Batched BLAS

Generated by Doxygen 1.8.14

Contents

Chapter 1

Main Page

Batched Basic Linear Algebra Subroutines

University of Manchester (UK)

University of Tennessee (US)

Download BBLAS Software

About

BBLAS is the reference implementation of the Batched BLAS standard specification. A current trend in high-performance computing is to decompose a large linear algebra problem into batches containing thousands of smaller problems, which can be solved independently, before collating the results. To standardize the interface to these routines, the community developed an extension to the BLAS standard (the batched BLAS), enabling users to perform thousands of small BLAS operations in parallel whilst making efficient use of their hardware. Please visit BBLAS workshops for more information on the standardization efforts.

The main folders & files

- **compute**: contains the standard BBLAS group API functions
- **core**: contains the auxiliary batched BLAS functions which perform on groups of same size problems
- **control**: contains auxiliary functions for type conversions
- **test**: contains testing routines associated with the BBLAS functions and provides an insight on how the BBLAS functions should be called/used.
- **include**: contains header files
- *make.inc*: a configuration file to specify a C/C++ compiler, compilation flags and a BLAS library. The default configuration should work when MKL is installed.
- *Makefile*: the Makefile, normally, it should not be modified.

Requirements

BLAS & LAPACK

- **MKL** is now free for academics (students and researchers) available at https://software.intel.com/en-us/articles/free-mkl

OR

- **Netlib BLAS** no optimized BLAS routines, available at BLAS-3.8.0.tgz
- **Netlib LAPACK** no optimized LAPACK routines, available at LAPACK-3.8.0.tgz

Doxygen for documentation

- **Doxygen** can be install on Unix systems by *sudo apt-get install doxygen* or downloaded on the doxygen page.

Compilation

After the configuration of **make.inc**, the compilation is very simple:

- **make [all]** – make lib test
- **make lib** – make lib/libbblas.{a,so} lib/libcore.{a,so}
- **make test** – make test/test
- **make docs** – make docs/html
- **make generate** – generate precisions
- **make clean** – remove objects, libraries, and executables
- **make cleangen** – remove generated precision files
- **make distclean** – remove above, Makefile.∗.gen, and anything else that can be generated

Citing

Feel free to use the following publications to reference BBLAS:

- Jack Dongarra, Sven Hammarling, Nicholas J. Higham, Samuel D. Relton, Mawussi Zounon: **Optimized Batched Linear Algebra for Modern Architectures.** *Euro-Par 2017: 511-522*.
- Jack Dongarra, Sven Hammarling, Nicholas J. Higham, Samuel D. Relton, Pedro Valero-Lara, Mawussi Zounon: **The Design and Performance of Batched BLAS on Modern High-Performance Computing Systems**, *ICCS 2017: 495-504*
- Jack Dongarra, Iain Duff, Mark Gates, Azzam Haidar, Sven Hammarling, Nicholas J. Higham, Jonathan Hogg, Pedro Valero Lara, Mawussi Zounon, Samuel D. Relton, and Stanimire Tomov, **A Proposed API for Batched Basic Linear Algebra Subprograms**, Draft Report, May 2016.

Funding

Primary funding for BBLAS was provided the European Union grant:

- NLAFET: Parallel Numerical Linear Algebra for Future Extreme Scale Systems, Grant Agreement no. 671633

People

The following people listed in alphabetical order contributed to the BBLAS reference implementation:

- Jack Dongarra
- Mark Gates
- Srikara Pranesh
- Samuel Relton
- Pedro Valero Lara
- Mawussi Zounon

Chapter 2

Module Index

2.1 Routines

Here is a list of all modules:

Chapter 3

Module Documentation

3.1 Batched BLAS

Batched BLAS group API functions. Standard Batched BLAS routines.

Modules

- : Standard Batched matrix-matrix operations,

  Batched matrix-matrix operations that perform on many groups of different size matrices.

3.1.1 Detailed Description

Batched BLAS group API functions. Standard Batched BLAS routines.

3.2 : Standard Batched matrix-matrix operations,

Batched matrix-matrix operations that perform on many groups of different size matrices.

Modules

- gemm_batch: Batched general matrix multiply: C[i] = A[i]B[i] + C[i]
  $C[i] = \alpha[i]\ op(A[i])\ op(B[i]) + \beta[i]C[i]$
- hemm_batch: Batched hermitian matrix multiply
  $C[i] = \alpha[i]A[i]B[i] + \beta[i]C[i]$ *or* $C[i] = \alpha[i]B[i]A[i] + \beta C[i]$ *where* $A[i]$ *are hermitian*
- herk_batch: Batched hermitian rank k update
  $C[i] = \alpha[i]A[i]A[i]^T + \beta[i]C[i]$ *where* $C[i]$ *are hermitian*
- her2k_batch: Batched hermitian rank 2k update
  $C[i] = \alpha[i]A[i]B[i]^T + \alpha[i]B[i]A[i]^T + \beta[i]C[i]$ *where* $C[i]$ *are Hermitian*
- symm_batch: Batched symmetric matrix multiply
  $C[i] = \alpha[i]A[i]B[i] + \beta[i]C[i]$ *or* $C[i] = \alpha[i]B[i]A[i] + \beta[i]C[i]$ *where* $A[i]$ *are symmetric*
- syrk_batch: Batched symmetric rank k update
  $C[i] = \alpha[i]A[i]A[i]^T + \beta[i]C[i]$ *where* $C[i]$ *are symmetric*
- syr2k_batch: Batched symmetric rank 2k update
  $C[i] = \alpha[i]A[i]B[i]^T + \alpha[i]B[i]A[i]^T + \beta[i]C[i]$ *where* $C[i]$ *are symmetric*
- trmm_batch: Batched triangular matrix multiply
  $B[i] = \alpha[i]\ op(A[i])\ B[i]$ *or* $B[i] = \alpha[i]B[i]\ op(A[i])$ *where* $A[i]$ *are triangular*
- trsm_batch: Batched triangular solve matrix
  $C[i] = op(A[i])^{-1}B[i]$ *or* $C[i] = B[i]\ op(A[i])^{-1}$ *where* $A[i]$ *are triangular*

3.2.1 Detailed Description

Batched matrix-matrix operations that perform on many groups of different size matrices.

3.3 gemm_batch: Batched general matrix multiply: C[i] = A[i]B[i] + C[i]

$C[i] = \alpha[i]\ op(A[i])\ op(B[i]) + \beta[i] C[i]$

$C[i] = \alpha[i]\ op(A[i])\ op(B[i]) + \beta[i] C[i]$

3.4 hemm_batch: Batched hermitian matrix multiply

$C[i] = \alpha[i]A[i]B[i] + \beta[i]C[i]$ or $C[i] = \alpha[i]B[i]A[i] + \beta C[i]$ where $A[i]$ are hermitian

Functions

- void blas_chemm_batch (int group_count, const int *group_sizes, bblas_enum_t layout, const bblas_enum_t *side, const bblas_enum_t *uplo, const int *m, const int *n, const bblas_complex32_t *alpha, bblas_complex32_t const *const *A, const int *lda, bblas_complex32_t const *const *B, const int *ldb, const bblas_complex32_t *beta, bblas_complex32_t **C, const int *ldc, int *info)
- void blas_zhemm_batch (int group_count, const int *group_sizes, bblas_enum_t layout, const bblas_enum_t *side, const bblas_enum_t *uplo, const int *m, const int *n, const bblas_complex64_t *alpha, bblas_complex64_t const *const *A, const int *lda, bblas_complex64_t const *const *B, const int *ldb, const bblas_complex64_t *beta, bblas_complex64_t **C, const int *ldc, int *info)

3.4.1 Detailed Description

$C[i] = \alpha[i]A[i]B[i] + \beta[i]C[i]$ or $C[i] = \alpha[i]B[i]A[i] + \beta C[i]$ where $A[i]$ are hermitian

3.4.2 Function Documentation

3.4.2.1 blas_chemm_batch()

```
void blas_chemm_batch (
            int group_count,
            const int * group_sizes,
            bblas_enum_t layout,
            const bblas_enum_t * side,
            const bblas_enum_t * uplo,
            const int * m,
            const int * n,
            const bblas_complex32_t * alpha,
            bblas_complex32_t const *const * A,
            const int * lda,
            bblas_complex32_t const *const * B,
            const int * ldb,
            const bblas_complex32_t * beta,
            bblas_complex32_t ** C,
            const int * ldc,
            int * info )
```

Performs one of the batch matrix-matrix operations on each group of matrices

$$C[i] = \alpha[i] \times A[i] \times B[i] + \beta[i] \times C[i]$$

or

$$C[i] = \alpha[i] \times B[i] \times A[i] + \beta[i] \times C[i]$$

where alpha[i] and beta[i] are scalars, A[i]-s are Hermitian matrices B[i]-s and C[i]-s are m-by-n matrices.

Parameters

in	*group_count*	The number groups of matrices.
in	*group_sizes*	An array of integers of length group_count, where group_sizes[i] denotes the number of matrices in i-th group.
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*side*	An array of length group_count. side[i] Specifies whether the Hermitian matrices A[j]-s of i-th group appear on the left or right in the operation as follows: • BblasLeft: $C[j] = \alpha[i] \times A[j] \times B[j] + \beta[i] \times C[j]$ • BblasRight: $C[j] = \alpha[i] \times B[j] \times A[j] + \beta[i] \times C[j]$
in	*uplo*	An array of length group_count, where uplo[i] specifies whether the upper or lower triangular part of the Hermitian matrices A[j]-s of i-th group are to be referenced

- BblasLower: Only the lower triangular part of the Hermitian matrices A[j] is to be referenced.
- BblasUpper: Only the upper triangular part of the Hermitian matrices A[j] is to be referenced.

Parameters

in	*m*	An of array of integers of length group_count, where m[i] denotes the number of rows in the matrices C[j] of i-th group. m[i] >= 0.
in	*n*	An of array of integers of length group_count, where n[i] denotes the number of columns in the matrices C[j] of i-th group. n[i] >= 0.
in	*alpha*	An array of scalars of length group-count.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[batch_count-1], where each element A[j] of i-th group is a pointer to a matrix A[j] of size lda[i]-by-ka, where ka is m[i] when side[i] = BblasLeft, and is n[i] otherwise. Only the uplo triangular part is referenced. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*lda*	An array of length group_count, where lda[i] is the leading dimension of the arrays A[j] of i-th group. lda[i] >= max(1,ka).
in	*B*	B is an array of pointers to matrices B[0], B[1] .. B[batch_count-1], where each element B[j] of i-th group is a pointer to a matrix B[j] of size ldb[i]-by-n[i] matrix, where the leading m[i]-by-n[i] part of the array B[j] must contain the matrix B[j]. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldb*	An array of length group_count, where ldb[i] is the leading dimension of the arrays B[j] of i-th group. ldb[i] >= max(1,m[i]).
in	*beta*	An array of scalars of length group_count.
in,out	*C*	C is an array of pointers to matrices C[0], C[1],...,C[batch_count-1]. In i-th group each element C[j] is a pointer to a matrix C[j]. On exit, each array C[j] of i-th group is overwritten by the m[i]-by-n[i] updated matrix. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldc*	An array of integers of size group_count, which denotes the leading dimension of the arrays C[j] in i-th group. ldc[i] >= max(1,m[i]).

Parameters

in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast ({i=0}^{group_count-1}group_sizes[i])+1 • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast to (group_count+1). • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

chemm_batch
chemm_batch

3.4.2.2 blas_zhemm_batch()

```
void blas_zhemm_batch (
             int group_count,
             const int * group_sizes,
             bblas_enum_t layout,
             const bblas_enum_t * side,
             const bblas_enum_t * uplo,
             const int * m,
             const int * n,
             const bblas_complex64_t * alpha,
             bblas_complex64_t const *const * A,
             const int * lda,
             bblas_complex64_t const *const * B,
             const int * ldb,
             const bblas_complex64_t * beta,
             bblas_complex64_t ** C,
             const int * ldc,
             int * info )
```

Performs one of the batch matrix-matrix operations on each group of matrices

$$C[i] = \alpha[i] \times A[i] \times B[i] + \beta[i] \times C[i]$$

or

$$C[i] = \alpha[i] \times B[i] \times A[i] + \beta[i] \times C[i]$$

where alpha[i] and beta[i] are scalars, A[i]-s are Hermitian matrices B[i]-s and C[i]-s are m-by-n matrices.

Parameters

in	*group_count*	The number groups of matrices.
in	*group_sizes*	An array of integers of length group_count, where group_sizes[i] denotes the number of matrices in i-th group.
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*side*	An array of length group_count. side[i] Specifies whether the Hermitian matrices A[j]-s of i-th group appear on the left or right in the operation as follows: • BblasLeft: $C[j] = \alpha[i] \times A[j] \times B[j] + \beta[i] \times C[j]$ • BblasRight: $C[j] = \alpha[i] \times B[j] \times A[j] + \beta[i] \times C[j]$
in	*uplo*	An array of length group_count, where uplo[i] specifies whether the upper or lower triangular part of the Hermitian matrices A[j]-s of i-th group are to be referenced

- BblasLower: Only the lower triangular part of the Hermitian matrices A[j] is to be referenced.
- BblasUpper: Only the upper triangular part of the Hermitian matrices A[j] is to be referenced.

Parameters

in	*m*	An of array of integers of length group_count, where m[i] denotes the number of rows in the matrices C[j] of i-th group. m[i] >= 0.
in	*n*	An of array of integers of length group_count, where n[i] denotes the number of columns in the matrices C[j] of i-th group. n[i] >= 0.
in	*alpha*	An array of scalars of length group-count.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[batch_count-1], where each element A[j] of i-th group is a pointer to a matrix A[j] of size lda[i]-by-ka, where ka is m[i] when side[i] = BblasLeft, and is n[i] otherwise. Only the uplo triangular part is referenced. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*lda*	An array of length group_count, where lda[i] is the leading dimension of the arrays A[j] of i-th group. lda[i] >= max(1,ka).
in	*B*	B is an array of pointers to matrices B[0], B[1] .. B[batch_count-1], where each element B[j] of i-th group is a pointer to a matrix B[j] of size ldb[i]-by-n[i] matrix, where the leading m[i]-by-n[i] part of the array B[j] must contain the matrix B[j]. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldb*	An array of length group_count, where ldb[i] is the leading dimension of the arrays B[j] of i-th group. ldb[i] >= max(1,m[i]).
in	*beta*	An array of scalars of length group_count.
in,out	*C*	C is an array of pointers to matrices C[0], C[1],...,C[batch_count-1]. In i-th group each element C[j] is a pointer to a matrix C[j]. On exit, each array C[j] of i-th group is overwritten by the m[i]-by-n[i] updated matrix. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldc*	An array of integers of size group_count, which denotes the leading dimension of the arrays C[j] in i-th group. ldc[i] >= max(1,m[i]).

Parameters

in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast ({i=0}^{group_count-1}group_sizes[i])+1 • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast to (group_count+1). • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

zhemm_batch
chemm_batch

3.5 herk_batch: Batched hermitian rank k update

$C[i] = \alpha[i]A[i]A[i]^T + \beta[i]C[i]$ where $C[i]$ are hermitian

Functions

- void blas_cherk_batch (int group_count, const int *group_sizes, bblas_enum_t layout, const bblas_enum_t *uplo, const bblas_enum_t *trans, const int *n, const int *k, const float *alpha, bblas_complex32_t const *const *A, const int *lda, const float *beta, bblas_complex32_t **C, const int *ldc, int *info)
- void blas_zherk_batch (int group_count, const int *group_sizes, bblas_enum_t layout, const bblas_enum_t *uplo, const bblas_enum_t *trans, const int *n, const int *k, const double *alpha, bblas_complex64_t const *const *A, const int *lda, const double *beta, bblas_complex64_t **C, const int *ldc, int *info)

3.5.1 Detailed Description

$C[i] = \alpha[i]A[i]A[i]^T + \beta[i]C[i]$ where $C[i]$ are hermitian

3.5.2 Function Documentation

3.5.2.1 blas_cherk_batch()

```
void blas_cherk_batch (
            int group_count,
            const int * group_sizes,
            bblas_enum_t layout,
            const bblas_enum_t * uplo,
            const bblas_enum_t * trans,
            const int * n,
            const int * k,
            const float * alpha,
            bblas_complex32_t const *const * A,
            const int * lda,
            const float * beta,
            bblas_complex32_t ** C,
            const int * ldc,
            int * info )
```

Performs one of the batch Hermitian rank k operations on a group of matrices, where matrices in each group have constant properties

$$C[i] = \alpha A[i] \times A[i]^H + \beta C[i],$$

or

$$C[i] = \alpha A[i]^H \times A[i] + \beta C[i],$$

where alpha and beta are real scalars, C[i]-s are n-by-n Hermitian matrices, and A[i]-s are n-by-k matrices in the first case and k-by-n matrices in the second case.

Parameters

in	*group_count*	The number groups of matrices with fixed size.
in	*group_sizes*	An array of integers of length group_count, where group_sizes[i] denotes the number of matrices in i-th group.
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*uplo*	An array of length group_count, where uplo[i] specifies whether the upper or lower triangular part of the Hermitian matrices C[j]-s of i-th group are to be stored

- BblasLower: Only the lower triangular part of the Hermitian matrices C[j] are to be stored.
- BblasUpper: Only the upper triangular part of the Hermitian matrices C[j] are to be stored.

Parameters

in	*trans*	An array of length group_count, where for j-th matrix in i-th group • BblasNoTrans: $$C[j] = \alpha[i]A[j] \times B[j]^H + conjg(\alpha[i])B[j] \times A[j]^H + \beta[i]C[j];$$ • BblasConjTrans: $$C[j] = \alpha[i]A[j]^H \times B[j] + conjg(\alpha[i])B[j]^H \times A[j] + \beta[i]C[j].$$
in	*n*	An array of integers of length group count-1, where n[i] is the order of the matrices C[j] in i-th group. n[i] >= 0.
in	*k*	An array of integers of length group_count. For matrices in i-th group If trans = BblasNoTrans, number of columns of A[j]-s and B[j]-s matrices; if trans = BblasConjTrans, number of rows of A[j]-s and B[j]-s matrices.
in	*alpha*	An array of scalars of length group_count.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[batch_count-1]. In i-th group each element A[j] is a pointer to a matrix of A[j] of size lda[i]-by-ka. If trans[i] = BblasNoTrans, ka = k[i]; if trans[i] = BblasConjTrans, ka = n[i]. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*lda*	An array of integers of length group_count, are the leading dimension of the arrays A[j] in i-th group. If trans[i] = BblasNoTrans, lda[i] >= max(1, n[i]); if trans[i] = BblasConjTrans, lda[i] >= max(1, k[i]).
in	*beta*	An array of scalars of length group_count.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[batc_count-1]. In i-th group each element C[j] is a pointer to a matrix C[j] of size ldc[i]-by-n[i]. On exit, the uplo[i] part of the matrix is overwritten by the uplo[i] part of the updated matrix. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldc*	An array of integers of length group_count. Where ldc[i] is the leading dimension of the arrays C[j] in i-th group. ldc[i] >= max(1, n[i]).

Parameters

in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=1}^{group_count-1}group_sizes[i] +1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast to (group_count+1). • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

cherk_batch
cherk_batch

3.5.2.2 blas_zherk_batch()

```
void blas_zherk_batch (
            int group_count,
            const int * group_sizes,
            bblas_enum_t layout,
            const bblas_enum_t * uplo,
            const bblas_enum_t * trans,
            const int * n,
            const int * k,
            const double * alpha,
            bblas_complex64_t const *const * A,
            const int * lda,
            const double * beta,
            bblas_complex64_t ** C,
            const int * ldc,
            int * info )
```

Performs one of the batch Hermitian rank k operations on a group of matrices, where matrices in each group have constant properties

$$C[i] = \alpha A[i] \times A[i]^H + \beta C[i],$$

or

$$C[i] = \alpha A[i]^H \times A[i] + \beta C[i],$$

where alpha and beta are real scalars, C[i]-s are n-by-n Hermitian matrices, and A[i]-s are n-by-k matrices in the first case and k-by-n matrices in the second case.

Parameters

in	*group_count*	The number groups of matrices with fixed size.
in	*group_sizes*	An array of integers of length group_count, where group_sizes[i] denotes the number of matrices in i-th group.
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*uplo*	An array of length group_count, where uplo[i] specifies whether the upper or lower triangular part of the Hermitian matrices C[j]-s of i-th group are to be stored

- BblasLower: Only the lower triangular part of the Hermitian matrices C[j] are to be stored.
- BblasUpper: Only the upper triangular part of the Hermitian matrices C[j] are to be stored.

Parameters

in	*trans*	An array of length group_count, where for j-th matrix in i-th group • BblasNoTrans: $$C[j] = \alpha[i]A[j] \times B[j]^H + conjg(\alpha[i])B[j] \times A[j]^H + \beta[i]C[j];$$ • BblasConjTrans: $$C[j] = \alpha[i]A[j]^H \times B[j] + conjg(\alpha[i])B[j]^H \times A[j] + \beta[i]C[j].$$
in	*n*	An array of integers of length group count-1, where n[i] is the order of the matrices C[j] in i-th group. n[i] >= 0.
in	*k*	An array of integers of length group_count. For matrices in i-th group If trans = BblasNoTrans, number of columns of A[j]-s and B[j]-s matrices; if trans = BblasConjTrans, number of rows of A[j]-s and B[j]-s matrices.
in	*alpha*	An array of scalars of length group_count.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[batch_count-1]. In i-th group each element A[j] is a pointer to a matrix of A[j] of size lda[i]-by-ka. If trans[i] = BblasNoTrans, ka = k[i]; if trans[i] = BblasConjTrans, ka = n[i]. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*lda*	An array of integers of length group_count, are the leading dimension of the arrays A[j] in i-th group. If trans[i] = BblasNoTrans, lda[i] >= max(1, n[i]); if trans[i] = BblasConjTrans, lda[i] >= max(1, k[i]).
in	*beta*	An array of scalars of length group_count.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[batc_count-1]. In i-th group each element C[j] is a pointer to a matrix C[j] of size ldc[i]-by-n[i]. On exit, the uplo[i] part of the matrix is overwritten by the uplo[i] part of the updated matrix. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldc*	An array of integers of length group_count. Where ldc[i] is the leading dimension of the arrays C[j] in i-th group. ldc[i] >= max(1, n[i]).

Parameters

in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=1}^{group_count-1}group_sizes[i] +1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast to (group_count+1). • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

zherk_batch
cherk_batch

3.6 her2k_batch: Batched hermitian rank 2k update

$C[i] = \alpha[i]A[i]B[i]^T + \alpha[i]B[i]A[i]^T + \beta[i]C[i]$ where $C[i]$ are Hermitian

Functions

- void blas_cher2k_batch (int group_count, const int *group_sizes, bblas_enum_t layout, const bblas_enum_t *uplo, const bblas_enum_t *trans, const int *n, const int *k, const bblas_complex32_t *alpha, bblas_complex32_t const *const *A, const int *lda, bblas_complex32_t const *const *B, const int *ldb, const float *beta, bblas_complex32_t **C, const int *ldc, int *info)
- void blas_zher2k_batch (int group_count, const int *group_sizes, bblas_enum_t layout, const bblas_enum_t *uplo, const bblas_enum_t *trans, const int *n, const int *k, const bblas_complex64_t *alpha, bblas_complex64_t const *const *A, const int *lda, bblas_complex64_t const *const *B, const int *ldb, const double *beta, bblas_complex64_t **C, const int *ldc, int *info)

3.6.1 Detailed Description

$C[i] = \alpha[i]A[i]B[i]^T + \alpha[i]B[i]A[i]^T + \beta[i]C[i]$ where $C[i]$ are Hermitian

3.6.2 Function Documentation

3.6.2.1 blas_cher2k_batch()

```
void blas_cher2k_batch (
            int group_count,
            const int * group_sizes,
            bblas_enum_t layout,
            const bblas_enum_t * uplo,
            const bblas_enum_t * trans,
            const int * n,
            const int * k,
            const bblas_complex32_t * alpha,
            bblas_complex32_t const *const * A,
            const int * lda,
            bblas_complex32_t const *const * B,
            const int * ldb,
            const float * beta,
            bblas_complex32_t ** C,
            const int * ldc,
            int * info )
```

Performs one of the batch Hermitian rank 2k operations

$$C[i] = \alpha[i]A[i] \times B[i]^H + conjg(\alpha[i])B \times A[i]^H + \beta[i]C[i],$$

or

$$C[i] = \alpha[i]A[i]^H \times B[i] + conjg(\alpha[i])B[i]^H \times A[i] + \beta[i]C[i],$$

for a group of matrices, where alpha[i] is a complex scalar, beta[i] is a real scalar, C[i]-s are n-by-n Hermitian matrices, and A[i]-s and B[i]-s are n-by-k matrices in the first case and k-by-n matrices in the second case.

Parameters

in	*group_count*	The number groups of matrices.
in	*group_sizes*	An array of integers of length group_count, where group_sizes[i] denotes the number of matrices in i-th group.
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*uplo*	An array of length group_count, where uplo[i] specifies whether the upper or lower triangular part of the Hermitian matrices C[j]-s of i-th group are to be stored

- BblasLower: Only the lower triangular part of the Hermitian matrices C[j] are to be stored.
- BblasUpper: Only the upper triangular part of the Hermitian matrices C[j] are to be stored.

Parameters

in	*trans*	An array of length group_count, where for j-th matrix in i-th group • BblasNoTrans: $$C[j] = \alpha[i]A[j] \times B[j]^H + conjg(\alpha[i])B[j] \times A[j]^H + \beta[i]C[j];$$ • BblasConjTrans: $$C[j] = \alpha[i]A[j]^H \times B[j] + conjg(\alpha[i])B[j]^H \times A[j] + \beta[i]C[j].$$
in	*n*	An array of integers of length group count-1, where n[i] is the order of the matrices C[j] in i-th group. n[i] >= 0.
in	*k*	An array of integers of length group_count. For matrices in i-th group If trans = BblasNoTrans, number of columns of A[j]-s and B[j]-s matrices; if trans = BblasConjTrans, number of rows of A[j]-s and B[j]-s matrices.
in	*alpha*	An array of scalars of length group_count.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[batch_count-1]. In i-th group each element A[j] is a pointer to a matrix of A[j] of size lda[i]-by-ka. If trans[i] = BblasNoTrans, ka = k[i]; if trans[i] = BblasConjTrans, ka = n[i]. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*lda*	An array of integers of length group_count, are the leading dimension of the arrays A[j] in i-th group. If trans[i] = BblasNoTrans, lda[i] >= max(1, n[i]); if trans[i] = BblasConjTrans, lda[i] >= max(1, k[i]).
in	*B*	B is an array of pointers to matrices B[0], B[1] .. B[batch_count-1]. In i-th group each element B[j] is a pointer to a matrix B[j] of size ldb[i]-by-kb. If trans[i] = BblasNoTrans, kb = k[i]; if trans[i] = BblasConjTrans, kb = n[i]. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldb*	An array of integers of length group_count, are the leading dimension of the arrays B[j] in i-th group. If trans[i] = BblasNoTrans, ldb[i] >= max(1, n[i]); if trans[i] = BblasConjTrans, ldb[i] >= max(1, k[i]).
in	*beta*	An array of scalars of length group_count.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[batc_count-1]. In i-th group each element C[j] is a pointer to a matrix C[j] of size ldc[i]-by-n[i]. On exit, the uplo[i] part of the matrix is overwritten by the uplo[i] part of the updated matrix. batch_count = {i=0}^{group_count-1}group_sizes[i].

