ReproBLAS

Generated by Doxygen 1.8.11

Contents

Chapter 1

File Index

1.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 2

File Documentation

2.1 include/binned.h File Reference

binned.h defines the binned types and the lower level functions associated with their use.

```
#include <stddef.h>
#include <stdlib.h>
#include <float.h>
```

Macros

- #define DBWIDTH 40

    Binned double precision bin width.
- #define SBWIDTH 13

    Binned single precision bin width.
- #define binned_DBMAXINDEX (((DBL_MAX_EXP - DBL_MIN_EXP + DBL_MANT_DIG - 1)/DBWIDTH) - 1)

    Binned double precision maximum index.
- #define binned_SBMAXINDEX (((FLT_MAX_EXP - FLT_MIN_EXP + FLT_MANT_DIG - 1)/SBWIDTH) - 1)

    Binned single precision maximum index.
- #define binned_DBMAXFOLD (binned_DBMAXINDEX + 1)

    The maximum double precision fold supported by the library.
- #define binned_SBMAXFOLD (binned_SBMAXINDEX + 1)

    The maximum single precision fold supported by the library.
- #define binned_DBENDURANCE (1 << (DBL_MANT_DIG - DBWIDTH - 2))

    Binned double precision deposit endurance.
- #define binned_SBENDURANCE (1 << (FLT_MANT_DIG - SBWIDTH - 2))

    Binned single precision deposit endurance.
- #define binned_DBCAPACITY (binned_DBENDURANCE*(1.0/DBL_EPSILON - 1.0))

    Binned double precision capacity.
- #define binned_SBCAPACITY (binned_SBENDURANCE*(1.0/FLT_EPSILON - 1.0))

    Binned single precision capacity.
- #define binned_DMCOMPRESSION (1.0/(1 << (DBL_MANT_DIG - DBWIDTH + 1)))

    Binned double precision compression factor.
- #define binned_SMCOMPRESSION (1.0/(1 << (FLT_MANT_DIG - SBWIDTH + 1)))

    Binned single precision compression factor.
- #define binned_DMEXPANSION (1.0*(1 << (DBL_MANT_DIG - DBWIDTH + 1)))

    Binned double precision expansion factor.
- #define binned_SMEXPANSION (1.0*(1 << (FLT_MANT_DIG - SBWIDTH + 1)))

    Binned single precision expansion factor.

Typedefs

- typedef double double_binned

    The binned double datatype.
- typedef double double_complex_binned

    The binned complex double datatype.
- typedef float float_binned

    The binned float datatype.
- typedef float float_complex_binned

    The binned complex float datatype.

Functions

- size_t binned_dbsize (const int fold)

    binned double precision size
- size_t binned_zbsize (const int fold)

    binned complex double precision size
- size_t binned_sbsbze (const int fold)

    binned single precision size
- size_t binned_cbsize (const int fold)

    binned complex single precision size
- double_binned ∗ binned_dballoc (const int fold)

    binned double precision allocation
- double_complex_binned ∗ binned_zballoc (const int fold)

    binned complex double precision allocation
- float_binned ∗ binned_sballoc (const int fold)

    binned single precision allocation
- float_complex_binned ∗ binned_cballoc (const int fold)

    binned complex single precision allocation
- int binned_dbnum (const int fold)

    binned double precision size
- int binned_zbnum (const int fold)

    binned complex double precision size
- int binned_sbnum (const int fold)

    binned single precision size
- int binned_cbnum (const int fold)

    binned complex single precision size
- double binned_dbbound (const int fold, const int N, const double X, const double S)

    Get binned double precision summation error bound.
- float binned_sbbound (const int fold, const int N, const float X, const float S)

    Get binned single precision summation error bound.
- const double ∗ binned_dmbins (const int X)

    Get binned double precision reference bins.
- const float ∗ binned_smbins (const int X)

    Get binned single precision reference bins.
- int binned_dindex (const double X)

    Get index of double precision.
- int binned_dmindex (const double ∗priX)

    Get index of manually specified binned double precision.
- int binned_dmindex0 (const double ∗priX)

Check if index of manually specified binned double precision is 0.
- int binned_sindex (const float X)

  Get index of single precision.
- int binned_smindex (const float *priX)

  Get index of manually specified binned single precision.
- int binned_smindex0 (const float *priX)

  Check if index of manually specified binned single precision is 0.
- int binned_dmdenorm (const int fold, const double *priX)

  Check if binned type has denormal bits.
- int binned_zmdenorm (const int fold, const double *priX)

  Check if binned type has denormal bits.
- int binned_smdenorm (const int fold, const float *priX)

  Check if binned type has denormal bits.
- int binned_cmdenorm (const int fold, const float *priX)

  Check if binned type has denormal bits.
- void binned_dbprint (const int fold, const double_binned *X)

  Print binned double precision.
- void binned_dmprint (const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX)

  Print manually specified binned double precision.
- void binned_zbprint (const int fold, const double_complex_binned *X)

  Print binned complex double precision.
- void binned_zmprint (const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX)

  Print manually specified binned complex double precision.
- void binned_sbprint (const int fold, const float_binned *X)

  Print binned single precision.
- void binned_smprint (const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX)

  Print manually specified binned single precision.
- void binned_cbprint (const int fold, const float_complex_binned *X)

  Print binned complex single precision.
- void binned_cmprint (const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX)

  Print manually specified binned complex single precision.
- void binned_dbdbset (const int fold, const double_binned *X, double_binned *Y)

  Set binned double precision (Y = X)
- void binned_dmdmset (const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)

  Set manually specified binned double precision (Y = X)
- void binned_zbzbset (const int fold, const double_complex_binned *X, double_complex_binned *Y)

  Set binned complex double precision (Y = X)
- void binned_zmzmset (const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)

  Set manually specified binned complex double precision (Y = X)
- void binned_zbdbset (const int fold, const double_binned *X, double_complex_binned *Y)

  Set binned complex double precision to binned double precision (Y = X)
- void binned_zmdmset (const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)

  Set manually specified binned complex double precision to manually specified binned double precision (Y = X)
- void binned_sbsbset (const int fold, const float_binned *X, float_binned *Y)

  Set binned single precision (Y = X)

- void binned_smsmset (const int fold, const float ∗priX, const int incpriX, const float ∗carX, const int inccarX, float ∗priY, const int incpriY, float ∗carY, const int inccarY)

    Set manually specified binned single precision (Y = X)
- void binned_cbcbset (const int fold, const float_complex_binned ∗X, float_complex_binned ∗Y)

    Set binned complex single precision (Y = X)
- void binned_cmcmset (const int fold, const float ∗priX, const int incpriX, const float ∗carX, const int inccarX, float ∗priY, const int incpriY, float ∗carY, const int inccarY)

    Set manually specified binned complex single precision (Y = X)
- void binned_cbsbset (const int fold, const float_binned ∗X, float_complex_binned ∗Y)

    Set binned complex single precision to binned single precision (Y = X)
- void binned_cmsmset (const int fold, const float ∗priX, const int incpriX, const float ∗carX, const int inccarX, float ∗priY, const int incpriY, float ∗carY, const int inccarY)

    Set manually specified binned complex single precision to manually specified binned single precision (Y = X)
- void binned_dbsetzero (const int fold, double_binned ∗X)

    Set binned double precision to 0 (X = 0)
- void binned_dmsetzero (const int fold, double ∗priX, const int incpriX, double ∗carX, const int inccarX)

    Set manually specified binned double precision to 0 (X = 0)
- void binned_zbsetzero (const int fold, double_complex_binned ∗X)

    Set binned double precision to 0 (X = 0)
- void binned_zmsetzero (const int fold, double ∗priX, const int incpriX, double ∗carX, const int inccarX)

    Set manually specified binned complex double precision to 0 (X = 0)
- void binned_sbsetzero (const int fold, float_binned ∗X)

    Set binned single precision to 0 (X = 0)
- void binned_smsetzero (const int fold, float ∗priX, const int incpriX, float ∗carX, const int inccarX)

    Set manually specified binned single precision to 0 (X = 0)
- void binned_cbsetzero (const int fold, float_complex_binned ∗X)

    Set binned single precision to 0 (X = 0)
- void binned_cmsetzero (const int fold, float ∗priX, const int incpriX, float ∗carX, const int inccarX)

    Set manually specified binned complex single precision to 0 (X = 0)
- void binned_dbdbadd (const int fold, const double_binned ∗X, double_binned ∗Y)

    Add binned double precision (Y += X)
- void binned_dmdmadd (const int fold, const double ∗priX, const int incpriX, const double ∗carX, const int inccarX, double ∗priY, const int incpriY, double ∗carY, const int inccarY)

    Add manually specified binned double precision (Y += X)
- void binned_zbzbadd (const int fold, const double_complex_binned ∗X, double_complex_binned ∗Y)

    Add binned complex double precision (Y += X)
- void binned_zmzmadd (const int fold, const double ∗priX, const int incpriX, const double ∗carX, const int inccarX, double ∗priY, const int incpriY, double ∗carY, const int inccarY)

    Add manually specified binned complex double precision (Y += X)
- void binned_sbsbadd (const int fold, const float_binned ∗X, float_binned ∗Y)

    Add binned single precision (Y += X)
- void binned_smsmadd (const int fold, const float ∗priX, const int incpriX, const float ∗carX, const int inccarX, float ∗priY, const int incpriY, float ∗carY, const int inccarY)

    Add manually specified binned single precision (Y += X)
- void binned_cbcbadd (const int fold, const float_complex_binned ∗X, float_complex_binned ∗Y)

    Add binned complex single precision (Y += X)
- void binned_cmcmadd (const int fold, const float ∗priX, const int incpriX, const float ∗carX, const int inccarX, float ∗priY, const int incpriY, float ∗carY, const int inccarY)

    Add manually specified binned complex single precision (Y += X)
- void binned_dbdbaddv (const int fold, const int N, const double_binned ∗X, const int incX, double_binned ∗Y, const int incY)

    Add binned double precision vectors (Y += X)

