Clicking on the name of a routine will take you to description of that routine and its parameters. After the description of the routine, there will be an option to view an example of its use. If you choose to view this example, the call to the selected BLACS routine will be at the top of your screen, so you will probably want to scroll upward in order to see the code before the call.

Routine names preceded by a lower case 'v' are type-dependent routines. The v is replaced as shown in the following table to arrive at the correct routine name for a given data type:

```
    v                          MEANING
  -----     ------------------------------------------------
    I       Integer data is to be operated on.
    S       Single precision real data is to be operated on.
    D       Double precision real data is to be operated on.
    C       Single precision complex data is to operated on.
    Z       Double precision complex data is to operated on.
```

Support routines

.. Initialization ..

```
  BLACS_PINFO    ( MYPNUM, NPROCS )
  BLACS_SETUP    ( MYPNUM, NPROCS )
  SETPVMTIDS     ( NTASKS, TIDS )
  BLACS_GET      ( ICONTXT, WHAT, VAL )
  BLACS_SET      ( ICONTXT, WHAT, VAL )
  BLACS_GRIDINIT( ICONTXT, ORDER,           NPROW, NPCOL )
  BLACS_GRIDMAP ( ICONTXT, USERMAP, LDUMAP, NPROW, NPCOL )
```

.. Destruction ..

```
  BLACS_FREEBUFF( ICONTXT, WAIT )
  BLACS_GRIDEXIT( ICONTXT )
  BLACS_ABORT    ( ICONTXT, ERRORNUM )
  BLACS_EXIT     ( CONTINUE )
```

.. Informational ..

```
  BLACS_GRIDINFO( ICONTXT, NPROW, NPCOL, MYPROW, MYPCOL )
  BLACS_PNUM     ( ICONTXT,                PROW,   PCOL )
  BLACS_PCOORD   ( ICONTXT, PNUM,          PROW,   PCOL )
```

.. Miscellaneous ..

```
  BLACS_BARRIER( ICONTXT, SCOPE )
```

Point to point communication

.. Sends ..

```
  vGESD2D( ICONTXT,             M, N, A, LDA, RDEST, CDEST )
  vTRSD2D( ICONTXT, UPLO, DIAG, M, N, A, LDA, RDEST, CDEST )
```

.. Receives ..

```
  vGERV2D( ICONTXT,             M, N, A, LDA, RSRC,  CSRC )
  vTRRV2D( ICONTXT, UPLO, DIAG, M, N, A, LDA, RSRC,  CSRC )
```

Broadcasts

.. Broadcast/send ..

```
vGEBS2D( ICONTXT, SCOPE, TOP,             M, N, A, LDA )
vTRBS2D( ICONTXT, SCOPE, TOP, UPLO, DIAG, M, N, A, LDA )
```

.. Broadcast/receive ..

```
vGEBR2D( ICONTXT, SCOPE, TOP,             M, N, A, LDA, RSRC, CSRC )
vTRBR2D( ICONTXT, SCOPE, TOP, UPLO, DIAG, M, N, A, LDA, RSRC, CSRC )
```

Combine operations

```
vGSUM2D( ICONTXT, SCOPE, TOP, M, N, A, LDA,                 RDEST, CDEST )
vGAMX2D( ICONTXT, SCOPE, TOP, M, N, A, LDA, RA, CA, RCFLAG, RDEST, CDEST )
vGAMN2D( ICONTXT, SCOPE, TOP, M, N, A, LDA, RA, CA, RCFLAG, RDEST, CDEST )
```

BLACS_PINFO

BLACS_PINFO(MYPNUM, NPROCS)

MYPNUM
: (output) INTEGER
: An integer between 0 and (NPROCS - 1) which uniquely identifies each process.

NPROCS
: (output) INTEGER
: The number of processes available for BLACS use.

This routine is used when some initial system information is required before the BLACS are set up. On all platforms except PVM, NPROCS is the actual number of processes available for use (i.e. NPROWS * NPCOLS <= NPROCS). In PVM, the virtual machine may not have been set up before this call, and therefore no parallel machine exists. In this case, NPROCS will be returned as less than one. If a process has been spawned via the keyboard, it will receive MYPNUM of 0, and all other processes will get MYPNUM of -1. This allows the user to distinguish between processes, so that only one reads in data, etc. Only after the virtual machine has been set up (via a call to BLACS_SETUP or SETPVMTIDS) will this routine return the correct values for MYPNUM and NPROCS.

