**: HErmitian, Hermitian Banded, Hermitian Packed
    - **SY**, **SB**, **SP**: SYmmetric, Symmetric Banded, Symmetric Packed
    - **TR**, **TB**, **TP**: TRiangular, Triangular Banded, Triangular Packed
  - YY: Operation
    - **S**: "Solve"
    - **M**: "Matrix"
    - **V**: "Vector"
    - **R**: Rank-1
    - **R2**: Rank-2
    - **RK**: Rank-k
    - **R2K**: Rank-2k
- Example:
  - **DTRSM**:
    - Double precision
    - TRiangular
    - Solve
    - Multiple right hand sides

Memory Traffic - Limitations

- Memory bandwidth and latency can not match the high performance of floating point computations on the chip.

- Solution:

  - Exploit caches by data locality in space and time

- Solution requires:

  - An operation that has much inherent locality

- Metric for estimating inherent locality in linear algebra:

  - Number of floating point operations

    Number of memory locations referenced
  - i.e., flop/memref

Locality - Examples

- Example of poor inherent locality: `AXPY (a*x + y)`
  - 2 vector loads (x, y)
  - 1 vector store (y)
  - 2 vector operations (*, +)
  - flop/memref = 2/3

- Example of good inherent locality: `GEMM (a*A*B + b*C)`
  - ~2*N^3 flops
  - ~3*N^2 loads
  - ~1*N^2 stores
  - flop/memref ~ N/2

Level 1, 2, 3

- Level-1 BLAS: Vector operations (~1 flop/memref)
  - **`_dot`**
    `_axpy`
    `_swap`
    `_copy`
    `_scal`
    ...

- Level-2 BLAS: Matrix-Vector operations (~1 flop/memref)
  - **`_gemv`**
    `_symv`
    `_trsv`
    ...

- Level-3 BLAS: Matrix-Matrix operations (~N flop/memref)
  - **`_gemm`**
    `_syrk`
    `_trsm`
    ...

LAPACK

- Linear Algebra PACKage (LAPACK)
  - http://www.netlib.org/lapack/ Official LAPACK releases
  - http://www.netlib.org/lapack/lanws/

    Publications related to
    LAPACK and DLA

- Some vendors provide their own optimized LAPACK routines as well as BLAS routines:
  - IBM: ESSL   (Proprietary)
  - AMD: ACML   (Free?)
  - Intel: MKL   (Proprietary?)
  - Cray: libsci   (Proprietary?)

- Interfaces:
  - FORTRAN (official)
  - C, C++, Java, ... (unofficial)

- Language:
  - FORTRAN

LAPACK - Content

- Compared with BLAS, the high level algorithms and tricky numerical algorithms go into LAPACK.
  - Factorizing matrices
    - LU, Cholesky, QR, QL, RQ, LQ, ...
  - Applying factored-form orthogonal matrices
  - Solving linear equations
  - Solving linear least squares problems
  - Decomposing matrices
    - SVD, Schur, ...
  - Computing eigenvalues and eigenvectors
    - Symmetric, non-symmetric, ...
  - Error bounds, condition estimation

Workspace Management

- Many routines in LAPACK require auxilliary workspace to function and/or run faster.

- Users must provide this storage.

- Routines take workspace via their arguments, typically:
    - **WORK**: Workspace
    - **LWORK**: Length of workspace

- Routines requiring workspace allow workspace queries.
    - Workspace query:
      LWORK = -1
      WORK(1) contains required workspace
      Cast to **INTEGER**: **INT(WORK(1))**
    - If you do a workspace query the routine will not modify any of its arguments.

Error Reporting

- LAPACK routines have an extra `INTEGER` argument at the end of their argument lists: `INFO`
  - The value of `INFO` tells what went wrong (if anything):
    - `0`: Success
    - `< 0`: Argument number `-INFO` contained an illegal value (fatal, programming error)
    - `> 0`: Something went wrong during computation (exact interpretation is routine specific)
      - Example: `DGETRF` (LU factorization)
        `INFO > 0`: `U(INFO, INFO)` is exactly zero

LAPACK - Examples

- Solving a linear system after LU factorization
  - `DGETRS( TRANS, N, NRHS, A, LDA, IPIV, B, LDB, INFO )`