- void binned_zbzbaddv (const int fold, const int N, const double_complex_binned ∗X, const int incX, double_complex_binned ∗Y, const int incY)

  Add binned complex double precision vectors (Y += X)
- void binned_sbsbaddv (const int fold, const int N, const float_binned ∗X, const int incX, float_binned ∗Y, const int incY)

  Add binned single precision vectors (Y += X)
- void binned_cbcbaddv (const int fold, const int N, const float_complex_binned ∗X, const int incX, float_complex_binned ∗Y, const int incY)

  Add binned complex single precision vectors (Y += X)
- void binned_dbdadd (const int fold, const double X, double_binned ∗Y)

  Add double precision to binned double precision (Y += X)
- void binned_dmdadd (const int fold, const double X, double ∗priY, const int incpriY, double ∗carY, const int inccarY)

  Add double precision to manually specified binned double precision (Y += X)
- void binned_zbzadd (const int fold, const void ∗X, double_complex_binned ∗Y)

  Add complex double precision to binned complex double precision (Y += X)
- void binned_zmzadd (const int fold, const void ∗X, double ∗priY, const int incpriY, double ∗carY, const int inccarY)

  Add complex double precision to manually specified binned complex double precision (Y += X)
- void binned_sbsadd (const int fold, const float X, float_binned ∗Y)

  Add single precision to binned single precision (Y += X)
- void binned_smsadd (const int fold, const float X, float ∗priY, const int incpriY, float ∗carY, const int inccarY)

  Add single precision to manually specified binned single precision (Y += X)
- void binned_cbcadd (const int fold, const void ∗X, float_complex_binned ∗Y)

  Add complex single precision to binned complex single precision (Y += X)
- void binned_cmcadd (const int fold, const void ∗X, float ∗priY, const int incpriY, float ∗carY, const int inccarY)

  Add complex single precision to manually specified binned complex single precision (Y += X)
- void binned_dbdupdate (const int fold, const double X, double_binned ∗Y)

  Update binned double precision with double precision (X -> Y)
- void binned_dmdupdate (const int fold, const double X, double ∗priY, const int incpriY, double ∗carY, const int inccarY)

  Update manually specified binned double precision with double precision (X -> Y)
- void binned_zbzupdate (const int fold, const void ∗X, double_complex_binned ∗Y)

  Update binned complex double precision with complex double precision (X -> Y)
- void binned_zmzupdate (const int fold, const void ∗X, double ∗priY, const int incpriY, double ∗carY, const int inccarY)

  Update manually specified binned complex double precision with complex double precision (X -> Y)
- void binned_zbdupdate (const int fold, const double X, double_complex_binned ∗Y)

  Update binned complex double precision with double precision (X -> Y)
- void binned_zmdupdate (const int fold, const double X, double ∗priY, const int incpriY, double ∗carY, const int inccarY)

  Update manually specified binned complex double precision with double precision (X -> Y)
- void binned_sbsupdate (const int fold, const float X, float_binned ∗Y)

  Update binned single precision with single precision (X -> Y)
- void binned_smsupdate (const int fold, const float X, float ∗priY, const int incpriY, float ∗carY, const int inccarY)

  Update manually specified binned single precision with single precision (X -> Y)
- void binned_cbcupdate (const int fold, const void ∗X, float_complex_binned ∗Y)

  Update binned complex single precision with complex single precision (X -> Y)
- void binned_cmcupdate (const int fold, const void ∗X, float ∗priY, const int incpriY, float ∗carY, const int inccarY)

  Update manually specified binned complex single precision with complex single precision (X -> Y)
- void binned_cbsupdate (const int fold, const float X, float_complex_binned ∗Y)

Update binned complex single precision with single precision (X -> Y)

- void binned_cmsupdate (const int fold, const float X, float ∗priY, const int incpriY, float ∗carY, const int inccarY)

  Update manually specified binned complex single precision with single precision (X -> Y)
- void binned_dbddeposit (const int fold, const double X, double_binned ∗Y)

  Add double precision to suitably binned binned double precision (Y += X)
- void binned_dmddeposit (const int fold, const double X, double ∗priY, const int incpriY)

  Add double precision to suitably binned manually specified binned double precision (Y += X)
- void binned_zbzdeposit (const int fold, const void ∗X, double_complex_binned ∗Y)

  Add complex double precision to suitably binned binned complex double precision (Y += X)
- void binned_zmzdeposit (const int fold, const void ∗X, double ∗priY, const int incpriY)

  Add complex double precision to suitably binned manually specified binned complex double precision (Y += X)
- void binned_sbsdeposit (const int fold, const float X, float_binned ∗Y)

  Add single precision to suitably binned binned single precision (Y += X)
- void binned_smsdeposit (const int fold, const float X, float ∗priY, const int incpriY)

  Add single precision to suitably binned manually specified binned single precision (Y += X)
- void binned_cbcdeposit (const int fold, const void ∗X, float_complex_binned ∗Y)

  Add complex single precision to suitably binned binned complex single precision (Y += X)
- void binned_cmcdeposit (const int fold, const void ∗X, float ∗priY, const int incpriY)

  Add complex single precision to suitably binned manually specified binned complex single precision (Y += X)
- void binned_dbrenorm (const int fold, double_binned ∗X)

  Renormalize binned double precision.
- void binned_dmrenorm (const int fold, double ∗priX, const int incpriX, double ∗carX, const int inccarX)

  Renormalize manually specified binned double precision.
- void binned_zbrenorm (const int fold, double_complex_binned ∗X)

  Renormalize binned complex double precision.
- void binned_zmrenorm (const int fold, double ∗priX, const int incpriX, double ∗carX, const int inccarX)

  Renormalize manually specified binned complex double precision.
- void binned_sbrenorm (const int fold, float_binned ∗X)

  Renormalize binned single precision.
- void binned_smrenorm (const int fold, float ∗priX, const int incpriX, float ∗carX, const int inccarX)

  Renormalize manually specified binned single precision.
- void binned_cbrenorm (const int fold, float_complex_binned ∗X)

  Renormalize binned complex single precision.
- void binned_cmrenorm (const int fold, float ∗priX, const int incpriX, float ∗carX, const int inccarX)

  Renormalize manually specified binned complex single precision.
- void binned_dbdconv (const int fold, const double X, double_binned ∗Y)

  Convert double precision to binned double precision (X -> Y)
- void binned_dmdconv (const int fold, const double X, double ∗priY, const int incpriY, double ∗carY, const int inccarY)

  Convert double precision to manually specified binned double precision (X -> Y)
- void binned_zbzconv (const int fold, const void ∗X, double_complex_binned ∗Y)

  Convert complex double precision to binned complex double precision (X -> Y)
- void binned_zmzconv (const int fold, const void ∗X, double ∗priY, const int incpriY, double ∗carY, const int inccarY)

  Convert complex double precision to manually specified binned complex double precision (X -> Y)
- void binned_sbsconv (const int fold, const float X, float_binned ∗Y)

  Convert single precision to binned single precision (X -> Y)
- void binned_smsconv (const int fold, const float X, float ∗priY, const int incpriY, float ∗carY, const int inccarY)

  Convert single precision to manually specified binned single precision (X -> Y)
- void binned_cbcconv (const int fold, const void ∗X, float_complex_binned ∗Y)

  Convert complex single precision to binned complex single precision (X -> Y)

- void binned_cmcconv (const int fold, const void ∗X, float ∗priY, const int incpriY, float ∗carY, const int inccarY)

    Convert complex single precision to manually specified binned complex single precision (X -> Y)
- double binned_ddbconv (const int fold, const double_binned ∗X)

    Convert binned double precision to double precision (X -> Y)
- double binned_ddmconv (const int fold, const double ∗priX, const int incpriX, const double ∗carX, const int inccarX)

    Convert manually specified binned double precision to double precision (X -> Y)
- void binned_zzbconv_sub (const int fold, const double_complex_binned ∗X, void ∗conv)

    Convert binned complex double precision to complex double precision (X -> Y)
- void binned_zzmconv_sub (const int fold, const double ∗priX, const int incpriX, const double ∗carX, const int inccarX, void ∗conv)

    Convert manually specified binned complex double precision to complex double precision (X -> Y)
- float binned_ssbconv (const int fold, const float_binned ∗X)

    Convert binned single precision to single precision (X -> Y)
- float binned_ssmconv (const int fold, const float ∗priX, const int incpriX, const float ∗carX, const int inccarX)

    Convert manually specified binned single precision to single precision (X -> Y)
- void binned_ccbconv_sub (const int fold, const float_complex_binned ∗X, void ∗conv)

    Convert binned complex single precision to complex single precision (X -> Y)
- void binned_ccmconv_sub (const int fold, const float ∗priX, const int incpriX, const float ∗carX, const int inccarX, void ∗conv)

    Convert manually specified binned complex single precision to complex single precision (X -> Y)
- void binned_dbnegate (const int fold, double_binned ∗X)

    Negate binned double precision (X = -X)
- void binned_dmnegate (const int fold, double ∗priX, const int incpriX, double ∗carX, const int inccarX)

    Negate manually specified binned double precision (X = -X)
- void binned_zbnegate (const int fold, double_complex_binned ∗X)

    Negate binned complex double precision (X = -X)
- void binned_zmnegate (const int fold, double ∗priX, const int incpriX, double ∗carX, const int inccarX)

    Negate manually specified binned complex double precision (X = -X)
- void binned_sbnegate (const int fold, float_binned ∗X)

    Negate binned single precision (X = -X)
- void binned_smnegate (const int fold, float ∗priX, const int incpriX, float ∗carX, const int inccarX)

    Negate manually specified binned single precision (X = -X)
- void binned_cbnegate (const int fold, float_complex_binned ∗X)

    Negate binned complex single precision (X = -X)
- void binned_cmnegate (const int fold, float ∗priX, const int incpriX, float ∗carX, const int inccarX)

    Negate manually specified binned complex single precision (X = -X)
- double binned_dscale (const double X)

    Get a reproducible double precision scale.
- float binned_sscale (const float X)