See EXAMPLE

BLACS_SETUP

BLACS_SETUP(MYPNUM, NPROCS)

MYPNUM
: (output) INTEGER
: An integer between 0 and (NPROCS - 1) which uniquely identifies each process.

NPROCS

(input/output) INTEGER

On the process spawned from the keyboard (rather than from pvmspawn), this parameter is input, and indicates the number of processes to create when building the virtual machine. For all other processes, it will be output.

This routine only accomplishes meaningful work in the PVM BLACS. On all other platforms, it is functionally equivalent to BLACS_PINFO. The BLACS assume a static system: you start with a given number of processes, and that is all you will ever have. PVM supplies a dynamic system, allowing processes to be added to the system on the fly. BLACS_SETUP is used to actually allocate the virtual machine and spawn off processes. It reads in a file called `blacs_setup.dat`, who's first line must be the name of your executable. The second line is optional, but if it exists, it should be a PVM spawn flag. Legal values at this time are 0 (PvmTaskDefault), 4 (PvmTaskDebug), 8 (PvmTaskTrace), and 12 (PvmTaskDebug + PvmTaskTrace). The primary reason for this line is to allow the user to easily turn on and off PVM debugging. Additional lines, if any, specify what machines should be added to the current configuration before spawning NPROCS-1 processes to the machines in a round robin fashion. NPROCS is input on the process which has no PVM parent (i.e. MYPNUM=0), and both parameters are output for all processes. Therefore, on PVM systems, the call to BLACS_PINFO informs you that the virtual machine has not been set up, and a call to BLACS_SETUP then sets up the machine and returns the real values for MYPNUM and NPROCS. Note that if the file `blacs_setup.dat` does not exist, the BLACS will prompt the user for the executable name, and processes will be spawned to the current PVM configuration.

See EXAMPLE See blacs_setup.dat EXAMPLE

SETPVMTIDS

SETPVMTIDS(NTASKS, TIDS)

NTASKS

(input) INTEGER

The number of PVM tasks the user has spawned.

TIDS

(input) INTEGER array of dimension (NTASKS)

This array contains the list of NTASKS PVM TIDS to participate in the BLACS.

SETPVMTIDS, as its name implies, is a PVM specific routine. SETPVMTIDS is the advanced PVM user's BLACS_SETUP. BLACS_SETUP may be too restrictive for someone who is using PVM outside the BLACS. For example, they may want to start the main process (process {0,0}) via a call to pvm_spawn, rather than starting it from the keyboard as BLACS_SETUP requires. SETPVMTIDS requires two parameters from the user. The first is the total number of processes (or tasks) that any BLACS grid will use. Remember that the BLACS is a static system: if you have P processes at the beginning of its execution, you must have those same P processes when the BLACS finish execution. Therefore, the user must set NTASKS to be the largest number of nodes of any grid he will use.

The second argument required by SETPVMTIDS is a list of tids in an integer array of at least length NTASKS. All processes require these inputs. This means that in order to use SETPVMTIDS, the PVM user should spawn off all of his processes, keeping their TIDS in an integer array, then send that array to all participating processes, and finally have them all call SETPVMTIDS. At this point, he has performed the actions inherent in a BLACS_SETUP call, and he may then proceed to use the BLACS as usual (can make calls to BLACS_PINFO, and then to BLACS_GRIDINIT or BLACS_GRIDMAP, and then proceed with the normal BLACS code).

BLACS_GRIDINIT

BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)

ICONTXT
: (input/output) INTEGER
: On input, an integer handle indicating the system context to be used in creating the BLACS context. The user may obtain a default system context via a call to BLACS_GET. On output, the integer handle to the created BLACS context.

ORDER
: (input) CHARACTER*1
: Indicates how to map processes to BLACS grid. Choices are:
  - 'R' : Use row-major natural ordering.
  - 'C' : Use column-major natural ordering.
  - ELSE: Use row-major natural ordering.

NPROW
: (input) INTEGER
: Indicates how many process rows the process grid should contain.

NPCOL
: (input) INTEGER
: Indicates how many process columns the process grid should contain.