Parameters

in	*ldc*	An array of integers of length group_count. Where ldc[i] is the leading dimension of the arrays C[j] in i-th group. ldc[i] >= max(1, n[i]).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=1}^{group_count-1}group_sizes[i] +1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast (group_count+1). • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

cher2k_batch
cher2k_batch

3.6.2.2 blas_zher2k_batch()

```
void blas_zher2k_batch (
            int group_count,
            const int * group_sizes,
            bblas_enum_t layout,
            const bblas_enum_t * uplo,
            const bblas_enum_t * trans,
            const int * n,
            const int * k,
            const bblas_complex64_t * alpha,
            bblas_complex64_t const *const * A,
            const int * lda,
            bblas_complex64_t const *const * B,
            const int * ldb,
            const double * beta,
            bblas_complex64_t ** C,
            const int * ldc,
            int * info )
```

Performs one of the batch Hermitian rank 2k operations

$$C[i] = \alpha[i]A[i] \times B[i]^H + conjg(\alpha[i])B \times A[i]^H + \beta[i]C[i],$$

or

$$C[i] = \alpha[i]A[i]^H \times B[i] + conjg(\alpha[i])B[i]^H \times A[i] + \beta[i]C[i],$$

for a group of matrices, where alpha[i] is a complex scalar, beta[i] is a real scalar, C[i]-s are n-by-n Hermitian matrices, and A[i]-s and B[i]-s are n-by-k matrices in the first case and k-by-n matrices in the second case.

Parameters

in	*group_count*	The number groups of matrices.
in	*group_sizes*	An array of integers of length group_count, where group_sizes[i] denotes the number of matrices in i-th group.
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*uplo*	An array of length group_count, where uplo[i] specifies whether the upper or lower triangular part of the Hermitian matrices C[j]-s of i-th group are to be stored

- BblasLower: Only the lower triangular part of the Hermitian matrices C[j] are to be stored.
- BblasUpper: Only the upper triangular part of the Hermitian matrices C[j] are to be stored.

Parameters

in	*trans*	An array of length group_count, where for j-th matrix in i-th group • BblasNoTrans: $$C[j] = \alpha[i]A[j] \times B[j]^H + conjg(\alpha[i])B[j] \times A[j]^H + \beta[i]C[j];$$ • BblasConjTrans: $$C[j] = \alpha[i]A[j]^H \times B[j] + conjg(\alpha[i])B[j]^H \times A[j] + \beta[i]C[j].$$
in	*n*	An array of integers of length group count-1, where n[i] is the order of the matrices C[j] in i-th group. n[i] >= 0.
in	*k*	An array of integers of length group_count. For matrices in i-th group If trans = BblasNoTrans, number of columns of A[j]-s and B[j]-s matrices; if trans = BblasConjTrans, number of rows of A[j]-s and B[j]-s matrices.
in	*alpha*	An array of scalars of length group_count.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[batch_count-1]. In i-th group each element A[j] is a pointer to a matrix of A[j] of size lda[i]-by-ka. If trans[i] = BblasNoTrans, ka = k[i]; if trans[i] = BblasConjTrans, ka = n[i]. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*lda*	An array of integers of length group_count, are the leading dimension of the arrays A[j] in i-th group. If trans[i] = BblasNoTrans, lda[i] >= max(1, n[i]); if trans[i] = BblasConjTrans, lda[i] >= max(1, k[i]).
in	*B*	B is an array of pointers to matrices B[0], B[1] .. B[batch_count-1]. In i-th group each element B[j] is a pointer to a matrix B[j] of size ldb[i]-by-kb. If trans[i] = BblasNoTrans, kb = k[i]; if trans[i] = BblasConjTrans, kb = n[i]. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldb*	An array of integers of length group_count, are the leading dimension of the arrays B[j] in i-th group. If trans[i] = BblasNoTrans, ldb[i] >= max(1, n[i]); if trans[i] = BblasConjTrans, ldb[i] >= max(1, k[i]).
in	*beta*	An array of scalars of length group_count.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[batc_count-1]. In i-th group each element C[j] is a pointer to a matrix C[j] of size ldc[i]-by-n[i]. On exit, the uplo[i] part of the matrix is overwritten by the uplo[i] part of the updated matrix. batch_count = {i=0}^{group_count-1}group_sizes[i].

Parameters

in	*ldc*	An array of integers of length group_count. Where ldc[i] is the leading dimension of the arrays C[j] in i-th group. ldc[i] >= max(1, n[i]).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=1}^{group_count-1}group_sizes[i] +1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast (group_count+1). • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

zher2k_batch
cher2k_batch

3.7 symm_batch: Batched symmetric matrix multiply

$C[i] = \alpha[i]A[i]B[i] + \beta[i]C[i]$ or $C[i] = \alpha[i]B[i]A[i] + \beta[i]C[i]$ where $A[i]$ are symmetric

Functions

- void blas_csymm_batch (int group_count, const int *group_sizes, bblas_enum_t layout, const bblas_enum_t *side, const bblas_enum_t *uplo, const int *m, const int *n, const bblas_complex32_t *alpha, bblas_complex32_t const *const *A, const int *lda, bblas_complex32_t const *const *B, const int *ldb, const bblas_complex32_t *beta, bblas_complex32_t **C, const int *ldc, int *info)
- void blas_dsymm_batch (int group_count, const int *group_sizes, bblas_enum_t layout, const bblas_enum_t *side, const bblas_enum_t *uplo, const int *m, const int *n, const double *alpha, double const *const *A, const int *lda, double const *const *B, const int *ldb, const double *beta, double **C, const int *ldc, int *info)
- void blas_ssymm_batch (int group_count, const int *group_sizes, bblas_enum_t layout, const bblas_enum_t *side, const bblas_enum_t *uplo, const int *m, const int *n, const float *alpha, float const *const *A, const int *lda, float const *const *B, const int *ldb, const float *beta, float **C, const int *ldc, int *info)
- void blas_zsymm_batch (int group_count, const int *group_sizes, bblas_enum_t layout, const bblas_enum_t *side, const bblas_enum_t *uplo, const int *m, const int *n, const bblas_complex64_t *alpha, bblas_complex64_t const *const *A, const int *lda, bblas_complex64_t const *const *B, const int *ldb, const bblas_complex64_t *beta, bblas_complex64_t **C, const int *ldc, int *info)

3.7.1 Detailed Description

$C[i] = \alpha[i]A[i]B[i] + \beta[i]C[i]$ or $C[i] = \alpha[i]B[i]A[i] + \beta[i]C[i]$ where $A[i]$ are symmetric

3.7.2 Function Documentation

3.7.2.1 blas_csymm_batch()

```
void blas_csymm_batch (
            int group_count,
            const int * group_sizes,
            bblas_enum_t layout,
            const bblas_enum_t * side,
            const bblas_enum_t * uplo,
            const int * m,
            const int * n,
            const bblas_complex32_t * alpha,
            bblas_complex32_t const *const * A,
            const int * lda,
            bblas_complex32_t const *const * B,
            const int * ldb,
            const bblas_complex32_t * beta,
            bblas_complex32_t ** C,
            const int * ldc,
            int * info )
```

Performs one of the batch matrix-matrix operations

$$C[i] = \alpha \times A[i] \times B[i] + \beta \times C[i]$$

or

$$C[i] = \alpha \times B[i] \times A[i] + \beta \times C[i]$$

for a group of matrices, where alpha[i] and beta[i] are scalars, A[i]-s are symmetric matrices and B[i]-s are C[i] are m[i]-by-n[i] matrices.

Parameters

in	*group_count*	The number groups of matrices.
in	*group_sizes*	An array of integers of length group_count, where group_sizes[i] denotes the number of matrices in i-th group.
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*side*	An array of length group_count. side[i] Specifies whether the Hermitian matrices A[j]-s of i-th group appear on the left or right in the operation as follows: • BblasLeft: $C[j] = \alpha[i] \times A[j] \times B[j] + \beta[i] \times C[j]$ • BblasRight: $C[j] = \alpha[i] \times B[j] \times A[j] + \beta[i] \times C[j]$
in	*uplo*	An array of length group_count, where uplo[i] specifies whether the upper or lower triangular part of the symmetric matrices A[j]-s of i-th group are to be referenced

- BblasLower: Only the lower triangular part of the symmetric matrices A[j] is to be referenced.
- BblasUpper: Only the upper triangular part of the symmetric matrices A[j] is to be referenced.

Parameters

in	*m*	An of array of integers of length group_count, where m[i] denotes the number of rows in the matrices C[j] of i-th group. m[i] >= 0.
in	*n*	An of array of integers of length group_count, where n[i] denotes the number of columns in the matrices C[j] of i-th group. n[i] >= 0.
in	*alpha*	An array of scalars of length group-count.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[batch_count-1], where each element A[j] of i-th group is a pointer to a matrix A[j] of size lda[i]-by-ka, where ka is m[i] when side[i] = BblasLeft, and is n[i] otherwise. Only the uplo triangular part is referenced. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*lda*	An array of length group_count, where lda[i] is the leading dimension of the arrays A[j] of i-th group. lda[i] >= max(1,ka).
in	*B*	B is an array of pointers to matrices B[0], B[1] .. B[batch_count-1], where each element B[j] of i-th group is a pointer to a matrix B[j] of size ldb[i]-by-n[i] matrix, where the leading m[i]-by-n[i] part of the array B[j] must contain the matrix B[j]. batch_count = {i=0}^{group_count-1}group_sizes[i].

Parameters

in	*ldb*	An array of length group_count, where ldb[i] is the leading dimension of the arrays B[j] of i-th group. ldb[i] >= max(1,m[i]).
in	*beta*	An array of scalars of length group_count.
in,out	*C*	C is an array of pointers to matrices C[0], C[1],...,C[batch_count-1]. In i-th group each element C[j] is a pointer to a matrix C[j]. On exit, each array C[j] of i-th group is overwritten by the m[i]-by-n[i] updated matrix. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldc*	An array of integers of length group_count, which denotes the leading dimension of the arrays C[j] in i-th group. ldc[i] >= max(1,m[i]).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast ({i=0}^{group_count-1}group_sizes[i] +1). • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast to (group_count+1). • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

csymm_batch
csymm_batch
dsymm_batch
ssymm_batch

3.7.2.2 blas_dsymm_batch()

```
void blas_dsymm_batch (
            int group_count,
            const int * group_sizes,
            bblas_enum_t layout,
            const bblas_enum_t * side,
            const bblas_enum_t * uplo,
            const int * m,
            const int * n,
            const double * alpha,
            double const *const * A,
            const int * lda,
            double const *const * B,
            const int * ldb,
            const double * beta,
```

```
double ** C,
const int * ldc,
int * info )
```

Performs one of the batch matrix-matrix operations

$$C[i] = \alpha \times A[i] \times B[i] + \beta \times C[i]$$

or

$$C[i] = \alpha \times B[i] \times A[i] + \beta \times C[i]$$

for a group of matrices, where alpha[i] and beta[i] are scalars, A[i]-s are symmetric matrices and B[i]-s are C[i] are m[i]-by-n[i] matrices.

Parameters

in	*group_count*	The number groups of matrices.
in	*group_sizes*	An array of integers of length group_count, where group_sizes[i] denotes the number of matrices in i-th group.
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*side*	An array of length group_count. side[i] Specifies whether the symmetric matrices A[j]-s of i-th group appear on the left or right in the operation as follows: • BblasLeft: $C[j] = \alpha[i] \times A[j] \times B[j] + \beta[i] \times C[j]$ • BblasRight: $C[j] = \alpha[i] \times B[j] \times A[j] + \beta[i] \times C[j]$
in	*uplo*	An array of length group_count, where uplo[i] specifies whether the upper or lower triangular part of the symmetric matrices A[j]-s of i-th group are to be referenced

- BblasLower: Only the lower triangular part of the symmetric matrices A[j] is to be referenced.
- BblasUpper: Only the upper triangular part of the symmetric matrices A[j] is to be referenced.

Parameters

in	*m*	An of array of integers of length group_count, where m[i] denotes the number of rows in the matrices C[j] of i-th group. m[i] >= 0.
in	*n*	An of array of integers of length group_count, where n[i] denotes the number of columns in the matrices C[j] of i-th group. n[i] >= 0.
in	*alpha*	An array of scalars of length group-count.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[batch_count-1], where each element A[j] of i-th group is a pointer to a matrix A[j] of size lda[i]-by-ka, where ka is m[i] when side[i] = BblasLeft, and is n[i] otherwise. Only the uplo triangular part is referenced. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*lda*	An array of length group_count, where lda[i] is the leading dimension of the arrays A[j] of i-th group. lda[i] >= max(1,ka).

Parameters

in	*B*	B is an array of pointers to matrices B[0], B[1] .. B[batch_count-1], where each element B[j] of i-th group is a pointer to a matrix B[j] of size ldb[i]-by-n[i] matrix, where the leading m[i]-by-n[i] part of the array B[j] must contain the matrix B[j]. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldb*	An array of length group_count, where ldb[i] is the leading dimension of the arrays B[j] of i-th group. ldb[i] >= max(1,m[i]).
in	*beta*	An array of scalars of length group_count.
in,out	*C*	C is an array of pointers to matrices C[0], C[1],...,C[batch_count-1]. In i-th group each element C[j] is a pointer to a matrix C[j]. On exit, each array C[j] of i-th group is overwritten by the m[i]-by-n[i] updated matrix. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldc*	An array of integers of length group_count, which denotes the leading dimension of the arrays C[j] in i-th group. ldc[i] >= max(1,m[i]).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast ({i=0}^{group_count-1}group_sizes[i] +1). • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast to (group_count+1). • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

dsymm_batch
csymm_batch
dsymm_batch
ssymm_batch

3.7.2.3 blas_ssymm_batch()

```
void blas_ssymm_batch (
            int group_count,
            const int * group_sizes,
            bblas_enum_t layout,
            const bblas_enum_t * side,
            const bblas_enum_t * uplo,
            const int * m,
            const int * n,
            const float * alpha,
            float const *const * A,
```

```
const int * lda,
float const *const * B,
const int * ldb,
const float * beta,
float ** C,
const int * ldc,
int * info )
```

Performs one of the batch matrix-matrix operations

$$C[i] = \alpha \times A[i] \times B[i] + \beta \times C[i]$$

or

$$C[i] = \alpha \times B[i] \times A[i] + \beta \times C[i]$$

for a group of matrices, where alpha[i] and beta[i] are scalars, A[i]-s are symmetric matrices and B[i]-s are C[i] are m[i]-by-n[i] matrices.

Parameters

in	*group_count*	The number groups of matrices.
in	*group_sizes*	An array of integers of length group_count, where group_sizes[i] denotes the number of matrices in i-th group.
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*side*	An array of length group_count. side[i] Specifies whether the symmetric matrices A[j]-s of i-th group appear on the left or right in the operation as follows: • BblasLeft: $C[j] = \alpha[i] \times A[j] \times B[j] + \beta[i] \times C[j]$ • BblasRight: $C[j] = \alpha[i] \times B[j] \times A[j] + \beta[i] \times C[j]$
in	*uplo*	An array of length group_count, where uplo[i] specifies whether the upper or lower triangular part of the symmetric matrices A[j]-s of i-th group are to be referenced

- BblasLower: Only the lower triangular part of the symmetric matrices A[j] is to be referenced.
- BblasUpper: Only the upper triangular part of the symmetric matrices A[j] is to be referenced.

Parameters

in	*m*	An of array of integers of length group_count, where m[i] denotes the number of rows in the matrices C[j] of i-th group. m[i] >= 0.
in	*n*	An of array of integers of length group_count, where n[i] denotes the number of columns in the matrices C[j] of i-th group. n[i] >= 0.
in	*alpha*	An array of scalars of length group-count.

Parameters

in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[batch_count-1], where each element A[j] of i-th group is a pointer to a matrix A[j] of size lda[i]-by-ka, where ka is m[i] when side[i] = BblasLeft, and is n[i] otherwise. Only the uplo triangular part is referenced. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*lda*	An array of length group_count, where lda[i] is the leading dimension of the arrays A[j] of i-th group. lda[i] >= max(1,ka).
in	*B*	B is an array of pointers to matrices B[0], B[1] .. B[batch_count-1], where each element B[j] of i-th group is a pointer to a matrix B[j] of size ldb[i]-by-n[i] matrix, where the leading m[i]-by-n[i] part of the array B[j] must contain the matrix B[j]. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldb*	An array of length group_count, where ldb[i] is the leading dimension of the arrays B[j] of i-th group. ldb[i] >= max(1,m[i]).
in	*beta*	An array of scalars of length group_count.
in,out	*C*	C is an array of pointers to matrices C[0], C[1],...,C[batch_count-1]. In i-th group each element C[j] is a pointer to a matrix C[j]. On exit, each array C[j] of i-th group is overwritten by the m[i]-by-n[i] updated matrix. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldc*	An array of integers of length group_count, which denotes the leading dimension of the arrays C[j] in i-th group. ldc[i] >= max(1,m[i]).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast ({i=0}^{group_count-1}group_sizes[i] +1). • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast to (group_count+1). • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

ssymm_batch
csymm_batch
dsymm_batch
ssymm_batch

3.7.2.4 blas_zsymm_batch()

```
void blas_zsymm_batch (
            int group_count,
            const int * group_sizes,
            bblas_enum_t layout,
```

```
const bblas_enum_t * side,
const bblas_enum_t * uplo,
const int * m,
const int * n,
const bblas_complex64_t * alpha,
bblas_complex64_t const *const * A,
const int * lda,
bblas_complex64_t const *const * B,
const int * ldb,
const bblas_complex64_t * beta,
bblas_complex64_t ** C,
const int * ldc,
int * info )
```

Performs one of the batch matrix-matrix operations

$$C[i] = \alpha \times A[i] \times B[i] + \beta \times C[i]$$

or

$$C[i] = \alpha \times B[i] \times A[i] + \beta \times C[i]$$

for a group of matrices, where alpha[i] and beta[i] are scalars, A[i]-s are symmetric matrices and B[i]-s are C[i] are m[i]-by-n[i] matrices.

Parameters

in	*group_count*	The number groups of matrices.
in	*group_sizes*	An array of integers of length group_count, where group_sizes[i] denotes the number of matrices in i-th group.
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*side*	An array of length group_count. side[i] Specifies whether the Hermitian matrices A[j]-s of i-th group appear on the left or right in the operation as follows: • BblasLeft: $C[j] = \alpha[i] \times A[j] \times B[j] + \beta[i] \times C[j]$ • BblasRight: $C[j] = \alpha[i] \times B[j] \times A[j] + \beta[i] \times C[j]$
in	*uplo*	An array of length group_count, where uplo[i] specifies whether the upper or lower triangular part of the symmetric matrices A[j]-s of i-th group are to be referenced

- BblasLower: Only the lower triangular part of the symmetric matrices A[j] is to be referenced.
- BblasUpper: Only the upper triangular part of the symmetric matrices A[j] is to be referenced.

Parameters

in	*m*	An of array of integers of length group_count, where m[i] denotes the number of rows in the matrices C[j] of i-th group. m[i] >= 0.
in	*n*	An of array of integers of length group_count, where n[i] denotes the number of columns in the matrices C[j] of i-th group. n[i] >= 0.
in	*alpha*	An array of scalars of length group-count.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[batch_count-1], where each element A[j] of i-th group is a pointer to a matrix A[j] of size lda[i]-by-ka, where ka is m[i] when side[i] = BblasLeft, and is n[i] otherwise. Only the uplo triangular part is referenced. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*lda*	An array of length group_count, where lda[i] is the leading dimension of the arrays A[j] of i-th group. lda[i] >= max(1,ka).
in	*B*	B is an array of pointers to matrices B[0], B[1] .. B[batch_count-1], where each element B[j] of i-th group is a pointer to a matrix B[j] of size ldb[i]-by-n[i] matrix, where the leading m[i]-by-n[i] part of the array B[j] must contain the matrix B[j]. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldb*	An array of length group_count, where ldb[i] is the leading dimension of the arrays B[j] of i-th group. ldb[i] >= max(1,m[i]).
in	*beta*	An array of scalars of length group_count.
in,out	*C*	C is an array of pointers to matrices C[0], C[1],...,C[batch_count-1]. In i-th group each element C[j] is a pointer to a matrix C[j]. On exit, each array C[j] of i-th group is overwritten by the m[i]-by-n[i] updated matrix. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldc*	An array of integers of length group_count, which denotes the leading dimension of the arrays C[j] in i-th group. ldc[i] >= max(1,m[i]).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast ({i=0}^{group_count-1}group_sizes[i] +1). • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast to (group_count+1). • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

zsymm_batch
csymm_batch
dsymm_batch
ssymm_batch

3.8 syrk_batch: Batched symmetric rank k update

$C[i] = \alpha[i]A[i]A[i]^T + \beta[i]C[i]$ where $C[i]$ are symmetric

Functions

- void blas_csyrk_batch (int group_count, const int *group_sizes, bblas_enum_t layout, const bblas_enum_t *uplo, const bblas_enum_t *trans, const int *n, const int *k, const bblas_complex32_t *alpha, bblas_complex32_t const *const *A, const int *lda, const bblas_complex32_t *beta, bblas_complex32_t **C, const int *ldc, int *info)
- void blas_dsyrk_batch (int group_count, const int *group_sizes, bblas_enum_t layout, const bblas_enum_t *uplo, const bblas_enum_t *trans, const int *n, const int *k, const double *alpha, double const *const *A, const int *lda, const double *beta, double **C, const int *ldc, int *info)
- void blas_ssyrk_batch (int group_count, const int *group_sizes, bblas_enum_t layout, const bblas_enum_t *uplo, const bblas_enum_t *trans, const int *n, const int *k, const float *alpha, float const *const *A, const int *lda, const float *beta, float **C, const int *ldc, int *info)
- void blas_zsyrk_batch (int group_count, const int *group_sizes, bblas_enum_t layout, const bblas_enum_t *uplo, const bblas_enum_t *trans, const int *n, const int *k, const bblas_complex64_t *alpha, bblas_complex64_t const *const *A, const int *lda, const bblas_complex64_t *beta, bblas_complex64_t **C, const int *ldc, int *info)

3.8.1 Detailed Description

$C[i] = \alpha[i]A[i]A[i]^T + \beta[i]C[i]$ where $C[i]$ are symmetric

3.8.2 Function Documentation

3.8.2.1 blas_csyrk_batch()

```
void blas_csyrk_batch (
            int group_count,
            const int * group_sizes,
            bblas_enum_t layout,
            const bblas_enum_t * uplo,
            const bblas_enum_t * trans,
            const int * n,
            const int * k,
            const bblas_complex32_t * alpha,
            bblas_complex32_t const *const * A,
            const int * lda,
            const bblas_complex32_t * beta,
            bblas_complex32_t ** C,
            const int * ldc,
            int * info )
```

Performs one of the batch symmetric rank k operations

$$C[i] = \alpha A[i] \times A[i]^T + \beta C[i],$$

or

$$C[i] = \alpha A[i]^T \times A[i] + \beta C[i],$$

for a group of matrices, where alpha[i] and beta[i] are scalars, C[i]-s are n[i]-by-n[i] symmetric matrices, and A[i]-s are n[i]-by-k[i] matrices in the first case and a k[i]-by-n[i] matrices in the second case.

Parameters

in	*group_count*	The number groups of matrices with fixed size.
in	*group_sizes*	An array of integers of length group_count, where group_sizes[i] denotes the number of matrices in i-th group.
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*uplo*	An array of length group_count, where uplo[i] specifies whether the upper or lower triangular part of the Hermitian matrices C[j]-s of i-th group are to be stored

- BblasLower: Only the lower triangular part of the Symmetric matrices C[j] are to be stored.
- BblasUpper: Only the upper triangular part of the Symmetric matrices C[j] are to be stored.

Parameters

in	*trans*	An array of length group_count, where for j-th matrix in i-th group • BblasNoTrans: $$C[j] = \alpha[i]A[j] \times A[j]^T + \beta[i]C[j];$$ • BblasTrans: $$C[j] = \alpha[i]A[j]^T \times A[j] + \beta[i]C[j].$$
in	*n*	An array of integers of length group count-1, where n[i] is the order of the matrices C[j] in i-th group. n[i] >= 0.
in	*k*	An array of integers of length group_count. For matrices in i-th group If trans = BblasNoTrans, number of columns of A[j]-s and B[j]-s matrices; if trans = BblasTrans, number of rows of A[j]-s and B[j]-s matrices.
in	*alpha*	An array of scalars of length group_count.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[batch_count-1]. In i-th group each element A[j] is a pointer to a matrix of A[j] of size lda[i]-by-ka. If trans[i] = BblasNoTrans, ka = k[i]; if trans[i] = BblasTrans, ka = n[i]. batch_count={i=1}^{group_count}group_sizes[i].
in	*lda*	An array of integers of length group_count, are the leading dimension of the arrays A[j] in i-th group. If trans[i] = BblasNoTrans, lda[i] >= max(1, n[i]); if trans[i] = BblasTrans, lda[i] >= max(1, k[i]).
in	*beta*	An array of scalars of length group_count.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[batc_count-1]. In i-th group each element C[j] is a pointer to a matrix C[j] of size ldc[i]-by-n[i]. On exit, the uplo[i] part of the matrix is overwritten by the uplo[i] part of the updated matrix. batch_count={i=1}^{group_count}group_sizes[i].
in	*ldc*	An array of integers of length group_count. Where ldc[i] is the leading dimension of the arrays C[j] in i-th group. ldc[i] >= max(1, n[i]).