    Get a reproducible single precision scale.
- void binned_dmdrescale (const int fold, const double X, const double scaleY, double ∗priY, const int incpriY, double ∗carY, const int inccarY)

    rescale manually specified binned double precision sum of squares
- void binned_zmdrescale (const int fold, const double X, const double scaleY, double ∗priY, const int incpriY, double ∗carY, const int inccarY)

    rescale manually specified binned complex double precision sum of squares
- void binned_smsrescale (const int fold, const float X, const float scaleY, float ∗priY, const int incpriY, float ∗carY, const int inccarY)

    rescale manually specified binned single precision sum of squares
- void binned_cmsrescale (const int fold, const float X, const float scaleY, float ∗priY, const int incpriY, float ∗carY, const int inccarY)

rescale manually specified binned complex single precision sum of squares
- double binned_dmdmaddsq (const int fold, const double scaleX, const double ∗priX, const int incpriX, const double ∗carX, const int inccarX, const double scaleY, double ∗priY, const int incpriY, double ∗carY, const int inccarY)

  Add manually specified binned double precision scaled sums of squares (Y += X)
- double binned_dbdbaddsq (const int fold, const double scaleX, const double_binned ∗X, const double scaleY, double_binned ∗Y)

  Add binned double precision scaled sums of squares (Y += X)
- float binned_smsmaddsq (const int fold, const float scaleX, const float ∗priX, const int incpriX, const float ∗carX, const int inccarX, const float scaleY, float ∗priY, const int incpriY, float ∗carY, const int inccarY)

  Add manually specified binned single precision scaled sums of squares (Y += X)
- float binned_sbsbaddsq (const int fold, const float scaleX, const float_binned ∗X, const float scaleY, float_↩ binned ∗Y)

  Add binned single precision scaled sums of squares (Y += X)
- double binned_ufp (const double X)

  unit in the first place
- float binned_ufpf (const float X)

  unit in the first place

2.1.1 Detailed Description

binned.h defines the binned types and the lower level functions associated with their use.

This header is modeled after cblas.h, and as such functions are prefixed with character sets describing the data types they operate upon. For example, the function `dfoo` would perform the function `foo` on `double` possibly returning a `double`.

If two character sets are prefixed, the first set of characters describes the output and the second the input type. For example, the function `dzbar` would perform the function `bar` on `double complex` and return a `double`.

Such character sets are listed as follows:

- d - double (`double`)
- z - complex double (`*void`)
- s - float (`float`)
- c - complex float (`*void`)
- db - binned double (double_binned)
- zb - binned complex double (double_complex_binned)
- sb - binned float (float_binned)
- cb - binned complex float (float_complex_binned)
- dm - manually specified binned double (`double`, `double`)
- zm - manually specified binned complex double (`double`, `double`)
- sm - manually specified binned float (`float`, `float`)
- cm - manually specified binned complex float (`float`, `float`)

Throughout the library, complex types are specified via *void pointers. These routines will sometimes be suffixed by sub, to represent that a function has been made into a subroutine. This allows programmers to use whatever complex types they are already using, as long as the memory pointed to is of the form of two adjacent floating point types, the first and second representing real and imaginary components of the complex number.

The goal of using binned types is to obtain either more accurate or reproducible summation of floating point numbers. In reproducible summation, floating point numbers are split into several slices along predefined boundaries in the exponent range. The space between two boundaries is called a bin. Binned types are composed of several accumulators, each accumulating the slices in a particular bin. The accumulators correspond to the largest consecutive nonzero bins seen so far.

The parameter `fold` describes how many accumulators are used in the binned types supplied to a subroutine (an binned type with `k` accumulators is `k-fold`). The default value for this parameter can be set in config.h. If you are unsure of what value to use for `fold`, we recommend 3. Note that the `fold` of binned types must be the same for all binned types that interact with each other. Operations on more than one binned type assume all binned types being operated upon have the same `fold`. Note that the `fold` of an binned type may not be changed once the type has been allocated. A common use case would be to set the value of `fold` as a global macro in your code and supply it to all binned functions that you use.Power users of the library may find themselves wanting to manually specify the underlying primary and carry vectors of an binned type themselves. If you do not know what these are, don't worry about the manually specified binned types.

2.1.2 Macro Definition Documentation

2.1.2.1 #define binned_DBCAPACITY (binned_DBENDURANCE*(1.0/DBL_EPSILON - 1.0))

Binned double precision capacity.

The maximum number of double precision numbers that can be summed using binned double precision. Applies also to binned complex double precision.

Author

Peter Ahrens

Date

27 Apr 2015

2.1.2.2 #define binned_DBENDURANCE (1 << (DBL_MANT_DIG - DBWIDTH - 2))

Binned double precision deposit endurance.

The number of deposits that can be performed before a renorm is necessary. Applies also to binned complex double precision.

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.2.3 #define binned_DBMAXFOLD (binned_DBMAXINDEX + 1)

The maximum double precision fold supported by the library.

Author

Peter Ahrens

Date

14 Jan 2016

2.1.2.4 #define binned_DBMAXINDEX (((DBL_MAX_EXP - DBL_MIN_EXP + DBL_MANT_DIG - 1)/DBWIDTH) - 1)

Binned double precision maximum index.

maximum index (inclusive)

Author

Peter Ahrens

Date

24 Jun 2015

2.1.2.5 #define binned_DMCOMPRESSION (1.0/(1 << (DBL_MANT_DIG - DBWIDTH + 1)))

Binned double precision compression factor.

This factor is used to scale down inputs before deposition into the bin of highest index

Author

Peter Ahrens

Date

19 May 2015

2.1.2.6 #define binned_DMEXPANSION (1.0*(1 << (DBL_MANT_DIG - DBWIDTH + 1)))

Binned double precision expansion factor.

This factor is used to scale up inputs after deposition into the bin of highest index

Author

Peter Ahrens

Date

19 May 2015

2.1.2.7 #define binned_SBCAPACITY (binned_SBENDURANCE*(1.0/FLT_EPSILON - 1.0))

Binned single precision capacity.

The maximum number of single precision numbers that can be summed using binned single precision. Applies also to binned complex double precision.

Author

Peter Ahrens

Date

27 Apr 2015

2.1.2.8 #define binned_SBENDURANCE (1 << (FLT_MANT_DIG - SBWIDTH - 2))

Binned single precision deposit endurance.

The number of deposits that can be performed before a renorm is necessary. Applies also to binned complex single precision.

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.2.9 #define binned_SBMAXFOLD (binned_SBMAXINDEX + 1)

The maximum single precision fold supported by the library.

Author

Peter Ahrens

Date

14 Jan 2016

2.1.2.10 #define binned_SBMAXINDEX (((FLT_MAX_EXP - FLT_MIN_EXP + FLT_MANT_DIG - 1)/SBWIDTH) - 1)

Binned single precision maximum index.

maximum index (inclusive)

Author

Peter Ahrens

Date

24 Jun 2015

2.1.2.11 #define binned_SMCOMPRESSION (1.0/(1 $<<$ (FLT_MANT_DIG - SBWIDTH + 1)))

Binned single precision compression factor.

This factor is used to scale down inputs before deposition into the bin of highest index

Author

Peter Ahrens

Date

19 May 2015

2.1.2.12 #define binned_SMEXPANSION (1.0$*$(1 $<<$ (FLT_MANT_DIG - SBWIDTH + 1)))

Binned single precision expansion factor.

This factor is used to scale up inputs after deposition into the bin of highest index

Author

Peter Ahrens

Date

19 May 2015

2.1.2.13 #define DBWIDTH 40

Binned double precision bin width.

bin width (in bits)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.2.14 #define SBWIDTH 13

Binned single precision bin width.

bin width (in bits)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.3 Typedef Documentation

2.1.3.1 typedef double **double_binned**

The binned double datatype.

To allocate a double_binned, call binned_dballoc()

Warning

A double_binned is, under the hood, an array of `double`. Therefore, if you have defined an array of double_binned, you must index it by multiplying the index into the array by the number of underlying `double` that make up the double_binned. This number can be obtained by a call to binned_dbnum()

2.1.3.2 typedef double **double_complex_binned**

The binned complex double datatype.

To allocate a double_complex_binned, call binned_zballoc()

Warning

A double_complex_binned is, under the hood, an array of `double`. Therefore, if you have defined an array of double_complex_binned, you must index it by multiplying the index into the array by the number of underlying `double` that make up the double_complex_binned. This number can be obtained by a call to binned_zbnum()

2.1.3.3 typedef float **float_binned**

The binned float datatype.

To allocate a float_binned, call binned_sballoc()

Warning

A float_binned is, under the hood, an array of `float`. Therefore, if you have defined an array of float_binned, you must index it by multiplying the index into the array by the number of underlying `float` that make up the float_binned. This number can be obtained by a call to binned_sbnum()

2.1.3.4 typedef float **float_complex_binned**

The binned complex float datatype.

