UPCBLAS REFERENCE MANUAL

VERSION 1.0

Contents

1 Introduction

UPCBLAS contains a subset of representative BLAS routines. Table 1 lists all the implemented routines. Unlike message-passing based codes which require the input data to be distributed in the local memory of each process, UPC functions simplify this data distribution by using shared arrays. In UPC shared arrays implicity distribute their elements among the parts of the shared memory with affinity to the different threads.

BLAS level	Tblasname	Action
BLAS1	Tcopy	Copies a vector
	Tswap	Swaps the elements of two vectors
	Tscal	Scales a vector by a scalar
	Taxpy	Updates a vector using another one: $y = \alpha * x + y$
	Tdot	Dot product between two vectors
	Tnrm2	Euclidean norm of a vector
	Tasum	Sums the absolute value of the elements of a vector
BLAS2	Tgemv	Matrix-vector product
	Tger	Outer product between two vectors
	Ttrsv	Solves a triangular system of equations
BLAS3	Tgemm	Matrix-matrix product
	Ttrsm	Solves a block of triangular systems of equations

Table 1: UPCBLAS routines. All the functions follow the naming convention: `upc_blas_Tblasname`, where "T" represents the data type (s=float; d=double; c=single precision complex; z=double precision complex) and `blasname` is the name of the routine in the sequential BLAS library

2 Installation Instructions

1. A sequential numerical library with support for BLAS routines (e.g. MKL, GSL, LibSci...) must be available in the system previously to the installation.

2. Create the appropriate file Smake.inc within the main folder selecting the options for the compiler and the underlying BLAS library. There is one configuration example in SmakeExample.inc. You only have to change the variables for the appropriate information in your system. The variables to specify are:

    - UPCBLLIBDIR: Directory for objects.
    - UPCBLBINDIR: Directory for test executables. Not necessary if you do not compile the tests.
    - UPC_COMPILER: Name of UPC compiler. Now UPCBLAS can work with BERKELEY, HP and CRAY
    - UPCC: Path to the UPC compiler.
    - UPC_OPTS: Optional flags to use with the UPC compiler.
    - UPC_NET: Network configuration of for the UPC compiler. This information is available by calling the UPC compile with –version
    - UPC_NET_OPTS: Additional options to the network selected for the compiler (for instance -pthreads).
    - THREADS: Number of static threads.
    - THREAD_INDIC: Way to indicate the number of static threads (-T for Berkeley UPC, -X for Cray UPC...).
    - SEQBLAS: Name of the sequential underlying numerical library. The options are: GSL, GOTOBLAS, MKL and LIBSCI.
    - SEQBLAS_INCL: Path to the headers of the sequential library.
    - SEQBLAS_LIBS_PATH: Path to the objects of the sequential library.
    - SEQBLAS_LIBS: Objects of the sequential library that need to be linked.
    - CC: C compiler for some auxiliar non-upc functions.
    - CC_FLAGS: Optional flags to use with the C compiler.

3. Type ”make install“ from the main folder.

4. If you want to compile some tests, just type ”make tests“ from the main folder.

If you need support for a different configuration or you find any problem, please contact to `jgonzalezd@udc.es`.

3 Enumerated Values

The UPCBLAS routines use enumerated values to specify some characteristics of the input matrices. Their declaration is available in the file *include/types.h*.

- `UPC_PBLAS_DIMMDIST`: It indicates if the matrix is distributed by rows or columns.
  - `upc_pblas_rowDist`
  - `upc_pblas_colDist`
- `UPC_PBLAS_TRANSPOSE`: It indicates if the matrix is transposed.
  - `upc_pblas_noTrans`
  - `upc_pblas_trans`
  - `upc_pblas_conjTrans`
- `UPC_PBLAS_UPLO`: It indicates if the matrix is upper or lower triangular.
  - `upc_pblas_upper`
  - `upc_pblas_lower`
- `UPC_PBLAS_DIAG`: It indicates if all the elements in the main diagonal of the triangular matrix are equal to 1 or not.
  - `upc_pblas_nonUnit`
  - `upc_pblas_unit`
- `UPC_PBLAS_SIDE`: In the BLAS3 triangular solver it indicates if the triangular matrix is on the left or on the right side of the equation.
  - `upc_pblas_left`
  - `upc_pblas_right`

4 BLAS1 Routines

These routines perform vector-vector operations. Their headers are available in the file *include/pblas1.h*.

4.1 upc_blas_Tcopy

Function to copy vector `x` to vector `y`.

SYNTAX:

- `int upc_blas_scopy(int block_size, int size, shared void *x, shared void *y)`
- `int upc_blas_dcopy(int block_size, int size, shared void *x, shared void *y)`
- `int upc_blas_ccopy(int block_size, int size, shared void *x, shared void *y)`
- `int upc_blas_zcopy(int block_size, int size, shared void *x, shared void *y)`

PARAMETERS:

- `IN block_size`: Storage block size for vectors `x` and `y`.
- `IN size`: Vectors length.
- `IN x`: Pointer to the position of the shared array where the source vector `x` is stored.
- `OUT y`: Pointer to the position of the shared array where the destination vector `y` is stored.
- returns:
  - 0 if everything is ok.
  - < 0 if a parameter error occurs. The exact value is -j if the wrong parameter is the jth one.