Parameters

in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

csyrk_batch
csyrk_batch
dsyrk_batch
ssyrk_batch

3.8.2.2 blas_dsyrk_batch()

```
void blas_dsyrk_batch (
            int group_count,
            const int * group_sizes,
            bblas_enum_t layout,
            const bblas_enum_t * uplo,
            const bblas_enum_t * trans,
            const int * n,
            const int * k,
            const double * alpha,
            double const *const * A,
            const int * lda,
            const double * beta,
            double ** C,
            const int * ldc,
            int * info )
```

Performs one of the batch symmetric rank k operations

$$C[i] = \alpha A[i] \times A[i]^T + \beta C[i],$$

or

$$C[i] = \alpha A[i]^T \times A[i] + \beta C[i],$$

for a group of matrices, where alpha[i] and beta[i] are scalars, C[i]-s are n[i]-by-n[i] symmetric matrices, and A[i]-s are n[i]-by-k[i] matrices in the first case and a k[i]-by-n[i] matrices in the second case.

Parameters

in	*group_count*	The number groups of matrices with fixed size.
in	*group_sizes*	An array of integers of length group_count, where group_sizes[i] denotes the number of matrices in i-th group.
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*uplo*	An array of length group_count, where uplo[i] specifies whether the upper or lower triangular part of the symmetric matrices C[j]-s of i-th group are to be stored

- BblasLower: Only the lower triangular part of the Symmetric matrices C[j] are to be stored.
- BblasUpper: Only the upper triangular part of the Symmetric matrices C[j] are to be stored.

Parameters

in	*trans*	An array of length group_count, where for j-th matrix in i-th group • BblasNoTrans: $$C[j] = \alpha[i]A[j] \times A[j]^T + \beta[i]C[j];$$ • BblasTrans: $$C[j] = \alpha[i]A[j]^T \times A[j] + \beta[i]C[j].$$
in	*n*	An array of integers of length group count-1, where n[i] is the order of the matrices C[j] in i-th group. n[i] >= 0.
in	*k*	An array of integers of length group_count. For matrices in i-th group If trans = BblasNoTrans, number of columns of A[j]-s and B[j]-s matrices; if trans = BblasTrans, number of rows of A[j]-s and B[j]-s matrices.
in	*alpha*	An array of scalars of length group_count.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[batch_count-1]. In i-th group each element A[j] is a pointer to a matrix of A[j] of size lda[i]-by-ka. If trans[i] = BblasNoTrans, ka = k[i]; if trans[i] = BblasTrans, ka = n[i]. batch_count={i=1}^{group_count}group_sizes[i].
in	*lda*	An array of integers of length group_count, are the leading dimension of the arrays A[j] in i-th group. If trans[i] = BblasNoTrans, lda[i] >= max(1, n[i]); if trans[i] = BblasTrans, lda[i] >= max(1, k[i]).
in	*beta*	An array of scalars of length group_count.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[batc_count-1]. In i-th group each element C[j] is a pointer to a matrix C[j] of size ldc[i]-by-n[i]. On exit, the uplo[i] part of the matrix is overwritten by the uplo[i] part of the updated matrix. batch_count={i=1}^{group_count}group_sizes[i].
in	*ldc*	An array of integers of length group_count. Where ldc[i] is the leading dimension of the arrays C[j] in i-th group. ldc[i] >= max(1, n[i]).

Parameters

in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

dsyrk_batch
csyrk_batch
dsyrk_batch
ssyrk_batch

3.8.2.3 blas_ssyrk_batch()

```
void blas_ssyrk_batch (
            int group_count,
            const int * group_sizes,
            bblas_enum_t layout,
            const bblas_enum_t * uplo,
            const bblas_enum_t * trans,
            const int * n,
            const int * k,
            const float * alpha,
            float const *const * A,
            const int * lda,
            const float * beta,
            float ** C,
            const int * ldc,
            int * info )
```

Performs one of the batch symmetric rank k operations

$$C[i] = \alpha A[i] \times A[i]^T + \beta C[i],$$

or

$$C[i] = \alpha A[i]^T \times A[i] + \beta C[i],$$

for a group of matrices, where alpha[i] and beta[i] are scalars, C[i]-s are n[i]-by-n[i] symmetric matrices, and A[i]-s are n[i]-by-k[i] matrices in the first case and a k[i]-by-n[i] matrices in the second case.

Parameters

in	*group_count*	The number groups of matrices with fixed size.
in	*group_sizes*	An array of integers of length group_count, where group_sizes[i] denotes the number of matrices in i-th group.
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*uplo*	An array of length group_count, where uplo[i] specifies whether the upper or lower triangular part of the symmetric matrices C[j]-s of i-th group are to be stored

- BblasLower: Only the lower triangular part of the Symmetric matrices C[j] are to be stored.
- BblasUpper: Only the upper triangular part of the Symmetric matrices C[j] are to be stored.

Parameters

in	*trans*	An array of length group_count, where for j-th matrix in i-th group • BblasNoTrans: $$C[j] = \alpha[i]A[j] \times A[j]^T + \beta[i]C[j];$$ • BblasTrans: $$C[j] = \alpha[i]A[j]^T \times A[j] + \beta[i]C[j].$$
in	*n*	An array of integers of length group count-1, where n[i] is the order of the matrices C[j] in i-th group. n[i] >= 0.
in	*k*	An array of integers of length group_count. For matrices in i-th group If trans = BblasNoTrans, number of columns of A[j]-s and B[j]-s matrices; if trans = BblasTrans, number of rows of A[j]-s and B[j]-s matrices.
in	*alpha*	An array of scalars of length group_count.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[batch_count-1]. In i-th group each element A[j] is a pointer to a matrix of A[j] of size lda[i]-by-ka. If trans[i] = BblasNoTrans, ka = k[i]; if trans[i] = BblasTrans, ka = n[i]. batch_count={i=1}^{group_count}group_sizes[i].
in	*lda*	An array of integers of length group_count, are the leading dimension of the arrays A[j] in i-th group. If trans[i] = BblasNoTrans, lda[i] >= max(1, n[i]); if trans[i] = BblasTrans, lda[i] >= max(1, k[i]).
in	*beta*	An array of scalars of length group_count.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[batc_count-1]. In i-th group each element C[j] is a pointer to a matrix C[j] of size ldc[i]-by-n[i]. On exit, the uplo[i] part of the matrix is overwritten by the uplo[i] part of the updated matrix. batch_count={i=1}^{group_count}group_sizes[i].
in	*ldc*	An array of integers of length group_count. Where ldc[i] is the leading dimension of the arrays C[j] in i-th group. ldc[i] >= max(1, n[i]).

Parameters

in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

ssyrk_batch
csyrk_batch
dsyrk_batch
ssyrk_batch

3.8.2.4 blas_zsyrk_batch()

```
void blas_zsyrk_batch (
            int group_count,
            const int * group_sizes,
            bblas_enum_t layout,
            const bblas_enum_t * uplo,
            const bblas_enum_t * trans,
            const int * n,
            const int * k,
            const bblas_complex64_t * alpha,
            bblas_complex64_t const *const * A,
            const int * lda,
            const bblas_complex64_t * beta,
            bblas_complex64_t ** C,
            const int * ldc,
            int * info )
```

Performs one of the batch symmetric rank k operations

$$C[i] = \alpha A[i] \times A[i]^T + \beta C[i],$$

or

$$C[i] = \alpha A[i]^T \times A[i] + \beta C[i],$$

for a group of matrices, where alpha[i] and beta[i] are scalars, C[i]-s are n[i]-by-n[i] symmetric matrices, and A[i]-s are n[i]-by-k[i] matrices in the first case and a k[i]-by-n[i] matrices in the second case.

Parameters

in	*group_count*	The number groups of matrices with fixed size.
in	*group_sizes*	An array of integers of length group_count, where group_sizes[i] denotes the number of matrices in i-th group.
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*uplo*	An array of length group_count, where uplo[i] specifies whether the upper or lower triangular part of the Hermitian matrices C[j]-s of i-th group are to be stored

- BblasLower: Only the lower triangular part of the Symmetric matrices C[j] are to be stored.
- BblasUpper: Only the upper triangular part of the Symmetric matrices C[j] are to be stored.

Parameters

in	*trans*	An array of length group_count, where for j-th matrix in i-th group • BblasNoTrans: $$C[j] = \alpha[i]A[j] \times A[j]^T + \beta[i]C[j];$$ • BblasTrans: $$C[j] = \alpha[i]A[j]^T \times A[j] + \beta[i]C[j].$$
in	*n*	An array of integers of length group count-1, where n[i] is the order of the matrices C[j] in i-th group. n[i] >= 0.
in	*k*	An array of integers of length group_count. For matrices in i-th group If trans = BblasNoTrans, number of columns of A[j]-s and B[j]-s matrices; if trans = BblasTrans, number of rows of A[j]-s and B[j]-s matrices.
in	*alpha*	An array of scalars of length group_count.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[batch_count-1]. In i-th group each element A[j] is a pointer to a matrix of A[j] of size lda[i]-by-ka. If trans[i] = BblasNoTrans, ka = k[i]; if trans[i] = BblasTrans, ka = n[i]. batch_count={i=1}^{group_count}group_sizes[i].
in	*lda*	An array of integers of length group_count, are the leading dimension of the arrays A[j] in i-th group. If trans[i] = BblasNoTrans, lda[i] >= max(1, n[i]); if trans[i] = BblasTrans, lda[i] >= max(1, k[i]).
in	*beta*	An array of scalars of length group_count.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[batc_count-1]. In i-th group each element C[j] is a pointer to a matrix C[j] of size ldc[i]-by-n[i]. On exit, the uplo[i] part of the matrix is overwritten by the uplo[i] part of the updated matrix. batch_count={i=1}^{group_count}group_sizes[i].
in	*ldc*	An array of integers of length group_count. Where ldc[i] is the leading dimension of the arrays C[j] in i-th group. ldc[i] >= max(1, n[i]).

Parameters

in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

zsyrk_batch
csyrk_batch
dsyrk_batch
ssyrk_batch

3.9 syr2k_batch: Batched symmetric rank 2k update

$C[i] = \alpha[i]A[i]B[i]^T + \alpha[i]B[i]A[i]^T + \beta[i]C[i]$ where $C[i]$ are symmetric

Functions

- void blas_csyr2k_batch (int group_count, const int *group_sizes, bblas_enum_t layout, const bblas_enum_t *uplo, const bblas_enum_t *trans, const int *n, const int *k, const bblas_complex32_t *alpha, bblas_complex32_t const *const *A, const int *lda, bblas_complex32_t const *const *B, const int *ldb, const bblas_complex32_t *beta, bblas_complex32_t **C, const int *ldc, int *info)
- void blas_dsyr2k_batch (int group_count, const int *group_sizes, bblas_enum_t layout, const bblas_enum_t *uplo, const bblas_enum_t *trans, const int *n, const int *k, const double *alpha, double const *const *A, const int *lda, double const *const *B, const int *ldb, const double *beta, double **C, const int *ldc, int *info)
- void blas_ssyr2k_batch (int group_count, const int *group_sizes, bblas_enum_t layout, const bblas_enum_t *uplo, const bblas_enum_t *trans, const int *n, const int *k, const float *alpha, float const *const *A, const int *lda, float const *const *B, const int *ldb, const float *beta, float **C, const int *ldc, int *info)
- void blas_zsyr2k_batch (int group_count, const int *group_sizes, bblas_enum_t layout, const bblas_enum_t *uplo, const bblas_enum_t *trans, const int *n, const int *k, const bblas_complex64_t *alpha, bblas_complex64_t const *const *A, const int *lda, bblas_complex64_t const *const *B, const int *ldb, const bblas_complex64_t *beta, bblas_complex64_t **C, const int *ldc, int *info)

3.9.1 Detailed Description

$C[i] = \alpha[i]A[i]B[i]^T + \alpha[i]B[i]A[i]^T + \beta[i]C[i]$ where $C[i]$ are symmetric

3.9.2 Function Documentation

3.9.2.1 blas_csyr2k_batch()

```
void blas_csyr2k_batch (
            int group_count,
            const int * group_sizes,
            bblas_enum_t layout,
            const bblas_enum_t * uplo,
            const bblas_enum_t * trans,
            const int * n,
            const int * k,
            const bblas_complex32_t * alpha,
            bblas_complex32_t const *const * A,
            const int * lda,
            bblas_complex32_t const *const * B,
            const int * ldb,
            const bblas_complex32_t * beta,
            bblas_complex32_t ** C,
            const int * ldc,
            int * info )
```

Performs one of the symmetric rank 2k operations on a group of matrices, .

$$C[i] = \alpha A[i] \times B[i]^T + \alpha B[i] \times A[i]^T + \beta C[i],$$

or

$$C[i] = \alpha A[i]^T \times B[i] + \alpha B[i]^T \times A[i] + \beta C[i],$$

where alpha[i]-s and beta[i]-s are scalars, C[i]-s are n-by-n symmetric matrices and A[i]-s and B[i]-s are n-by-k matrices in the first case and k-by-n matrices in the second case.

Parameters

in	*group_count*	The number groups of matrices.
in	*group_sizes*	An array of integers of length group_count, where group_sizes[i] denotes the number of matrices in i-th group.
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*uplo*	An array of length group_count, where uplo[i] specifies whether the upper or lower triangular part of the Hermitian matrices C[j]-s of i-th group are to be stored

- BblasLower: Only the lower triangular part of the symmetric matrices C[j] are to be stored.
- BblasUpper: Only the upper triangular part of the symmetric matrices C[j] are to be stored.

Parameters

in	*trans*	An array of length group_count, where for j-th matrix in i-th group • BblasNoTrans: $$C[i] = \alpha[i]A[j] \times B[j]^T + \alpha[i]B[j] \times A[j]^T + \beta[i]C[j];$$ • BblasTrans: $$C[j] = \alpha[i]A[j]^T \times B[j] + \alpha[i]B[j]^T \times A[j] + \beta[i]C[j].$$
in	*n*	An array of integers of length group_count, where n[i] is the order of the matrices C[j] in i-th group. n[i] >= 0.
in	*k*	An array of integers of length group_count. For matrices in i-th group If trans = BblasNoTrans, number of columns of A[j]-s and B[j]-s matrices; if trans = BblasTrans, number of rows of A[j]-s and B[j]-s matrices.
in	*alpha*	An array of scalars of length group_count.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[batch_count-1]. In i-th group each element A[j] is a pointer to a matrix of A[j] of size lda[i]-by-ka. If trans[i] = BblasNoTrans, ka = k[i]; if trans[i] = BblasTrans, ka = n[i]. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*lda*	An array of integers of length group_count, are the leading dimension of the arrays A[j] in i-th group. If trans[i] = BblasNoTrans, lda[i] >= max(1, n[i]); if trans[i] = BblasTrans, lda[i] >= max(1, k[i]).
in	*B*	B is an array of pointers to matrices B[0], B[1] .. B[batch_count-1]. In i-th group each element B[j] is a pointer to a matrix B[j] of size ldb[i]-by-kb. If trans[i] = BblasNoTrans, kb = k[i]; if trans[i] = BblasTrans, kb = n[i]. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldb*	An array of integers of size group_count, are the leading dimension of the arrays B[j] in i-th group. If trans[i] = BblasNoTrans, ldb[i] >= max(1, n[i]); if trans[i] = BblasTrans, ldb[i] >= max(1, k[i]).
in	*beta*	An array of scalars of size group_count.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[batc_count-1]. In i-th group each element C[j] is a pointer to a matrix C[j] of size ldc[i]-by-n[i]. On exit, the uplo[i] part of the matrix is overwritten by the uplo[i] part of the updated matrix. batch_count = {i=0}^{group_count-1}group_sizes[i].

Parameters

in	*ldc*	An array of integers of length group_count. Where ldc[i] is the leading dimension of the arrays C[j] in i-th group. ldc[i] >= max(1, n[i]).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

csyr2k_batch
csyr2k_batch
dsyr2k_batch
ssyr2k_batch

3.9.2.2 blas_dsyr2k_batch()

```
void blas_dsyr2k_batch (
            int group_count,
            const int * group_sizes,
            bblas_enum_t layout,
            const bblas_enum_t * uplo,
            const bblas_enum_t * trans,
            const int * n,
            const int * k,
            const double * alpha,
            double const *const * A,
            const int * lda,
            double const *const * B,
            const int * ldb,
            const double * beta,
            double ** C,
            const int * ldc,
            int * info )
```

Performs one of the symmetric rank 2k operations on a group of matrices, .

$$C[i] = \alpha A[i] \times B[i]^T + \alpha B[i] \times A[i]^T + \beta C[i],$$

or

$$C[i] = \alpha A[i]^T \times B[i] + \alpha B[i]^T \times A[i] + \beta C[i],$$

where alpha[i]-s and beta[i]-s are scalars, C[i]-s are n-by-n symmetric matrices and A[i]-s and B[i]-s are n-by-k matrices in the first case and k-by-n matrices in the second case.

Parameters

in	*group_count*	The number groups of matrices.
in	*group_sizes*	An array of integers of length group_count, where group_sizes[i] denotes the number of matrices in i-th group.
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*uplo*	An array of length group_count, where uplo[i] specifies whether the upper or lower triangular part of the symmetric matrices C[j]-s of i-th group are to be stored

- BblasLower: Only the lower triangular part of the symmetric matrices C[j] are to be stored.
- BblasUpper: Only the upper triangular part of the symmetric matrices C[j] are to be stored.

Parameters

in	*trans*	An array of length group_count, where for j-th matrix in i-th group • BblasNoTrans: $$C[i] = \alpha[i]A[j] \times B[j]^T + \alpha[i]B[j] \times A[j]^T + \beta[i]C[j];$$ • BblasTrans: $$C[j] = \alpha[i]A[j]^T \times B[j] + \alpha[i]B[j]^T \times A[j] + \beta[i]C[j].$$
in	*n*	An array of integers of length group_count, where n[i] is the order of the matrices C[j] in i-th group. n[i] >= 0.
in	*k*	An array of integers of length group_count. For matrices in i-th group If trans = BblasNoTrans, number of columns of A[j]-s and B[j]-s matrices; if trans = BblasTrans, number of rows of A[j]-s and B[j]-s matrices.
in	*alpha*	An array of scalars of length group_count.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[batch_count-1]. In i-th group each element A[j] is a pointer to a matrix of A[j] of size lda[i]-by-ka. If trans[i] = BblasNoTrans, ka = k[i]; if trans[i] = BblasTrans, ka = n[i]. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*lda*	An array of integers of length group_count, are the leading dimension of the arrays A[j] in i-th group. If trans[i] = BblasNoTrans, lda[i] >= max(1, n[i]); if trans[i] = BblasTrans, lda[i] >= max(1, k[i]).
in	*B*	B is an array of pointers to matrices B[0], B[1] .. B[batch_count-1]. In i-th group each element B[j] is a pointer to a matrix B[j] of size ldb[i]-by-kb. If trans[i] = BblasNoTrans, kb = k[i]; if trans[i] = BblasTrans, kb = n[i]. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldb*	An array of integers of size group_count, are the leading dimension of the arrays B[j] in i-th group. If trans[i] = BblasNoTrans, ldb[i] >= max(1, n[i]); if trans[i] = BblasTrans, ldb[i] >= max(1, k[i]).

Parameters

in	*beta*	An array of scalars of size group_count.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[batc_count-1]. In i-th group each element C[j] is a pointer to a matrix C[j] of size ldc[i]-by-n[i]. On exit, the uplo[i] part of the matrix is overwritten by the uplo[i] part of the updated matrix. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldc*	An array of integers of length group_count. Where ldc[i] is the leading dimension of the arrays C[j] in i-th group. ldc[i] >= max(1, n[i]).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

dsyr2k_batch
csyr2k_batch
dsyr2k_batch
ssyr2k_batch

3.9.2.3 blas_ssyr2k_batch()

```
void blas_ssyr2k_batch (
            int group_count,
            const int * group_sizes,
            bblas_enum_t layout,
            const bblas_enum_t * uplo,
            const bblas_enum_t * trans,
            const int * n,
            const int * k,
            const float * alpha,
            float const *const * A,
            const int * lda,
            float const *const * B,
            const int * ldb,
            const float * beta,
            float ** C,
```

```
const int * ldc,
int * info )
```

Performs one of the symmetric rank 2k operations on a group of matrices, .

$$C[i] = \alpha A[i] \times B[i]^T + \alpha B[i] \times A[i]^T + \beta C[i],$$

or

$$C[i] = \alpha A[i]^T \times B[i] + \alpha B[i]^T \times A[i] + \beta C[i],$$

where alpha[i]-s and beta[i]-s are scalars, C[i]-s are n-by-n symmetric matrices and A[i]-s and B[i]-s are n-by-k matrices in the first case and k-by-n matrices in the second case.

Parameters

in	*group_count*	The number groups of matrices.
in	*group_sizes*	An array of integers of length group_count, where group_sizes[i] denotes the number of matrices in i-th group.
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*uplo*	An array of length group_count, where uplo[i] specifies whether the upper or lower triangular part of the symmetric matrices C[j]-s of i-th group are to be stored

- BblasLower: Only the lower triangular part of the symmetric matrices C[j] are to be stored.
- BblasUpper: Only the upper triangular part of the symmetric matrices C[j] are to be stored.

Parameters

in	*trans*	An array of length group_count, where for j-th matrix in i-th group • BblasNoTrans: $C[i] = \alpha[i]A[j] \times B[j]^T + \alpha[i]B[j] \times A[j]^T + \beta[i]C[j];$ • BblasTrans: $C[j] = \alpha[i]A[j]^T \times B[j] + \alpha[i]B[j]^T \times A[j] + \beta[i]C[j].$
in	*n*	An array of integers of length group_count, where n[i] is the order of the matrices C[j] in i-th group. n[i] >= 0.
in	*k*	An array of integers of length group_count. For matrices in i-th group If trans = BblasNoTrans, number of columns of A[j]-s and B[j]-s matrices; if trans = BblasTrans, number of rows of A[j]-s and B[j]-s matrices.
in	*alpha*	An array of scalars of length group_count.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[batch_count-1]. In i-th group each element A[j] is a pointer to a matrix of A[j] of size lda[i]-by-ka. If trans[i] = BblasNoTrans, ka = k[i]; if trans[i] = BblasTrans, ka = n[i]. batch_count = {i=0}^{group_count-1}group_sizes[i].

Parameters

in	*lda*	An array of integers of length group_count, are the leading dimension of the arrays A[j] in i-th group. If trans[i] = BblasNoTrans, lda[i] >= max(1, n[i]); if trans[i] = BblasTrans, lda[i] >= max(1, k[i]).
in	*B*	B is an array of pointers to matrices B[0], B[1] .. B[batch_count-1]. In i-th group each element B[j] is a pointer to a matrix B[j] of size ldb[i]-by-kb. If trans[i] = BblasNoTrans, kb = k[i]; if trans[i] = BblasTrans, kb = n[i]. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldb*	An array of integers of size group_count, are the leading dimension of the arrays B[j] in i-th group. If trans[i] = BblasNoTrans, ldb[i] >= max(1, n[i]); if trans[i] = BblasTrans, ldb[i] >= max(1, k[i]).
in	*beta*	An array of scalars of size group_count.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[batc_count-1]. In i-th group each element C[j] is a pointer to a matrix C[j] of size ldc[i]-by-n[i]. On exit, the uplo[i] part of the matrix is overwritten by the uplo[i] part of the updated matrix. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldc*	An array of integers of length group_count. Where ldc[i] is the leading dimension of the arrays C[j] in i-th group. ldc[i] >= max(1, n[i]).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

ssyr2k_batch
csyr2k_batch
dsyr2k_batch
ssyr2k_batch

3.9.2.4 blas_zsyr2k_batch()

```
void blas_zsyr2k_batch (
            int group_count,
            const int * group_sizes,
            bblas_enum_t layout,
            const bblas_enum_t * uplo,
            const bblas_enum_t * trans,
```

```
const int * n,
const int * k,
const bblas_complex64_t * alpha,
bblas_complex64_t const *const * A,
const int * lda,
bblas_complex64_t const *const * B,
const int * ldb,
const bblas_complex64_t * beta,
bblas_complex64_t ** C,
const int * ldc,
int * info )
```

Performs one of the symmetric rank 2k operations on a group of matrices, .

$$C[i] = \alpha A[i] \times B[i]^T + \alpha B[i] \times A[i]^T + \beta C[i],$$

or

$$C[i] = \alpha A[i]^T \times B[i] + \alpha B[i]^T \times A[i] + \beta C[i],$$

where alpha[i]-s and beta[i]-s are scalars, C[i]-s are n-by-n symmetric matrices and A[i]-s and B[i]-s are n-by-k matrices in the first case and k-by-n matrices in the second case.

Parameters

in	*group_count*	The number groups of matrices.
in	*group_sizes*	An array of integers of length group_count, where group_sizes[i] denotes the number of matrices in i-th group.
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*uplo*	An array of length group_count, where uplo[i] specifies whether the upper or lower triangular part of the Hermitian matrices C[j]-s of i-th group are to be stored

- BblasLower: Only the lower triangular part of the symmetric matrices C[j] are to be stored.
- BblasUpper: Only the upper triangular part of the symmetric matrices C[j] are to be stored.

Parameters

in	*trans*	An array of length group_count, where for j-th matrix in i-th group • BblasNoTrans: $C[i] = \alpha[i]A[j] \times B[j]^T + \alpha[i]B[j] \times A[j]^T + \beta[i]C[j];$ • BblasTrans: $C[j] = \alpha[i]A[j]^T \times B[j] + \alpha[i]B[j]^T \times A[j] + \beta[i]C[j].$
in	*n*	An array of integers of length group_count, where n[i] is the order of the matrices C[j] in i-th group. n[i] $>=$ 0.