To allocate a float_complex_binned, call binned_cballoc()

Warning

A float_complex_binned is, under the hood, an array of `float`. Therefore, if you have defined an array of float_complex_binned, you must index it by multiplying the index into the array by the number of underlying `float` that make up the float_complex_binned. This number can be obtained by a call to binned_cbnum()

2.1.4 Function Documentation

2.1.4.1 **float_complex_binned*** binned_cballoc (const int *fold*)

binned complex single precision allocation

Parameters

fold	the fold of the binned type

Returns

a freshly allocated binned type. (free with `free()`)

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.2 void binned_cbcadd (const int *fold*, const void ∗ *X*, float_complex_binned ∗ *Y*)

Add complex single precision to binned complex single precision (Y += X)

Performs the operation Y += X on an binned type Y

Parameters

fold	the fold of the binned types
X	scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.3 void binned_cbcbadd (const int *fold*, const float_complex_binned ∗ *X*, float_complex_binned ∗ *Y*)

Add binned complex single precision (Y += X)

Performs the operation Y += X

Parameters

fold	the fold of the binned types
X	binned scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.4 void binned_cbcbaddv (const int *fold*, const int *N*, const float_complex_binned * *X*, const int *incX*, float_complex_binned * *Y*, const int *incY*)

Add binned complex single precision vectors (Y += X)

Performs the operation Y += X

Parameters

fold	the fold of the binned types
N	vector length
X	binned vector X
incX	X vector stride (use every incX'th element)
Y	binned vector Y
incY	Y vector stride (use every incY'th element)

Author

Peter Ahrens

Date

25 Jun 2015

2.1.4.5 void binned_cbcbset (const int *fold*, const float_complex_binned * *X*, float_complex_binned * *Y*)

Set binned complex single precision (Y = X)

Performs the operation Y = X

Parameters

fold	the fold of the binned types
X	binned scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.6 void binned_cbcconv (const int *fold*, const void * *X*, float_complex_binned * *Y*)

Convert complex single precision to binned complex single precision (X -> Y)

Parameters

fold	the fold of the binned types
X	scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.7 void binned_cbcdeposit (const int *fold*, const void * *X*, float_complex_binned * *Y*)

Add complex single precision to suitably binned binned complex single precision (Y += X)

Performs the operation Y += X on an binned type Y where the index of Y is larger than the index of X

Note

This routine was provided as a means of allowing the you to optimize your code. After you have called binned_cbcupdate() on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call binned_cbcdeposit() to deposit a maximum of binned_SBENDURANCE elements into Y before renormalizing Y with binned_cbrenorm(). After any number of successive calls of binned_cbcdeposit() on Y, you must renormalize Y with binned_cbrenorm() before using any other function on Y.

Parameters

fold	the fold of the binned types
X	scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.8 void binned_cbcupdate (const int *fold*, const void * *X*, float_complex_binned * *Y*)

Update binned complex single precision with complex single precision (X -> Y)

This method updates Y to an index suitable for adding numbers with absolute value of real and imaginary components less than absolute value of real and imaginary components of X respectively.

Parameters

fold	the fold of the binned types
X	scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.9 void binned_cbnegate (const int *fold*, float_complex_binned * *X*)

Negate binned complex single precision (X = -X)

Performs the operation X = -X

Parameters

fold	the fold of the binned types
X	binned scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.10 int binned_cbnum (const int *fold*)

binned complex single precision size

Parameters

fold	the fold of the binned type

Returns

the size (in `float`) of the binned type

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.11 void binned_cbprint (const int *fold*, const float_complex_binned ∗ *X*)

Print binned complex single precision.

Parameters

fold	the fold of the binned types
X	binned scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.12 void binned_cbrenorm (const int *fold*, float_complex_binned ∗ *X*)

Renormalize binned complex single precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

Parameters

fold	the fold of the binned types
X	binned scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.13 void binned_cbsbset (const int *fold*, const **float_binned** ∗ *X*, **float_complex_binned** ∗ *Y*)

Set binned complex single precision to binned single precision (Y = X)

Performs the operation Y = X

Parameters

fold	the fold of the binned types
X	binned scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.14 void binned_cbsetzero (const int *fold*, **float_complex_binned** ∗ *X*)

Set binned single precision to 0 (X = 0)

Performs the operation X = 0

Parameters

fold	the fold of the binned types
X	binned scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.15 size_t binned_cbsize (const int *fold*)

binned complex single precision size

Parameters

fold	the fold of the binned type

Returns

the size (in bytes) of the binned type

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.16 void binned_cbsupdate (const int *fold*, const float *X*, float_complex_binned $*$ *Y*)

Update binned complex single precision with single precision (X -> Y)

This method updates Y to an index suitable for adding numbers with absolute value less than X

Parameters

fold	the fold of the binned types
X	scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.17 void binned_ccbconv_sub (const int *fold*, const float_complex_binned $*$ *X*, void $*$ *conv*)

Convert binned complex single precision to complex single precision (X -> Y)

Parameters

fold	the fold of the binned types
X	binned scalar X
conv	scalar return

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.18 void binned_ccmconv_sub (const int *fold*, const float ∗ *priX*, const int *incpriX*, const float ∗ *carX*, const int *inccarX*, void ∗ *conv*)

Convert manually specified binned complex single precision to complex single precision (X -> Y)

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)
conv	scalar return

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.19 void binned_cmcadd (const int *fold*, const void ∗ *X*, float ∗ *priY*, const int *incpriY*, float ∗ *carY*, const int *inccarY*)

Add complex single precision to manually specified binned complex single precision (Y += X)

Performs the operation Y += X on an binned type Y

Parameters

fold	the fold of the binned types
X	scalar X
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.20 void binned_cmcconv (const int *fold*, const void ∗ *X*, float ∗ *priY*, const int *incpriY*, float ∗ *carY*, const int *inccarY*)

Convert complex single precision to manually specified binned complex single precision (X -> Y)

Parameters

fold	the fold of the binned types
X	scalar X
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.21 void binned_cmcdeposit (const int *fold*, const void ∗ *X*, float ∗ *priY*, const int *incpriY*)

Add complex single precision to suitably binned manually specified binned complex single precision (Y += X)

Performs the operation Y += X on an binned type Y where the index of Y is larger than the index of X

Note

This routine was provided as a means of allowing the you to optimize your code. After you have called binned_cmcupdate() on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call binned_cmcdeposit() to deposit a maximum of binned_SBENDURANCE elements into Y before renormalizing Y with binned_cmrenorm(). After any number of successive calls of binned_cmcdeposit() on Y, you must renormalize Y with binned_cmrenorm() before using any other function on Y.

Parameters

fold	the fold of the binned types
X	scalar X
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.22 void binned_cmcmadd (const int *fold*, const float ∗ *priX*, const int *incpriX*, const float ∗ *carX*, const int *inccarX*, float ∗ *priY*, const int *incpriY*, float ∗ *carY*, const int *inccarY*)

Add manually specified binned complex single precision (Y += X)

Performs the operation Y += X

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.23 void binned_cmcmset (const int *fold*, const float ∗ *priX*, const int *incpriX*, const float ∗ *carX*, const int *inccarX*, float ∗ *priY*, const int *incpriY*, float ∗ *carY*, const int *inccarY*)

Set manually specified binned complex single precision (Y = X)

Performs the operation Y = X

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)

Parameters

carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.24 void binned_cmcupdate (const int *fold,* const void ∗ *X,* float ∗ *priY,* const int *incpriY,* float ∗ *carY,* const int *inccarY*)

Update manually specified binned complex single precision with complex single precision (X -> Y)

This method updates Y to an index suitable for adding numbers with absolute value of real and imaginary components less than absolute value of real and imaginary components of X respectively.

Parameters

fold	the fold of the binned types
X	scalar X
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.25 int binned_cmdenorm (const int *fold,* const float ∗ *priX*)

Check if binned type has denormal bits.

A quick check to determine if calculations involving X cannot be performed with "denormals are zero"

Parameters

fold	the fold of the binned type
priX	X's primary vector

Returns

$>$0 if x has denormal bits, 0 otherwise.

Author

Peter Ahrens

Date

23 Jun 2015

2.1.4.26 void binned_cmnegate (const int *fold,* float $*$ *priX,* const int *incpriX,* float $*$ *carX,* const int *inccarX*)

Negate manually specified binned complex single precision (X = -X)

Performs the operation X = -X

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.27 void binned_cmprint (const int *fold,* const float $*$ *priX,* const int *incpriX,* const float $*$ *carX,* const int *inccarX*)

Print manually specified binned complex single precision.

Parameters

fold	the fold of the binned types

Parameters

priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.28 void binned_cmrenorm (const int *fold*, float * *priX*, const int *incpriX*, float * *carX*, const int *inccarX*)

Renormalize manually specified binned complex single precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.29 void binned_cmsetzero (const int *fold*, float * *priX*, const int *incpriX*, float * *carX*, const int *inccarX*)

Set manually specified binned complex single precision to 0 (X = 0)

Performs the operation X = 0

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.30 void binned_cmsmset (const int *fold,* const float $*$ *priX,* const int *incpriX,* const float $*$ *carX,* const int *inccarX,* float $*$ *priY,* const int *incpriY,* float $*$ *carY,* const int *inccarY*)

Set manually specified binned complex single precision to manually specified binned single precision (Y = X)

Performs the operation Y = X

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.31 void binned_cmsrescale (const int *fold*, const float *X*, const float *scaleY*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

rescale manually specified binned complex single precision sum of squares

Rescale an binned complex single precision sum of squares Y

Parameters

fold	the fold of the binned types
X	Y's new scaleY (X == binned_sscale (f) for some `float` f) (X >= scaleY)
scaleY	Y's current scaleY (scaleY == binned_sscale (f) for some `float` f) (X >= scaleY)
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Peter Ahrens

Date

19 Jun 2015

2.1.4.32 void binned_cmsupdate (const int *fold*, const float *X*, float ∗ *priY*, const int *incpriY*, float ∗ *carY*, const int *inccarY*)

Update manually specified binned complex single precision with single precision (X -> Y)

This method updates Y to an index suitable for adding numbers with absolute value less than X

Parameters

fold	the fold of the binned types
X	scalar X
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.33 double_binned∗ binned_dballoc (const int *fold*)

binned double precision allocation

Parameters

fold	the fold of the binned type

Returns

a freshly allocated binned type. (free with `free()`)

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.34 double binned_dbbound (const int *fold*, const int *N*, const double *X*, const double *S*)

Get binned double precision summation error bound.

This is a bound on the absolute error of a summation using binned types

Parameters

fold	the fold of the binned types
N	the number of double precision floating point summands
X	the summand of maximum absolute value
S	the value of the sum computed using binned types

Returns

error bound

Author

Peter Ahrens

Date

31 Jul 2015

2.1.4.35 void binned_dbdadd (const int *fold*, const double *X*, double_binned * *Y*)

Add double precision to binned double precision (Y += X)

Performs the operation Y += X on an binned type Y

Parameters

fold	the fold of the binned types
X	scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.36 void binned_dbdbadd (const int *fold*, const **double_binned** ∗ *X*, **double_binned** ∗ *Y*)

Add binned double precision (Y += X)

Performs the operation Y += X

Parameters

fold	the fold of the binned types
X	binned scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.37 double binned_dbdbaddsq (const int *fold*, const double *scaleX*, const **double_binned** ∗ *X*, const double *scaleY*, **double_binned** ∗ *Y*)

Add binned double precision scaled sums of squares (Y += X)

Performs the operation Y += X, where X and Y represent scaled sums of squares.