RESTRICTIONS:

- This function treats pointers `x` and `y` as if they had type `shared [block_size] type[size]`.
- The address of the first element of `x` and `y` must have phase 0.
- The first element of `x` and `y` must be in parts of the shared memory with affinity to the same thread.
- If `x` or `y` overlap, the behavior is undefined.

4.2 upc_blas_Tswap

Function to swap the elements of two vectors.

<u>SYNTAX:</u>

- `int upc_blas_sswap(int block_size, int size, shared void *x, shared void *y)`
- `int upc_blas_dswap(int block_size, int size, shared void *x, shared void *y)`
- `int upc_blas_cswap(int block_size, int size, shared void *x, shared void *y)`
- `int upc_blas_zswap(int block_size, int size, shared void *x, shared void *y)`

<u>PARAMETERS:</u>

- `IN block_size`: Storage block size for the vectors to swap (`x` and `y`).
- `IN size`: Vectors length.
- `IN/OUT x,y`: Pointers to the positions of the shared arrays where vectors `x` and `y` are stored.
- returns:
  - 0 if everything is ok.
  - < 0 if a parameter error occurs. The exact value is -j if the wrong parameter is the jth one.

<u>RESTRICTIONS:</u>

- This function treats pointers `x` and `y` as if they had type `shared [block_size] type[size]`.
- The address of the first element of `x` and `y` must have phase 0.
- The first element of `x` and `y` must be in parts of the shared memory with affinity to the same thread.
- If `x` or `y` overlap, the behavior is undefined.

4.3 upc_blas_Tscal

Function to scale a vector by a scalar.

<u>SYNTAX:</u>

- `int upc_blas_sscal(int block_size, int size, float alpha, shared void *x)`
- `int upc_blas_dscal(int block_size, int size, double alpha, shared void *x)`
- `int upc_blas_cscal(int block_size, int size, void *alpha, shared void *x)`
- `int upc_blas_zscal(int block_size, int size, void *alpha, shared void *x)`
- `int upc_blas_csscal(int block_size, int size, float alpha, shared void *x)`
- `int upc_blas_zdscal(int block_size, int size, double alpha, shared void *x)`

<u>PARAMETERS:</u>

- `IN block_size`: Storage block size for the vector.
- `IN size`: Vector length.
- `IN alpha`: Scale factor.
- `IN/OUT x`: Pointer to the position of the shared array where vector `x` is stored.
- returns:
  - 0 if everything is ok.
  - < 0 if a parameter error occurs. The exact value is -j if the wrong parameter is the jth one.

<u>RESTRICTIONS:</u>

- This function treats pointer `x` as if it had type `shared [block_size] type[size]`.
- The address of the first element of `x` must have phase 0.

4.4 upc_blas_Taxpy

Function to update a vector by adding to it another vector scaled by a factor. These vectors are stored in shared arrays. The equivalent function is: $y = \alpha * x + y$.

SYNTAX:

- `int upc_blas_saxpy(int block_size, int size, float alpha, shared void *x, shared void *y)`
- `int upc_blas_daxpy(int block_size, int size, double alpha, shared void *x, shared void *y)`
- `int upc_blas_caxpy(int block_size, int size, void * alpha, shared void *x, shared void *y)`
- `int upc_blas_zaxpy(int block_size, int size, void * alpha, shared void *x, shared void *y)`

PARAMETERS:

- `IN block_size`: Storage block size for the vectors.
- `IN size`: Vectors length.
- `IN alpha`: Scale factor.
- `IN x`: Pointer to the position of the shared array where the source vector `x` is stored.
- `IN/OUT y`: Pointer to the position of the shared array where vector `y` is stored.
- returns:
  - 0 if everything is ok.
  - < 0 if a parameter error occurs. The exact value is -j if the wrong parameter is the jth one.
  - +1 if an internal memory error occurs.

RESTRICTIONS:

- This function treats pointers `x` and `y` as if they had type `shared [block_size] type[size]`.
- The address of the first element of `x` and `y` must have phase 0.
- The first element of `x` and `y` must be in parts of the shared memory with affinity to the same thread.
- If `x` or `y` overlap, the behavior is undefined.

4.5 upc_blas_Tsdot

Function to perform the dot product between two vectors stored in shared arrays.

SYNTAX:

- `int upc_blas_sdot(int block_size, int size, shared void *x, shared void *y, shared float *dst)`
- `int upc_blas_ddot(int block_size, int size, shared void *x, shared void *y, shared double *dst)`
- `int upc_blas_cdotc(int block_size, int size, shared void *x, shared void *y, shared void *_dst)`
- `int upc_blas_zdotc(int block_size, int size, shared void *x, shared void *y, shared void *_dst)`
- `int upc_blas_cdotu(int block_size, int size, shared void *x, shared void *y, shared void *_dst)`
- `int upc_blas_zdotu(int block_size, int size, shared void *x, shared void *y, shared void *_dst)`

PARAMETERS:

- `IN block_size`: Storage block size for the vectors.
- `IN size`: Vectors length.
- `IN x,y`: Pointers to the positions of the shared arrays where the source vectors `x` and `y` are stored.
- `OUT dst`: Pointer to shared memory where the dot product result will be written. This pointer must have memory allocated for one element.
- returns:
  - 0 if everything is ok.
  - <0 if a parameter error occurs. The exact value is -j if the wrong parameter is the jth one.
  - +1 if an internal memory error occurs.