Parameters

in	*k*	An array of integers of length group_count. For matrices in i-th group If trans = BblasNoTrans, number of columns of A[j]-s and B[j]-s matrices; if trans = BblasTrans, number of rows of A[j]-s and B[j]-s matrices.
in	*alpha*	An array of scalars of length group_count.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[batch_count-1]. In i-th group each element A[j] is a pointer to a matrix of A[j] of size lda[i]-by-ka. If trans[i] = BblasNoTrans, ka = k[i]; if trans[i] = BblasTrans, ka = n[i]. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*lda*	An array of integers of length group_count, are the leading dimension of the arrays A[j] in i-th group. If trans[i] = BblasNoTrans, lda[i] >= max(1, n[i]); if trans[i] = BblasTrans, lda[i] >= max(1, k[i]).
in	*B*	B is an array of pointers to matrices B[0], B[1] .. B[batch_count-1]. In i-th group each element B[j] is a pointer to a matrix B[j] of size ldb[i]-by-kb. If trans[i] = BblasNoTrans, kb = k[i]; if trans[i] = BblasTrans, kb = n[i]. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldb*	An array of integers of size group_count, are the leading dimension of the arrays B[j] in i-th group. If trans[i] = BblasNoTrans, ldb[i] >= max(1, n[i]); if trans[i] = BblasTrans, ldb[i] >= max(1, k[i]).
in	*beta*	An array of scalars of size group_count.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[batc_count-1]. In i-th group each element C[j] is a pointer to a matrix C[j] of size ldc[i]-by-n[i]. On exit, the uplo[i] part of the matrix is overwritten by the uplo[i] part of the updated matrix. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldc*	An array of integers of length group_count. Where ldc[i] is the leading dimension of the arrays C[j] in i-th group. ldc[i] >= max(1, n[i]).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

zsyr2k_batch
csyr2k_batch
dsyr2k_batch
ssyr2k_batch

3.10 trmm_batch: Batched triangular matrix multiply

$B[i] = \alpha[i]\ op(A[i])\ B[i]$ or $B[i] = \alpha[i]B[i]\ op(A[i])$ where $A[i]$ are triangular

Functions

- void blas_ctrmm_batch (int group_count, const int *group_sizes, bblas_enum_t layout, const bblas_enum_t *side, const bblas_enum_t *uplo, const bblas_enum_t *transa, const bblas_enum_t *diag, const int *m, const int *n, const bblas_complex32_t *alpha, bblas_complex32_t const *const *A, const int *lda, bblas_complex32_t **B, int const *ldb, int *info)
- void blas_dtrmm_batch (int group_count, const int *group_sizes, bblas_enum_t layout, const bblas_enum_t *side, const bblas_enum_t *uplo, const bblas_enum_t *transa, const bblas_enum_t *diag, const int *m, const int *n, const double *alpha, double const *const *A, const int *lda, double **B, int const *ldb, int *info)
- void blas_strmm_batch (int group_count, const int *group_sizes, bblas_enum_t layout, const bblas_enum_t *side, const bblas_enum_t *uplo, const bblas_enum_t *transa, const bblas_enum_t *diag, const int *m, const int *n, const float *alpha, float const *const *A, const int *lda, float **B, int const *ldb, int *info)
- void blas_ztrmm_batch (int group_count, const int *group_sizes, bblas_enum_t layout, const bblas_enum_t *side, const bblas_enum_t *uplo, const bblas_enum_t *transa, const bblas_enum_t *diag, const int *m, const int *n, const bblas_complex64_t *alpha, bblas_complex64_t const *const *A, const int *lda, bblas_complex64_t **B, int const *ldb, int *info)

3.10.1 Detailed Description

$B[i] = \alpha[i]\ op(A[i])\ B[i]$ or $B[i] = \alpha[i]B[i]\ op(A[i])$ where $A[i]$ are triangular

3.10.2 Function Documentation

3.10.2.1 blas_ctrmm_batch()

```
void blas_ctrmm_batch (
            int group_count,
            const int * group_sizes,
            bblas_enum_t layout,
            const bblas_enum_t * side,
            const bblas_enum_t * uplo,
            const bblas_enum_t * transa,
            const bblas_enum_t * diag,
            const int * m,
            const int * n,
            const bblas_complex32_t * alpha,
            bblas_complex32_t const *const * A,
            const int * lda,
            bblas_complex32_t ** B,
            int const * ldb,
            int * info )
```

Performs a triangular batch matrix-matrix multiply of the form

$$B[i] = \alpha[op(A[i]) \times B[i]]$$

, if side = BblasLeft or

$$B[i] = \alpha[B[i] \times op(A[i])]$$

, if side = BblasRight

for a group of matrices, and op(X) is one of:

```
- op(A[i]) = A[i]   or
- op(A[i]) = A[i]^T or
- op(A[i]) = A[i]^H
```

alpha[i]-s are scalars, B[i]-s are m-by-n matrices and A[i]-s are a unit or non-unit, upper or lower triangular matrix.

Parameters

in	*group_count*	The number groups of matrices.
in	*group_sizes*	The number of matrices in each group.
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*side*	An array of length group_count, for matrices of i-th group it specifies whether op(A[j]) appears on the left or on the right of B[j]: • BblasLeft: alpha[i]∗op(A[j])∗B[j] • BblasRight: alpha[i]∗B[j]∗op(A[j])
in	*uplo*	An array of length group_count, where uplo[i] specifies whether the upper or lower triangular part of the symmetric matrices A[j]-s of i-th group are to be referenced

- BblasLower: Only the lower triangular part of the symmetric matrices A[j] is to be referenced.
- BblasUpper: Only the upper triangular part of the symmetric matrices A[j] is to be referenced.

Parameters

in	*transa*	An array of length group_count, where • BblasNoTrans: A[j]-s in i-th group are not transposed, • BblasTrans: A[j]-s in i-th group are transposed, • BblasConjTrans: A[j]-s in i-th group are conjugate transposed.
in	*diag*	An array of length group_count, which specifies whether or not A[j]-s of i-th group are unit triangular: • BblasNonUnit: A[j]-s are non-unit triangular; • BblasUnit: A[j]-s are unit triangular.

Parameters

in	*m*	An array integers of length group_count, which specified the number of rows of matrices B[j] in i-th group. m[i] >= 0.
in	*n*	An array integers of length group_count, which specified the number of columns of matrices B[j] in i-th group. n[i] >= 0.
in	*alpha*	An array of length group_count, where alpha[i] is a scalar.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[batch_count-1], where for i-th group each element A[j] is a pointer to a triangular matrix of dimension lda[i]-by-k, where k is m[i] when side='L' or 'l' and k is n[i] when when side='R' or 'r'. If uplo = BblasUpper, the leading k-by-k upper triangular part of the array A[j] contains the upper triangular matrix, and the strictly lower triangular part of A[j] is not referenced. If uplo = BblasLower, the leading k-by-k lower triangular part of the array A[j] contains the lower triangular matrix, and the strictly upper triangular part of A[j] is not referenced. If diag = BblasUnit, the diagonal elements of A[j] are also not referenced and are assumed to be 1. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*lda*	An array of integers of length group_count, where lda[i] denotes the leading dimension of the arrays A[j] of i-th group. When side='L' or 'l', lda[i] >= max(1,m[i]), when side='R' or 'r' then lda[i] >= max(1,n[i]).
in	*B*	B is an array of pointers to matrices B[0], B[1],..,B[batch_count-1], where for i-th group each element B[j] is a pointer to a matrix. On entry, the matrices B[j] are of dimension ldb[i]-by-n[i]. On exit, the result of a triangular matrix-matrix multiply (alpha[i]*op(A[j])*B[j]) or (alpha[i]*B[j]*op(A[j])). batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldb*	An array of integers of length group_count, where ldb[i] is the leading dimension of the arrays B[j] of i-th group. ldb[i] >= max(1,m[i]).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

ctrmm_batch
ctrmm_batch
dtrmm_batch
strmm_batch

3.10.2.2 blas_dtrmm_batch()

```
void blas_dtrmm_batch (
            int group_count,
            const int * group_sizes,
            bblas_enum_t layout,
            const bblas_enum_t * side,
            const bblas_enum_t * uplo,
            const bblas_enum_t * transa,
            const bblas_enum_t * diag,
            const int * m,
            const int * n,
            const double * alpha,
            double const *const * A,
            const int * lda,
            double ** B,
            int const * ldb,
            int * info )
```

Performs a triangular batch matrix-matrix multiply of the form

$$B[i] = \alpha[op(A[i]) \times B[i]]$$

, if side = BblasLeft or

$$B[i] = \alpha[B[i] \times op(A[i])]$$

, if side = BblasRight

for a group of matrices, and op(X) is one of:

```
- op(A[i]) = A[i]   or
- op(A[i]) = A[i]^T or
- op(A[i]) = A[i]^T
```

alpha[i]-s are scalars, B[i]-s are m-by-n matrices and A[i]-s are a unit or non-unit, upper or lower triangular matrix.

Parameters

in	*group_count*	The number groups of matrices.
in	*group_sizes*	The number of matrices in each group.
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*side*	An array of length group_count, for matrices of i-th group it specifies whether op(A[j]) appears on the left or on the right of B[j]: • BblasLeft: alpha[i]∗op(A[j])∗B[j] • BblasRight: alpha[i]∗B[j]∗op(A[j])
in	*uplo*	An array of length group_count, where uplo[i] specifies whether the upper or lower triangular part of the symmetric matrices A[j]-s of i-th group are to be referenced

- BblasLower: Only the lower triangular part of the symmetric matrices A[j] is to be referenced.
- BblasUpper: Only the upper triangular part of the symmetric matrices A[j] is to be referenced.

Parameters

in	*transa*	An array of length group_count, where • BblasNoTrans: A[j]-s in i-th group are not transposed, • BblasTrans: A[j]-s in i-th group are transposed, • BblasConjTrans: A[j]-s in i-th group are conjugate transposed.
in	*diag*	An array of length group_count, which specifies whether or not A[j]-s of i-th group are unit triangular: • BblasNonUnit: A[j]-s are non-unit triangular; • BblasUnit: A[j]-s are unit triangular.
in	*m*	An array integers of length group_count, which specified the number of rows of matrices B[j] in i-th group. m[i] >= 0.
in	*n*	An array integers of length group_count, which specified the number of columns of matrices B[j] in i-th group. n[i] >= 0.
in	*alpha*	An array of length group_count, where alpha[i] is a scalar.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[batch_count-1], where for i-th group each element A[j] is a pointer to a triangular matrix of dimension lda[i]-by-k, where k is m[i] when side='L' or 'l' and k is n[i] when when side='R' or 'r'. If uplo = BblasUpper, the leading k-by-k upper triangular part of the array A[j] contains the upper triangular matrix, and the strictly lower triangular part of A[j] is not referenced. If uplo = BblasLower, the leading k-by-k lower triangular part of the array A[j] contains the lower triangular matrix, and the strictly upper triangular part of A[j] is not referenced. If diag = BblasUnit, the diagonal elements of A[j] are also not referenced and are assumed to be 1. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*lda*	An array of integers of length group_count, where lda[i] denotes the leading dimension of the arrays A[j] of i-th group. When side='L' or 'l', lda[i] >= max(1,m[i]), when side='R' or 'r' then lda[i] >= max(1,n[i]).
in	*B*	B is an array of pointers to matrices B[0], B[1],..,B[batch_count-1], where for i-th group each element B[j] is a pointer to a matrix. On entry, the matrices B[j] are of dimension ldb[i]-by-n[i]. On exit, the result of a triangular matrix-matrix multiply (alpha[i]∗op(A[j])∗B[j]) or (alpha[i]∗B[j]∗op(A[j])). batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldb*	An array of integers of length group_count, where ldb[i] is the leading dimension of the arrays B[j] of i-th group. ldb[i] >= max(1,m[i]).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

dtrmm_batch
ctrmm_batch
dtrmm_batch
strmm_batch

3.10.2.3 blas_strmm_batch()

```
void blas_strmm_batch (
            int group_count,
            const int * group_sizes,
            bblas_enum_t layout,
            const bblas_enum_t * side,
            const bblas_enum_t * uplo,
            const bblas_enum_t * transa,
            const bblas_enum_t * diag,
            const int * m,
            const int * n,
            const float * alpha,
            float const *const * A,
            const int * lda,
            float ** B,
            int const * ldb,
            int * info )
```

Performs a triangular batch matrix-matrix multiply of the form

$$B[i] = \alpha[op(A[i]) \times B[i]]$$

, if side = BblasLeft or

$$B[i] = \alpha[B[i] \times op(A[i])]$$

, if side = BblasRight

for a group of matrices, and op(X) is one of:

```
- op(A[i]) = A[i]   or
- op(A[i]) = A[i]^T or
- op(A[i]) = A[i]^T
```

alpha[i]-s are scalars, B[i]-s are m-by-n matrices and A[i]-s are a unit or non-unit, upper or lower triangular matrix.

Parameters

in	*group_count*	The number groups of matrices.
in	*group_sizes*	The number of matrices in each group.

Parameters

in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*side*	An array of length group_count, for matrices of i-th group it specifies whether op(A[j]) appears on the left or on the right of B[j]: • BblasLeft: alpha[i]∗op(A[j])∗B[j] • BblasRight: alpha[i]∗B[j]∗op(A[j])
in	*uplo*	An array of length group_count, where uplo[i] specifies whether the upper or lower triangular part of the symmetric matrices A[j]-s of i-th group are to be referenced

- BblasLower: Only the lower triangular part of the symmetric matrices A[j] is to be referenced.
- BblasUpper: Only the upper triangular part of the symmetric matrices A[j] is to be referenced.

Parameters

in	*transa*	An array of length group_count, where • BblasNoTrans: A[j]-s in i-th group are not transposed, • BblasTrans: A[j]-s in i-th group are transposed, • BblasConjTrans: A[j]-s in i-th group are conjugate transposed.
in	*diag*	An array of length group_count, which specifies whether or not A[j]-s of i-th group are unit triangular: • BblasNonUnit: A[j]-s are non-unit triangular; • BblasUnit: A[j]-s are unit triangular.
in	*m*	An array integers of length group_count, which specified the number of rows of matrices B[j] in i-th group. m[i] >= 0.
in	*n*	An array integers of length group_count, which specified the number of columns of matrices B[j] in i-th group. n[i] >= 0.
in	*alpha*	An array of length group_count, where alpha[i] is a scalar.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[batch_count-1], where for i-th group each element A[j] is a pointer to a triangular matrix of dimension lda[i]-by-k, where k is m[i] when side='L' or 'l' and k is n[i] when when side='R' or 'r'. If uplo = BblasUpper, the leading k-by-k upper triangular part of the array A[j] contains the upper triangular matrix, and the strictly lower triangular part of A[j] is not referenced. If uplo = BblasLower, the leading k-by-k lower triangular part of the array A[j] contains the lower triangular matrix, and the strictly upper triangular part of A[j] is not referenced. If diag = BblasUnit, the diagonal elements of A[j] are also not referenced and are assumed to be 1. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*lda*	An array of integers of length group_count, where lda[i] denotes the leading dimension of the arrays A[j] of i-th group. When side='L' or 'l', lda[i] >= max(1,m[i]), when side='R' or 'r' then lda[i] >= max(1,n[i]).
in	*B*	B is an array of pointers to matrices B[0], B[1],..,B[batch_count-1], where for i-th group each element B[j] is a pointer to a matrix. On entry, the matrices B[j] are of dimension ldb[i]-by-n[i]. On exit, the result of a triangular matrix-matrix multiply (alpha[i]∗op(A[j])∗B[j]) or (alpha[i]∗B[j]∗op(A[j])). batch_count = {i=0}^{group_count-1}group_sizes[i].

Parameters

in	*ldb*	An array of integers of length group_count, where ldb[i] is the leading dimension of the arrays B[j] of i-th group. ldb[i] >= max(1,m[i]).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

strmm_batch
ctrmm_batch
dtrmm_batch
strmm_batch

3.10.2.4 blas_ztrmm_batch()

```
void blas_ztrmm_batch (
            int group_count,
            const int * group_sizes,
            bblas_enum_t layout,
            const bblas_enum_t * side,
            const bblas_enum_t * uplo,
            const bblas_enum_t * transa,
            const bblas_enum_t * diag,
            const int * m,
            const int * n,
            const bblas_complex64_t * alpha,
            bblas_complex64_t const *const * A,
            const int * lda,
            bblas_complex64_t ** B,
            int const * ldb,
            int * info )
```

Performs a triangular batch matrix-matrix multiply of the form

$$B[i] = \alpha[op(A[i]) \times B[i]]$$

, if side = BblasLeft or

$$B[i] = \alpha[B[i] \times op(A[i])]$$

, if side = BblasRight

for a group of matrices, and op(X) is one of:

```
- op(A[i]) = A[i]   or
- op(A[i]) = A[i]^T or
- op(A[i]) = A[i]^H
```

alpha[i]-s are scalars, B[i]-s are m-by-n matrices and A[i]-s are a unit or non-unit, upper or lower triangular matrix.

Parameters

in	*group_count*	The number groups of matrices.
in	*group_sizes*	The number of matrices in each group.
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*side*	An array of length group_count, for matrices of i-th group it specifies whether op(A[j]) appears on the left or on the right of B[j]: • BblasLeft: alpha[i]∗op(A[j])∗B[j] • BblasRight: alpha[i]∗B[j]∗op(A[j])
in	*uplo*	An array of length group_count, where uplo[i] specifies whether the upper or lower triangular part of the symmetric matrices A[j]-s of i-th group are to be referenced

- BblasLower: Only the lower triangular part of the symmetric matrices A[j] is to be referenced.
- BblasUpper: Only the upper triangular part of the symmetric matrices A[j] is to be referenced.

Parameters

in	*transa*	An array of length group_count, where • BblasNoTrans: A[j]-s in i-th group are not transposed, • BblasTrans: A[j]-s in i-th group are transposed, • BblasConjTrans: A[j]-s in i-th group are conjugate transposed.
in	*diag*	An array of length group_count, which specifies whether or not A[j]-s of i-th group are unit triangular: • BblasNonUnit: A[j]-s are non-unit triangular; • BblasUnit: A[j]-s are unit triangular.
in	*m*	An array integers of length group_count, which specified the number of rows of matrices B[j] in i-th group. m[i] >= 0.
in	*n*	An array integers of length group_count, which specified the number of columns of matrices B[j] in i-th group. n[i] >= 0.

Parameters

in	*alpha*	An array of length group_count, where alpha[i] is a scalar.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[batch_count-1], where for i-th group each element A[j] is a pointer to a triangular matrix of dimension lda[i]-by-k, where k is m[i] when side='L' or 'l' and k is n[i] when when side='R' or 'r'. If uplo = BblasUpper, the leading k-by-k upper triangular part of the array A[j] contains the upper triangular matrix, and the strictly lower triangular part of A[j] is not referenced. If uplo = BblasLower, the leading k-by-k lower triangular part of the array A[j] contains the lower triangular matrix, and the strictly upper triangular part of A[j] is not referenced. If diag = BblasUnit, the diagonal elements of A[j] are also not referenced and are assumed to be 1. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*lda*	An array of integers of length group_count, where lda[i] denotes the leading dimension of the arrays A[j] of i-th group. When side='L' or 'l', lda[i] >= max(1,m[i]), when side='R' or 'r' then lda[i] >= max(1,n[i]).
in	*B*	B is an array of pointers to matrices B[0], B[1],..,B[batch_count-1], where for i-th group each element B[j] is a pointer to a matrix. On entry, the matrices B[j] are of dimension ldb[i]-by-n[i]. On exit, the result of a triangular matrix-matrix multiply (alpha[i]*op(A[j])*B[j]) or (alpha[i]*B[j]*op(A[j])). batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldb*	An array of integers of length group_count, where ldb[i] is the leading dimension of the arrays B[j] of i-th group. ldb[i] >= max(1,m[i]).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

ztrmm_batch
ctrmm_batch
dtrmm_batch
strmm_batch

3.11 trsm_batch: Batched triangular solve matrix

$C[i] = op(A[i])^{-1}B[i]$ or $C[i] = B[i]\ op(A[i])^{-1}$ where $A[i]$ are triangular

Functions

- void blas_ctrsm_batch (int group_count, const int *group_sizes, bblas_enum_t layout, const bblas_enum_t *side, const bblas_enum_t *uplo, const bblas_enum_t *transa, const bblas_enum_t *diag, const int *m, const int *n, const bblas_complex32_t *alpha, bblas_complex32_t const *const *A, const int *lda, bblas_complex32_t **B, const int *ldb, int *info)
- void blas_dtrsm_batch (int group_count, const int *group_sizes, bblas_enum_t layout, const bblas_enum_t *side, const bblas_enum_t *uplo, const bblas_enum_t *transa, const bblas_enum_t *diag, const int *m, const int *n, const double *alpha, double const *const *A, const int *lda, double **B, const int *ldb, int *info)
- void blas_strsm_batch (int group_count, const int *group_sizes, bblas_enum_t layout, const bblas_enum_t *side, const bblas_enum_t *uplo, const bblas_enum_t *transa, const bblas_enum_t *diag, const int *m, const int *n, const float *alpha, float const *const *A, const int *lda, float **B, const int *ldb, int *info)
- void blas_ztrsm_batch (int group_count, const int *group_sizes, bblas_enum_t layout, const bblas_enum_t *side, const bblas_enum_t *uplo, const bblas_enum_t *transa, const bblas_enum_t *diag, const int *m, const int *n, const bblas_complex64_t *alpha, bblas_complex64_t const *const *A, const int *lda, bblas_complex64_t **B, const int *ldb, int *info)

3.11.1 Detailed Description

$C[i] = op(A[i])^{-1}B[i]$ or $C[i] = B[i]\ op(A[i])^{-1}$ where $A[i]$ are triangular

3.11.2 Function Documentation

3.11.2.1 blas_ctrsm_batch()

```
void blas_ctrsm_batch (
            int group_count,
            const int * group_sizes,
            bblas_enum_t layout,
            const bblas_enum_t * side,
            const bblas_enum_t * uplo,
            const bblas_enum_t * transa,
            const bblas_enum_t * diag,
            const int * m,
            const int * n,
            const bblas_complex32_t * alpha,
            bblas_complex32_t const *const * A,
            const int * lda,
            bblas_complex32_t ** B,
            const int * ldb,
            int * info )
```

Solves one of the batch matrix equations

$$op(A[i]) \times X[i] = \alpha[i]B[i],$$

or

$$X[i] \times op(A[i]) = \alpha[i]B[i],$$

for a group of matrices, and op(A[i]) is one of:

$$op(A[i]) = A[i],$$
$$op(A[i]) = A[i]^T,$$
$$op(A[i]) = A[i]^H,$$

alpha[i] is a scalar, X[i]-s and B[i]-s are m-by-n matrices, and A[i]-s are unit or non-unit, upper or lower triangular matrices. The matrix X[i] overwrites B[i].

Parameters

in	*group_count*	The number groups of matrices.
in	*group_sizes*	An array of integers of length group_count-1 denoting the number of matrices in each group.
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*side*	An array of length group_count-1, where for matrices of i-th group specifies whether op(A[j]) appears on the left or on the right of X[j]: • BblasLeft: op(A[j])∗X[j] = B[j], • BblasRight: X[j]∗op(A[j]) = B[j].
in	*uplo*	An array of length group_count-1, where for matrices of i-th group specifies whether the matrices A[j]-s are upper triangular or lower triangular: • BblasUpper: Upper triangle of A[j] is stored; • BblasLower: Lower triangle of A[j] is stored.
in	*transa*	An array of length group_count-1, where • BblasNoTrans: A[j]-s in i-th group are not transposed, • BblasTrans: A[j]-s in i-th group are transposed, • BblasConjTrans: A[j]-s in i-th group are conjugate transposed.
in	*diag*	An array of length group_count-1, which specifies whether or not A[j]-s of i-th group are unit triangular: • BblasNonUnit: A[j]-s are non-unit triangular; • BblasUnit: A[j]-s are unit triangular.
in	*m*	An array integers of length group_count-1, which specified the number of rows of matrices B[j] in i-th group. m[i] >= 0.

Parameters

in	*n*	An array integers of length group_count-1, which specified the number of columns of matrices B[j] in i-th group. n[i] >= 0.
in	*alpha*	An array of length group_count-1, where alpha[i] is a scalar.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[batch_count-1], where for i-th group each element A[j] is a pointer to a triangular matrix of dimension lda[i]-by-ka triangular, where ka = m[i] if side = BblasLeft, and ka = n[i] if side = BblasRight. If uplo[i] = BblasUpper, the leading k-by-k upper triangular part of the array A[j] contains the upper triangular matrix, and the strictly lower triangular part of A[j] is not referenced. If uplo[i] = BblasLower, the leading k-by-k lower triangular part of the array A[j] contains the lower triangular matrix, and the strictly upper triangular part of A[j] is not referenced. If diag[i] = BblasUnit, the diagonal elements of A[j] are also not referenced and are assumed to be 1. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*lda*	An arrray of integers of length group_count-1, where lda[i] is the leading dimension of the arrays A[j] in i-th group. lda[i] >= max(1,k[i]).
in,out	*B*	B is an array of pointers to matrices B[0], B[1] .. B[batch_count-1], On entry, for i-th group each B[j]-s are ldb[i]-by-n[i] right hand side matrix. On exit, if return value = 0, the ldb[i]-by-n[i] solution matrix X. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldb*	An array of integers of length group_count-1, where ldb[i] is the leading dimension of the arrays B[j] in i-th group. ldb[i] >= max(1,m[i]).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

ctrsm_batch
ctrsm_batch
dtrsm_batch
strsm_batch

3.11.2.2 blas_dtrsm_batch()

```
void blas_dtrsm_batch (
            int group_count,
```

```
const int * group_sizes,
bblas_enum_t layout,
const bblas_enum_t * side,
const bblas_enum_t * uplo,
const bblas_enum_t * transa,
const bblas_enum_t * diag,
const int * m,
const int * n,
const double * alpha,
double const *const * A,
const int * lda,
double ** B,
const int * ldb,
int * info )
```

Solves one of the batch matrix equations

$$op(A[i]) \times X[i] = \alpha[i]B[i],$$

or

$$X[i] \times op(A[i]) = \alpha[i]B[i],$$

for a group of matrices, and op(A[i]) is one of:

$$op(A[i]) = A[i],$$
$$op(A[i]) = A[i]^T,$$
$$op(A[i]) = A[i]^T,$$

alpha[i] is a scalar, X[i]-s and B[i]-s are m-by-n matrices, and A[i]-s are unit or non-unit, upper or lower triangular matrices. The matrix X[i] overwrites B[i].

Parameters

in	*group_count*	The number groups of matrices.
in	*group_sizes*	An array of integers of length group_count-1 denoting the number of matrices in each group.
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*side*	An array of length group_count-1, where for matrices of i-th group specifies whether op(A[j]) appears on the left or on the right of X[j]: • BblasLeft: op(A[j])∗X[j] = B[j], • BblasRight: X[j]∗op(A[j]) = B[j].
in	*uplo*	An array of length group_count-1, where for matrices of i-th group specifies whether the matrices A[j]-s are upper triangular or lower triangular: • BblasUpper: Upper triangle of A[j] is stored; • BblasLower: Lower triangle of A[j] is stored.