Parameters

fold	the fold of the binned types
scaleX	scale of X (scaleX == binned_dscale (Z) for some `double` Z)
X	binned scalar X
scaleY	scale of Y (scaleY == binned_dscale (Z) for some `double` Z)
Y	binned scalar Y

Returns

updated scale of Y

Author

Peter Ahrens

Date

2 Dec 2015

2.1.4.38 void binned_dbdbaddv (const int *fold*, const int *N*, const double_binned * *X*, const int *incX*, double_binned * *Y*, const int *incY*)

Add binned double precision vectors (Y += X)

Performs the operation Y += X

Parameters

fold	the fold of the binned types
N	vector length
X	binned vector X
incX	X vector stride (use every incX'th element)
Y	binned vector Y
incY	Y vector stride (use every incY'th element)

Author

Peter Ahrens

Date

25 Jun 2015

2.1.4.39 void binned_dbdbset (const int *fold*, const double_binned * *X*, double_binned * *Y*)

Set binned double precision (Y = X)

Performs the operation Y = X

Parameters

fold	the fold of the binned types
X	binned scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.40 void binned_dbdconv (const int *fold*, const double *X*, double_binned ∗ *Y*)

Convert double precision to binned double precision (X -> Y)

Parameters

fold	the fold of the binned types
X	scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.41 void binned_dbddeposit (const int *fold*, const double *X*, double_binned ∗ *Y*)

Add double precision to suitably binned binned double precision (Y += X)

Performs the operation Y += X on an binned type Y where the index of Y is larger than the index of X

Note

This routine was provided as a means of allowing the you to optimize your code. After you have called binned_dbdupdate() on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call binned_dbddeposit() to deposit a maximum of binned_DBENDURANCE elements into Y before renormalizing Y with binned_dbrenorm(). After any number of successive calls of binned_dbddeposit() on Y, you must renormalize Y with binned_dbrenorm() before using any other function on Y.

Parameters

fold	the fold of the binned types
X	scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.42 void binned_dbdupdate (const int *fold,* const double *X,* double_binned * *Y*)

Update binned double precision with double precision (X -> Y)

This method updates Y to an index suitable for adding numbers with absolute value less than X

Parameters

fold	the fold of the binned types
X	scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.43 void binned_dbnegate (const int *fold,* double_binned * *X*)

Negate binned double precision (X = -X)

Performs the operation X = -X

Parameters

fold	the fold of the binned types
X	binned scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.44 int binned_dbnum (const int *fold*)

binned double precision size

Parameters

fold	the fold of the binned type

Returns

the size (in `double`) of the binned type

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.45 void binned_dbprint (const int *fold*, const **double_binned** * *X*)

Print binned double precision.

Parameters

fold	the fold of the binned types
X	binned scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.46 void binned_dbrenorm (const int *fold*, **double_binned** * *X*)

Renormalize binned double precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

Parameters

fold	the fold of the binned types
X	binned scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.47 void binned_dbsetzero (const int *fold*, double_binned * *X*)

Set binned double precision to 0 (X = 0)

Performs the operation X = 0

Parameters

fold	the fold of the binned types
X	binned scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.48 size_t binned_dbsize (const int *fold*)

binned double precision size

Parameters

fold	the fold of the binned type

Returns

the size (in bytes) of the binned type

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.49 double binned_ddbconv (const int *fold*, const double_binned * *X*)

Convert binned double precision to double precision (X -> Y)

Parameters

fold	the fold of the binned types
X	binned scalar X

Returns

scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.50 double binned_ddmconv (const int *fold*, const double * *priX*, const int *incpriX*, const double * *carX*, const int *inccarX*)

Convert manually specified binned double precision to double precision (X -> Y)

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)

Returns

scalar Y

Author

Peter Ahrens

Date

31 Jul 2015

2.1.4.51 int binned_dindex (const double *X*)

Get index of double precision.

The index of a non-binned type is the smallest index an binned type would need to have to sum it reproducibly. Higher indicies correspond to smaller bins.

Parameters

X	scalar X

Returns

X's index

Author

Peter Ahrens
Hong Diep Nguyen

Date

19 Jun 2015

2.1.4.52 const double* binned_dmbins (const int *X*)

Get binned double precision reference bins.

returns a pointer to the bins corresponding to the given index

Parameters

X	index

Returns

pointer to constant double precision bins of index X

Author

Peter Ahrens
Hong Diep Nguyen

Date

19 Jun 2015

2.1.4.53 void binned_dmdadd (const int *fold*, const double *X*, double * *priY*, const int *incpriY*, double * *carY*, const int *inccarY*)

Add double precision to manually specified binned double precision (Y += X)

Performs the operation Y += X on an binned type Y

Parameters

fold	the fold of the binned types
X	scalar X
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.54 void binned_dmdconv (const int *fold*, const double *X*, double * *priY*, const int *incpriY*, double * *carY*, const int *inccarY*)

Convert double precision to manually specified binned double precision (X -> Y)

Parameters

fold	the fold of the binned types
X	scalar X
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

30 Apr 2015

2.1.4.55 void binned_dmddeposit (const int *fold*, const double *X*, double $*$ *priY*, const int *incpriY*)

Add double precision to suitably binned manually specified binned double precision (Y += X)

Performs the operation Y += X on an binned type Y where the index of Y is larger than the index of X

Note

This routine was provided as a means of allowing the you to optimize your code. After you have called binned_dmdupdate() on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call binned_dmddeposit() to deposit a maximum of binned_DBENDURANCE elements into Y before renormalizing Y with binned_dmrenorm(). After any number of successive calls of binned_dmddeposit() on Y, you must renormalize Y with binned_dmrenorm() before using any other function on Y.

Parameters

fold	the fold of the binned types
X	scalar X
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.56 int binned_dmdenorm (const int *fold*, const double $*$ *priX*)

Check if binned type has denormal bits.

A quick check to determine if calculations involving X cannot be performed with "denormals are zero"

Parameters

fold	the fold of the binned type
priX	X's primary vector

Returns

$>$0 if x has denormal bits, 0 otherwise.

Author

Peter Ahrens

Date

23 Jun 2015

2.1.4.57 void binned_dmdmadd (const int *fold*, const double * *priX*, const int *incpriX*, const double * *carX*, const int *inccarX*, double * *priY*, const int *incpriY*, double * *carY*, const int *inccarY*)

Add manually specified binned double precision (Y += X)

Performs the operation Y += X

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.58 double binned_dmdmaddsq (const int *fold*, const double *scaleX*, const double * *priX*, const int *incpriX*, const double * *carX*, const int *inccarX*, const double *scaleY*, double * *priY*, const int *incpriY*, double * *carY*, const int *inccarY*)

Add manually specified binned double precision scaled sums of squares (Y += X)

Performs the operation Y += X, where X and Y represent scaled sums of squares.

Parameters

fold	the fold of the binned types
scaleX	scale of X (scaleX == binned_dscale (Z) for some `double` Z)
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)
scaleY	scale of Y (scaleY == binned_dscale (Z) for some `double` Z)
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Returns

updated scale of Y

Author

Peter Ahrens

Date

1 Jun 2015

2.1.4.59 void binned_dmdmset (const int *fold,* const double * *priX,* const int *incpriX,* const double * *carX,* const int *inccarX,* double * *priY,* const int *incpriY,* double * *carY,* const int *inccarY*)

Set manually specified binned double precision (Y = X)

Performs the operation Y = X

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.60 void binned_dmdrescale (const int *fold,* const double *X,* const double *scaleY,* double * *priY,* const int *incpriY,* double * *carY,* const int *inccarY*)

rescale manually specified binned double precision sum of squares

Rescale an binned double precision sum of squares Y

Parameters

fold	the fold of the binned types
X	Y's new scaleY (X == binned_dscale (f) for some `double` f) (X >= scaleY)
scaleY	Y's current scaleY (scaleY == binned_dscale (f) for some `double` f) (X >= scaleY)
priY	Y's primary vector
incpriY	stride within Y's primary vector
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Peter Ahrens

Date

19 Jun 2015

2.1.4.61 void binned_dmdupdate (const int *fold*, const double *X*, double * *priY*, const int *incpriY*, double * *carY*, const int *inccarY*)

Update manually specified binned double precision with double precision (X -> Y)

This method updates Y to an index suitable for adding numbers with absolute value less than X

Parameters

fold	the fold of the binned types
X	scalar X
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

5 May 2015

2.1.4.62 int binned_dmindex (const double * *priX*)

Get index of manually specified binned double precision.

The index of an binned type is the bin that it corresponds to. Higher indicies correspond to smaller bins.

Parameters

priX	X's primary vector

Returns

X's index

Author

Peter Ahrens
Hong Diep Nguyen

Date

23 Sep 2015

2.1.4.63 int binned_dmindex0 (const double $*$ *priX*)

Check if index of manually specified binned double precision is 0.

A quick check to determine if the index is 0

Parameters

priX	X's primary vector

Returns

$>$0 if x has index 0, 0 otherwise.

Author

Peter Ahrens

Date

19 May 2015

2.1.4.64 void binned_dmnegate (const int *fold,* double $*$ *priX,* const int *incpriX,* double $*$ *carX,* const int *inccarX*)

Negate manually specified binned double precision (X = -X)

Performs the operation X = -X

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.65 void binned_dmprint (const int *fold*, const double ∗ *priX*, const int *incpriX*, const double ∗ *carX*, const int *inccarX*)

Print manually specified binned double precision.