All BLACS codes must call this routine, or its sister routine BLACS_GRIDMAP. These routines take the available processes, and assign, or map, them into a BLACS process grid. In other words, they establish how the BLACS coordinate system will map into the native machine's process numbering system. Each BLACS grid is contained in a context (its own message passing universe), so that it does not interfere with distributed operations which occur within other grids/contexts. These grid creation routines may be called repeatedly in order to define additional contexts/grids.

The creation of a grid requires input from all processes which are defined to be in it. It is therefore a globally blocking operation. Processes belonging to more than one grid will have to agree on which grid formation will be serviced first, much like the globally blocking sum or broadcast.

These grid creation routines set up various internals for the BLACS, and so one of them must be called before any calls are made to the non-initialization BLACS.

Note that these routines map already-existing processes to a grid: the processes are not created dynamically. On most parallel machines, the processes will be actual processors (hardware), and they are "created" when the user runs his executable. When using the PVM BLACS, if the virtual machine has not been set up yet, the routine BLACS_SETUP should be used to create the virtual machine. If the PVM user wishes to use a virtual machine he has already set up using explicit PVM calls, the routine SETPVMTIDS may be used.

This routine creates a simple `NPROW x NPCOL` process grid. This process grid will use the first `NPROW * NPCOL` processes, and assign them to the grid in a row- or column-major natural ordering.

If these process-to-grid mappings are unacceptable, BLACS_GRIDINIT's more complex sister routine BLACS_GRIDMAP must be called instead.

See EXAMPLE

BLACS_GRIDMAP

BLACS_GRIDMAP(ICONTXT, USERMAP, LDUMAP, NPROW, NPCOL)

ICONTXT
: (input/output) INTEGER
: On input, an integer handle indicating the system context to be used in creating the BLACS context. The user may obtain a default system context via a call to BLACS_GET. On output, the integer handle to the created BLACS context.

USERMAP
: (input) INTEGER array, dimension (LDUMAP, NPCOL)
: Input array indicating the process-to-grid mapping.

LDUMAP
: (input) INTEGER, LDUMAP >= NPROW
: The leading dimension of the 2D array USERMAP

NPROW
: (input) INTEGER
: Indicates how many process rows the process grid should contain.

NPCOL
: (input) INTEGER
: Indicates how many process columns the process grid should contain.

All BLACS codes must call this routine, or its sister routine BLACS_GRIDINIT. These routines take the available processes, and assign, or map, them into a BLACS process grid. In other words, they establish how the BLACS coordinate system will map into the native machine's process numbering system. Each BLACS grid is contained in a context (its own message passing universe), so that it does not interfere with distributed operations which occur within other grids/contexts. These grid creation routines may be called repeatedly in order to define additional contexts/grids.

The creation of a grid requires input from all processes which are defined to be in it. It is therefore a globally blocking operation. Processes belonging to more than one grid will have to agree on which grid formation will be serviced first, much like the globally blocking sum or broadcast.

These grid creation routines set up various internals for the BLACS, and so one of them must be called before any calls are made to the non-initialization BLACS.

Note that these routines map already-existing processes to a grid: the processes are not created dynamically. On most parallel machines, the processes will be actual processors (hardware), and they are "created" when the user runs his executable. When using the PVM BLACS, if the virtual machine has not been set up yet, the routine BLACS_SETUP should be used to create the virtual machine. If the PVM user wishes to use a virtual machine he has already set up using explicit PVM calls, the routine SETPVMTIDS may be used.

This routine allows the user to map processes to the process grid in an arbitrary manner. USERMAP(i,j) holds the process number of the process to be placed in {i, j} of the process grid.

On most distributed systems, this process number will simply by a machine defined number between 0 ... NPROCS-1. For PVM these node numbers will be the PVM TIDS (Task IDs). BLACS_GRIDMAP is not for the inexperienced user -- BLACS_GRIDINIT is much simpler. BLACS_GRIDINIT simply performs a GRIDMAP where the first `NPROW * NPCOL` processes are mapped into the current grid in a row-major natural ordering. BLACS_GRIDMAP allows the experienced user to take advantage of his system's actual layout (i.e. he can map nodes that are physically connected to be neighbors in the BLACS grid, etc.). BLACS_GRIDMAP also opens the way for {\it multigridding}: the user can separate his nodes into arbitrary grids, join them together at some later date, and then re-split them into new grids. BLACS_GRIDMAP also provides the ability to make arbitrary grids or subgrids (e.g., a "nearest neighbor" grid), which can greatly facilitate operations among processes which do not fall on a row or column of the main process grid.