Parameters

in	*transa*	An array of length group_count-1, where • BblasNoTrans: A[j]-s in i-th group are not transposed, • BblasTrans: A[j]-s in i-th group are transposed, • BblasConjTrans: A[j]-s in i-th group are conjugate transposed.
in	*diag*	An array of length group_count-1, which specifies whether or not A[j]-s of i-th group are unit triangular: • BblasNonUnit: A[j]-s are non-unit triangular; • BblasUnit: A[j]-s are unit triangular.
in	*m*	An array integers of length group_count-1, which specified the number of rows of matrices B[j] in i-th group. m[i] >= 0.
in	*n*	An array integers of length group_count-1, which specified the number of columns of matrices B[j] in i-th group. n[i] >= 0.
in	*alpha*	An array of length group_count-1, where alpha[i] is a scalar.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[batch_count-1], where for i-th group each element A[j] is a pointer to a triangular matrix of dimension lda[i]-by-ka triangular, where ka = m[i] if side = BblasLeft, and ka = n[i] if side = BblasRight. If uplo[i] = BblasUpper, the leading k-by-k upper triangular part of the array A[j] contains the upper triangular matrix, and the strictly lower triangular part of A[j] is not referenced. If uplo[i] = BblasLower, the leading k-by-k lower triangular part of the array A[j] contains the lower triangular matrix, and the strictly upper triangular part of A[j] is not referenced. If diag[i] = BblasUnit, the diagonal elements of A[j] are also not referenced and are assumed to be 1. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*lda*	An arrray of integers of length group_count-1, where lda[i] is the leading dimension of the arrays A[j] in i-th group. lda[i] >= max(1,k[i]).
in,out	*B*	B is an array of pointers to matrices B[0], B[1] .. B[batch_count-1], On entry, for i-th group each B[j]-s are ldb[i]-by-n[i] right hand side matrix. On exit, if return value = 0, the ldb[i]-by-n[i] solution matrix X. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldb*	An array of integers of length group_count-1, where ldb[i] is the leading dimension of the arrays B[j] in i-th group. ldb[i] >= max(1,m[i]).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

dtrsm_batch
ctrsm_batch
dtrsm_batch
strsm_batch

3.11.2.3 blas_strsm_batch()

```
void blas_strsm_batch (
            int group_count,
            const int * group_sizes,
            bblas_enum_t layout,
            const bblas_enum_t * side,
            const bblas_enum_t * uplo,
            const bblas_enum_t * transa,
            const bblas_enum_t * diag,
            const int * m,
            const int * n,
            const float * alpha,
            float const *const * A,
            const int * lda,
            float ** B,
            const int * ldb,
            int * info )
```

Solves one of the batch matrix equations

$$op(A[i]) \times X[i] = \alpha[i]B[i],$$

or

$$X[i] \times op(A[i]) = \alpha[i]B[i],$$

for a group of matrices, and op(A[i]) is one of:

$$op(A[i]) = A[i],$$
$$op(A[i]) = A[i]^T,$$
$$op(A[i]) = A[i]^T,$$

alpha[i] is a scalar, X[i]-s and B[i]-s are m-by-n matrices, and A[i]-s are unit or non-unit, upper or lower triangular matrices. The matrix X[i] overwrites B[i].

Parameters

in	*group_count*	The number groups of matrices.
in	*group_sizes*	An array of integers of length group_count-1 denoting the number of matrices in each group.

Parameters

in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*side*	An array of length group_count-1, where for matrices of i-th group specifies whether op(A[j]) appears on the left or on the right of X[j]: • BblasLeft: op(A[j])∗X[j] = B[j], • BblasRight: X[j]∗op(A[j]) = B[j].
in	*uplo*	An array of length group_count-1, where for matrices of i-th group specifies whether the matrices A[j]-s are upper triangular or lower triangular: • BblasUpper: Upper triangle of A[j] is stored; • BblasLower: Lower triangle of A[j] is stored.
in	*transa*	An array of length group_count-1, where • BblasNoTrans: A[j]-s in i-th group are not transposed, • BblasTrans: A[j]-s in i-th group are transposed, • BblasConjTrans: A[j]-s in i-th group are conjugate transposed.
in	*diag*	An array of length group_count-1, which specifies whether or not A[j]-s of i-th group are unit triangular: • BblasNonUnit: A[j]-s are non-unit triangular; • BblasUnit: A[j]-s are unit triangular.
in	*m*	An array integers of length group_count-1, which specified the number of rows of matrices B[j] in i-th group. m[i] >= 0.
in	*n*	An array integers of length group_count-1, which specified the number of columns of matrices B[j] in i-th group. n[i] >= 0.
in	*alpha*	An array of length group_count-1, where alpha[i] is a scalar.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[batch_count-1], where for i-th group each element A[j] is a pointer to a triangular matrix of dimension lda[i]-by-ka triangular, where ka = m[i] if side = BblasLeft, and ka = n[i] if side = BblasRight. If uplo[i] = BblasUpper, the leading k-by-k upper triangular part of the array A[j] contains the upper triangular matrix, and the strictly lower triangular part of A[j] is not referenced. If uplo[i] = BblasLower, the leading k-by-k lower triangular part of the array A[j] contains the lower triangular matrix, and the strictly upper triangular part of A[j] is not referenced. If diag[i] = BblasUnit, the diagonal elements of A[j] are also not referenced and are assumed to be 1. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*lda*	An arrray of integers of length group_count-1, where lda[i] is the leading dimension of the arrays A[j] in i-th group. lda[i] >= max(1,k[i]).
in,out	*B*	B is an array of pointers to matrices B[0], B[1] .. B[batch_count-1], On entry, for i-th group each B[j]-s are ldb[i]-by-n[i] right hand side matrix. On exit, if return value = 0, the ldb[i]-by-n[i] solution matrix X. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldb*	An array of integers of length group_count-1, where ldb[i] is the leading dimension of the arrays B[j] in i-th group. ldb[i] >= max(1,m[i]).

Parameters

in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

strsm_batch
ctrsm_batch
dtrsm_batch
strsm_batch

3.11.2.4 blas_ztrsm_batch()

```
void blas_ztrsm_batch (
            int group_count,
            const int * group_sizes,
            bblas_enum_t layout,
            const bblas_enum_t * side,
            const bblas_enum_t * uplo,
            const bblas_enum_t * transa,
            const bblas_enum_t * diag,
            const int * m,
            const int * n,
            const bblas_complex64_t * alpha,
            bblas_complex64_t const *const * A,
            const int * lda,
            bblas_complex64_t ** B,
            const int * ldb,
            int * info )
```

Solves one of the batch matrix equations

$$op(A[i]) \times X[i] = \alpha[i]B[i],$$

or

$$X[i] \times op(A[i]) = \alpha[i]B[i],$$

for a group of matrices, and op(A[i]) is one of:

$$op(A[i]) = A[i],$$
$$op(A[i]) = A[i]^T,$$
$$op(A[i]) = A[i]^H,$$

alpha[i] is a scalar, X[i]-s and B[i]-s are m-by-n matrices, and A[i]-s are unit or non-unit, upper or lower triangular matrices. The matrix X[i] overwrites B[i].

Parameters

in	*group_count*	The number groups of matrices.
in	*group_sizes*	An array of integers of length group_count-1 denoting the number of matrices in each group.
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*side*	An array of length group_count-1, where for matrices of i-th group specifies whether op(A[j]) appears on the left or on the right of X[j]: • BblasLeft: op(A[j])∗X[j] = B[j], • BblasRight: X[j]∗op(A[j]) = B[j].
in	*uplo*	An array of length group_count-1, where for matrices of i-th group specifies whether the matrices A[j]-s are upper triangular or lower triangular: • BblasUpper: Upper triangle of A[j] is stored; • BblasLower: Lower triangle of A[j] is stored.
in	*transa*	An array of length group_count-1, where • BblasNoTrans: A[j]-s in i-th group are not transposed, • BblasTrans: A[j]-s in i-th group are transposed, • BblasConjTrans: A[j]-s in i-th group are conjugate transposed.
in	*diag*	An array of length group_count-1, which specifies whether or not A[j]-s of i-th group are unit triangular: • BblasNonUnit: A[j]-s are non-unit triangular; • BblasUnit: A[j]-s are unit triangular.
in	*m*	An array integers of length group_count-1, which specified the number of rows of matrices B[j] in i-th group. m[i] >= 0.
in	*n*	An array integers of length group_count-1, which specified the number of columns of matrices B[j] in i-th group. n[i] >= 0.
in	*alpha*	An array of length group_count-1, where alpha[i] is a scalar.

Parameters

in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[batch_count-1], where for i-th group each element A[j] is a pointer to a triangular matrix of dimension lda[i]-by-ka triangular, where ka = m[i] if side = BblasLeft, and ka = n[i] if side = BblasRight. If uplo[i] = BblasUpper, the leading k-by-k upper triangular part of the array A[j] contains the upper triangular matrix, and the strictly lower triangular part of A[j] is not referenced. If uplo[i] = BblasLower, the leading k-by-k lower triangular part of the array A[j] contains the lower triangular matrix, and the strictly upper triangular part of A[j] is not referenced. If diag[i] = BblasUnit, the diagonal elements of A[j] are also not referenced and are assumed to be 1. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*lda*	An arrray of integers of length group_count-1, where lda[i] is the leading dimension of the arrays A[j] in i-th group. lda[i] >= max(1,k[i]).
in,out	*B*	B is an array of pointers to matrices B[0], B[1] .. B[batch_count-1], On entry, for i-th group each B[j]-s are ldb[i]-by-n[i] right hand side matrix. On exit, if return value = 0, the ldb[i]-by-n[i] solution matrix X. batch_count = {i=0}^{group_count-1}group_sizes[i].
in	*ldb*	An array of integers of length group_count-1, where ldb[i] is the leading dimension of the arrays B[j] in i-th group. ldb[i] >= max(1,m[i]).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

ztrsm_batch
ctrsm_batch
dtrsm_batch
strsm_batch

3.12 Fixed Batched BLAS API

Fixed Batched BLAS functions.

Modules

- : Fixed Batched matrix-matrix operations,

  Fixed Batched matrix-matrix operations that perform on problems of same size.

3.12.1 Detailed Description

Fixed Batched BLAS functions.

3.13 : Fixed Batched matrix-matrix operations,

Fixed Batched matrix-matrix operations that perform on problems of same size.

Modules

- gemm_batchf: Batch of same size general matrix multiply: C[i] = A[i]B[i] + C[i]
  $C[i] = \alpha[i]\ op(A[i])\ op(B[i]) + \beta[i]C[i]$
- hemm_batchf: Batch of same size hermitian matrix multiply
  $C[i] = \alpha[i]A[i]B[i] + \beta[i]C[i]$ *or* $C[i] = \alpha[i]B[i]A[i] + \beta C[i]$ *where* $A[i]$ *are hermitian*
- herk_batchf: Batch of same size hermitian rank k update
  $C[i] = \alpha[i]A[i]A[i]^T + \beta[i]C[i]$ *where* $C[i]$ *are hermitian*
- her2k_batchf: Batch of same size hermitian rank 2k update
  $C[i] = \alpha[i]A[i]B[i]^T + \alpha[i]B[i]A[i]^T + \beta[i]C[i]$ *where* $C[i]$ *are Hermitian*
- symm_batchf: Batch of same size symmetric matrix multiply
  $C[i] = \alpha[i]A[i]B[i] + \beta[i]C[i]$ *or* $C[i] = \alpha[i]B[i]A[i] + \beta[i]C[i]$ *where* $A[i]$ *are symmetric*
- syrk_batchf: Batch of same size symmetric rank k update
  $C[i] = \alpha[i]A[i]A[i]^T + \beta[i]C[i]$ *where* $C[i]$ *are symmetric*
- syr2k_batchf: Batch of same size symmetric rank 2k update
  $C[i] = \alpha[i]A[i]B[i]^T + \alpha[i]B[i]A[i]^T + \beta[i]C[i]$ *where* $C[i]$ *are symmetric*
- trmm_batchf: Batch of same size triangular matrix multiply
  $B[i] = \alpha[i]\ op(A[i])\ B[i]$ *or* $B[i] = \alpha[i]B[i]\ op(A[i])$ *where* $A[i]$ *are triangular*
- trsm_batchf: Batch of same size triangular solve matrix
  $C[i] = op(A[i])^{-1}B[i]$ *or* $C[i] = B[i]\ op(A[i])^{-1}$ *where* $A[i]$ *are triangular*

3.13.1 Detailed Description

Fixed Batched matrix-matrix operations that perform on problems of same size.

3.14 gemm_batchf: Batch of same size general matrix multiply: C[i] = A[i]B[i] + C[i]

$C[i] = \alpha[i]\ op(A[i])\ op(B[i]) + \beta[i]C[i]$

Functions

- void blas_cgemm_batchf (int group_size, bblas_enum_t layout, bblas_enum_t transa, bblas_enum_t transb, int m, int n, int k, bblas_complex32_t alpha, bblas_complex32_t const ∗const ∗A, int lda, bblas_complex32_t const ∗const ∗B, int ldb, bblas_complex32_t beta, bblas_complex32_t ∗∗C, int ldc, int ∗info)
- void blas_dgemm_batchf (int group_size, bblas_enum_t layout, bblas_enum_t transa, bblas_enum_t transb, int m, int n, int k, double alpha, double const ∗const ∗A, int lda, double const ∗const ∗B, int ldb, double beta, double ∗∗C, int ldc, int ∗info)
- void blas_sgemm_batchf (int group_size, bblas_enum_t layout, bblas_enum_t transa, bblas_enum_t transb, int m, int n, int k, float alpha, float const ∗const ∗A, int lda, float const ∗const ∗B, int ldb, float beta, float ∗∗C, int ldc, int ∗info)
- void blas_zgemm_batchf (int group_size, bblas_enum_t layout, bblas_enum_t transa, bblas_enum_t transb, int m, int n, int k, bblas_complex64_t alpha, bblas_complex64_t const ∗const ∗A, int lda, bblas_complex64_t const ∗const ∗B, int ldb, bblas_complex64_t beta, bblas_complex64_t ∗∗C, int ldc, int ∗info)

3.14.1 Detailed Description

$C[i] = \alpha[i]\ op(A[i])\ op(B[i]) + \beta[i]C[i]$

3.14.2 Function Documentation

3.14.2.1 blas_cgemm_batchf()

```
void blas_cgemm_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t transa,
            bblas_enum_t transb,
            int m,
            int n,
            int k,
            bblas_complex32_t alpha,
            bblas_complex32_t const *const * A,
            int lda,
            bblas_complex32_t const *const * B,
            int ldb,
            bblas_complex32_t beta,
            bblas_complex32_t ** C,
            int ldc,
            int * info )
```

cgemm_batchf is a batch version of cgemm. It performs matrix-matrix multiplication of matrices, where all the matrices of the batch have a fixed size.

$$C[i] = \alpha[op(A[i]) \times op(B[i])] + \beta C[i],$$

where op(X) is one of:

$$op(X) = X,$$
$$op(X) = X^T,$$
$$op(X) = X^H,$$

alpha and beta are scalars, and A[i], B[i] and C[i] are matrices, with op(A[i]) an m-by-k matrix, op(B[i]) a k-by-n matrix and C[i] an m-by-n matrix.

i

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*transa*	• BblasNoTrans: A[i] is not transposed, • BblasTrans: A[i] is transposed, • BblasConjTrans: A[i] is conjugate transposed.
in	*transb*	• BblasNoTrans: B[i] is not transposed, • BblasTrans: B[i] is transposed, • BblasConjTrans: B[i] is conjugate transposed.
in	*m*	The number of rows of the matrix op(A[i]) and of the matrix C[i]. m >= 0.
in	*n*	The number of columns of the matrix op(B[i]) and of the matrix C[i]. n >= 0.
in	*k*	The number of columns of the matrix op(A[i]) and the number of rows of the matrix op(B[i]). k >= 0.
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a matrix of dimension lda-by-ka, where ka is k when transa = BblasNoTrans, and is m otherwise. When using transa = BblasNoTrans the leading m-by-k part of A[i] must contain the matrix elements, otherwise the leading k-by-m part of A[i] must contain the matrix elements.
in	*lda*	The leading dimension of the array A[i]. When transa = BblasNoTrans, lda >= max(1,m), otherwise, lda >= max(1,k).
in	*B*	B is an array of pointers to matrices B[0], B[1],..,B[group_size-1], where each element B[i] is a pointer to a matrix of dimension lda-by-kb, where kb is n when transb = BblasNoTrans, and is k otherwise. When using transb = BblasNoTrans the leading k-by-n part of B[i] must contain the matrix elements, otherwise the leading n-by-k part of B[i] must contain the matrix elements.
in	*ldb*	The leading dimension of the array B[i]. When transb = BblasNoTrans, ldb >= max(1,k), otherwise, ldb >= max(1,n).

Parameters

in	*beta*	The scalar beta.
in,out	*C*	C is an array of pointers to matrices C[0], C[1],...,C[group_size-1], where each element of C[i] is a pointer to a matrix of dimension ldc-by-n. On exit, each array C[i] is overwritten by the m-by-n matrix (alpha∗op(A[i])∗op(B[i]) + beta∗C[i]).
in	*ldc*	The leading dimension of the array C[i]. ldc >= max(1,m).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

cgemm_batchf
cgemm_batchf
dgemm_batchf
sgemm_batchf

3.14.2.2 blas_dgemm_batchf()

```
void blas_dgemm_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t transa,
            bblas_enum_t transb,
            int m,
            int n,
            int k,
            double alpha,
            double const *const * A,
            int lda,
            double const *const * B,
            int ldb,
            double beta,
            double ** C,
            int ldc,
            int * info )
```

dgemm_batchf is a batch version of dgemm. It performs matrix-matrix multiplication of matrices, where all the matrices of the batch have a fixed size.

$$C[i] = \alpha[op(A[i]) \times op(B[i])] + \beta C[i],$$

where op(X) is one of:

$$op(X) = X,$$
$$op(X) = X^T,$$
$$op(X) = X^T,$$

alpha and beta are scalars, and A[i], B[i] and C[i] are matrices, with op(A[i]) an m-by-k matrix, op(B[i]) a k-by-n matrix and C[i] an m-by-n matrix.

i

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*transa*	• BblasNoTrans: A[i] is not transposed, • BblasTrans: A[i] is transposed, • BblasConjTrans: A[i] is conjugate transposed.
in	*transb*	• BblasNoTrans: B[i] is not transposed, • BblasTrans: B[i] is transposed, • BblasConjTrans: B[i] is conjugate transposed.
in	*m*	The number of rows of the matrix op(A[i]) and of the matrix C[i]. m >= 0.
in	*n*	The number of columns of the matrix op(B[i]) and of the matrix C[i]. n >= 0.
in	*k*	The number of columns of the matrix op(A[i]) and the number of rows of the matrix op(B[i]). k >= 0.
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a matrix of dimension lda-by-ka, where ka is k when transa = BblasNoTrans, and is m otherwise. When using transa = BblasNoTrans the leading m-by-k part of A[i] must contain the matrix elements, otherwise the leading k-by-m part of A[i] must contain the matrix elements.
in	*lda*	The leading dimension of the array A[i]. When transa = BblasNoTrans, lda >= max(1,m), otherwise, lda >= max(1,k).
in	*B*	B is an array of pointers to matrices B[0], B[1],..,B[group_size-1], where each element B[i] is a pointer to a matrix of dimension lda-by-kb, where kb is n when transb = BblasNoTrans, and is k otherwise. When using transb = BblasNoTrans the leading k-by-n part of B[i] must contain the matrix elements, otherwise the leading n-by-k part of B[i] must contain the matrix elements.

Parameters

in	*ldb*	The leading dimension of the array B[i]. When transb = BblasNoTrans, ldb >= max(1,k), otherwise, ldb >= max(1,n).
in	*beta*	The scalar beta.
in,out	*C*	C is an array of pointers to matrices C[0], C[1],...,C[group_size-1], where each element of C[i] is a pointer to a matrix of dimension ldc-by-n. On exit, each array C[i] is overwritten by the m-by-n matrix (alpha∗op(A[i])∗op(B[i]) + beta∗C[i]).
in	*ldc*	The leading dimension of the array C[i]. ldc >= max(1,m).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

dgemm_batchf
cgemm_batchf
dgemm_batchf
sgemm_batchf

3.14.2.3 blas_sgemm_batchf()

```
void blas_sgemm_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t transa,
            bblas_enum_t transb,
            int m,
            int n,
            int k,
            float alpha,
            float const *const * A,
            int lda,
            float const *const * B,
            int ldb,
            float beta,
            float ** C,
```

```
int ldc,
int * info )
```

sgemm_batchf is a batch version of sgemm. It performs matrix-matrix multiplication of matrices, where all the matrices of the batch have a fixed size.

$$C[i] = \alpha[op(A[i]) \times op(B[i])] + \beta C[i],$$

where op(X) is one of:

$$op(X) = X,$$
$$op(X) = X^T,$$
$$op(X) = X^T,$$

alpha and beta are scalars, and A[i], B[i] and C[i] are matrices, with op(A[i]) an m-by-k matrix, op(B[i]) a k-by-n matrix and C[i] an m-by-n matrix.

i

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*transa*	• BblasNoTrans: A[i] is not transposed, • BblasTrans: A[i] is transposed, • BblasConjTrans: A[i] is conjugate transposed.
in	*transb*	• BblasNoTrans: B[i] is not transposed, • BblasTrans: B[i] is transposed, • BblasConjTrans: B[i] is conjugate transposed.
in	*m*	The number of rows of the matrix op(A[i]) and of the matrix C[i]. m >= 0.
in	*n*	The number of columns of the matrix op(B[i]) and of the matrix C[i]. n >= 0.
in	*k*	The number of columns of the matrix op(A[i]) and the number of rows of the matrix op(B[i]). k >= 0.
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a matrix of dimension lda-by-ka, where ka is k when transa = BblasNoTrans, and is m otherwise. When using transa = BblasNoTrans the leading m-by-k part of A[i] must contain the matrix elements, otherwise the leading k-by-m part of A[i] must contain the matrix elements.
in	*lda*	The leading dimension of the array A[i]. When transa = BblasNoTrans, lda >= max(1,m), otherwise, lda >= max(1,k).

Parameters

in	*B*	B is an array of pointers to matrices B[0], B[1],..,B[group_size-1], where each element B[i] is a pointer to a matrix of dimension lda-by-kb, where kb is n when transb = BblasNoTrans, and is k otherwise. When using transb = BblasNoTrans the leading k-by-n part of B[i] must contain the matrix elements, otherwise the leading n-by-k part of B[i] must contain the matrix elements.
in	*ldb*	The leading dimension of the array B[i]. When transb = BblasNoTrans, ldb >= max(1,k), otherwise, ldb >= max(1,n).
in	*beta*	The scalar beta.
in,out	*C*	C is an array of pointers to matrices C[0], C[1],...,C[group_size-1], where each element of C[i] is a pointer to a matrix of dimension ldc-by-n. On exit, each array C[i] is overwritten by the m-by-n matrix (alpha∗op(A[i])∗op(B[i]) + beta∗C[i]).
in	*ldc*	The leading dimension of the array C[i]. ldc >= max(1,m).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

sgemm_batchf
cgemm_batchf
dgemm_batchf
sgemm_batchf

3.14.2.4 blas_zgemm_batchf()

```
void blas_zgemm_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t transa,
            bblas_enum_t transb,
            int m,
            int n,
            int k,
            bblas_complex64_t alpha,
            bblas_complex64_t const *const * A,
```

```
int lda,
bblas_complex64_t const *const * B,
int ldb,
bblas_complex64_t beta,
bblas_complex64_t ** C,
int ldc,
int * info )
```

zgemm_batchf is a batch version of zgemm. It performs matrix-matrix multiplication of matrices, where all the matrices of the batch have a fixed size.

$$C[i] = \alpha[op(A[i]) \times op(B[i])] + \beta C[i],$$

where op(X) is one of:

$$op(X) = X,$$
$$op(X) = X^T,$$
$$op(X) = X^H,$$

alpha and beta are scalars, and A[i], B[i] and C[i] are matrices, with op(A[i]) an m-by-k matrix, op(B[i]) a k-by-n matrix and C[i] an m-by-n matrix.

i

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*transa*	• BblasNoTrans: A[i] is not transposed, • BblasTrans: A[i] is transposed, • BblasConjTrans: A[i] is conjugate transposed.
in	*transb*	• BblasNoTrans: B[i] is not transposed, • BblasTrans: B[i] is transposed, • BblasConjTrans: B[i] is conjugate transposed.
in	*m*	The number of rows of the matrix op(A[i]) and of the matrix C[i]. m >= 0.
in	*n*	The number of columns of the matrix op(B[i]) and of the matrix C[i]. n >= 0.
in	*k*	The number of columns of the matrix op(A[i]) and the number of rows of the matrix op(B[i]). k >= 0.
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a matrix of dimension lda-by-ka, where ka is k when transa = BblasNoTrans, and is m otherwise. When using transa = BblasNoTrans the leading m-by-k part of A[i] must contain the matrix elements, otherwise the leading k-by-m part of A[i] must contain the matrix elements.

Parameters

in	*lda*	The leading dimension of the array A[i]. When transa = BblasNoTrans, lda >= max(1,m), otherwise, lda >= max(1,k).
in	*B*	B is an array of pointers to matrices B[0], B[1],..,B[group_size-1], where each element B[i] is a pointer to a matrix of dimension lda-by-kb, where kb is n when transb = BblasNoTrans, and is k otherwise. When using transb = BblasNoTrans the leading k-by-n part of B[i] must contain the matrix elements, otherwise the leading n-by-k part of B[i] must contain the matrix elements.
in	*ldb*	The leading dimension of the array B[i]. When transb = BblasNoTrans, ldb >= max(1,k), otherwise, ldb >= max(1,n).
in	*beta*	The scalar beta.
in,out	*C*	C is an array of pointers to matrices C[0], C[1],...,C[group_size-1], where each element of C[i] is a pointer to a matrix of dimension ldc-by-n. On exit, each array C[i] is overwritten by the m-by-n matrix (alpha∗op(A[i])∗op(B[i]) + beta∗C[i]).
in	*ldc*	The leading dimension of the array C[i]. ldc >= max(1,m).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

zgemm_batchf
cgemm_batchf
dgemm_batchf
sgemm_batchf

3.15 hemm_batchf: Batch of same size hermitian matrix multiply

$C[i] = \alpha[i]A[i]B[i] + \beta[i]C[i]$ or $C[i] = \alpha[i]B[i]A[i] + \beta C[i]$ where $A[i]$ are hermitian

Functions

- void blas_chemm_batchf (int group_size, bblas_enum_t layout, bblas_enum_t side, bblas_enum_t uplo, int m, int n, bblas_complex32_t alpha, bblas_complex32_t const *const *A, int lda, bblas_complex32_t const *const *B, int ldb, bblas_complex32_t beta, bblas_complex32_t **C, int ldc, int *info)
- void blas_zhemm_batchf (int group_size, bblas_enum_t layout, bblas_enum_t side, bblas_enum_t uplo, int m, int n, bblas_complex64_t alpha, bblas_complex64_t const *const *A, int lda, bblas_complex64_t const *const *B, int ldb, bblas_complex64_t beta, bblas_complex64_t **C, int ldc, int *info)

3.15.1 Detailed Description

$C[i] = \alpha[i]A[i]B[i] + \beta[i]C[i]$ or $C[i] = \alpha[i]B[i]A[i] + \beta C[i]$ where $A[i]$ are hermitian

3.15.2 Function Documentation

3.15.2.1 blas_chemm_batchf()

```
void blas_chemm_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t side,
            bblas_enum_t uplo,
            int m,
            int n,
            bblas_complex32_t alpha,
            bblas_complex32_t const *const * A,
            int lda,
            bblas_complex32_t const *const * B,
            int ldb,
            bblas_complex32_t beta,
            bblas_complex32_t ** C,
            int ldc,
            int * info )
```

Performs one of the batch matrix-matrix operations

$$C[i] = \alpha \times A[i] \times B[i] + \beta \times C[i]$$

or

$$C[i] = \alpha \times B[i] \times A[i] + \beta \times C[i]$$

where alpha and beta are scalars, A[i]-s are Hermitian matrices B[i]-s and C[i]-s are m-by-n matrices.