RESTRICTIONS:

- This function treats pointers `x` and `y` as if they had type: `shared [block_size] type[size]`.
- The address of the first element of `x` and `y` must have phase 0.

- The first element of `x` and `y` must be in parts of the shared memory with affinity to the same thread.
- If `x` or `y` overlap, the behavior is undefined.

4.6 upc_blas_Tnrm2

Function to perform the euclidean norm of a vector stored in a shared array.

SYNTAX:

- `int upc_blas_snrm2(int block_size, int size, shared void *x, shared float *dst)`
- `int upc_blas_dnrm2(int block_size, int size, shared void *x, shared double *dst)`
- `int upc_blas_scnrm2(int block_size, int size, shared void * x, shared float *dst)`
- `int upc_blas_dznrm2(int block_size, int size, shared void * x, shared double *dst)`

PARAMETERS:

- `IN block_size`: Storage block size for vector `x`.
- `IN size`: Vector length.
- `IN x`: Pointer to the position of the shared array where the source vector `x` is stored.
- `OUT dst`: Pointer to the position of shared memory where the norm result is stored. This pointer must have memory allocated for one element.
- returns:
  - 0 if everything is ok.
  - < 0 if a parameter error occurs. The exact value is -j if the wrong parameter is the jth one.

RESTRICTIONS:

- This function treats pointer `x` as if it had type `shared [block_size] type[size]`.
- The address of the first element of `x` must have phase 0.

4.7 upc_blas_Tasum

Function to perform the sum of the absolute values of all the elements of a vector.

SYNTAX:

- `int upc_blas_sasum(int block_size, int size, shared void * x, shared float * dst)`
- `int upc_blas_dasum(int block_size, int size, shared void * x, shared double * dst)`
- `int upc_blas_scasum(int block_size, int size, shared void * x, shared float * dst)`
- `int upc_blas_dzasum(int block_size, int size, shared void * x, shared double * dst)`

PARAMETERS:

- `IN block_size`: Storage block size for vector `x`.
- `IN size`: Vector length.
- `IN x`: Pointer to the position of the shared array where the source vector `x` is stored.
- `OUT dst`: Pointer to the position of shared memory where the sum result is stored. This pointer must have memory allocated for one element.
- returns:
  - 0 if everything is ok.
  - < 0 if a parameter error occurs. The exact value is -j if the wrong parameter is the jth one.

RESTRICTIONS:

- This function treats pointer `x` as if it had type `shared [block_size] type[size]`.
- The address of the first element of `x` must have phase 0.

5 BLAS2 Routines

These routines perform matrix-vector operations. Their headers are available in the file *include/pblas2.h*.

5.1 upc_blas_Tgemv

Function to perform the matrix-vector product: $y = \alpha * A * x + \beta * y$ (with transpose variants of matrix `A`).

<u>SYNTAX:</u>

- `int upc_blas_sgemv(UPC_PBLAS_DIMMDIST dimmDist, int block_size, int sec_block_size, UPC_PBLAS_TRANSPOSE transpose, int m, int n, float alpha, shared void *A, int lda, shared void *x, float beta, shared void *y)`
- `int upc_blas_dgemv(UPC_PBLAS_DIMMDIST dimmDist, int block_size, int sec_block_size, UPC_PBLAS_TRANSPOSE transpose, int m, int n, double alpha, shared void *A, int lda, shared void *x, double beta, shared void *y)`
- `int upc_blas_cgemv(UPC_PBLAS_DIMMDIST dimmDist, int block_size, int sec_block_size, UPC_PBLAS_TRANSPOSE transpose, int m, int n, void *alpha, shared void *A, int lda, shared void *x, void beta, shared void *y)`
- `int upc_blas_zgemv(UPC_PBLAS_DIMMDIST dimmDist, int block_size, int sec_block_size, UPC_PBLAS_TRANSPOSE transpose, int m, int n, void *alpha, shared void *A, int lda, shared void *x, void beta, shared void *y)`

<u>PARAMETERS:</u>

- `IN dimmDist`: Enumerated value representing the matrix dimension that will be distributed among threads. Each thread must have one or more complete rows or columns depending on this parameter. If `upc_pblas_rowDist` is selected, each thread has complete rows. Columns are distributed entirely in the `upc_pblas_colDist` case.
- `IN block_size`: Number of rows or columns (depending on `dimmDist`) of the matrix that are consecutively distributed among all threads. For example, in the `upc_pblas_rowDist` case, the first `block_size` rows will correspond to thread 0, the second `block_size` ones to thread 1, etc. Remember that blocks are built using whole rows/columns and not individual elements.