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.66 void binned_dmrenorm (const int *fold*, double ∗ *priX*, const int *incpriX*, double ∗ *carX*, const int *inccarX*)

Renormalize manually specified binned double precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

23 Sep 2015

2.1.4.67 void binned_dmsetzero (const int *fold*, double * *priX*, const int *incpriX*, double * *carX*, const int *inccarX*)

Set manually specified binned double precision to 0 (X = 0)

Performs the operation X = 0

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.68 double binned_dscale (const double *X*)

Get a reproducible double precision scale.

For any given X, return a reproducible scaling factor Y of the form

2^(DBWIDTH * z) where z is an integer

such that

Y * 2^(-DBL_MANT_DIG - DBWIDTH - 1) < X < Y * 2\^(DBWIDTH + 2)

Parameters

X	double precision number to be scaled

Returns

reproducible scaling factor

Author

Peter Ahrens

Date

19 Jun 2015

2.1.4.69 float_binned* binned_sballoc (const int *fold*)

binned single precision allocation

Parameters

fold	the fold of the binned type

Returns

a freshly allocated binned type. (free with `free()`)

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.70 float binned_sbbound (const int *fold,* const int *N,* const float *X,* const float *S*)

Get binned single precision summation error bound.

This is a bound on the absolute error of a summation using binned types

Parameters

fold	the fold of the binned types
N	the number of single precision floating point summands
X	the summand of maximum absolute value
S	the value of the sum computed using binned types

Returns

error bound

Author

Peter Ahrens

Date

31 Jul 2015

Author

Peter Ahrens
Hong Diep Nguyen

Date

21 May 2015

2.1.4.71 void binned_sbnegate (const int *fold*, float_binned * *X*)

Negate binned single precision (X = -X)

Performs the operation X = -X

Parameters

fold	the fold of the binned types
X	binned scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.72 int binned_sbnum (const int *fold*)

binned single precision size

Parameters

fold	the fold of the binned type

Returns

the size (in `float`) of the binned type

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.73 void binned_sbprint (const int *fold*, const float_binned * *X*)

Print binned single precision.

Parameters

fold	the fold of the binned types
X	binned scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.74 void binned_sbrenorm (const int *fold*, float_binned * *X*)

Renormalize binned single precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

Parameters

fold	the fold of the binned types
X	binned scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.75 void binned_sbsadd (const int *fold*, const float *X*, float_binned * *Y*)

Add single precision to binned single precision (Y += X)

Performs the operation Y += X on an binned type Y

Parameters

fold	the fold of the binned types
X	scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.76 void binned_sbsbadd (const int *fold*, const float_binned * *X*, float_binned * *Y*)

Add binned single precision (Y += X)

Performs the operation Y += X

Parameters

fold	the fold of the binned types
X	binned scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.77 float binned_sbsbaddsq (const int *fold*, const float *scaleX*, const **float_binned** * *X*, const float *scaleY*, **float_binned** * *Y*)

Add binned single precision scaled sums of squares (Y += X)

Performs the operation Y += X, where X and Y represent scaled sums of squares.

Parameters

fold	the fold of the binned types
scaleX	scale of X (scaleX == binned_sscale (Z) for some `float` Z)
X	binned scalar X
scaleY	scale of Y (scaleY == binned_sscale (Z) for some `float` Z)
Y	binned scalar Y

Returns

updated scale of Y

Author

Peter Ahrens

Date

2 Dec 2015

2.1.4.78 void binned_sbsbaddv (const int *fold*, const int *N*, const **float_binned** * *X*, const int *incX*, **float_binned** * *Y*, const int *incY*)

Add binned single precision vectors (Y += X)

Performs the operation Y += X

Parameters

fold	the fold of the binned types
N	vector length
X	binned vector X
incX	X vector stride (use every incX'th element)
Y	binned vector Y
incY	Y vector stride (use every incY'th element)

Author

Peter Ahrens

Date

25 Jun 2015

2.1.4.79 void binned_sbsbset (const int *fold*, const float_binned * *X*, float_binned * *Y*)

Set binned single precision (Y = X)

Performs the operation Y = X

Parameters

fold	the fold of the binned types
X	binned scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.80 size_t binned_sbsbze (const int *fold*)

binned single precision size

Parameters

fold	the fold of the binned type

Returns

the size (in bytes) of the binned type

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.81 void binned_sbsconv (const int *fold*, const float *X*, float_binned * *Y*)

Convert single precision to binned single precision (X -> Y)

Parameters

fold	the fold of the binned types
X	scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.82 void binned_sbsdeposit (const int *fold*, const float *X*, float_binned * *Y*)

Add single precision to suitably binned binned single precision (Y += X)

Performs the operation Y += X on an binned type Y where the index of Y is larger than the index of X

Note

This routine was provided as a means of allowing the you to optimize your code. After you have called binned_sbsupdate() on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call binned_sbsdeposit() to deposit a maximum of binned_SBENDURANCE elements into Y before renormalizing Y with binned_sbrenorm(). After any number of successive calls of binned_sbsdeposit() on Y, you must renormalize Y with binned_sbrenorm() before using any other function on Y.

Parameters

fold	the fold of the binned types
X	scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.83 void binned_sbsetzero (const int *fold*, float_binned * *X*)

Set binned single precision to 0 (X = 0)

Performs the operation X = 0

Parameters

fold	the fold of the binned types
X	binned scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.84 void binned_sbsupdate (const int *fold*, const float *X*, float_binned * *Y*)

Update binned single precision with single precision (X -> Y)

This method updates Y to an index suitable for adding numbers with absolute value less than X

Parameters

fold	the fold of the binned types
X	scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.85 int binned_sindex (const float *X*)

Get index of single precision.

The index of a non-binned type is the smallest index an binned type would need to have to sum it reproducibly. Higher indicies correspond to smaller bins.

Parameters

X	scalar X

Returns

X's index

Author

Peter Ahrens
Hong Diep Nguyen

Date

19 Jun 2015

2.1.4.86 const float* binned_smbins (const int *X*)

Get binned single precision reference bins.

returns a pointer to the bins corresponding to the given index

Parameters

X	index

Returns

pointer to constant single precision bins of index X

Author

Peter Ahrens
Hong Diep Nguyen

Date

19 Jun 2015

2.1.4.87 int binned_smdenorm (const int *fold*, const float * *priX*)

Check if binned type has denormal bits.

A quick check to determine if calculations involving X cannot be performed with "denormals are zero"

Parameters

fold	the fold of the binned type
priX	X's primary vector

Returns

$>$0 if x has denormal bits, 0 otherwise.

Author

Peter Ahrens

Date

23 Jun 2015

2.1.4.88 int binned_smindex (const float $*$ *priX*)

Get index of manually specified binned single precision.

The index of an binned type is the bin that it corresponds to. Higher indicies correspond to smaller bins.

Parameters

priX	X's primary vector

Returns

X's index

Author

Peter Ahrens
Hong Diep Nguyen

Date

23 Sep 2015

2.1.4.89 int binned_smindex0 (const float $*$ *priX*)

Check if index of manually specified binned single precision is 0.

A quick check to determine if the index is 0

Parameters

priX	X's primary vector

Returns

$>$0 if x has index 0, 0 otherwise.

Author

Peter Ahrens

Date

19 May 2015

2.1.4.90 void binned_smnegate (const int *fold*, float $*$ *priX*, const int *incpriX*, float $*$ *carX*, const int *inccarX*)

Negate manually specified binned single precision (X = -X)

Performs the operation X = -X

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.91 void binned_smprint (const int *fold*, const float $*$ *priX*, const int *incpriX*, const float $*$ *carX*, const int *inccarX*)

Print manually specified binned single precision.

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.92 void binned_smrenorm (const int *fold*, float * *priX*, const int *incpriX*, float * *carX*, const int *inccarX*)

Renormalize manually specified binned single precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

23 Sep 2015

2.1.4.93 void binned_smsadd (const int *fold*, const float *X*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Add single precision to manually specified binned single precision (Y += X)

Performs the operation Y += X on an binned type Y

Parameters

fold	the fold of the binned types
X	scalar X
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.94 void binned_smsconv (const int *fold*, const float *X*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Convert single precision to manually specified binned single precision (X -> Y)

Parameters

fold	the fold of the binned types
X	scalar X
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

30 Apr 2015

2.1.4.95 void binned_smsdeposit (const int *fold*, const float *X*, float * *priY*, const int *incpriY*)

Add single precision to suitably binned manually specified binned single precision (Y += X)

Performs the operation Y += X on an binned type Y where the index of Y is larger than the index of X

Note

This routine was provided as a means of allowing the you to optimize your code. After you have called binned_smsupdate() on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call binned_smsdeposit() to deposit a maximum of binned_SBENDURANCE elements into Y before renormalizing Y with binned_smrenorm(). After any number of successive calls of binned_smsdeposit() on Y, you must renormalize Y with binned_smrenorm() before using any other function on Y.

Parameters

fold	the fold of the binned types
X	scalar X
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.96 void binned_smsetzero (const int *fold,* float * *priX,* const int *incpriX,* float * *carX,* const int *inccarX*)

Set manually specified binned single precision to 0 (X = 0)

Performs the operation X = 0

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.97 void binned_smsmadd (const int *fold,* const float * *priX,* const int *incpriX,* const float * *carX,* const int *inccarX,* float * *priY,* const int *incpriY,* float * *carY,* const int *inccarY*)

Add manually specified binned single precision (Y += X)

Performs the operation Y += X

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.98 float binned_smsmaddsq (const int *fold*, const float *scaleX*, const float ∗ *priX*, const int *incpriX*, const float ∗ *carX*, const int *inccarX*, const float *scaleY*, float ∗ *priY*, const int *incpriY*, float ∗ *carY*, const int *inccarY*)

Add manually specified binned single precision scaled sums of squares (Y += X)

Performs the operation Y += X, where X and Y represent scaled sums of squares.