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*side*	Specifies whether the Hermitian matrices A[i] appear on the left or right in the operation as follows: • BblasLeft: $C[i] = \alpha \times A[i] \times B[i] + \beta \times C[i]$ • BblasRight: $C[i] = \alpha \times B[i] \times A[i] + \beta \times C[i]$
in	*uplo*	Specifies whether the upper or lower triangular part of the Hermitian matrices A[i] are to be referenced as follows: • BblasLower: Only the lower triangular part of the Hermitian matrices A[i] is to be referenced. • BblasUpper: Only the upper triangular part of the Hermitian matrices A[i] is to be referenced.
in	*m*	The number of rows in the matrices C[i]. m >= 0.
in	*n*	The number of columns in the matrices C[i]. n >= 0.
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a matrix A[i] of size lda-by-ka, where ka is m when side = BblasLeft, and is n otherwise. Only the uplo triangular part is referenced.
in	*lda*	The leading dimension of the arrays A[i]. lda >= max(1,ka).
in	*B*	B is an array of pointers to matrices B[0], B[1] .. B[group_size-1], where each element B[i] is a pointer to a matrix B[i] of size ldb-by-n matrix, where the leading m-by-n part of the array B[i] must contain the matrix B[i].
in	*ldb*	The leading dimension of the arrays B[i]. ldb >= max(1,m).
in	*beta*	The scalar beta.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[group_size-1], where each element C[i] is a pointer to a matrix C[i]. On exit, the array is overwritten by the m-by-n updated matrix.
in	*ldc*	The leading dimension of the arrays C[i]. ldc >= max(1,m).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

chemm_batchf
chemm_batchf

3.15.2.2 blas_zhemm_batchf()

```
void blas_zhemm_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t side,
            bblas_enum_t uplo,
            int m,
            int n,
            bblas_complex64_t alpha,
            bblas_complex64_t const *const * A,
            int lda,
            bblas_complex64_t const *const * B,
            int ldb,
            bblas_complex64_t beta,
            bblas_complex64_t ** C,
            int ldc,
            int * info )
```

Performs one of the batch matrix-matrix operations

$$C[i] = \alpha \times A[i] \times B[i] + \beta \times C[i]$$

or

$$C[i] = \alpha \times B[i] \times A[i] + \beta \times C[i]$$

where alpha and beta are scalars, A[i]-s are Hermitian matrices B[i]-s and C[i]-s are m-by-n matrices.

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*side*	Specifies whether the Hermitian matrices A[i] appear on the left or right in the operation as follows: • BblasLeft: $C[i] = \alpha \times A[i] \times B[i] + \beta \times C[i]$ • BblasRight: $C[i] = \alpha \times B[i] \times A[i] + \beta \times C[i]$

Parameters

in	*uplo*	Specifies whether the upper or lower triangular part of the Hermitian matrices A[i] are to be referenced as follows: • BblasLower: Only the lower triangular part of the Hermitian matrices A[i] is to be referenced. • BblasUpper: Only the upper triangular part of the Hermitian matrices A[i] is to be referenced.
in	*m*	The number of rows in the matrices C[i]. m >= 0.
in	*n*	The number of columns in the matrices C[i]. n >= 0.
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a matrix A[i] of size lda-by-ka, where ka is m when side = BblasLeft, and is n otherwise. Only the uplo triangular part is referenced.
in	*lda*	The leading dimension of the arrays A[i]. lda >= max(1,ka).
in	*B*	B is an array of pointers to matrices B[0], B[1] .. B[group_size-1], where each element B[i] is a pointer to a matrix B[i] of size ldb-by-n matrix, where the leading m-by-n part of the array B[i] must contain the matrix B[i].
in	*ldb*	The leading dimension of the arrays B[i]. ldb >= max(1,m).
in	*beta*	The scalar beta.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[group_size-1], where each element C[i] is a pointer to a matrix C[i]. On exit, the array is overwritten by the m-by-n updated matrix.
in	*ldc*	The leading dimension of the arrays C[i]. ldc >= max(1,m).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

zhemm_batchf
chemm_batchf

3.16 herk_batchf: Batch of same size hermitian rank k update

$C[i] = \alpha[i]A[i]A[i]^T + \beta[i]C[i]$ where $C[i]$ are hermitian

Functions

- void blas_cherk_batchf (int group_size, bblas_enum_t layout, bblas_enum_t uplo, bblas_enum_t trans, int n, int k, const float alpha, bblas_complex32_t const *const *A, int lda, const float beta, bblas_complex32_t **C, int ldc, int *info)
- void blas_zherk_batchf (int group_size, bblas_enum_t layout, bblas_enum_t uplo, bblas_enum_t trans, int n, int k, const double alpha, bblas_complex64_t const *const *A, int lda, const double beta, bblas_complex64_t **C, int ldc, int *info)

3.16.1 Detailed Description

$C[i] = \alpha[i]A[i]A[i]^T + \beta[i]C[i]$ where $C[i]$ are hermitian

3.16.2 Function Documentation

3.16.2.1 blas_cherk_batchf()

```
void blas_cherk_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t uplo,
            bblas_enum_t trans,
            int n,
            int k,
            const float alpha,
            bblas_complex32_t const *const * A,
            int lda,
            const float beta,
            bblas_complex32_t ** C,
            int ldc,
            int * info )
```

Performs one of the batch Hermitian rank k operations

$$C[i] = \alpha A[i] \times A[i]^H + \beta C[i],$$

or

$$C[i] = \alpha A[i]^H \times A[i] + \beta C[i],$$

where alpha and beta are real scalars, C[i]-s are n-by-n Hermitian matrices, and A[i]-s are n-by-k matrices in the first case and k-by-n matrices in the second case.

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*uplo*	• BblasUpper: Upper triangle of C[i]-s are stored; • BblasLower: Lower triangle of C[i]-s are stored.
in	*trans*	• BblasNoTrans: $C[i] = \alpha A[i] \times A[i]^H + \beta C[i].$ • BblasConjTrans: $C[i] = \alpha A[i]^H \times A[i] + \beta C[i].$
in	*n*	The order of the matrices C[i]. n >= 0.
in	*k*	If trans = BblasNoTrans, number of columns of the matrices A[i]; if trans = BblasConjTrans, number of rows of the matrices A[i].
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a matrix A[i] of size lda-by-ka. If trans = BblasNoTrans, ka = k; if trans = BblasConjTrans, ka = n.
in	*lda*	The leading dimension of the arrays A[i]. If trans = BblasNoTrans, lda >= max(1, n); if trans = BblasConjTrans, lda >= max(1, k).
in	*beta*	The scalar beta.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[group_size-1], where each element C[i] is a pointer to a matrix C[i] of size ldc-by-n. On exit, the uplo part of the matrix is overwritten by the uplo part of the updated matrix.
in	*ldc*	The leading dimension of the arrays C[i]. ldc >= max(1, n).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

cherk_batchf
cherk_batchf

3.16.2.2 blas_zherk_batchf()

```
void blas_zherk_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t uplo,
            bblas_enum_t trans,
            int n,
            int k,
            const double alpha,
            bblas_complex64_t const *const * A,
            int lda,
            const double beta,
            bblas_complex64_t ** C,
            int ldc,
            int * info )
```

Performs one of the batch Hermitian rank k operations

$$C[i] = \alpha A[i] \times A[i]^H + \beta C[i],$$

or

$$C[i] = \alpha A[i]^H \times A[i] + \beta C[i],$$

where alpha and beta are real scalars, C[i]-s are n-by-n Hermitian matrices, and A[i]-s are n-by-k matrices in the first case and k-by-n matrices in the second case.

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*uplo*	• BblasUpper: Upper triangle of C[i]-s are stored; • BblasLower: Lower triangle of C[i]-s are stored.
in	*trans*	• BblasNoTrans: $C[i] = \alpha A[i] \times A[i]^H + \beta C[i].$ • BblasConjTrans: $C[i] = \alpha A[i]^H \times A[i] + \beta C[i].$
in	*n*	The order of the matrices C[i]. n >= 0.
in	*k*	If trans = BblasNoTrans, number of columns of the matrices A[i]; if trans = BblasConjTrans, number of rows of the matrices A[i].
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a matrix A[i] of size lda-by-ka. If trans = BblasNoTrans, ka = k; if trans = BblasConjTrans, ka = n.

Parameters

in	*lda*	The leading dimension of the arrays A[i]. If trans = BblasNoTrans, lda >= max(1, n); if trans = BblasConjTrans, lda >= max(1, k).
in	*beta*	The scalar beta.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[group_size-1], where each element C[i] is a pointer to a matrix C[i] of size ldc-by-n. On exit, the uplo part of the matrix is overwritten by the uplo part of the updated matrix.
in	*ldc*	The leading dimension of the arrays C[i]. ldc >= max(1, n).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

zherk_batchf
cherk_batchf

3.17 her2k_batchf: Batch of same size hermitian rank 2k update

$C[i] = \alpha[i]A[i]B[i]^T + \alpha[i]B[i]A[i]^T + \beta[i]C[i]$ where $C[i]$ are Hermitian

Functions

- void blas_cher2k_batchf (int group_size, bblas_enum_t layout, bblas_enum_t uplo, bblas_enum_t trans, int n, int k, bblas_complex32_t alpha, bblas_complex32_t const ∗const ∗A, int lda, bblas_complex32_t const ∗const ∗B, int ldb, const float beta, bblas_complex32_t ∗∗C, int ldc, int ∗info)
- void blas_zher2k_batchf (int group_size, bblas_enum_t layout, bblas_enum_t uplo, bblas_enum_t trans, int n, int k, bblas_complex64_t alpha, bblas_complex64_t const ∗const ∗A, int lda, bblas_complex64_t const ∗const ∗B, int ldb, const double beta, bblas_complex64_t ∗∗C, int ldc, int ∗info)

3.17.1 Detailed Description

$C[i] = \alpha[i]A[i]B[i]^T + \alpha[i]B[i]A[i]^T + \beta[i]C[i]$ where $C[i]$ are Hermitian

3.17.2 Function Documentation

3.17.2.1 blas_cher2k_batchf()

```
void blas_cher2k_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t uplo,
            bblas_enum_t trans,
            int n,
            int k,
            bblas_complex32_t alpha,
            bblas_complex32_t const *const * A,
            int lda,
            bblas_complex32_t const *const * B,
            int ldb,
            const float beta,
            bblas_complex32_t ** C,
            int ldc,
            int * info )
```

Performs one of the batch Hermitian rank 2k operations

$$C[i] = \alpha A[i] \times B[i]^H + conjg(\alpha)B \times A[i]^H + \beta C[i],$$

or

$$C[i] = \alpha A[i]^H \times B[i] + conjg(\alpha)B[i]^H \times A[i] + \beta C[i],$$

where alpha is a complex scalar, beta is a real scalar, C[i]-s are n-by-n Hermitian matrices, and A[i]-s and B[i]-s are n-by-k matrices in the first case and k-by-n matrices in the second case.

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*uplo*	• BblasUpper: Upper triangle of C[i]-s is stored; • BblasLower: Lower triangle of C[i]-s is stored.
in	*trans*	• BblasNoTrans: $$C[i] = \alpha A[i] \times B[i]^H + conjg(\alpha) B[i] \times A[i]^H + \beta C[i];$$ • BblasConjTrans: $$C[i] = \alpha A[i]^H \times B[i] + conjg(\alpha) B[i]^H \times A[i] + \beta C[i].$$
in	*n*	The order of the matrix C[i]. n >= zero.
in	*k*	If trans = BblasNoTrans, number of columns of the A[i] and B[i] matrices; if trans = BblasConjTrans, number of rows of the A[i] and B[i] matrices.
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a matrix A[i] of size lda-by-ka. If trans = BblasNoTrans, ka = k; if trans = BblasConjTrans, ka = n.
in	*lda*	The leading dimension of the arrays A[i]. If trans = BblasNoTrans, lda >= max(1, n); if trans = BblasConjTrans, lda >= max(1, k).
in	*B*	B is an array of pointers to matrices B[0], B[1] .. B[group_size-1], where each element B[i] is a pointer to a matrix B[i] of size ldb-by-kb. If trans = BblasNoTrans, kb = k; if trans = BblasConjTrans, kb = n.
in	*ldb*	The leading dimension of the arrays B[i]. If trans = BblasNoTrans, ldb >= max(1, n); if trans = BblasConjTrans, ldb >= max(1, k).
in	*beta*	The scalar beta.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[group_size-1], where each element C[i] is a pointer to a matrix C[i] of size ldc-by-n. On exit, the uplo part of the matrix is overwritten by the uplo part of the updated matrix.
in	*ldc*	The leading dimension of the arrays C[i]. ldc >= max(1, n).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

cher2k_batchf
cher2k_batchf

3.17.2.2 blas_zher2k_batchf()

```
void blas_zher2k_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t uplo,
            bblas_enum_t trans,
            int n,
            int k,
            bblas_complex64_t alpha,
            bblas_complex64_t const *const * A,
            int lda,
            bblas_complex64_t const *const * B,
            int ldb,
            const double beta,
            bblas_complex64_t ** C,
            int ldc,
            int * info )
```

Performs one of the batch Hermitian rank 2k operations

$$C[i] = \alpha A[i] \times B[i]^H + conjg(\alpha) B \times A[i]^H + \beta C[i],$$

or

$$C[i] = \alpha A[i]^H \times B[i] + conjg(\alpha) B[i]^H \times A[i] + \beta C[i],$$

where alpha is a complex scalar, beta is a real scalar, C[i]-s are n-by-n Hermitian matrices, and A[i]-s and B[i]-s are n-by-k matrices in the first case and k-by-n matrices in the second case.

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*uplo*	• BblasUpper: Upper triangle of C[i]-s is stored; • BblasLower: Lower triangle of C[i]-s is stored.

Parameters

in	*trans*	• BblasNoTrans: $$C[i] = \alpha A[i] \times B[i]^H + conjg(\alpha) B[i] \times A[i]^H + \beta C[i];$$ • BblasConjTrans: $$C[i] = \alpha A[i]^H \times B[i] + conjg(\alpha) B[i]^H \times A[i] + \beta C[i].$$
in	*n*	The order of the matrix C[i]. n >= zero.
in	*k*	If trans = BblasNoTrans, number of columns of the A[i] and B[i] matrices; if trans = BblasConjTrans, number of rows of the A[i] and B[i] matrices.
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a matrix A[i] of size lda-by-ka. If trans = BblasNoTrans, ka = k; if trans = BblasConjTrans, ka = n.
in	*lda*	The leading dimension of the arrays A[i]. If trans = BblasNoTrans, lda >= max(1, n); if trans = BblasConjTrans, lda >= max(1, k).
in	*B*	B is an array of pointers to matrices B[0], B[1] .. B[group_size-1], where each element B[i] is a pointer to a matrix B[i] of size ldb-by-kb. If trans = BblasNoTrans, kb = k; if trans = BblasConjTrans, kb = n.
in	*ldb*	The leading dimension of the arrays B[i]. If trans = BblasNoTrans, ldb >= max(1, n); if trans = BblasConjTrans, ldb >= max(1, k).
in	*beta*	The scalar beta.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[group_size-1], where each element C[i] is a pointer to a matrix C[i] of size ldc-by-n. On exit, the uplo part of the matrix is overwritten by the uplo part of the updated matrix.
in	*ldc*	The leading dimension of the arrays C[i]. ldc >= max(1, n).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

zher2k_batchf
cher2k_batchf

3.18 symm_batchf: Batch of same size symmetric matrix multiply

$C[i] = \alpha[i]A[i]B[i] + \beta[i]C[i]$ or $C[i] = \alpha[i]B[i]A[i] + \beta[i]C[i]$ where $A[i]$ are symmetric

Functions

- void blas_csymm_batchf (int group_size, bblas_enum_t layout, bblas_enum_t side, bblas_enum_t uplo, int m, int n, bblas_complex32_t alpha, bblas_complex32_t const *const *A, int lda, bblas_complex32_t const *const *B, int ldb, bblas_complex32_t beta, bblas_complex32_t **C, int ldc, int *info)
- void blas_dsymm_batchf (int group_size, bblas_enum_t layout, bblas_enum_t side, bblas_enum_t uplo, int m, int n, double alpha, double const *const *A, int lda, double const *const *B, int ldb, double beta, double **C, int ldc, int *info)
- void blas_ssymm_batchf (int group_size, bblas_enum_t layout, bblas_enum_t side, bblas_enum_t uplo, int m, int n, float alpha, float const *const *A, int lda, float const *const *B, int ldb, float beta, float **C, int ldc, int *info)
- void blas_zsymm_batchf (int group_size, bblas_enum_t layout, bblas_enum_t side, bblas_enum_t uplo, int m, int n, bblas_complex64_t alpha, bblas_complex64_t const *const *A, int lda, bblas_complex64_t const *const *B, int ldb, bblas_complex64_t beta, bblas_complex64_t **C, int ldc, int *info)

3.18.1 Detailed Description

$C[i] = \alpha[i]A[i]B[i] + \beta[i]C[i]$ or $C[i] = \alpha[i]B[i]A[i] + \beta[i]C[i]$ where $A[i]$ are symmetric

3.18.2 Function Documentation

3.18.2.1 blas_csymm_batchf()

```
void blas_csymm_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t side,
            bblas_enum_t uplo,
            int m,
            int n,
            bblas_complex32_t alpha,
            bblas_complex32_t const *const * A,
            int lda,
            bblas_complex32_t const *const * B,
            int ldb,
            bblas_complex32_t beta,
            bblas_complex32_t ** C,
            int ldc,
            int * info )
```

Performs one of the matrix-matrix operations

$$C[i] = \alpha \times A[i] \times B[i] + \beta \times C[i]$$

or

$$C[i] = \alpha \times B[i] \times A[i] + \beta \times C[i]$$

where alpha and beta are scalars, A[i]-s are symmetric matrices and B[i]-s are C[i] are m-by-n matrices.

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*side*	Specifies whether the symmetric matrices A[i] appears on the left or right in the operation as follows: • BblasLeft: $C[i] = \alpha \times A[i] \times B[i] + \beta \times C[i]$ • BblasRight: $C[i] = \alpha \times B[i] \times A[i] + \beta \times C[i]$
in	*uplo*	Specifies whether the upper or lower triangular part of the symmetric matrices A[i] are to be referenced as follows: • BblasLower: Only the lower triangular part of the symmetric matrices A[i] is to be referenced. • BblasUpper: Only the upper triangular part of the symmetric matrices A[i] is to be referenced.
in	*m*	The number of rows of the matrices C[i]. m >= 0.
in	*n*	The number of columns of the matrices C[i]. n >= 0.
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a matrix A[i] of size lda-by-ka, where ka is m when side = BblasLeft, and is n otherwise. Only the uplo triangular part is referenced.
in	*lda*	The leading dimension of the arrays A[i]. lda >= max(1,ka).
in	*B*	B is an array of pointers to matrices B[0], B[1] .. B[group_size-1], where each element B[i] is a pointer to a matrix B[i] of size ldb-by-n matrix, where the leading m-by-n part of the array B[i] must contain the matrix B[i].
in	*ldb*	The leading dimension of the arrays B[i]. ldb >= max(1,m).
in	*beta*	The scalar beta.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[group_size-1], where each element C[i] is a pointer to a matrix C[i]. On exit, the array is overwritten by the m-by-n updated matrix.
in	*ldc*	The leading dimension of the arrays C[i]. ldc >= max(1,m).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

csymm_batchf
csymm_batchf
dsymm_batchf
ssymm_batchf

3.18.2.2 blas_dsymm_batchf()

```
void blas_dsymm_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t side,
            bblas_enum_t uplo,
            int m,
            int n,
            double alpha,
            double const *const * A,
            int lda,
            double const *const * B,
            int ldb,
            double beta,
            double ** C,
            int ldc,
            int * info )
```

Performs one of the matrix-matrix operations

$$C[i] = \alpha \times A[i] \times B[i] + \beta \times C[i]$$

or

$$C[i] = \alpha \times B[i] \times A[i] + \beta \times C[i]$$

where alpha and beta are scalars, A[i]-s are symmetric matrices and B[i]-s are C[i] are m-by-n matrices.

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format

Parameters

in	*side*	Specifies whether the symmetric matrices A[i] appears on the left or right in the operation as follows: • BblasLeft: $C[i] = \alpha \times A[i] \times B[i] + \beta \times C[i]$ • BblasRight: $C[i] = \alpha \times B[i] \times A[i] + \beta \times C[i]$
in	*uplo*	Specifies whether the upper or lower triangular part of the symmetric matrices A[i] are to be referenced as follows: • BblasLower: Only the lower triangular part of the symmetric matrices A[i] is to be referenced. • BblasUpper: Only the upper triangular part of the symmetric matrices A[i] is to be referenced.
in	*m*	The number of rows of the matrices C[i]. m >= 0.
in	*n*	The number of columns of the matrices C[i]. n >= 0.
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a matrix A[i] of size lda-by-ka, where ka is m when side = BblasLeft, and is n otherwise. Only the uplo triangular part is referenced.
in	*lda*	The leading dimension of the arrays A[i]. lda >= max(1,ka).
in	*B*	B is an array of pointers to matrices B[0], B[1] .. B[group_size-1], where each element B[i] is a pointer to a matrix B[i] of size ldb-by-n matrix, where the leading m-by-n part of the array B[i] must contain the matrix B[i].
in	*ldb*	The leading dimension of the arrays B[i]. ldb >= max(1,m).
in	*beta*	The scalar beta.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[group_size-1], where each element C[i] is a pointer to a matrix C[i]. On exit, the array is overwritten by the m-by-n updated matrix.
in	*ldc*	The leading dimension of the arrays C[i]. ldc >= max(1,m).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

dsymm_batchf
csymm_batchf
dsymm_batchf
ssymm_batchf

3.18.2.3 blas_ssymm_batchf()

```
void blas_ssymm_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t side,
            bblas_enum_t uplo,
            int m,
            int n,
            float alpha,
            float const *const * A,
            int lda,
            float const *const * B,
            int ldb,
            float beta,
            float ** C,
            int ldc,
            int * info )
```

Performs one of the matrix-matrix operations

$$C[i] = \alpha \times A[i] \times B[i] + \beta \times C[i]$$

or

$$C[i] = \alpha \times B[i] \times A[i] + \beta \times C[i]$$

where alpha and beta are scalars, A[i]-s are symmetric matrices and B[i]-s are C[i] are m-by-n matrices.

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*side*	Specifies whether the symmetric matrices A[i] appears on the left or right in the operation as follows: • BblasLeft: $C[i] = \alpha \times A[i] \times B[i] + \beta \times C[i]$ • BblasRight: $C[i] = \alpha \times B[i] \times A[i] + \beta \times C[i]$

Parameters

in	*uplo*	Specifies whether the upper or lower triangular part of the symmetric matrices A[i] are to be referenced as follows: • BblasLower: Only the lower triangular part of the symmetric matrices A[i] is to be referenced. • BblasUpper: Only the upper triangular part of the symmetric matrices A[i] is to be referenced.
in	*m*	The number of rows of the matrices C[i]. m >= 0.
in	*n*	The number of columns of the matrices C[i]. n >= 0.
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a matrix A[i] of size lda-by-ka, where ka is m when side = BblasLeft, and is n otherwise. Only the uplo triangular part is referenced.
in	*lda*	The leading dimension of the arrays A[i]. lda >= max(1,ka).
in	*B*	B is an array of pointers to matrices B[0], B[1] .. B[group_size-1], where each element B[i] is a pointer to a matrix B[i] of size ldb-by-n matrix, where the leading m-by-n part of the array B[i] must contain the matrix B[i].
in	*ldb*	The leading dimension of the arrays B[i]. ldb >= max(1,m).
in	*beta*	The scalar beta.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[group_size-1], where each element C[i] is a pointer to a matrix C[i]. On exit, the array is overwritten by the m-by-n updated matrix.
in	*ldc*	The leading dimension of the arrays C[i]. ldc >= max(1,m).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

ssymm_batchf
csymm_batchf
dsymm_batchf
ssymm_batchf

3.18.2.4 blas_zsymm_batchf()

```
void blas_zsymm_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t side,
            bblas_enum_t uplo,
            int m,
            int n,
            bblas_complex64_t alpha,
            bblas_complex64_t const *const * A,
            int lda,
            bblas_complex64_t const *const * B,
            int ldb,
            bblas_complex64_t beta,
            bblas_complex64_t ** C,
            int ldc,
            int * info )
```

Performs one of the matrix-matrix operations

$$C[i] = \alpha \times A[i] \times B[i] + \beta \times C[i]$$

or

$$C[i] = \alpha \times B[i] \times A[i] + \beta \times C[i]$$

where alpha and beta are scalars, A[i]-s are symmetric matrices and B[i]-s are C[i] are m-by-n matrices.

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*side*	Specifies whether the symmetric matrices A[i] appears on the left or right in the operation as follows: • BblasLeft: $C[i] = \alpha \times A[i] \times B[i] + \beta \times C[i]$ • BblasRight: $C[i] = \alpha \times B[i] \times A[i] + \beta \times C[i]$
in	*uplo*	Specifies whether the upper or lower triangular part of the symmetric matrices A[i] are to be referenced as follows: • BblasLower: Only the lower triangular part of the symmetric matrices A[i] is to be referenced. • BblasUpper: Only the upper triangular part of the symmetric matrices A[i] is to be referenced.
in	*m*	The number of rows of the matrices C[i]. m >= 0.
in	*n*	The number of columns of the matrices C[i]. n >= 0.
in	*alpha*	The scalar alpha.