- Computing QR factorization
  - `DGEQRF( M, N, A, LDA, TAU, WORK, LWORK, INFO )`

BLACS

- Basic Linear Algebra Communication Subroutines
  - http://www.netlib.org/blacs/   Official BLACS releases
- Purpose:
  - Communication of submatrices appropriate for dense linear algebra algorithms (e.g., ScaLAPACK)
- Interfaces:
  - FORTRAN, C (official)
- Language:
  - C

2D-Grid, Scope, Context

- Processes are arranged in a logical 2D-grid.
- Each process is a member of three scopes:
  - **'All':** All processes in the grid
  - **'Row':** All processes on the same row of the grid
  - **'Column':** All processes on the same column of the grid
- BLACS communication is tied to a context (think of MPI communicators) which is an integer.

Submatrix Communication

- The BLACS unit of communication is a submatrix of some specified size and shape.
- Two types of submatrices:
  - General submatrices:
    - Parameters: `M, N, A, LDA`
  - Trapezoidal submatrices (generalization of triangular):
    - Parameters: `M, N, A, LDA, UPLO, DIAG`
- Packing of matrices hidden from user
- Types supported:
  - `I`: Integer
  - `S`: Single precision
  - `D`: Double precision
  - `C`: Complex single precision
  - `Z`: Complex double precision

Point-to-Point

- Send:
  - **xGESD2D(CTXT,   M, N, A, LDA, RDST, CDST)**
  - **xTRSD2D(CTXT, UPLO, DIAG, M, N, A, LDA, RDST, CDST)**

- Receive:
  - **xGERV2D(CTXT,   M, N, A, LDA, RSRC, CSRC)**
  - **xTRRV2D(CTXT, UPLO, DIAG, M, N, A, LDA, RSRC, CSRC)**

Collectives

- Broadcast (send):
  - `xGEBS2D(CTXT, SCOPE, TOP,                M, N, A, LDA)`
  - `xTRBS2D(CTXT, SCOPE, TOP, UPLO, DIAG, M, N, A, LDA)`

- Broadcast (receive):
  - `xGEBR2D(CTXT, SCOPE, TOP,                M, N, A, LDA,`
    `        RSRC, CSRC)`
  - `xTRBR2D(CTXT, SCOPE, TOP, UPLO, DIAG, M, N, A, LDA,`
    `        RSRC, CSRC)`

- Combine operations (SUM, MAX, MIN):
  - `xGSUM2D(CTXT, SCOPE, TOP, M, N, A, LDA, RDST, CDST)`
  - `xGAMX2D(CTXT, SCOPE, TOP, M, N, A, LDA, RA, CA,`
    `        RCFLAG, RDST, CDST)`
  - `xGAMN2D(CTXT, SCOPE, TOP, M, N, A, LDA, RA, CA,`
    `        RCFLAG, RDST, CDST)`

Collectives: Topology

- Topologies (TOP) specify the communication pattern.
  - **'I':** Increasing ring
  - **'D':** Decreasing ring
  - **'S':** Split ring
  - **'M':** Multi-ring
  - **'1':** 1-tree
  - **'B':** Bidirectional exchange
  - **' ':** Default (may use MPI_Bcast)

BLACS – Setup (FORTRAN)

- Initializing BLACS:
  - `CALL BLACS_PINFO(ME, NP)`

- Initializing context:
  - `CALL BLACS_GET(0, 0, CTXT)`
    `CALL BLACS_GRIDINIT(CTXT, 'Row', P, Q)`
    `CALL BLACS_GRIDINFO(CTXT, P, Q, MYROW, MYCOL)`

- Getting someones rank from coordinates
  - `RANK = BLACS_PNUM(CTXT, ROW, COL)`

- Getting someones coordinates from rank
  - `CALL BLACS_PCOORD(CTXT, RANK, ROW, COL)`

- Exiting BLACS
  - `CALL BLACS_EXIT(0)`

PBLAS

- Parallel BLAS
  - http://www.netlib.org/scalapack/   PBLAS reference impl.
    is part of ScaLAPACK

- Interfaces:
  - FORTRAN
  - C?

- Language:
  - C

2D Block Cyclic Distribution

- PBLAS operates on data distributed using the 2D block cyclic distribution.