- `IN sec_block_size`: Depending on the selected `dimmDist`, the block size of one of the source vectors is automatically determined (see RESTRICTIONS below) but the distribution of the other vector can vary. This parameter is used to indicate the block size of the vector which is not automatically defined.
- `IN transpose`: Enumerated value to indicate if the matrix is transposed.
- `IN m`: Number of rows of matrix `A`.
- `IN n`: Number of columns of matrix `A`.
- `IN alpha`: Scale factor for matrix `A`.
- `IN A`: Pointer to the position of the shared array where the source matrix `A` is stored.
- `IN lda`: It specifies the first dimension of `A` as declared in the calling program (used to work with submatrices). It must be at least `n`.
- `IN x`: Pointer to the position of the shared array where the source vector `x` is stored.
- `IN beta`: Scale factor for vector `y`.
- `IN/OUT y`: Pointer to the position of the shared array where vector `y` is stored.
- returns:
  - 0 if everything is ok.
  - < 0 if a parameter error occurs. The exact value is -j if the wrong parameter is the jth one.
  - +1 if an internal memory error occurs.

<u>RESTRICTIONS:</u>

The `block_size` specifications are different depending on the `dimmDist` selection:

1. `upc_pblas_rowDist`: All the elements in a row must have affinity to the same thread but the number of consecutive rows in each thread can change. In this case, this function treats array pointers as if they had type:

   - `A:shared [block_size*lda] type [m*lda]`
   - `x:shared [sec_block_size] type [n]`
   - `y:shared [block_size] type [m]`

   In transpose case:

   - `A:shared [block_size] type [m*lda]`
   - `x:shared [sec_block_size] type [m]`
   - `y:shared [block_size] type [n]`

Besides, there are some limitations about the `block_size` and `sec_block_size` parameters:

- The address of the first element of `y` must have phase 0.
- The address of the first element of `x` must have phase 0.
- The first element of `A` and `y` must be in parts of the shared memory with affinity to the same thread.
- If `A` non-transpose:
  - The first element of `A` must be in the first row of a block in the shared memory space.
- If `A` transpose:
  - The address of the first element of `A` must have phase 0.
  - The first element of all rows must have affinity to thread 0 with phase 0. To achieve this, the following condition must hold: $lda\%(block\_size * THREADS) == 0$.

2. `upc_pblas_colDist`: All the elements in a column must have affinity to the same thread but the number of consecutive columns in each thread can change. In this case, this function treats array pointers as if they had type:

   - `A:shared [block_size] type [m*lda]`
   - `x:shared [block_size] type [n]`
   - `y:shared [sec_block_size] type [m]`

   In transpose case:

   - `A:shared [block_size*lda] type [m*lda]`
   - `x:shared [block_size] type [m]`
   - `y:shared [sec_block_size] type [n]`

   Besides, there are some limitations about the `block_size` and `sec_block_size` parameters:

   - The address of the first element of `y` must have phase 0.
   - The address of the first element of `x` must have phase 0.
   - The first element of `A` and `x` must be in parts of the shared memory with affinity to the same thread.
   - If `A` non-transpose:
     - The address of the first element of `A` must have phase 0.
     - The first element of all rows must have affinity to thread 0 with phase 0. To achieve this, the following condition must hold: $lda\%(block\_size * THREADS) == 0$.

- If `A` transpose:
  - The first element of `A` must be in the first row of a block in the shared memory space.

If any array overlaps, the behavior is undefined.

ADVICE:

The `upc_pblas_rowDist` case is much more efficient than the`upc_pblas_colDist` case.

5.2 upc_blas_Tger

Function to perform the outer product of two vectors: $A = \alpha * x * y' + A$.

SYNTAX:

- `int upc_blas_sger(UPC_PBLAS_DIMMDIST dimmDist, int block_size, int sec_block_size, int m, int n, float alpha, shared void *x, shared void *y, shared void *A, int lda)`
- `int upc_blas_dger(UPC_PBLAS_DIMMDIST dimmDist, int block_size, int sec_block_size, int m, int n, double alpha, shared void *x, shared void *y, shared void *A, int lda)`
- `int upc_blas_cgerc(UPC_PBLAS_DIMMDIST dimmDist, int block_size, int sec_block_size, int m, int n, void *alpha, shared void *x, shared void *y, shared void *A, int lda)`
- `int upc_blas_zgerc(UPC_PBLAS_DIMMDIST dimmDist, int block_size, int sec_block_size, int m, int n, void *alpha, shared void *x, shared void *y, shared void *A, int lda)`
- `int upc_blas_cgeru(UPC_PBLAS_DIMMDIST dimmDist, int block_size, int sec_block_size, int m, int n, void *alpha, shared void *x, shared void *y, shared void *A, int lda)`
- `int upc_blas_zgeru(UPC_PBLAS_DIMMDIST dimmDist, int block_size, int sec_block_size, int m, int n, void *alpha, shared void *x, shared void *y, shared void *A, int lda)`

PARAMETERS:

- `IN dimmDist`: Enumerated value representing the matrix dimension that will be distributed among threads. Each thread must have one or more complete rows orcolumns depending on this parameter. If `upc_pblas_rowDist` is selected, each thread has complete rows. Columns are distributed entirely in the `upc_pblas_colDist` case.