See EXAMPLE

BLACS_ABORT

BLACS_ABORT(ICONTXT, ERRORNUM)

ICONTXT
: (input) INTEGER.
: Integer handle indicating the context which is aborting.

ERRORNUM
: (input) INTEGER.
: User defined integer error number.

When a catastrophic error occurs, the user may need to abort all processes. BLACS_ABORT exists for this reason. Note that both parameters are input, but that BLACS_ABORT uses them only in printing out the error message. The context handle passed in may be anything (i.e., it need not be a valid context handle). This routine kills *all* BLACS processes, not just those confined to a particular context.

BLACS_GRIDEXIT

BLACS_GRIDEXIT(ICONTXT)

ICONTXT
: (input) INTEGER.
: Integer handle indicating the context to be freed.

Contexts consume resources, and therefore the user should release them when they are no longer needed. BLACS_GRIDEXIT frees a context. After the freeing of a context, the context no longer exists, and its handle may be re-used if new contexts are defined.

BLACS_BARRIER

BLACS_BARRIER(ICONTXT, SCOPE)

ICONTXT
    (input) INTEGER.
    Integer handle indicating the context.
SCOPE
    (input) CHARACTER*1.
    Indicates whether a process row (`SCOPE='R'`), column (`'C'`), or entire grid (`'A'`) will participate in barrier.

This routines holds up execution of all processes within the indicated scope until they have all called the routine.

BLACS_GRIDINFO

BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYPROW, MYPCOL)

ICONTXT
    (input) INTEGER.
    Integer handle indicating the context.
NPROW
    (output) INTEGER.
    On output, it contains the number of process rows in the current process grid.
NPCOL
    (output) INTEGER.
    On output, it contains the number of process columns in the current process grid.
MYPROW
    (output) INTEGER.
    On output, it contains the calling process's row coordinate in the process grid.
MYPCOL
    (output) INTEGER.
    On output, it contains the calling process's column coordinate in the process grid.

Returns information on the current grid. If the context handle does not point at a valid context, all quantities are returned as -1.

See EXAMPLE

BLACS_PCOORD

BLACS_PCOORD(ICONTXT, PNUM, PROW, PCOL)

ICONTXT
    (input) INTEGER.
    Integer handle indicating the context.
PNUM
    (input) INTEGER.
    The process number who's coordinates are to be determined. This is the process number of

the underlying machine (i.e., it will be a TID for PVM).

PROW
: (output) INTEGER.
  On output, the row coordinate of process PNUM in the BLACS grid.

PCOL
: (output) INTEGER.
  On output, the column coordinate of process PNUM in the BLACS grid.

Given the system process number, returns the row and column coordinates in the BLACS' process grid.

See EXAMPLE

BLACS_PNUM

INTEGER FUNCTION BLACS_PNUM(ICONTXT, PROW, PCOL)

ICONTXT
: (input) INTEGER.
  Integer handle indicating the context.

PROW
: (input) INTEGER.
  The row coordinate of the process who's system process number is to be determined.

PCOL
: (input) INTEGER.
  The column coordinate of the process who's system process number is to be determined.

This function returns the system process number of the process at {PROW, PCOL} in the process grid.

See EXAMPLE

BLACS_GET

BLACS_GET(ICONTXT, WHAT, VAL)

ICONTXT
: (input) INTEGER.
  On WHATs that are tied to a particular context, this is the integer handle indicating the context. Otherwise, it is ignored.

WHAT
: (input) INTEGER.
  What BLACS internal(s) should be returned in VAL. Present options are:
  - WHAT = 0 : Handle indicating default system context;
  - WHAT = 1 : The BLACS message ID range;
  - WHAT = 2 : The BLACS debug level the library was compiled with;
  - WHAT = 10: Handle indicating the system context used to define the BLACS context whose handle is ICONTXT;

- WHAT = 11: Number of rings multiring topology is presently using;
- WHAT = 12: Number of branches general tree topology is presently using.