Parameters

in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a matrix A[i] of size lda-by-ka, where ka is m when side = BblasLeft, and is n otherwise. Only the uplo triangular part is referenced.
in	*lda*	The leading dimension of the arrays A[i]. lda >= max(1,ka).
in	*B*	B is an array of pointers to matrices B[0], B[1] .. B[group_size-1], where each element B[i] is a pointer to a matrix B[i] of size ldb-by-n matrix, where the leading m-by-n part of the array B[i] must contain the matrix B[i].
in	*ldb*	The leading dimension of the arrays B[i]. ldb >= max(1,m).
in	*beta*	The scalar beta.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[group_size-1], where each element C[i] is a pointer to a matrix C[i]. On exit, the array is overwritten by the m-by-n updated matrix.
in	*ldc*	The leading dimension of the arrays C[i]. ldc >= max(1,m).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

zsymm_batchf
csymm_batchf
dsymm_batchf
ssymm_batchf

3.19 syrk_batchf: Batch of same size symmetric rank k update

$C[i] = \alpha[i]A[i]A[i]^T + \beta[i]C[i]$ where $C[i]$ are symmetric

Functions

- void blas_csyrk_batchf (int group_size, bblas_enum_t layout, bblas_enum_t uplo, bblas_enum_t trans, int n, int k, const bblas_complex32_t alpha, bblas_complex32_t const *const *A, int lda, const bblas_complex32_t beta, bblas_complex32_t **C, int ldc, int *info)
- void blas_dsyrk_batchf (int group_size, bblas_enum_t layout, bblas_enum_t uplo, bblas_enum_t trans, int n, int k, const double alpha, double const *const *A, int lda, const double beta, double **C, int ldc, int *info)
- void blas_ssyrk_batchf (int group_size, bblas_enum_t layout, bblas_enum_t uplo, bblas_enum_t trans, int n, int k, const float alpha, float const *const *A, int lda, const float beta, float **C, int ldc, int *info)
- void blas_zsyrk_batchf (int group_size, bblas_enum_t layout, bblas_enum_t uplo, bblas_enum_t trans, int n, int k, const bblas_complex64_t alpha, bblas_complex64_t const *const *A, int lda, const bblas_complex64_t beta, bblas_complex64_t **C, int ldc, int *info)

3.19.1 Detailed Description

$C[i] = \alpha[i]A[i]A[i]^T + \beta[i]C[i]$ where $C[i]$ are symmetric

3.19.2 Function Documentation

3.19.2.1 blas_csyrk_batchf()

```
void blas_csyrk_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t uplo,
            bblas_enum_t trans,
            int n,
            int k,
            const bblas_complex32_t alpha,
            bblas_complex32_t const *const * A,
            int lda,
            const bblas_complex32_t beta,
            bblas_complex32_t ** C,
            int ldc,
            int * info )
```

Performs one of the batch symmetric rank k operations

$$C[i] = \alpha A[i] \times A[i]^T + \beta C[i],$$

or

$$C[i] = \alpha A[i]^T \times A[i] + \beta C[i],$$

where alpha and beta are scalars, C[i]-s are n-by-n symmetric matrices, and A[i]-s are n-by-k matrices in the first case and a k-by-n matrices in the second case.

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*uplo*	• BblasUpper: Upper triangle of C[i]-s are stored; • BblasLower: Lower triangle of C[i]-s are stored.
in	*trans*	• BblasNoTrans: $C[i] = \alpha A[i] \times A[i]^T + \beta C[i];$ • BblasTrans: $C[i] = \alpha A[i]^T \times A[i] + \beta C[i].$
in	*n*	The order of the matrices C[i]. n >= 0.
in	*k*	If trans = BblasNoTrans, number of columns of matrices A[i]; if trans = BblasTrans, number of rows of matrices A[i].
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a matrix A[i] of size lda-by-ka. If trans = BblasNoTrans, ka = k; if trans = BblasTrans, ka = n.
in	*lda*	The leading dimension of the arrays A[i]. If trans = BblasNoTrans, lda >= max(1, n); if trans = BblasTrans, lda >= max(1, k).
in	*beta*	The scalar beta.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[group_size-1], where each element C[i] is a pointer to a matrix C[i] of size ldc-by-n. On exit, the uplo part of the matrix is overwritten by the uplo part of the updated matrix.
in	*ldc*	The leading dimension of the arrays C[i]. ldc >= max(1, n).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

csyrk_batchf

csyrk_batchf
dsyrk_batchf
ssyrk_batchf

3.19.2.2 blas_dsyrk_batchf()

```
void blas_dsyrk_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t uplo,
            bblas_enum_t trans,
            int n,
            int k,
            const double alpha,
            double const *const * A,
            int lda,
            const double beta,
            double ** C,
            int ldc,
            int * info )
```

Performs one of the batch symmetric rank k operations

$$C[i] = \alpha A[i] \times A[i]^T + \beta C[i],$$

or

$$C[i] = \alpha A[i]^T \times A[i] + \beta C[i],$$

where alpha and beta are scalars, C[i]-s are n-by-n symmetric matrices, and A[i]-s are n-by-k matrices in the first case and a k-by-n matrices in the second case.

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*uplo*	• BblasUpper: Upper triangle of C[i]-s are stored; • BblasLower: Lower triangle of C[i]-s are stored.
in	*trans*	• BblasNoTrans: $C[i] = \alpha A[i] \times A[i]^T + \beta C[i];$ • BblasTrans: $C[i] = \alpha A[i]^T \times A[i] + \beta C[i].$
in	*n*	The order of the matrices C[i]. n >= 0.

Parameters

in	*k*	If trans = BblasNoTrans, number of columns of matrices A[i]; if trans = BblasTrans, number of rows of matrices A[i].
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a matrix A[i] of size lda-by-ka. If trans = BblasNoTrans, ka = k; if trans = BblasTrans, ka = n.
in	*lda*	The leading dimension of the arrays A[i]. If trans = BblasNoTrans, lda >= max(1, n); if trans = BblasTrans, lda >= max(1, k).
in	*beta*	The scalar beta.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[group_size-1], where each element C[i] is a pointer to a matrix C[i] of size ldc-by-n. On exit, the uplo part of the matrix is overwritten by the uplo part of the updated matrix.
in	*ldc*	The leading dimension of the arrays C[i]. ldc >= max(1, n).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

dsyrk_batchf
csyrk_batchf
dsyrk_batchf
ssyrk_batchf

3.19.2.3 blas_ssyrk_batchf()

```
void blas_ssyrk_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t uplo,
            bblas_enum_t trans,
            int n,
            int k,
            const float alpha,
            float const *const * A,
```

```
int lda,
const float beta,
float ** C,
int ldc,
int * info )
```

Performs one of the batch symmetric rank k operations

$$C[i] = \alpha A[i] \times A[i]^T + \beta C[i],$$

or

$$C[i] = \alpha A[i]^T \times A[i] + \beta C[i],$$

where alpha and beta are scalars, C[i]-s are n-by-n symmetric matrices, and A[i]-s are n-by-k matrices in the first case and a k-by-n matrices in the second case.

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*uplo*	• BblasUpper: Upper triangle of C[i]-s are stored; • BblasLower: Lower triangle of C[i]-s are stored.
in	*trans*	• BblasNoTrans: $C[i] = \alpha A[i] \times A[i]^T + \beta C[i];$ • BblasTrans: $C[i] = \alpha A[i]^T \times A[i] + \beta C[i].$
in	*n*	The order of the matrices C[i]. n >= 0.
in	*k*	If trans = BblasNoTrans, number of columns of matrices A[i]; if trans = BblasTrans, number of rows of matrices A[i].
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a matrix A[i] of size lda-by-ka. If trans = BblasNoTrans, ka = k; if trans = BblasTrans, ka = n.
in	*lda*	The leading dimension of the arrays A[i]. If trans = BblasNoTrans, lda >= max(1, n); if trans = BblasTrans, lda >= max(1, k).
in	*beta*	The scalar beta.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[group_size-1], where each element C[i] is a pointer to a matrix C[i] of size ldc-by-n. On exit, the uplo part of the matrix is overwritten by the uplo part of the updated matrix.
in	*ldc*	The leading dimension of the arrays C[i]. ldc >= max(1, n).

Parameters

in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

ssyrk_batchf
csyrk_batchf
dsyrk_batchf
ssyrk_batchf

3.19.2.4 blas_zsyrk_batchf()

```
void blas_zsyrk_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t uplo,
            bblas_enum_t trans,
            int n,
            int k,
            const bblas_complex64_t alpha,
            bblas_complex64_t const *const * A,
            int lda,
            const bblas_complex64_t beta,
            bblas_complex64_t ** C,
            int ldc,
            int * info )
```

Performs one of the batch symmetric rank k operations

$$C[i] = \alpha A[i] \times A[i]^T + \beta C[i],$$

or

$$C[i] = \alpha A[i]^T \times A[i] + \beta C[i],$$

where alpha and beta are scalars, C[i]-s are n-by-n symmetric matrices, and A[i]-s are n-by-k matrices in the first case and a k-by-n matrices in the second case.

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*uplo*	• BblasUpper: Upper triangle of C[i]-s are stored; • BblasLower: Lower triangle of C[i]-s are stored.
in	*trans*	• BblasNoTrans: $C[i] = \alpha A[i] \times A[i]^T + \beta C[i];$ • BblasTrans: $C[i] = \alpha A[i]^T \times A[i] + \beta C[i].$
in	*n*	The order of the matrices C[i]. n >= 0.
in	*k*	If trans = BblasNoTrans, number of columns of matrices A[i]; if trans = BblasTrans, number of rows of matrices A[i].
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a matrix A[i] of size lda-by-ka. If trans = BblasNoTrans, ka = k; if trans = BblasTrans, ka = n.
in	*lda*	The leading dimension of the arrays A[i]. If trans = BblasNoTrans, lda >= max(1, n); if trans = BblasTrans, lda >= max(1, k).
in	*beta*	The scalar beta.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[group_size-1], where each element C[i] is a pointer to a matrix C[i] of size ldc-by-n. On exit, the uplo part of the matrix is overwritten by the uplo part of the updated matrix.
in	*ldc*	The leading dimension of the arrays C[i]. ldc >= max(1, n).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

zsyrk_batchf

csyrk_batchf
dsyrk_batchf
ssyrk_batchf

3.20 syr2k_batchf: Batch of same size symmetric rank 2k update

$C[i] = \alpha[i]A[i]B[i]^T + \alpha[i]B[i]A[i]^T + \beta[i]C[i]$ where $C[i]$ are symmetric

Functions

- void blas_csyr2k_batchf (int group_size, bblas_enum_t layout, bblas_enum_t uplo, bblas_enum_t trans, int n, int k, bblas_complex32_t alpha, bblas_complex32_t const *const *A, int lda, bblas_complex32_t const *const *B, int ldb, bblas_complex32_t beta, bblas_complex32_t **C, int ldc, int *info)
- void blas_dsyr2k_batchf (int group_size, bblas_enum_t layout, bblas_enum_t uplo, bblas_enum_t trans, int n, int k, double alpha, double const *const *A, int lda, double const *const *B, int ldb, double beta, double **C, int ldc, int *info)
- void blas_ssyr2k_batchf (int group_size, bblas_enum_t layout, bblas_enum_t uplo, bblas_enum_t trans, int n, int k, float alpha, float const *const *A, int lda, float const *const *B, int ldb, float beta, float **C, int ldc, int *info)
- void blas_zsyr2k_batchf (int group_size, bblas_enum_t layout, bblas_enum_t uplo, bblas_enum_t trans, int n, int k, bblas_complex64_t alpha, bblas_complex64_t const *const *A, int lda, bblas_complex64_t const *const *B, int ldb, bblas_complex64_t beta, bblas_complex64_t **C, int ldc, int *info)

3.20.1 Detailed Description

$C[i] = \alpha[i]A[i]B[i]^T + \alpha[i]B[i]A[i]^T + \beta[i]C[i]$ where $C[i]$ are symmetric

3.20.2 Function Documentation

3.20.2.1 blas_csyr2k_batchf()

```
void blas_csyr2k_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t uplo,
            bblas_enum_t trans,
            int n,
            int k,
            bblas_complex32_t alpha,
            bblas_complex32_t const *const * A,
            int lda,
            bblas_complex32_t const *const * B,
            int ldb,
            bblas_complex32_t beta,
            bblas_complex32_t ** C,
            int ldc,
            int * info )
```

Performs one of the batch symmetric rank 2k operations

$$C[i] = \alpha A[i] \times B[i]^T + \alpha B[i] \times A[i]^T + \beta C[i],$$

or

$$C[i] = \alpha A[i]^T \times B[i] + \alpha B[i]^T \times A[i] + \beta C[i],$$

where alpha and beta are scalars, C[i]-s are n-by-n symmetric matrix, and A[i]-s and B[i]-s are n-by-k matrices in the first case and k-by-n matrices in the second case.

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*uplo*	• BblasUpper: Upper triangle of C[i]-s are stored; • BblasLower: Lower triangle of C[i]-s are stored.
in	*trans*	• BblasNoTrans: $$C[i] = \alpha A[i] \times B[i]^T + \alpha B[i] \times A[i]^T + \beta C[i];$$ • BblasTrans: $$C[i] = \alpha A[i]^T \times B[i] + \alpha B[i]^T \times A[i] + \beta C[i].$$
in	*n*	The order of the matrices C[i]. n >= zero.
in	*k*	If trans = BblasNoTrans, number of columns of the A[i] and B[i] matrices; if trans = BblasTrans, number of rows of the A[i] and B[i] matrices.
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a matrix A[i] of size lda-by-ka. If trans = BblasNoTrans, ka = k; if trans = BblasTrans, ka = n.
in	*lda*	The leading dimension of the arrays A[i]. If trans = BblasNoTrans, lda >= max(1, n); if trans = BblasTrans, lda >= max(1, k).
in	*B*	B is an array of pointers to matrices B[0], B[1] .. B[group_size-1], where each element B[i] is a pointer to a matrix B[i] of size ldb-by-kb. If trans = BblasNoTrans, kb = k; if trans = BblasTrans, kb = n.
in	*ldb*	The leading dimension of the arrays B[i]. If trans = BblasNoTrans, ldb >= max(1, n); if trans = BblasTrans, ldb >= max(1, k).
in	*beta*	The scalar beta.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[group_size-1], where each element C[i] is a pointer to a matrix C[i] of size ldc-by-n. On exit, the uplo part of the matrix is overwritten by the uplo part of the updated matrix.
in	*ldc*	The leading dimension of the arrays C[i]. ldc >= max(1, n).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

csyr2k_batchf
csyr2k_batchf
dsyr2k_batchf
ssyr2k_batchf

3.20.2.2 blas_dsyr2k_batchf()

```
void blas_dsyr2k_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t uplo,
            bblas_enum_t trans,
            int n,
            int k,
            double alpha,
            double const *const * A,
            int lda,
            double const *const * B,
            int ldb,
            double beta,
            double ** C,
            int ldc,
            int * info )
```

Performs one of the batch symmetric rank 2k operations

$$C[i] = \alpha A[i] \times B[i]^T + \alpha B[i] \times A[i]^T + \beta C[i],$$

or

$$C[i] = \alpha A[i]^T \times B[i] + \alpha B[i]^T \times A[i] + \beta C[i],$$

where alpha and beta are scalars, C[i]-s are n-by-n symmetric matrix, and A[i]-s and B[i]-s are n-by-k matrices in the first case and k-by-n matrices in the second case.

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*uplo*	• BblasUpper: Upper triangle of C[i]-s are stored; • BblasLower: Lower triangle of C[i]-s are stored.

Parameters

in	*trans*	• BblasNoTrans: $$C[i] = \alpha A[i] \times B[i]^T + \alpha B[i] \times A[i]^T + \beta C[i];$$ • BblasTrans: $$C[i] = \alpha A[i]^T \times B[i] + \alpha B[i]^T \times A[i] + \beta C[i].$$
in	*n*	The order of the matrices C[i]. n >= zero.
in	*k*	If trans = BblasNoTrans, number of columns of the A[i] and B[i] matrices; if trans = BblasTrans, number of rows of the A[i] and B[i] matrices.
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a matrix A[i] of size lda-by-ka. If trans = BblasNoTrans, ka = k; if trans = BblasTrans, ka = n.
in	*lda*	The leading dimension of the arrays A[i]. If trans = BblasNoTrans, lda >= max(1, n); if trans = BblasTrans, lda >= max(1, k).
in	*B*	B is an array of pointers to matrices B[0], B[1] .. B[group_size-1], where each element B[i] is a pointer to a matrix B[i] of size ldb-by-kb. If trans = BblasNoTrans, kb = k; if trans = BblasTrans, kb = n.
in	*ldb*	The leading dimension of the arrays B[i]. If trans = BblasNoTrans, ldb >= max(1, n); if trans = BblasTrans, ldb >= max(1, k).
in	*beta*	The scalar beta.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[group_size-1], where each element C[i] is a pointer to a matrix C[i] of size ldc-by-n. On exit, the uplo part of the matrix is overwritten by the uplo part of the updated matrix.
in	*ldc*	The leading dimension of the arrays C[i]. ldc >= max(1, n).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

dsyr2k_batchf
csyr2k_batchf
dsyr2k_batchf
ssyr2k_batchf

3.20.2.3 blas_ssyr2k_batchf()

```
void blas_ssyr2k_batchf (
             int group_size,
             bblas_enum_t layout,
             bblas_enum_t uplo,
             bblas_enum_t trans,
             int n,
             int k,
             float alpha,
             float const *const * A,
             int lda,
             float const *const * B,
             int ldb,
             float beta,
             float ** C,
             int ldc,
             int * info )
```

Performs one of the batch symmetric rank 2k operations

$$C[i] = \alpha A[i] \times B[i]^T + \alpha B[i] \times A[i]^T + \beta C[i],$$

or

$$C[i] = \alpha A[i]^T \times B[i] + \alpha B[i]^T \times A[i] + \beta C[i],$$

where alpha and beta are scalars, C[i]-s are n-by-n symmetric matrix, and A[i]-s and B[i]-s are n-by-k matrices in the first case and k-by-n matrices in the second case.

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*uplo*	• BblasUpper: Upper triangle of C[i]-s are stored; • BblasLower: Lower triangle of C[i]-s are stored.
in	*trans*	• BblasNoTrans: $C[i] = \alpha A[i] \times B[i]^T + \alpha B[i] \times A[i]^T + \beta C[i];$ • BblasTrans: $C[i] = \alpha A[i]^T \times B[i] + \alpha B[i]^T \times A[i] + \beta C[i].$
in	*n*	The order of the matrices C[i]. n >= zero.
in	*k*	If trans = BblasNoTrans, number of columns of the A[i] and B[i] matrices; if trans = BblasTrans, number of rows of the A[i] and B[i] matrices.
in	*alpha*	The scalar alpha.

Parameters

in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a matrix A[i] of size lda-by-ka. If trans = BblasNoTrans, ka = k; if trans = BblasTrans, ka = n.
in	*lda*	The leading dimension of the arrays A[i]. If trans = BblasNoTrans, lda >= max(1, n); if trans = BblasTrans, lda >= max(1, k).
in	*B*	B is an array of pointers to matrices B[0], B[1] .. B[group_size-1], where each element B[i] is a pointer to a matrix B[i] of size ldb-by-kb. If trans = BblasNoTrans, kb = k; if trans = BblasTrans, kb = n.
in	*ldb*	The leading dimension of the arrays B[i]. If trans = BblasNoTrans, ldb >= max(1, n); if trans = BblasTrans, ldb >= max(1, k).
in	*beta*	The scalar beta.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[group_size-1], where each element C[i] is a pointer to a matrix C[i] of size ldc-by-n. On exit, the uplo part of the matrix is overwritten by the uplo part of the updated matrix.
in	*ldc*	The leading dimension of the arrays C[i]. ldc >= max(1, n).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

ssyr2k_batchf
csyr2k_batchf
dsyr2k_batchf
ssyr2k_batchf

3.20.2.4 blas_zsyr2k_batchf()

```
void blas_zsyr2k_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t uplo,
            bblas_enum_t trans,
            int n,
            int k,
```

```
bblas_complex64_t alpha,
bblas_complex64_t const *const * A,
int lda,
bblas_complex64_t const *const * B,
int ldb,
bblas_complex64_t beta,
bblas_complex64_t ** C,
int ldc,
int * info )
```

Performs one of the batch symmetric rank 2k operations

$$C[i] = \alpha A[i] \times B[i]^T + \alpha B[i] \times A[i]^T + \beta C[i],$$

or

$$C[i] = \alpha A[i]^T \times B[i] + \alpha B[i]^T \times A[i] + \beta C[i],$$

where alpha and beta are scalars, C[i]-s are n-by-n symmetric matrix, and A[i]-s and B[i]-s are n-by-k matrices in the first case and k-by-n matrices in the second case.

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*uplo*	• BblasUpper: Upper triangle of C[i]-s are stored; • BblasLower: Lower triangle of C[i]-s are stored.
in	*trans*	• BblasNoTrans: $C[i] = \alpha A[i] \times B[i]^T + \alpha B[i] \times A[i]^T + \beta C[i];$ • BblasTrans: $C[i] = \alpha A[i]^T \times B[i] + \alpha B[i]^T \times A[i] + \beta C[i].$
in	*n*	The order of the matrices C[i]. n >= zero.
in	*k*	If trans = BblasNoTrans, number of columns of the A[i] and B[i] matrices; if trans = BblasTrans, number of rows of the A[i] and B[i] matrices.
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a matrix A[i] of size lda-by-ka. If trans = BblasNoTrans, ka = k; if trans = BblasTrans, ka = n.
in	*lda*	The leading dimension of the arrays A[i]. If trans = BblasNoTrans, lda >= max(1, n); if trans = BblasTrans, lda >= max(1, k).
in	*B*	B is an array of pointers to matrices B[0], B[1] .. B[group_size-1], where each element B[i] is a pointer to a matrix B[i] of size ldb-by-kb. If trans = BblasNoTrans, kb = k; if trans = BblasTrans, kb = n.
in	*ldb*	The leading dimension of the arrays B[i]. If trans = BblasNoTrans, ldb >= max(1, n); if trans = BblasTrans, ldb >= max(1, k).

Parameters

in	*beta*	The scalar beta.
in,out	*C*	C is an array of pointers to matrices C[0], C[1] .. C[group_size-1], where each element C[i] is a pointer to a matrix C[i] of size ldc-by-n. On exit, the uplo part of the matrix is overwritten by the uplo part of the updated matrix.
in	*ldc*	The leading dimension of the arrays C[i]. ldc >= max(1, n).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

zsyr2k_batchf
csyr2k_batchf
dsyr2k_batchf
ssyr2k_batchf

3.21 trmm_batchf: Batch of same size triangular matrix multiply

$B[i] = \alpha[i]\ op(A[i])\ B[i]$ or $B[i] = \alpha[i]B[i]\ op(A[i])$ where $A[i]$ are triangular

Functions

- void blas_ctrmm_batchf (int group_size, bblas_enum_t layout, bblas_enum_t side, bblas_enum_t uplo, bblas_enum_t transa, bblas_enum_t diag, int m, int n, bblas_complex32_t alpha, bblas_complex32_t const ∗const ∗A, int lda, bblas_complex32_t ∗∗B, int ldb, int ∗info)
- void blas_dtrmm_batchf (int group_size, bblas_enum_t layout, bblas_enum_t side, bblas_enum_t uplo, bblas_enum_t transa, bblas_enum_t diag, int m, int n, double alpha, double const ∗const ∗A, int lda, double ∗∗B, int ldb, int ∗info)
- void blas_strmm_batchf (int group_size, bblas_enum_t layout, bblas_enum_t side, bblas_enum_t uplo, bblas_enum_t transa, bblas_enum_t diag, int m, int n, float alpha, float const ∗const ∗A, int lda, float ∗∗B, int ldb, int ∗info)
- void blas_ztrmm_batchf (int group_size, bblas_enum_t layout, bblas_enum_t side, bblas_enum_t uplo, bblas_enum_t transa, bblas_enum_t diag, int m, int n, bblas_complex64_t alpha, bblas_complex64_t const ∗const ∗A, int lda, bblas_complex64_t ∗∗B, int ldb, int ∗info)

3.21.1 Detailed Description

$B[i] = \alpha[i]\ op(A[i])\ B[i]$ or $B[i] = \alpha[i]B[i]\ op(A[i])$ where $A[i]$ are triangular

3.21.2 Function Documentation

3.21.2.1 blas_ctrmm_batchf()

```
void blas_ctrmm_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t side,
            bblas_enum_t uplo,
            bblas_enum_t transa,
            bblas_enum_t diag,
            int m,
            int n,
            bblas_complex32_t alpha,
            bblas_complex32_t const *const * A,
            int lda,
            bblas_complex32_t ** B,
            int ldb,
            int * info )
```

Performs a triangular batch matrix-matrix multiply of the form

$$B[i] = \alpha[op(A[i]) \times B[i]]$$

, if side = BblasLeft or

$$B[i] = \alpha[B[i] \times op(A[i])]$$

, if side = BblasRight

where op(X) is one of:

```
- op(A[i]) = A[i]   or
- op(A[i]) = A[i]^T or
- op(A[i]) = A[i]^H
```

alpha is a scalar, B[i]-s are m-by-n matrices and A[i]-s are a unit or non-unit, upper or lower triangular matrix.

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*side*	Specifies whether op(A[i]) appears on the left or on the right of B[i]: • BblasLeft: alpha∗op(A[i])∗B[i] • BblasRight: alpha∗B[i]∗op(A[i])
in	*uplo*	Specifies whether the matrices A[i]-s are upper triangular or lower triangular: • BblasUpper: Upper triangle of A[i] is stored; • BblasLower: Lower triangle of A[i] is stored.
in	*transa*	Specifies whether the matrices A[i] are transposed, not transposed or conjugate transposed: • BblasNoTrans: A[i]-s are transposed; • BblasTrans: A[i]-s are not transposed; • BblasConjTrans: A[i]-s are conjugate transposed.
in	*diag*	Specifies whether or not A[i]-s are unit triangular: • BblasNonUnit: A[i]-s are non-unit triangular; • BblasUnit: A[i]-s are unit triangular.
in	*m*	The number of rows of matrices B[i]. m >= 0.
in	*n*	The number of columns of matrices B[i]. n >= 0.
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a triangular matrix of dimension lda-by-k, where k is m when side='L' or 'l' and k is n when when side='R' or 'r'. If uplo = BblasUpper, the leading k-by-k upper triangular part of the array A[i] contains the upper triangular matrix, and the strictly lower triangular part of A[i] is not referenced. If uplo = BblasLower, the leading k-by-k lower triangular part of the array A[i] contains the lower triangular matrix, and the strictly upper triangular part of A[i] is not referenced. If diag = BblasUnit, the diagonal elements of A[i] are also not referenced and are assumed to be 1.
in	*lda*	The leading dimension of the arrays A[i]. When side='L' or 'l', lda >= max(1,m), when side='R' or 'r' then lda >= max(1,n).
in	*B*	B is an array of pointers to matrices B[0], B[1],..,B[group_size-1], where each element B[i] is a pointer to a matrix. On entry, the matrices B[i] are of dimension ldb-by-n. On exit, the result of a triangular matrix-matrix multiply (alpha∗op(A[i])∗B[i]) or (alpha∗B[i]∗op(A[i])).
in	*ldb*	The leading dimension of the arrays B[i]. ldb >= max(1,m).