- `IN block_size`: Number of rows or columns (depending on `dimmDist`) of the matrix that are consecutively distributed among all threads. For example, in the `upc_pblas_rowDist` case, the first `block_size` rows will correspond to thread 0, the second `block_size` ones to thread 1, etc. Remember that blocks are built using whole rows/columns and not individual elements.
- `IN sec_block_size`: Depending on the selected `dimmDist`, the block size of one of the source vectors is automatically determined (see RESTRICTIONS below) but the distribution of the other vector can vary. This parameter is used to indicate the block size of the vector not automatically defined.
- `IN m`: Number of rows of matrix `A`.
- `IN n`: Number of columns of matrix `A`.
- `IN alpha`: Scale factor.
- `IN x`: Pointer to the position of the shared array where the source vector `x` is stored.
- `IN y`: Pointer to the position of the shared array where the source vector `y` is stored.
- `IN/OUT A`: Pointer to the position of the shared array where matrix `A` is stored.
- `IN lda`: It specifies the first dimension of `A` as declared in the calling program (used to work with submatrices). It must be at least `n`.
- returns:
  - 0 if everything is ok.
  - < 0 if a parameter error occurs. The exact value is -j if the wrong parameter is the jth one.
  - +1 if an internal memory error occurs.

RESTRICTIONS:

The `block_size` specifications are different depending on the `dimmDist` selection:

1. `upc_pblas_rowDist`: All the elements in a row must have affinity to the same thread but the number of consecutive rows in each thread can change. In this case, this function treats array pointers as if they had type:
   - `A:shared [block_size*lda] type [m*lda]`
   - `x:shared [sec_block_size] type [m]`
   - `y:shared [block_size] type [n]`

   Besides, there are some limitations about the `block_size` and `sec_block_size` parameters:

- The address of the first element of y must have phase 0.
- The address of the first element of x must have phase 0.
- The first element of A and x must be in parts of the shared memory with affinity to the same thread.
- The first element of A must be in the first row of a block in the shared memory space.

2. `upc_pblas_colDist`: All the elements in a column must have affinity to the same thread but the number of consecutive columns in each thread can change. In this case, this function treats array pointers as if they had type:

    - `A:shared [block_size] type [m*lda]`
    - `x:shared [block_size] type [m]`
    - `y:shared [sec_block_size] type [n]`

    Besides, there are some limitations about the `block_size` and `sec_block_size` parameters:

    - The address of the first element of y must have phase 0.
    - The address of the first element of x must have phase 0.
    - The first element of A and y must be in parts of the shared memory with affinity to the same thread.
    - The address of the first element of A must have phase 0.
    - The first element of all rows must have affinity to thread 0 with phase 0. To achieve this, the following condition must hold: $lda\%(block\_size * THREADS) == 0$.

If any array overlaps, the behavior is undefined.

5.3 upc_blas_Ttrsv

Function to solve the triangular system of equations $x = T^{-1} * x$ (with transpose variants of matrix T).

<u>SYNTAX:</u>

- `int upc_blas_strsv(UPC_PBLAS_DIMMDIST dimmDist, int block_size, UPC_PBLAS_UPLO uplo, UPC_PBLAS_TRANSPOSE transpose, UPC_PBLAS_DIAG diag, int n, shared void *T, int ldt, shared void *x)`
- `int upc_blas_dtrsv(UPC_PBLAS_DIMMDIST dimmDist, int block_size, UPC_PBLAS_UPLO uplo, UPC_PBLAS_TRANSPOSE transpose, UPC_PBLAS_DIAG diag, int n, shared void *T, int ldt, shared void *x)`

- `int upc_blas_ctrsv(UPC_PBLAS_DIMMDIST dimmDist, int block_size, UPC_PBLAS_UPLO uplo, UPC_PBLAS_TRANSPOSE transpose, UPC_PBLAS_DIAG diag, int n, shared void *T, int ldt, shared void *x)`
- `int upc_blas_ztrsv(UPC_PBLAS_DIMMDIST dimmDist, int block_size, UPC_PBLAS_UPLO uplo, UPC_PBLAS_TRANSPOSE transpose, UPC_PBLAS_DIAG diag, int n, shared void *T, int ldt, shared void *x)`

PARAMETERS:

- `IN dimmDist`: Enumerated value representing the matrix dimension that will be distributed among threads. Each thread must have one or more complete rows or columns depending on this parameter. If `upc_pblas_rowDist` is selected, each thread has complete rows. Columns are distributed entirely in the `upc_pblas_colDist` case.
- `IN block_size`: Number of rows or columns (depending on `dimmDist`) of the matrix that are consecutively distributed among all threads. For example, in the `upc_pblas_rowDist` case, the first `block_size` rows will correspond to thread 0, the second `block_size` ones to thread 1, etc. Remember that blocks are built using whole rows/columns and not individual elements.
- `IN uplo`: Enumerated value to indicate if matrix `T` is upper or lower triangular.
- `IN transpose`: Enumerated value to indicate if matrix `T` is transposed.
- `IN diag`: Enumerated value to indicate if all the diagonal values of `T` are equal to 1 or not.
- `IN n`: Number of rows and columns of matrix `T`.
- `IN T`: Pointer to the position of the shared array where the source triangular matrix `T` is stored.
- `IN ldt`: It specifies the first dimension of `T` as declared in the calling program (used to work with submatrices). It must be at least `n`.
- `IN/OUT x`: Pointer to the position of the shared array where the vector `x` is stored.
- returns:
  - 0 if everything is ok.
  - < 0 if a parameter error occurs. The exact value is -j if the wrong parameter is the jth one.
  - +1 if an internal memory error occurs.