VAL
: (output) INTEGER.
: The value the BLACS internal presently is.

Gets the values the BLACS are using for internal defaults. Some values are tied to a BLACS context, and some are more general. The most common use is in retrieving a default system context for input into BLACS_GRIDINIT or BLACS_GRIDMAP. Some systems, such as MPI, supply their own version of context. For those users who mix system code with BLACS code, we therefore need to be able to form a BLACS context in reference to a system context. Thus, the grid creation routines take a system context as input. If you wish to have strictly portable code, you may use BLACS_GET to retrieve a default system context which will include all available processes. This value is not tied to a BLACS context, obviously, so the parameter ICONTXT is unused.

BLACS_GET returns information on three quantities which are tied to an individual BLACS context, which is passed in as ICONTXT. The information which may be retrieved is:

- The handle of the system context upon which this BLACS context was defined.
- The number of rings for TOP = 'M' (multiring broadcast);
- The number of branches for TOP = 'T' (general tree broadcast/general tree gather).

See EXAMPLE

BLACS_SET

BLACS_SET(ICONTXT, WHAT, VAL)

ICONTXT
: (input) INTEGER.
: On WHATs that are tied to a particular context, this is the integer handle indicating the context. Otherwise, it is ignored.

WHAT
: (input) INTEGER.
: What BLACS internal should be set. Present values are:
  - 1 = The BLACS message ID range;
  - 11 = number of rings for multiring topology to use;
  - 12 = number of branches for general tree topology to use.

VAL (input) INTEGER array of dimension (*)
: The value(s) to set internals to. Its specific meaning is dependent on WHAT, as discussed below.

Sets BLACS internal defaults. The action taken is dependent upon WHAT, as follows:

WHAT = 1 : Setting the BLACS message ID range
: If the user wishes to mix the BLACS with other message-passing packages, he may restrict the BLACS to a certain message ID range, which he ensures is not used by the non-BLACS routines. The message ID range must be set *before* the first call to BLACS_GRIDMAP or BLACS_GRIDINIT. Subsequent calls will have no effect. Because the message ID range is not tied to a particular context, the parameter ICONTXT is ignored, and VAL is defined as:

VAL (input) INTEGER array of dimension (2)
VAL(1) : The smallest message ID (also called message type or message tag) the BLACS should use.
VAL(2) : The largest message ID (also called message type or message tag) the BLACS should use.
WHAT = 11 : Set number of rings for TOP = 'M' (multiring broadcast)
This quantity is tied to a context, thus ICONTXT is used, and VAL is defined as:
VAL (input) INTEGER array of dimension (1)
VAL(1): The number of rings for multiring topology to use.
WHAT = 12 : Set number of branches for TOP = 'T' (general tree broadcast/general tree gather).
This quantity is tied to a context, thus ICONTXT is used, and VAL is defined as:
VAL (input) INTEGER array of dimension (1)
VAL(1): The number of branches for general tree topology to use.

BLACS_FREEBUFF

BLACS_FREEBUFF(ICONTXT, WAIT)

ICONTXT
(input) INTEGER.
Integer handle indicating the context.
WHAT
(input) INTEGER.
Whether to wait on non-blocking operations or not.

The BLACS have at least one internal buffer that is used for packing messages (the number of internal buffers depends on what platform you are running the BLACS on). On systems where memory is tight, keeping this buffer(s) around may become expensive. Calling this routine will release the BLACS buffer. However, the next call to a communication routine which requires packing will cause the buffer to be reallocated.

The parameter WAIT determines whether the BLACS should wait for any non-blocking operations to complete or not. If WAIT = 0, the BLACS will free any buffers that can be freed without waiting. If WAIT is not 0, the BLACS will free all internal buffers, even if non-blocking operations must be completed first.

BLACS_EXIT

BLACS_EXIT(CONTINUE)

CONTINUE
(input) INTEGER.
If CONTINUE is non-zero, it is assume the user will continue using the machine after the BLACS are done. Otherwise, it is assumed that no message passing will be done after the BLACS_EXIT call.