- Recall:

11	12	13	14	15
21	22	23	24	25
31	32	33	34	35
41	42	43	44	45
51	52	53	54	55

5 x 5 matrix, 2 x 2 blocks

	0			1	
0	11	12	15	13	14
	21	22	25	23	24
	51	52	55	53	54
1	31	32	35	33	34
	41	42	45	43	44

2 x 2 process grid point of view

Matrix Descriptors

- Descriptors are used in PBLAS and ScaLAPACK to encapsulate information on a distributed matrix.

- A descriptor is a 9-item integer vector:
  - `INTEGER DESCA(9)`

  - `DESCA(1): (DTYPE)`   1
    `DESCA(2): (CTXT)`   BLACS context
    `DESCA(3): (M)`   Number of rows in global matrix
    `DESCA(4): (N)`   Number of columns in global matrix
    `DESCA(5): (MB)`   Row blocking factor
    `DESCA(6): (NB)`   Column blocking factor
    `DESCA(7): (RSRC)`   Row index of owner of A(1, 1)
    `DESCA(8): (CSRC)`   Column index of owner of A(1, 1)
    `DESCA(9): (LLD)`   Leading dimension of the local matrix

PBLAS - Example

- Parallel version of **DGEMM**
  - **CALL PDGEMM(TRANSA, TRANSB,**
    M, N, K,
    ALPHA, A, IA, JA, DESC_A,
    B, IB, JB, DESC_B,
    BETA, C, IC, JC, DESC_C)

- Notice:
  - PBLAS has interfaces that take descriptions of submatrices
  - BLAS, on the other hand, takes submatrices implicitly

ScaLAPACK

- SCAlable LAPACK (distributed memory)
  - http://www.netlib.org/scalapack/   Official ScaLAPACK releases

ScaLAPACK - Content

- Most of LAPACK
- No support for band and packed matrices
- Missing some more advanced algorithms
  - SVD and QR w/ pivoting least squares
  - Generalized least squares
  - Non-symmetric eigenvalue problems
  - D&C SVD
  - ...

ScaLAPACK – Coding Conventions

- Symbols are similar to LAPACK (just add `P`)
- Submatrices are referenced explicitly in interface:
  - `A(I, J), LDA`   LAPACK submatrix reference
  - `A, I, J, DESCA`   ScaLAPACK submatrix reference

Utilities: DESCINIT

- `SUBROUTINE DESCINIT(DESC, M, N, MB, NB, RSRC, CSRC, CTXT, LLD, INFO)`

- Initializes all elements of a descriptor.

- Arguments:
  - `DESC`   Descriptor to initialize (output)
  - `M, N`   Size of global matrix
  - `MB, NB`   Blocking factors
  - `RSRC, CSRC`   Coordinates of owner of (1, 1) matrix element
  - `CTXT`   BLACS context
  - `LLD`   Leading dimension of local matrix (use `NUMROC` to find)
  - `INFO`   Error reporting, 0: success (output)

Utilities: NUMROC

- `INTEGER FUNCTION NUMROC(N, NB, ME, SRC, NP)`

- Finds the number of rows (or columns) mapped to a specific grid row (or column).

- Arguments:
    - `N`   Extent of matrix dimension
    - `NB`   Blocking factor in matrix dimension
    - `ME`   Row (or column) index of processor of interest
    - `SRC`   Row (or column) index of source
    - `NP`   Number of processes in grid dimension

Utilities: INFOG2L

- `SUBROUTINE INFOG2L(GRINDX, GCINDX, DESC, NPROW, NPCOL, MYROW, MYCOL, LRINDX, LCINDX, RSRC, CSRC)`

- Given a global matrix element `(GRINDX, GCINDX),` returns the corresponding local matrix element `(LRINDX, LCINDX)` and coordinates of processor that owns that element `(RSRC, CSRC).`

- Arguments:

  - `GRINDX, GCINDX`   Global matrix element
  - `DESC`   Descriptor of matrix
  - `NPROW, NPCOL`   Grid size
  - `MYROW, MYCOL`   My coordinates
  - `LRINDX, LCINDX`   Local matrix element (output)
  - `RSRC, CSRC`   Owner of element (output)