RESTRICTIONS:

The `block_size` specifications are different depending on the `dimmDist` selection:

1. `upc_pblas_rowDist`: All the elements in a row must have affinity to the same thread but the number of consecutive rows in each thread can change. In this case, this function treats array pointers as if they had type:

    - `T:shared [block_size*ldt] type [n*ldt]`
    - `T:shared [block_size] type [n*ldt]` in transpose case
    - `x:shared [block_size] type [n]`

    Besides, there are some limitations about the `block_size` parameter:

    - The address of the first element of `x` must have phase 0.
    - The first element of `T` and `x` must be in parts of the shared memory with affinity to the same thread.
    - If `T` non-transpose:
        - The first element of `T` must be in the first row of a block in the shared memory space.
    - If `T` transpose:
        - The address of the first element of `T` must have phase 0.
        - The first element of all rows must have affinity to thread 0 with phase 0. To achieve this, the following condition must hold: $ldt\%(block\_size * THREADS) == 0$.

2. `upc_pblas_colDist`: All the elements in a column must have affinity to the same thread but the number of consecutive columns in each thread can change. In this case, this function treats array pointers as if they had type:

    - `T:shared [block_size] type [n*ldt]`
    - `T:shared [block_size*ldt] type [n*ldt]` in transpose case
    - `x:shared [block_size] type [n]`

    Besides, there are some limitations about the `block_size` and `sec_block_size` parameters:

    - The address of the first element of `x` must have phase 0.
    - The first element of `T` and `x` must be in parts of the shared memory with affinity to the same thread.
    - If `T` non-transpose:
        - The address of the first element of `T` must have phase 0.
        - The first element of all rows must have affinity to thread 0 with phase 0. To achieve this, the following condition must hold: $ldt\%(block\_size * THREADS) == 0$.
    - If `T` transpose:

  - The first element of `T` must be in the first row of a block in the shared memory space.

If any array overlaps, the behavior is undefined.

<u>ADVICE:</u>

No check for singularity or near singularity of matrix `T` is included in this function. It must be implemented before the function call if necessary.

These routines perform matrix-matrix operations. Their headers are available in the file *include/pblas3.h*.

5.4 upc_blas_Tgemm

Function to perform the matrix-matrix product: $C = \alpha * A * B + \beta * C$ (with transpose variants of matrices `A` and `B`).

SYNTAX:

- `int upc_blas_sgemm(UPC_PBLAS_DIMMDIST dimmDist, int block_size, int sec_block_size, UPC_PBLAS_TRANSPOSE transposeA, UPC_PBLAS_TRANSPOSE transposeB, int m, int n, int k, float alpha, shared void *A, int lda, shared void *B, int ldb, float beta, shared void *C, int ldc)`
- `int upc_blas_dgemm(UPC_PBLAS_DIMMDIST dimmDist, int block_size, int sec_block_size, UPC_PBLAS_TRANSPOSE transposeA, UPC_PBLAS_TRANSPOSE transposeB, int m, int n, int k, double alpha, shared void *A, int lda, shared void *B, int ldb, double beta, shared void *C, int ldc)`
- `int upc_blas_cgemm(UPC_PBLAS_DIMMDIST dimmDist, int block_size, int sec_block_size, UPC_PBLAS_TRANSPOSE transposeA, UPC_PBLAS_TRANSPOSE transposeB, int m, int n, int k, void alpha, shared void *A, int lda, shared void *B, int ldb, void beta, shared void *C, int ldc)`
- `int upc_blas_zgemm(UPC_PBLAS_DIMMDIST dimmDist, int block_size, int sec_block_size, UPC_PBLAS_TRANSPOSE transposeA, UPC_PBLAS_TRANSPOSE transposeB, int m, int n, int k, void alpha, shared void *A, int lda, shared void *B, int ldb, void beta, shared void *C, int ldc)`

PARAMETERS:

- `IN dimmDist`: Enumerated value representing the dimesion of matrix `C` that will be distributed among threads. Each thread must have one or more complete rows or-columns depending on this parameter. If `upc_pblas_rowDist` is selected, each thread has complete rows. Columns are distributed entirely in `upc_pblas_colDist` case.
- `IN block_size`: Number of rows or columns (depending on `dimmDist`) of matrix `C` that are consecutively distributed among all threads. For example, in the `upc_pblas_rowDist` case, the first `block_size` rows will correspond to thread 0, the second `block_size` ones to thread 1, etc. Remember that blocks are built using whole rows/columns and not individual elements.