This routine should be called when a process has finished all use of the BLACS. It frees all BLACS contexts and releases all memory the BLACS have allocated. CONTINUE indicates whether the user will be using the underlying communication platform after the BLACS are finished. This information is most important for the PVM BLACS. If CONTINUE is set to 0, then `pvm_exit` will be called; otherwise, it will not. If the user sets CONTINUE not equal to 0, he is indicating that he will be calling explicit PVM send/recvs after the BLACS are done, so that the process cannot tell the virtual machine that it is done. It then becomes the user's responsibility to make sure his code calls `pvm_exit`. PVM users should either call BLACS_EXIT or explicitly call `pvm_exit` to avoid PVM problems.

See EXAMPLE

Point to point sends

`vGESD2D( ICONTXT,             M, N, A, LDA, RDEST, CDEST )`

`vTRSD2D( ICONTXT, UPLO, DIAG, M, N, A, LDA, RDEST, CDEST )`

ICONTXT
: (input) INTEGER.
  Integer handle indicating the context.

UPLO, DIAG, M, N, A, LDA
: (input) These parameters describe the matrix to be sent. See MATRIX SHAPES for details.

RDEST
: (input) INTEGER
  The process row coordinate of the process to send the message to.

CDEST
: (input) INTEGER
  The process column coordinate of the process to send the message to.

These calls take the indicated matrix and send it to the destination process located at {RDEST, CDEST} in the process grid. Return from the routine indicates that the buffer (the matrix A) may be reused. The routine is locally blocking, i.e. it will return even if the corresponding receive is not posted.

See EXAMPLE

Point to point receives

`vGERV2D( ICONTXT,             M, N, A, LDA, RSRC, CSRC )`

`vTRRV2D( ICONTXT, UPLO, DIAG, M, N, A, LDA, RSRC, CSRC )`

ICONTXT
: (input) INTEGER.
  Integer handle indicating the context.

UPLO, DIAG, M, N, LDA
: (input) These parameters describe the matrix to be sent. See MATRIX SHAPES for details.

A

(output) array of dimension (LDA,N)
Array to receive the incoming message into.

RSRC

(input) INTEGER
The process row coordinate of the source of the message.

CSRC

(input) INTEGER
The process column coordinate of the source of the message.

These calls receive a message from process {RSRC, CSRC} into the matrix A. This routine is globally-blocking, i.e. return from the routine indicates that the message has been received into A.

See EXAMPLE

Broadcast/send

```
vGEBS2D( ICONTXT, SCOPE, TOP,             M, N, A, LDA )

vTRBS2D( ICONTXT, SCOPE, TOP, UPLO, DIAG, M, N, A, LDA )
```

ICONTXT

(input) INTEGER.
Integer handle indicating the context.

SCOPE

(input) CHARACTER*1
Indicates what scope the broadcast should proceed on. Limited to 'Row', 'Column', or 'All'.

TOP

(input) CHARACTER*1
Indicates the communication pattern to use for the broadcast. See BLACS TOPOLOGY for details.

UPLO, DIAG, M, N, A, LDA

(input) These parameters describe the matrix to be sent. See MATRIX SHAPES for details.

These routines start a broadcast along a scope. All other processes within the scope must call broadcast/receive for the broadcast to proceed. At the end of a broadcast, all processes within the scope will possess the data in the matrix A. Broadcasts may be globally-blocking. This means no process is guaranteed to return from a broadcast until all processes in the scope have called the appropriate routine (broadcast/send or broadcast/receive).

See EXAMPLE

Broadcast/receive

```
vGEBR2D( ICONTXT, SCOPE, TOP,             M, N, A, LDA, RSRC, CSRC )

vTRBR2D( ICONTXT, SCOPE, TOP, UPLO, DIAG, M, N, A, LDA, RSRC, CSRC )
```

ICONTXT
  (input) INTEGER.
  Integer handle indicating the context.
SCOPE
  (input) CHARACTER*1
  Indicates what scope the broadcast should proceed on. Limited to 'Row', 'Column', or 'All'.
TOP
  (input) CHARACTER*1
  Indicates the communication pattern to use for the broadcast. See BLACS TOPOLOGY for details.
UPLO, DIAG, M, N, LDA
  (input) These parameters describe the matrix to be sent. See MATRIX SHAPES for details.
A array of dimension (LDA,N)
  (output)
  Array to receive the incoming message into.
RSRC
  (input) INTEGER
  The process row coordinate of the process who called broadcast/send.
CSRC
  (input) INTEGER
  The process column coordinate of the process who called broadcast/send.