Parameters

fold	the fold of the binned types
scaleX	scale of X (scaleX == binned_sscale (Z) for some `float` Z)
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)
scaleY	scale of Y (scaleY == binned_sscale (Z) for some `double` Z)
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Returns

updated scale of Y

Author

Peter Ahrens

Date

1 Jun 2015

2.1.4.99 void binned_smsmset (const int *fold*, const float ∗ *priX*, const int *incpriX*, const float ∗ *carX*, const int *inccarX*, float ∗ *priY*, const int *incpriY*, float ∗ *carY*, const int *inccarY*)

Set manually specified binned single precision (Y = X)

Performs the operation Y = X

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.100 void binned_smsrescale (const int *fold*, const float *X*, const float *scaleY*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

rescale manually specified binned single precision sum of squares

Rescale an binned single precision sum of squares Y

Parameters

fold	the fold of the binned types
X	Y's new scaleY (X == binned_sscale (f) for some `float` f) (X >= scaleY)
scaleY	Y's current scaleY (scaleY == binned_sscale (f) for some `float` f) (X >= scaleY)
priY	Y's primary vector
incpriY	stride within Y's primary vector
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Peter Ahrens

Date

1 Jun 2015

2.1.4.101 void binned_smsupdate (const int *fold,* const float *X,* float * *priY,* const int *incpriY,* float * *carY,* const int *inccarY*)

Update manually specified binned single precision with single precision (X -> Y)

This method updates Y to an index suitable for adding numbers with absolute value less than X

Parameters

fold	the fold of the binned types
X	scalar X
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

5 May 2015

2.1.4.102 float binned_ssbconv (const int *fold*, const **float_binned** ∗ ***X***)

Convert binned single precision to single precision (X -> Y)

Parameters

fold	the fold of the binned types
X	binned scalar X

Returns

scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.103 float binned_sscale (const float ***X***)

Get a reproducible single precision scale.

For any given X, return a reproducible scaling factor Y of the form

2^(SBWIDTH ∗ z) where z is an integer

such that

Y ∗ 2^(-`FLT_MANT_DIG` - SBWIDTH - 1) < X < Y ∗ 2\^(SBWIDTH + 2)

Parameters

X	single precision number to be scaled

Returns

reproducible scaling factor

Author

Peter Ahrens

Date

19 Jun 2015

2.1.4.104 float binned_ssmconv (const int *fold,* const float ∗ *priX,* const int *incpriX,* const float ∗ *carX,* const int *inccarX*)

Convert manually specified binned single precision to single precision (X -> Y)

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)

Returns

scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.105 double binned_ufp (double *X*)

unit in the first place

This method returns just the implicit 1 in the mantissa of a `double`

Parameters

X	scalar X

Returns

unit in the first place

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.106 float binned_ufpf (float *X*)

unit in the first place

This method returns just the implicit 1 in the mantissa of a `float`

Parameters

X	scalar X

Returns

unit in the first place

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.107 double_complex_binned* binned_zballoc (const int *fold*)

binned complex double precision allocation

Parameters

fold	the fold of the binned type

Returns

a freshly allocated binned type. (free with `free()`)

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.108 void binned_zbdbset (const int *fold*, const double_binned * *X*, double_complex_binned * *Y*)

Set binned complex double precision to binned double precision (Y = X)

Performs the operation Y = X

Parameters

fold	the fold of the binned types
X	binned scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.109 void binned_zbdupdate (const int *fold*, const double *X*, double_complex_binned * *Y*)

Update binned complex double precision with double precision (X -> Y)

This method updates Y to an index suitable for adding numbers with absolute value less than X

Parameters

fold	the fold of the binned types
X	scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.110 void binned_zbnegate (const int *fold,* double_complex_binned * *X*)

Negate binned complex double precision (X = -X)

Performs the operation X = -X

Parameters

fold	the fold of the binned types
X	binned scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.111 int binned_zbnum (const int *fold*)

binned complex double precision size

Parameters

fold	the fold of the binned type

Returns

the size (in `double`) of the binned type

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.112 void binned_zbprint (const int *fold*, const double_complex_binned * *X*)

Print binned complex double precision.

Parameters

fold	the fold of the binned types
X	binned scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.113 void binned_zbrenorm (const int *fold*, double_complex_binned * *X*)

Renormalize binned complex double precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

Parameters

fold	the fold of the binned types
X	binned scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.114 void binned_zbsetzero (const int *fold*, double_complex_binned * *X*)

Set binned double precision to 0 (X = 0)

Performs the operation X = 0

Parameters

fold	the fold of the binned types
X	binned scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.115 size_t binned_zbsize (const int *fold*)

binned complex double precision size

Parameters

fold	the fold of the binned type

Returns

the size (in bytes) of the binned type

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.116 void binned_zbzadd (const int *fold*, const void ∗ *X*, double_complex_binned ∗ *Y*)

Add complex double precision to binned complex double precision (Y += X)

Performs the operation Y += X on an binned type Y

Parameters

fold	the fold of the binned types
X	scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.117 void binned_zbzbadd (const int *fold*, const double_complex_binned * *X*, double_complex_binned * *Y*)

Add binned complex double precision (Y += X)

Performs the operation Y += X

Parameters

fold	the fold of the binned types
X	binned scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.118 void binned_zbzbaddv (const int *fold*, const int *N*, const double_complex_binned * *X*, const int *incX*, double_complex_binned * *Y*, const int *incY*)

Add binned complex double precision vectors (Y += X)

Performs the operation Y += X

Parameters

fold	the fold of the binned types
N	vector length
X	binned vector X
incX	X vector stride (use every incX'th element)
Y	binned vector Y
incY	Y vector stride (use every incY'th element)

Author

Peter Ahrens

Date

25 Jun 2015

2.1.4.119 void binned_zbzbset (const int *fold*, const **double_complex_binned** ∗ *X*, **double_complex_binned** ∗ *Y*)

Set binned complex double precision (Y = X)

Performs the operation Y = X

Parameters

fold	the fold of the binned types
X	binned scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.120 void binned_zbzconv (const int *fold*, const void ∗ *X*, **double_complex_binned** ∗ *Y*)

Convert complex double precision to binned complex double precision (X -> Y)

Parameters

fold	the fold of the binned types
X	scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.121 void binned_zbzdeposit (const int *fold*, const void ∗ *X*, **double_complex_binned** ∗ *Y*)

Add complex double precision to suitably binned binned complex double precision (Y += X)

Performs the operation Y += X on an binned type Y where the index of Y is larger than the index of X

Note

This routine was provided as a means of allowing the you to optimize your code. After you have called binned_zbzupdate() on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call binned_zbzdeposit() to deposit a maximum of binned_DBENDURANCE elements into Y before renormalizing Y with binned_zbrenorm(). After any number of successive calls of binned_zbzdeposit() on Y, you must renormalize Y with binned_zbrenorm() before using any other function on Y.

Parameters

fold	the fold of the binned types
X	scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.122 void binned_zbzupdate (const int *fold*, const void * *X*, double_complex_binned * *Y*)

Update binned complex double precision with complex double precision (X -> Y)

This method updates Y to an index suitable for adding numbers with absolute value of real and imaginary components less than absolute value of real and imaginary components of X respectively.

Parameters

fold	the fold of the binned types
X	scalar X
Y	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.123 int binned_zmdenorm (const int *fold*, const double * *priX*)

Check if binned type has denormal bits.

A quick check to determine if calculations involving X cannot be performed with "denormals are zero"

Parameters

fold	the fold of the binned type
priX	X's primary vector

Returns

> $>$0 if x has denormal bits, 0 otherwise.

Author

> Peter Ahrens

Date

> 23 Jun 2015

2.1.4.124 void binned_zmdmset (const int *fold*, const double ∗ *priX*, const int *incpriX*, const double ∗ *carX*, const int *inccarX*, double ∗ *priY*, const int *incpriY*, double ∗ *carY*, const int *inccarY*)

Set manually specified binned complex double precision to manually specified binned double precision (Y = X)

Performs the operation Y = X

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

> Hong Diep Nguyen
> Peter Ahrens

Date

> 27 Apr 2015

2.1.4.125 void binned_zmdrescale (const int *fold*, const double *X*, const double *scaleY*, double ∗ *priY*, const int *incpriY*, double ∗ *carY*, const int *inccarY*)

rescale manually specified binned complex double precision sum of squares

Rescale an binned complex double precision sum of squares Y

Parameters

fold	the fold of the binned types
X	Y's new scaleY (X == binned_dscale (f) for some double f) (X >= scaleY)
scaleY	Y's current scaleY (scaleY == binned_dscale (f) for some double f) (X >= scaleY)
priY	Y's primary vector
incpriY	stride within Y's primary vector
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Peter Ahrens

Date

1 Jun 2015

2.1.4.126 void binned_zmdupdate (const int *fold*, const double *X*, double * *priY*, const int *incpriY*, double * *carY*, const int *inccarY*)

Update manually specified binned complex double precision with double precision (X -> Y)

This method updates Y to an index suitable for adding numbers with absolute value less than X

Parameters

fold	the fold of the binned types
X	scalar X
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.127 void binned_zmnegate (const int *fold*, double * *priX*, const int *incpriX*, double * *carX*, const int *inccarX*)

Negate manually specified binned complex double precision (X = -X)

Performs the operation X = -X

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.128 void binned_zmprint (const int *fold*, const double ∗ *priX*, const int *incpriX*, const double ∗ *carX*, const int *inccarX*)

Print manually specified binned complex double precision.