- IN `sec_block_size`: Depending on the selected `dimmDist`, the block size of one of the source matrices (`A` or `B`) is automatically determined (see RESTRICTIONS below) but the distribution of the other matrix can vary. This parameter is used to indicate the block size of the matrix not automatically defined.
- IN `transposeA`: Enumerated value to indicate if matrix `A` is transposed.
- IN `transposeB`: Enumerated value to indicate if matrix `B` is transposed.
- IN `m`: Number of rows of matrices `A` and `C`.
- IN `n`: Number of columns of matrices `B` and `C`.
- IN `k`: Number of columns of matrix `A` and rows of matrix `B`.
- IN `alpha`: Scale factor for matrix `A`.
- IN `A`: Pointer to the position of the shared array where the source matrix `A` is stored.
- IN `lda`: It specifies the first dimension of `A` as declared in the calling program (used to work with submatrices). It must be at least `k` in the non-transpose case and `m` in the transpose case.
- IN `B`: Pointer to the position of the shared array where the source matrix `B` is stored.
- IN `ldb`: It specifies the first dimension of `B` as declared in the calling program (used to work with submatrices). It must be at least `n` in the non-transpose case and `k` in the transpose case.
- IN `beta`: Scale factor for matrix `C`.
- IN/OUT `C`: Pointer to the position of the shared array where matrix `C` is stored.
- IN `ldc`: It specifies the first dimension of `C` as declared in the calling program (used to work with submatrices). It must be at least `n`.
- returns:
  - 0 if everything is ok.
  - < 0 if a parameter error occurs. The exact value is -j if the wrong parameter is the jth one.
  - +1 if an internal memory error occurs.

RESTRICTIONS:

The `block_size` specifications are different depending on the `dimmDist` selection:

1. `upc_pblas_rowDist`: All the elements in a row of matrix `C` must have affinity to the same thread but the number of consecutive rows in each thread can change. In this case, this function treats array pointers as if they had type:

- `A:shared [block_size*lda] type [m*lda]`
- `A:shared [block_size] type [k*lda]` in transpose case
- `B:shared [sec_block_size] type [k*ldb]`
- `B:shared [sec_block_size] type [n*ldb]` in transpose case
- `C:shared [block_size*ldc] type [m*ldc]`

Besides, there are some limitations about the `block_size` and `sec_block_size` parameters:

- `ldb` and `sec_block_size` must be multiples of each other.
- The first element of `B` must be in the first row of a block in the shared array.
- The first element of `C` must be in the first row of a block in the shared array.
- The first element of `A` and `C` must be in parts of the shared memory with affinity to the same thread.
- If `A` non-transpose:
  - The first element of `A` must be in the first row of a block in the shared memory space.
- If `A` transpose:
  - The first element of `A` must be in the first column of a block in the shared memory space.
  - The first element of all rows must have affinity to thread 0 with phase 0. To achieve this, the following condition must hold: $lda\%(block\_size * THREADS) == 0$.

2. `upc_pblas_colDist`: All the elements in a column of matrix `C` must have affinity to the same thread but the number of consecutive columns in each thread can change. In this case, this function treats array pointers as if they had type:

   - `A:shared [sec_block_size] type [m*lda]`
   - `A:shared [sec_block_size] type [k*lda]` in transpose case
   - `B:shared [block_size] type [k*ldb]`
   - `B:shared [block_size*ldb] type [n*ldb]` in transpose case
   - `C:shared [block_size] type [m*ldc]`

   Besides, there are some limitations about the `block_size` and `sec_block_size` parameters:

   - `lda` and `sec_block_size` must be multiples of each other.
   - The first element of `A` must be in the first row of a block in the shared array.
   - The first element of `C` must be in the first row of a block in the shared array.

- The first element of `B` and `C` must be in parts of the shared memory with affinity to the same thread.
- If `B` non-transpose:
  - The first element of `B` must be in the first column of a block in the shared memory space.
  - The first element of all rows must have affinity to thread 0 with phase 0. To achieve this, the following condition must hold: $ldb\%(block\_size * THREADS) == 0$.
- If `B` transpose:
  - The first element of `B` must be in the first row of a block in the shared memory space.

If any array overlaps, the behavior is undefined.

5.5 upc_blas_Ttrsm

Function to solve the triangular systems of equations $X = T^{-1} * \alpha * X$ or $X = \alpha * X * T^{-1}$ (with transpose variants of matrix `T`).

<u>SYNTAX:</u>

- `int upc_blas_strsm(UPC_PBLAS_DIMMDIST dimmDist, int block_size, int sec_block_size, UPC_PBLAS_SIDE side, UPC_PBLAS_UPLO uplo, UPC_PBLAS_TRANSPOSE transpose, UPC_PBLAS_DIAG diag, int m, int n, float alpha, shared void *T, int ldt, shared void *X, int ldx);`

- `int upc_blas_dtrsm(UPC_PBLAS_DIMMDIST dimmDist, int block_size, int sec_block_size, UPC_PBLAS_SIDE side, UPC_PBLAS_UPLO uplo, UPC_PBLAS_TRANSPOSE transpose, UPC_PBLAS_DIAG diag, int m, int n, double alpha, shared void *T, int ldt, shared void *X, int ldx);`

- `int upc_blas_ctrsm(UPC_PBLAS_DIMMDIST dimmDist, int block_size, int sec_block_size, UPC_PBLAS_SIDE side, UPC_PBLAS_UPLO uplo, UPC_PBLAS_TRANSPOSE transpose, UPC_PBLAS_DIAG diag, int m, int n, void *alpha, shared void *T, int ldt, shared void *X, int ldx);`

- `int upc_blas_ztrsm(UPC_PBLAS_DIMMDIST dimmDist, int block_size, int sec_block_size, UPC_PBLAS_SIDE side, UPC_PBLAS_UPLO uplo, UPC_PBLAS_TRANSPOSE transpose, UPC_PBLAS_DIAG diag, int m, int n, void *alpha, shared void *T, int ldt, shared void *X, int ldx);`