These routines receive and participate in a broadcast along a scope. At the end of a broadcast, all processes within the scope will possess the data in the matrix A. Broadcasts may be globally-blocking. This means no process is guaranteed to return from a broadcast until all processes in the scope have called the appropriate routine (broadcast/send or broadcast/receive).

See EXAMPLE

Combine: element-wise summation

`vGSUM2D( ICONTXT, SCOPE, TOP, M, N, A, LDA, RDEST, CDEST )`

ICONTXT
  (input) INTEGER.
  Integer handle indicating the context.
SCOPE
  (input) CHARACTER*1
  Indicates what scope the combine should proceed on. Limited to 'Row', 'Column', or 'All'.
TOP
  (input) CHARACTER*1
  Indicates the communication pattern to use during the combine. See BLACS TOPOLOGY for details.
UPLO, DIAG, M, N, LDA
  (input) These parameters describe the matrix to be sent. See MATRIX SHAPES for details.
A
  (input/output) array of dimension (LDA,N)

On input, the matrix to be added to produce the sum. On output, it contains the result (if this process is selected to receive the answer), or intermediate (useless) results (if process not selected to receive the result).

RDEST
: (input) INTEGER
  The process row coordinate of the process who should receive the result. If RDEST = -1, all processes within the indicated scope receive the answer.

CDEST
: (input) INTEGER
  The process column coordinate of the process who should receive the result. If RDEST = -1, CDEST is ignored.

The sum is element-wise, i.e., each element of A is summed with the corresponding element of the other process's matrices. Combines may be globally-blocking, so they must be programmed as if no process returns until all have called the routine.

See EXAMPLE

Combine: element-wise absolute value maximization/minimization

vGAMX2D(ICONTXT, SCOPE, TOP, M, N, A, LDA, RA, CA, RCFLAG, RDEST, CDEST)

vGAMN2D(ICONTXT, SCOPE, TOP, M, N, A, LDA, RA, CA, RCFLAG, RDEST, CDEST)

ICONTXT
: (input) INTEGER.
  Integer handle indicating the context.

SCOPE
: (input) CHARACTER*1
  Indicates what scope the combine should proceed on. Limited to 'Row', 'Column', or 'All'.

TOP
: (input) CHARACTER*1
  Indicates the communication pattern to use during the combine. See BLACS TOPOLOGY for details.

UPLO, DIAG, M, N, LDA
: (input) These parameters describe the matrix to be operated on. See MATRIX SHAPES for details.

RCFLAG
: (input) INTEGER
  If RCFLAG = -1, then the arrays RA and CA are not referenced and need not exist.
  Otherwise, RCFLAG indicates the leading dimension of these arrays, and so must be >= M.

A
: (input/output) array of dimension (LDA,N)
  On input, the matrix to be compared with to produce the max/min. On output, it contains the result (if this process is selected to receive the answer), or intermediate (useless) results (if process not selected to receive the result).

RA

(output) INTEGER array of dimension (RCFLAG,N)
If RCFLAG = -1, this array will not be referenced, and need not exist. Otherwise it is an integer array (of size at least RCFLAG x N) indicating the row index of the process that provided the maximum/minimum. If the calling process is not selected to receive the result, this array will contain intermediate (useless) results.

CA
: (output) INTEGER array of dimension (LDIA,N)
If RCFLAG = -1, this array will not be referenced, and need not exist. Otherwise it is an integer array (of size at least RCFLAG x N) indicating the column index of the process that provided the maximum/minimum. If the calling process is not selected to receive the result, this array will contain intermediate (useless) results.

RDEST
: (input) INTEGER
The process row coordinate of the process who should receive the result. If RDEST = -1, all processes within the indicated scope receive the answer.

CDEST
: (input) INTEGER
The process column coordinate of the process who should receive the result. If RDEST = -1, CDEST is ignored.

Performs an elementwise absolute value max/min, i.e., each element of A is compared with the corresponding element of the other process's matrices. Remember: the value of A is returned, but the absolute value is used to determine the max/min (the 1-norm is used for complex numbers). Combines may be globally-blocking, so they must be programmed as if no process returns until all have called the routine.

See EXAMPLE