Parameters

in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

ctrmm_batchf
ctrmm_batchf
dtrmm_batchf
strmm_batchf

3.21.2.2 blas_dtrmm_batchf()

```
void blas_dtrmm_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t side,
            bblas_enum_t uplo,
            bblas_enum_t transa,
            bblas_enum_t diag,
            int m,
            int n,
            double alpha,
            double const *const * A,
            int lda,
            double ** B,
            int ldb,
            int * info )
```

Performs a triangular batch matrix-matrix multiply of the form

$$B[i] = \alpha[op(A[i]) \times B[i]]$$

, if side = BblasLeft or

$$B[i] = \alpha[B[i] \times op(A[i])]$$

, if side = BblasRight

where op(X) is one of:

```
- op(A[i]) = A[i]   or
- op(A[i]) = A[i]^T or
- op(A[i]) = A[i]^T
```

alpha is a scalar, B[i]-s are m-by-n matrices and A[i]-s are a unit or non-unit, upper or lower triangular matrix.

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*side*	Specifies whether op(A[i]) appears on the left or on the right of B[i]: • BblasLeft: alpha∗op(A[i])∗B[i] • BblasRight: alpha∗B[i]∗op(A[i])
in	*uplo*	Specifies whether the matrices A[i]-s are upper triangular or lower triangular: • BblasUpper: Upper triangle of A[i] is stored; • BblasLower: Lower triangle of A[i] is stored.
in	*transa*	Specifies whether the matrices A[i] are transposed, not transposed or conjugate transposed: • BblasNoTrans: A[i]-s are transposed; • BblasTrans: A[i]-s are not transposed; • BblasConjTrans: A[i]-s are conjugate transposed.
in	*diag*	Specifies whether or not A[i]-s are unit triangular: • BblasNonUnit: A[i]-s are non-unit triangular; • BblasUnit: A[i]-s are unit triangular.
in	*m*	The number of rows of matrices B[i]. m >= 0.
in	*n*	The number of columns of matrices B[i]. n >= 0.
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a triangular matrix of dimension lda-by-k, where k is m when side='L' or 'l' and k is n when when side='R' or 'r'. If uplo = BblasUpper, the leading k-by-k upper triangular part of the array A[i] contains the upper triangular matrix, and the strictly lower triangular part of A[i] is not referenced. If uplo = BblasLower, the leading k-by-k lower triangular part of the array A[i] contains the lower triangular matrix, and the strictly upper triangular part of A[i] is not referenced. If diag = BblasUnit, the diagonal elements of A[i] are also not referenced and are assumed to be 1.
in	*lda*	The leading dimension of the arrays A[i]. When side='L' or 'l', lda >= max(1,m), when side='R' or 'r' then lda >= max(1,n).
in	*B*	B is an array of pointers to matrices B[0], B[1],..,B[group_size-1], where each element B[i] is a pointer to a matrix. On entry, the matrices B[i] are of dimension ldb-by-n. On exit, the result of a triangular matrix-matrix multiply (alpha∗op(A[i])∗B[i]) or (alpha∗B[i]∗op(A[i])).
in	*ldb*	The leading dimension of the arrays B[i]. ldb >= max(1,m).

Parameters

in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

dtrmm_batchf
ctrmm_batchf
dtrmm_batchf
strmm_batchf

3.21.2.3 blas_strmm_batchf()

```
void blas_strmm_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t side,
            bblas_enum_t uplo,
            bblas_enum_t transa,
            bblas_enum_t diag,
            int m,
            int n,
            float alpha,
            float const *const * A,
            int lda,
            float ** B,
            int ldb,
            int * info )
```

Performs a triangular batch matrix-matrix multiply of the form

$$B[i] = \alpha[op(A[i]) \times B[i]]$$

, if side = BblasLeft or

$$B[i] = \alpha[B[i] \times op(A[i])]$$

, if side = BblasRight

where op(X) is one of:

```
- op(A[i]) = A[i]   or
- op(A[i]) = A[i]^T or
- op(A[i]) = A[i]^T
```

alpha is a scalar, B[i]-s are m-by-n matrices and A[i]-s are a unit or non-unit, upper or lower triangular matrix.

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*side*	Specifies whether op(A[i]) appears on the left or on the right of B[i]: • BblasLeft: alpha∗op(A[i])∗B[i] • BblasRight: alpha∗B[i]∗op(A[i])
in	*uplo*	Specifies whether the matrices A[i]-s are upper triangular or lower triangular: • BblasUpper: Upper triangle of A[i] is stored; • BblasLower: Lower triangle of A[i] is stored.
in	*transa*	Specifies whether the matrices A[i] are transposed, not transposed or conjugate transposed: • BblasNoTrans: A[i]-s are transposed; • BblasTrans: A[i]-s are not transposed; • BblasConjTrans: A[i]-s are conjugate transposed.
in	*diag*	Specifies whether or not A[i]-s are unit triangular: • BblasNonUnit: A[i]-s are non-unit triangular; • BblasUnit: A[i]-s are unit triangular.
in	*m*	The number of rows of matrices B[i]. m >= 0.
in	*n*	The number of columns of matrices B[i]. n >= 0.
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a triangular matrix of dimension lda-by-k, where k is m when side='L' or 'l' and k is n when when side='R' or 'r'. If uplo = BblasUpper, the leading k-by-k upper triangular part of the array A[i] contains the upper triangular matrix, and the strictly lower triangular part of A[i] is not referenced. If uplo = BblasLower, the leading k-by-k lower triangular part of the array A[i] contains the lower triangular matrix, and the strictly upper triangular part of A[i] is not referenced. If diag = BblasUnit, the diagonal elements of A[i] are also not referenced and are assumed to be 1.
in	*lda*	The leading dimension of the arrays A[i]. When side='L' or 'l', lda >= max(1,m), when side='R' or 'r' then lda >= max(1,n).
in	*B*	B is an array of pointers to matrices B[0], B[1],..,B[group_size-1], where each element B[i] is a pointer to a matrix. On entry, the matrices B[i] are of dimension ldb-by-n. On exit, the result of a triangular matrix-matrix multiply (alpha∗op(A[i])∗B[i]) or (alpha∗B[i]∗op(A[i])).
in	*ldb*	The leading dimension of the arrays B[i]. ldb >= max(1,m).

Parameters

in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

strmm_batchf
ctrmm_batchf
dtrmm_batchf
strmm_batchf

3.21.2.4 blas_ztrmm_batchf()

```
void blas_ztrmm_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t side,
            bblas_enum_t uplo,
            bblas_enum_t transa,
            bblas_enum_t diag,
            int m,
            int n,
            bblas_complex64_t alpha,
            bblas_complex64_t const *const * A,
            int lda,
            bblas_complex64_t ** B,
            int ldb,
            int * info )
```

Performs a triangular batch matrix-matrix multiply of the form

$$B[i] = \alpha[op(A[i]) \times B[i]]$$

, if side = BblasLeft or

$$B[i] = \alpha[B[i] \times op(A[i])]$$

, if side = BblasRight

where op(X) is one of:

```
- op(A[i]) = A[i]   or
- op(A[i]) = A[i]^T or
- op(A[i]) = A[i]^H
```

alpha is a scalar, B[i]-s are m-by-n matrices and A[i]-s are a unit or non-unit, upper or lower triangular matrix.

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*side*	Specifies whether op(A[i]) appears on the left or on the right of B[i]: • BblasLeft: alpha∗op(A[i])∗B[i] • BblasRight: alpha∗B[i]∗op(A[i])
in	*uplo*	Specifies whether the matrices A[i]-s are upper triangular or lower triangular: • BblasUpper: Upper triangle of A[i] is stored; • BblasLower: Lower triangle of A[i] is stored.
in	*transa*	Specifies whether the matrices A[i] are transposed, not transposed or conjugate transposed: • BblasNoTrans: A[i]-s are transposed; • BblasTrans: A[i]-s are not transposed; • BblasConjTrans: A[i]-s are conjugate transposed.
in	*diag*	Specifies whether or not A[i]-s are unit triangular: • BblasNonUnit: A[i]-s are non-unit triangular; • BblasUnit: A[i]-s are unit triangular.
in	*m*	The number of rows of matrices B[i]. m >= 0.
in	*n*	The number of columns of matrices B[i]. n >= 0.
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a triangular matrix of dimension lda-by-k, where k is m when side='L' or 'l' and k is n when when side='R' or 'r'. If uplo = BblasUpper, the leading k-by-k upper triangular part of the array A[i] contains the upper triangular matrix, and the strictly lower triangular part of A[i] is not referenced. If uplo = BblasLower, the leading k-by-k lower triangular part of the array A[i] contains the lower triangular matrix, and the strictly upper triangular part of A[i] is not referenced. If diag = BblasUnit, the diagonal elements of A[i] are also not referenced and are assumed to be 1.
in	*lda*	The leading dimension of the arrays A[i]. When side='L' or 'l', lda >= max(1,m), when side='R' or 'r' then lda >= max(1,n).
in	*B*	B is an array of pointers to matrices B[0], B[1],..,B[group_size-1], where each element B[i] is a pointer to a matrix. On entry, the matrices B[i] are of dimension ldb-by-n. On exit, the result of a triangular matrix-matrix multiply (alpha∗op(A[i])∗B[i]) or (alpha∗B[i]∗op(A[i])).
in	*ldb*	The leading dimension of the arrays B[i]. ldb >= max(1,m).

Parameters

in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

ztrmm_batchf
ctrmm_batchf
dtrmm_batchf
strmm_batchf

3.22 trsm_batchf: Batch of same size triangular solve matrix

$C[i] = op(A[i])^{-1}B[i]$ or $C[i] = B[i]\ op(A[i])^{-1}$ where $A[i]$ are triangular

Functions

- void blas_ctrsm_batchf (int group_size, bblas_enum_t layout, bblas_enum_t side, bblas_enum_t uplo, bblas_enum_t transa, bblas_enum_t diag, int m, int n, bblas_complex32_t alpha, bblas_complex32_t const *const *A, int lda, bblas_complex32_t **B, int ldb, int *info)
- void blas_dtrsm_batchf (int group_size, bblas_enum_t layout, bblas_enum_t side, bblas_enum_t uplo, bblas_enum_t transa, bblas_enum_t diag, int m, int n, double alpha, double const *const *A, int lda, double **B, int ldb, int *info)
- void blas_strsm_batchf (int group_size, bblas_enum_t layout, bblas_enum_t side, bblas_enum_t uplo, bblas_enum_t transa, bblas_enum_t diag, int m, int n, float alpha, float const *const *A, int lda, float **B, int ldb, int *info)
- void blas_ztrsm_batchf (int group_size, bblas_enum_t layout, bblas_enum_t side, bblas_enum_t uplo, bblas_enum_t transa, bblas_enum_t diag, int m, int n, bblas_complex64_t alpha, bblas_complex64_t const *const *A, int lda, bblas_complex64_t **B, int ldb, int *info)

3.22.1 Detailed Description

$C[i] = op(A[i])^{-1}B[i]$ or $C[i] = B[i]\ op(A[i])^{-1}$ where $A[i]$ are triangular

3.22.2 Function Documentation

3.22.2.1 blas_ctrsm_batchf()

```
void blas_ctrsm_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t side,
            bblas_enum_t uplo,
            bblas_enum_t transa,
            bblas_enum_t diag,
            int m,
            int n,
            bblas_complex32_t alpha,
            bblas_complex32_t const *const * A,
            int lda,
            bblas_complex32_t ** B,
            int ldb,
            int * info )
```

Solves one of the batch matrix equations

$$op(A[i]) \times X[i] = \alpha B[i],$$

or

$$X[i] \times op(A[i]) = \alpha B[i],$$

where op(A[i]) is one of:

$$op(A[i]) = A[i],$$
$$op(A[i]) = A[i]^T,$$
$$op(A[i]) = A[i]^H,$$

alpha is a scalar, X[i]-s and B[i]-s are m-by-n matrices, and A[i]-s are unit or non-unit, upper or lower triangular matrices. The matrix X[i] overwrites B[i].

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*side*	Specifies whether op(A[i]) appears on the left or on the right of X[i]: • BblasLeft: op(A[i])∗X[i] = B[i], • BblasRight: X[i]∗op(A[i]) = B[i].
in	*uplo*	Specifies whether the matrices A[i]-s are upper triangular or lower triangular: • BblasUpper: Upper triangle of A[i] is stored; • BblasLower: Lower triangle of A[i] is stored.
in	*transa*	Specifies whether the matrices A[i] are transposed, not transposed or conjugate transposed: • BblasNoTrans: A[i]-s are transposed; • BblasTrans: A[i]-s are not transposed; • BblasConjTrans: A[i]-s are conjugate transposed.
in	*diag*	Specifies whether or not A[i]-s are unit triangular: • BblasNonUnit: A[i]-s are non-unit triangular; • BblasUnit: A[i]-s are unit triangular.
in	*m*	The number of rows of matrices B[i]. m >= 0.
in	*n*	The number of columns of matrices B[i]. n >= 0.
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a triangular matrix of dimension lda-by-ka triangular, where ka = m if side = BblasLeft, and ka = n if side = BblasRight. If uplo = BblasUpper, the leading k-by-k upper triangular part of the array A[i] contains the upper triangular matrix, and the strictly lower triangular part of A[i] is not referenced. If uplo = BblasLower, the leading k-by-k lower triangular part of the array A[i] contains the lower triangular matrix, and the strictly upper triangular part of A[i] is not referenced. If diag = BblasUnit, the diagonal elements of A[i] are also not referenced and are assumed to be 1.
in	*lda*	The leading dimension of the array A. lda >= max(1,k).

Parameters

in,out	*B*	B is an array of pointers to matrices B[0], B[1] .. B[group_size-1], On entry, each B[i]-s are ldb-by-n right hand side matrix. On exit, if return value = 0, the ldb-by-n solution matrix X.
in	*ldb*	The leading dimension of the array B. ldb >= max(1,m).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

ctrsm_batchf
ctrsm_batchf
dtrsm_batchf
strsm_batchf

3.22.2.2 blas_dtrsm_batchf()

```
void blas_dtrsm_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t side,
            bblas_enum_t uplo,
            bblas_enum_t transa,
            bblas_enum_t diag,
            int m,
            int n,
            double alpha,
            double const *const * A,
            int lda,
            double ** B,
            int ldb,
            int * info )
```

Solves one of the batch matrix equations

$$op(A[i]) \times X[i] = \alpha B[i],$$

or

$$X[i] \times op(A[i]) = \alpha B[i],$$

where op(A[i]) is one of:

$$op(A[i]) = A[i],$$
$$op(A[i]) = A[i]^T,$$
$$op(A[i]) = A[i]^T,$$

alpha is a scalar, X[i]-s and B[i]-s are m-by-n matrices, and A[i]-s are unit or non-unit, upper or lower triangular matrices. The matrix X[i] overwrites B[i].

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*side*	Specifies whether op(A[i]) appears on the left or on the right of X[i]: • BblasLeft: op(A[i])∗X[i] = B[i], • BblasRight: X[i]∗op(A[i]) = B[i].
in	*uplo*	Specifies whether the matrices A[i]-s are upper triangular or lower triangular: • BblasUpper: Upper triangle of A[i] is stored; • BblasLower: Lower triangle of A[i] is stored.
in	*transa*	Specifies whether the matrices A[i] are transposed, not transposed or conjugate transposed: • BblasNoTrans: A[i]-s are transposed; • BblasTrans: A[i]-s are not transposed; • BblasConjTrans: A[i]-s are conjugate transposed.
in	*diag*	Specifies whether or not A[i]-s are unit triangular: • BblasNonUnit: A[i]-s are non-unit triangular; • BblasUnit: A[i]-s are unit triangular.
in	*m*	The number of rows of matrices B[i]. m >= 0.
in	*n*	The number of columns of matrices B[i]. n >= 0.
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a triangular matrix of dimension lda-by-ka triangular, where ka = m if side = BblasLeft, and ka = n if side = BblasRight. If uplo = BblasUpper, the leading k-by-k upper triangular part of the array A[i] contains the upper triangular matrix, and the strictly lower triangular part of A[i] is not referenced. If uplo = BblasLower, the leading k-by-k lower triangular part of the array A[i] contains the lower triangular matrix, and the strictly upper triangular part of A[i] is not referenced. If diag = BblasUnit, the diagonal elements of A[i] are also not referenced and are assumed to be 1.
in	*lda*	The leading dimension of the array A. lda >= max(1,k).

Parameters

in,out	*B*	B is an array of pointers to matrices B[0], B[1] .. B[group_size-1], On entry, each B[i]-s are ldb-by-n right hand side matrix. On exit, if return value = 0, the ldb-by-n solution matrix X.
in	*ldb*	The leading dimension of the array B. ldb >= max(1,m).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

dtrsm_batchf
ctrsm_batchf
dtrsm_batchf
strsm_batchf

3.22.2.3 blas_strsm_batchf()

```
void blas_strsm_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t side,
            bblas_enum_t uplo,
            bblas_enum_t transa,
            bblas_enum_t diag,
            int m,
            int n,
            float alpha,
            float const *const * A,
            int lda,
            float ** B,
            int ldb,
            int * info )
```

Solves one of the batch matrix equations

$$op(A[i]) \times X[i] = \alpha B[i],$$

or

$$X[i] \times op(A[i]) = \alpha B[i],$$

where op(A[i]) is one of:

$$op(A[i]) = A[i],$$
$$op(A[i]) = A[i]^T,$$
$$op(A[i]) = A[i]^T,$$

alpha is a scalar, X[i]-s and B[i]-s are m-by-n matrices, and A[i]-s are unit or non-unit, upper or lower triangular matrices. The matrix X[i] overwrites B[i].

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*side*	Specifies whether op(A[i]) appears on the left or on the right of X[i]: • BblasLeft: op(A[i])∗X[i] = B[i], • BblasRight: X[i]∗op(A[i]) = B[i].
in	*uplo*	Specifies whether the matrices A[i]-s are upper triangular or lower triangular: • BblasUpper: Upper triangle of A[i] is stored; • BblasLower: Lower triangle of A[i] is stored.
in	*transa*	Specifies whether the matrices A[i] are transposed, not transposed or conjugate transposed: • BblasNoTrans: A[i]-s are transposed; • BblasTrans: A[i]-s are not transposed; • BblasConjTrans: A[i]-s are conjugate transposed.
in	*diag*	Specifies whether or not A[i]-s are unit triangular: • BblasNonUnit: A[i]-s are non-unit triangular; • BblasUnit: A[i]-s are unit triangular.
in	*m*	The number of rows of matrices B[i]. m >= 0.
in	*n*	The number of columns of matrices B[i]. n >= 0.
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a triangular matrix of dimension lda-by-ka triangular, where ka = m if side = BblasLeft, and ka = n if side = BblasRight. If uplo = BblasUpper, the leading k-by-k upper triangular part of the array A[i] contains the upper triangular matrix, and the strictly lower triangular part of A[i] is not referenced. If uplo = BblasLower, the leading k-by-k lower triangular part of the array A[i] contains the lower triangular matrix, and the strictly upper triangular part of A[i] is not referenced. If diag = BblasUnit, the diagonal elements of A[i] are also not referenced and are assumed to be 1.
in	*lda*	The leading dimension of the array A. lda >= max(1,k).

Parameters

in,out	*B*	B is an array of pointers to matrices B[0], B[1] .. B[group_size-1], On entry, each B[i]-s are ldb-by-n right hand side matrix. On exit, if return value = 0, the ldb-by-n solution matrix X.
in	*ldb*	The leading dimension of the array B. ldb >= max(1,m).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

strsm_batchf
ctrsm_batchf
dtrsm_batchf
strsm_batchf

3.22.2.4 blas_ztrsm_batchf()

```
void blas_ztrsm_batchf (
            int group_size,
            bblas_enum_t layout,
            bblas_enum_t side,
            bblas_enum_t uplo,
            bblas_enum_t transa,
            bblas_enum_t diag,
            int m,
            int n,
            bblas_complex64_t alpha,
            bblas_complex64_t const *const * A,
            int lda,
            bblas_complex64_t ** B,
            int ldb,
            int * info )
```

Solves one of the batch matrix equations

$$op(A[i]) \times X[i] = \alpha B[i],$$

or

$$X[i] \times op(A[i]) = \alpha B[i],$$

where op(A[i]) is one of:

$$op(A[i]) = A[i],$$
$$op(A[i]) = A[i]^T,$$
$$op(A[i]) = A[i]^H,$$

alpha is a scalar, X[i]-s and B[i]-s are m-by-n matrices, and A[i]-s are unit or non-unit, upper or lower triangular matrices. The matrix X[i] overwrites B[i].

Parameters

in	*group_size*	The number of matrices to operate on
in	*layout*	Specifies if the matrix is stored in row major or column major format: • BblasRowMajor: Row major format • BblasColMajor: Column major format
in	*side*	Specifies whether op(A[i]) appears on the left or on the right of X[i]: • BblasLeft: op(A[i])∗X[i] = B[i], • BblasRight: X[i]∗op(A[i]) = B[i].
in	*uplo*	Specifies whether the matrices A[i]-s are upper triangular or lower triangular: • BblasUpper: Upper triangle of A[i] is stored; • BblasLower: Lower triangle of A[i] is stored.
in	*transa*	Specifies whether the matrices A[i] are transposed, not transposed or conjugate transposed: • BblasNoTrans: A[i]-s are transposed; • BblasTrans: A[i]-s are not transposed; • BblasConjTrans: A[i]-s are conjugate transposed.
in	*diag*	Specifies whether or not A[i]-s are unit triangular: • BblasNonUnit: A[i]-s are non-unit triangular; • BblasUnit: A[i]-s are unit triangular.
in	*m*	The number of rows of matrices B[i]. m >= 0.
in	*n*	The number of columns of matrices B[i]. n >= 0.
in	*alpha*	The scalar alpha.
in	*A*	A is an array of pointers to matrices A[0], A[1] .. A[group_size-1], where each element A[i] is a pointer to a triangular matrix of dimension lda-by-ka triangular, where ka = m if side = BblasLeft, and ka = n if side = BblasRight. If uplo = BblasUpper, the leading k-by-k upper triangular part of the array A[i] contains the upper triangular matrix, and the strictly lower triangular part of A[i] is not referenced. If uplo = BblasLower, the leading k-by-k lower triangular part of the array A[i] contains the lower triangular matrix, and the strictly upper triangular part of A[i] is not referenced. If diag = BblasUnit, the diagonal elements of A[i] are also not referenced and are assumed to be 1.
in	*lda*	The leading dimension of the array A. lda >= max(1,k).

Parameters

in,out	*B*	B is an array of pointers to matrices B[0], B[1] .. B[group_size-1], On entry, each B[i]-s are ldb-by-n right hand side matrix. On exit, if return value = 0, the ldb-by-n solution matrix X.
in	*ldb*	The leading dimension of the array B. ldb >= max(1,m).
in,out	*info*	Array of int for error handling. On entry info[0] should have one of the following values • BblasErrorsReportAll : All errors will be specified on output. Length of the array should be atleast {i=0}^{group_count-1}group_sizes[i]+1. • BblasErrorsReportGroup : Single error from each group will be reported. Length of the array should be atleast group_count+1. • BblasErrorsReportAny : Occurence of an error will be indicated by a single integer value, and length of the array should be atleast 1. • BblasErrorsReportNone : No error will be reported on output, and length of the array should be atleast 1.

Return values

BblasSuccess	successful exit

See also

ztrsm_batchf
ctrsm_batchf
dtrsm_batchf
strsm_batchf

3.23 Bblas_const

Functions

- bblas_enum_t bblas_diag_const (char lapack_char)
- bblas_enum_t bblas_info_const (char lapack_char)
- bblas_enum_t bblas_direct_const (char lapack_char)
- bblas_enum_t bblas_norm_const (char lapack_char)
- bblas_enum_t bblas_side_const (char lapack_char)
- bblas_enum_t bblas_storev_const (char lapack_char)
- bblas_enum_t bblas_trans_const (char lapack_char)
- bblas_enum_t bblas_uplo_const (char lapack_char)
- static char lapack_const (int bblas_const)

3.23.1 Detailed Description

Convert LAPACK character constants to BBLAS constants. This is a one-to-many mapping, requiring multiple translators (e.g., "N" can be NoTrans or NonUnit or NoVec). Matching is case-insensitive.

3.23.2 Function Documentation

3.23.2.1 bblas_diag_const()

```
bblas_enum_t bblas_diag_const (
            char lapack_char )
```

Return values

BblasNonUnit	if lapack_char = 'N'
BblasUnit	if lapack_char = 'U'

3.23.2.2 bblas_info_const()

```
bblas_enum_t bblas_info_const (
            char lapack_char )
```

Return values

BblasErrorsReportAll	if lapack_char = 'a'
BblasErrorsReportGroup	if lapack_char = 'g'
BblasErrorsReportAny	if lapack_char = 'o'
BblasErrorsReportNone	if lapack_char = 'n'

3.23.2.3 bblas_direct_const()

```
bblas_enum_t bblas_direct_const (
             char lapack_char )
```

Return values

BblasForward	if lapack_char = 'F'
BblasBackward	if lapack_char = 'B'

3.23.2.4 bblas_norm_const()

```
bblas_enum_t bblas_norm_const (
             char lapack_char )
```

Return values

BblasOneNorm	if lapack_char = 'O|o|1'
BblasTwoNorm	if lapack_char = '2'
BblasFrobeniusNorm	if lapack_char = 'F|f|E|e'
BblasInfNorm	if lapack_char = 'I|i'
BblasMaxNorm	if lapack_char = 'M|m'

3.23.2.5 bblas_side_const()

```
bblas_enum_t bblas_side_const (
             char lapack_char )
```

Return values

BblasLeft	if lapack_char = 'L'
BblasRight	if lapack_char = 'R'

3.23.2.6 bblas_storev_const()

```
bblas_enum_t bblas_storev_const (
             char lapack_char )
```

Return values

BblasColumnwise	if lapack_char = 'C'
BblasRowwise	if lapack_char = 'R'

3.23.2.7 bblas_trans_const()

```
bblas_enum_t bblas_trans_const (
             char lapack_char )
```

Return values

BblasNoTrans	if lapack_char = 'N'
BblasTrans	if lapack_char = 'T'
BblasConjTrans	if lapack_char = 'C'

3.23.2.8 bblas_uplo_const()

```
bblas_enum_t bblas_uplo_const (
             char lapack_char )
```

Return values

BblasUpper	if lapack_char = 'U'
BblasLower	if lapack_char = 'L'
BblasGeneral	otherwise

3.23.2.9 lapack_const()

```
static char lapack_const (
             int bblas_const )  [inline], [static]
```

Return values

LAPACK	character constant corresponding to BBLAS constant