<u>PARAMETERS:</u>

- `IN dimmDist`: Enumerated value representing the dimension of matrix `X` that will be distributed among threads. Each thread must have one or more complete rows or columns depending on this parameter. If `upc_pblas_rowDist` is selected, each thread has complete rows. Columns are distributed entirely in the `upc_pblas_colDist` case.
- `IN block_size`: Number of rows or columns (depending on `dimmDist`) of matrix `X` that are consecutively distributed among all threads. For example, in the `upc_pblas_rowDist` case, the first `block_size` rows will correspond to thread 0, the second `block_size` ones to thread 1, etc. Remember that blocks are built using whole rows/columns and not individual elements.
- `IN sec_block_size`: Number of elements consecutively distributed among all threads in matrix `T` in the options: `upc_pblas_colDist & upc_pblas_left` or `upc_pblas_rowDist & upc_pblas_right`.
- `IN side`: Enumerated value to indicate if matrix `T` is on the left or right side of the equation.
- `IN uplo`: Enumerated value to indicate if matrix `T` is upper or lower triangular.
- `IN transpose`: Enumerated value to indicate if matrix `T` is transposed.
- `IN diag`: Enumerated value to indicate if all the diagonal values of `T` are equal to 1 or not.
- `IN m`: Number of rows of matrix `X`. The dimension of matrix `T` if `upc_pblas_left`.
- `IN n`: Number of columns of matrix `X`. The dimension of matrix `T` if `upc_pblas_right`.
- `IN alpha`: Scale factor for matrix `T`.
- `IN T`: Pointer to the position of the shared array where the source triangular matrix `T` is stored.
- `IN ldt`: It specifies the first dimension of `T` as declared in the calling program (used to work with submatrices). It must be at least `m` in the `upc_pblas_left` case and `n` in the `upc_pblas_right` case.
- `IN/OUT X`: Pointer to the position of the shared array where matrix `X` is stored.
- `IN ldx`: It specifies the first dimension of `X` as declared in the calling program (used to work with submatrices). It must be at least `n`.
- returns:
  - 0 if everything is ok.
  - < 0 if a parameter error occurs. The exact value is `-j` if the wrong parameter is the jth one.
  - +1 if an internal memory error occurs.

RESTRICTIONS:

The `block_size` specifications are different depending on the `dimmDist` and `side` selection:

1. `upc_pblas_rowDist & upc_pblas_left`: All the elements in a row of matrix `X` must have affinity to the same thread but the number of consecutive rows in each thread can change. In this case, this function treats array pointers as if they had type:

    - `T:shared [block_size*ldt] type [m*ldt]`
    - `T:shared [block_size] type [m*ldt]` in transpose case
    - `X:shared [block_size*ldx] type [m*ldx]`

    Besides, there are some limitations about the `block_size` parameter:

    - The first element of `X` must be in the first row of a block in the shared memory space.
    - The first element of `T` and `X` must be in parts of the shared memory with affinity to the same thread.
    - If `T` non-transpose:
        - The first element of `T` must be in the first row of a block in the shared memory space.
    - If `T` transpose:
        - The first element of `T` must be in the first column of a block in the shared memory space.

2. `upc_pblas_colDist & upc_pblas_left`: All the elements in a column of matrix `X` must have affinity to the same thread but number of consecutive columns in each thread can change. In this case, this function treats array pointers as if they had type:

    - `T: shared [sec_block_size] type [m*ldt]`
    - `X: shared [block_size] type [m*ldx]`

    Besides, there are some limitations about the `block_size` and `sec_block_size` parameters:

    - `ldt` and `sec_block_size` must be multiples of each other.
    - The first element of `T` must be in the first row of a block in the shared memory space.
    - The first element of `X` must be in the first column of a block in the shared memory space.

3. `upc_pblas_rowDist & upc_pblas_right`: All the elements in a row of matrix `X` must have affinity to the same thread but the number of consecutive rows in each thread can change. In this case, this function treats array pointers as if they had type:

- `T:shared [sec_block_size] type [n*ldt]`
- `X:shared [block_size*ldx] type [m*ldx]`

Besides, there are some limitations about the `block_size` and `sec_block_size` parameters:

- `ldt` and `sec_block_size` must be multiples of each other.
- The first element of `T` must be in the first row of a block in the shared memory space.
- The first element of `X` must be in the first row of a block in the shared memory space.

4. `upc_pblas_colDist & upc_pblas_right`: All the elements in a column of matrix `X` must have affinity to the same thread but number of consecutive columns in each thread can change. In this case, this function treats array pointers as if they had type:

   - `T:shared [block_size] type [n*ldt]`
   - `T:shared [block_size*ldt] type [n*ldt]` in transpose case
   - `X:shared [block_size] type [m*ldx]`

   Besides, there are some limitations about `block_size` parameter:

   - The first element of `X` must be in the first column of a block in the shared array.
   - The first element of `T` and `X` must be in parts of the shared memory with affinity to the same thread.
   - If `T` non-transpose:
     - The first element of `T` must be in the first column of a block in the shared memory space.
   - If `T` transpose:
     - The first element of `T` must be in the first row of a block in the shared memory space.

If any array overlaps, the behavior is undefined.

No check for singularity or near singularity of matrix `T` is included in this function. It must be implemented before the function call if necessary.