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.129 void binned_zmrenorm (const int *fold*, double ∗ *priX*, const int *incpriX*, double ∗ *carX*, const int *inccarX*)

Renormalize manually specified binned complex double precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.130 void binned_zmsetzero (const int *fold*, double ∗ *priX*, const int *incpriX*, double ∗ *carX*, const int *inccarX*)

Set manually specified binned complex double precision to 0 (X = 0)

Performs the operation X = 0

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.131 void binned_zmzadd (const int *fold*, const void ∗ *X*, double ∗ *priY*, const int *incpriY*, double ∗ *carY*, const int *inccarY*)

Add complex double precision to manually specified binned complex double precision (Y += X)

Performs the operation Y += X on an binned type Y

Parameters

fold	the fold of the binned types
X	scalar X
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.132 void binned_zmzconv (const int *fold*, const void * *X*, double * *priY*, const int *incpriY*, double * *carY*, const int *inccarY*)

Convert complex double precision to manually specified binned complex double precision (X -> Y)

Parameters

fold	the fold of the binned types
X	scalar X
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.133 void binned_zmzdeposit (const int *fold*, const void * *X*, double * *priY*, const int *incpriY*)

Add complex double precision to suitably binned manually specified binned complex double precision (Y += X)

Performs the operation Y += X on an binned type Y where the index of Y is larger than the index of X

Note

This routine was provided as a means of allowing the you to optimize your code. After you have called binned_zmzupdate() on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call binned_zmzdeposit() to deposit a maximum of binned_DBENDURANCE elements into Y before renormalizing Y with binned_zmrenorm(). After any number of successive calls of binned_zmzdeposit() on Y, you must renormalize Y with binned_zmrenorm() before using any other function on Y.

Parameters

fold	the fold of the binned types
X	scalar X
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.134 void binned_zmzmadd (const int *fold*, const double ∗ *priX*, const int *incpriX*, const double ∗ *carX*, const int *inccarX*, double ∗ *priY*, const int *incpriY*, double ∗ *carY*, const int *inccarY*)

Add manually specified binned complex double precision (Y += X)

Performs the operation Y += X

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.135 void binned_zmzmset (const int *fold*, const double ∗ *priX*, const int *incpriX*, const double ∗ *carX*, const int *inccarX*, double ∗ *priY*, const int *incpriY*, double ∗ *carY*, const int *inccarY*)

Set manually specified binned complex double precision (Y = X)

Performs the operation Y = X

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)

Parameters

carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.136 void binned_zmzupdate (const int *fold,* const void ∗ *X,* double ∗ *priY,* const int *incpriY,* double ∗ *carY,* const int *inccarY*)

Update manually specified binned complex double precision with complex double precision (X -> Y)

This method updates Y to an index suitable for adding numbers with absolute value of real and imaginary components less than absolute value of real and imaginary components of X respectively.

Parameters

fold	the fold of the binned types
X	scalar X
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.137 void binned_zzbconv_sub (const int *fold,* const double_complex_binned ∗ *X,* void ∗ *conv*)

Convert binned complex double precision to complex double precision (X -> Y)

Parameters

fold	the fold of the binned types
X	binned scalar X
conv	scalar return

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.138 void binned_zzmconv_sub (const int *fold*, const double ∗ *priX*, const int *incpriX*, const double ∗ *carX*, const int *inccarX*, void ∗ *conv*)

Convert manually specified binned complex double precision to complex double precision (X -> Y)

Parameters

fold	the fold of the binned types
priX	X's primary vector
incpriX	stride within X's primary vector (use every incpriX'th element)
carX	X's carry vector
inccarX	stride within X's carry vector (use every inccarX'th element)
conv	scalar return

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.2 include/binnedBLAS.h File Reference

binnedBLAS.h defines BLAS Methods that operate on binned types.

```
#include "binned.h"
#include "reproBLAS.h"
```

Functions

- float binnedBLAS_samax (const int N, const float ∗X, const int incX)

  Find maximum absolute value in vector of single precision.
- double binnedBLAS_damax (const int N, const double ∗X, const int incX)

  Find maximum absolute value in vector of double precision.
- void binnedBLAS_camax_sub (const int N, const void ∗X, const int incX, void ∗amax)

  Find maximum magnitude in vector of complex single precision.
- void binnedBLAS_zamax_sub (const int N, const void ∗X, const int incX, void ∗amax)

  Find maximum magnitude in vector of complex double precision.
- float binnedBLAS_samaxm (const int N, const float ∗X, const int incX, const float ∗Y, const int incY)

  Find maximum absolute value pairwise product between vectors of single precision.
- double binnedBLAS_damaxm (const int N, const double ∗X, const int incX, const double ∗Y, const int incY)

  Find maximum absolute value pairwise product between vectors of double precision.
- void binnedBLAS_camaxm_sub (const int N, const void ∗X, const int incX, const void ∗Y, const int incY, void ∗amaxm)

  Find maximum magnitude pairwise product between vectors of complex single precision.
- void binnedBLAS_zamaxm_sub (const int N, const void ∗X, const int incX, const void ∗Y, const int incY, void ∗amaxm)

  Find maximum magnitude pairwise product between vectors of complex double precision.
- void binnedBLAS_dbdsum (const int fold, const int N, const double ∗X, const int incX, double_binned ∗Y)

  Add to binned double precision Y the sum of double precision vector X.
- void binnedBLAS_dmdsum (const int fold, const int N, const double ∗X, const int incX, double ∗priY, const int incpriY, double ∗carY, const int inccarY)

  Add to manually specified binned double precision Y the sum of double precision vector X.
- void binnedBLAS_dbdasum (const int fold, const int N, const double ∗X, const int incX, double_binned ∗Y)

  Add to binned double precision Y the absolute sum of double precision vector X.
- void binnedBLAS_dmdasum (const int fold, const int N, const double ∗X, const int incX, double ∗priY, const int incpriY, double ∗carY, const int inccarY)

  Add to manually specified binned double precision Y the absolute sum of double precision vector X.
- double binnedBLAS_dbdssq (const int fold, const int N, const double ∗X, const int incX, const double scaleY, double_binned ∗Y)

  Add to scaled binned double precision Y the scaled sum of squares of elements of double precision vector X.
- double binnedBLAS_dmdssq (const int fold, const int N, const double ∗X, const int incX, const double scaleY, double ∗priY, const int incpriY, double ∗carY, const int inccarY)

  Add to scaled manually specified binned double precision Y the scaled sum of squares of elements of double precision vector X.
- void binnedBLAS_dbddot (const int fold, const int N, const double ∗X, const int incX, const double ∗Y, const int incY, double_binned ∗Z)

  Add to binned double precision Z the dot product of double precision vectors X and Y.
- void binnedBLAS_dmddot (const int fold, const int N, const double ∗X, const int incX, const double ∗Y, const int incY, double ∗manZ, const int incmanZ, double ∗carZ, const int inccarZ)

  Add to manually specified binned double precision Z the dot product of double precision vectors X and Y.
- void binnedBLAS_zbzsum (const int fold, const int N, const void ∗X, const int incX, double_binned ∗Y)

  Add to binned complex double precision Y the sum of complex double precision vector X.
- void binnedBLAS_zmzsum (const int fold, const int N, const void ∗X, const int incX, double ∗priY, const int incpriY, double ∗carY, const int inccarY)

  Add to manually specified binned complex double precision Y the sum of complex double precision vector X.
- void binnedBLAS_dbzasum (const int fold, const int N, const void ∗X, const int incX, double_binned ∗Y)

  Add to binned double precision Y the absolute sum of complex double precision vector X.
- void binnedBLAS_dmzasum (const int fold, const int N, const void ∗X, const int incX, double ∗priY, const int incpriY, double ∗carY, const int inccarY)

Add to manually specified binned double precision Y the absolute sum of complex double precision vector X.
- double binnedBLAS_dbzssq (const int fold, const int N, const void ∗X, const int incX, const double scaleY, double_binned ∗Y)

  Add to scaled binned double precision Y the scaled sum of squares of elements of complex double precision vector X.
- double binnedBLAS_dmzssq (const int fold, const int N, const void ∗X, const int incX, const double scaleY, double ∗priY, const int incpriY, double ∗carY, const int inccarY)

  Add to scaled manually specified binned double precision Y the scaled sum of squares of elements of complex double precision vector X.
- void binnedBLAS_zbzdotu (const int fold, const int N, const void ∗X, const int incX, const void ∗Y, const int incY, double_binned ∗Z)

  Add to binned complex double precision Z the unconjugated dot product of complex double precision vectors X and Y.
- void binnedBLAS_zmzdotu (const int fold, const int N, const void ∗X, const int incX, const void ∗Y, const int incY, double ∗manZ, const int incmanZ, double ∗carZ, const int inccarZ)

  Add to manually specified binned complex double precision Z the unconjugated dot product of complex double precision vectors X and Y.
- void binnedBLAS_zbzdotc (const int fold, const int N, const void ∗X, const int incX, const void ∗Y, const int incY, double_binned ∗Z)

  Add to binned complex double precision Z the conjugated dot product of complex double precision vectors X and Y.
- void binnedBLAS_zmzdotc (const int fold, const int N, const void ∗X, const int incX, const void ∗Y, const int incY, double ∗manZ, const int incmanZ, double ∗carZ, const int inccarZ)

  Add to manually specified binned complex double precision Z the conjugated dot product of complex double precision vectors X and Y.
- void binnedBLAS_sbssum (const int fold, const int N, const float ∗X, const int incX, float_binned ∗Y)

  Add to binned single precision Y the sum of single precision vector X.
- void binnedBLAS_smssum (const int fold, const int N, const float ∗X, const int incX, float ∗priY, const int incpriY, float ∗carY, const int inccarY)

  Add to manually specified binned single precision Y the sum of single precision vector X.
- void binnedBLAS_sbsasum (const int fold, const int N, const float ∗X, const int incX, float_binned ∗Y)

  Add to binned single precision Y the absolute sum of single precision vector X.
- void binnedBLAS_smsasum (const int fold, const int N, const float ∗X, const int incX, float ∗priY, const int incpriY, float ∗carY, const int inccarY)

  Add to manually specified binned single precision Y the absolute sum of double precision vector X.
- float binnedBLAS_sbsssq (const int fold, const int N, const float ∗X, const int incX, const float scaleY, float↩_binned ∗Y)

  Add to scaled binned single precision Y the scaled sum of squares of elements of single precision vector X.
- float binnedBLAS_smsssq (const int fold, const int N, const float ∗X, const int incX, const float scaleY, float ∗priY, const int incpriY, float ∗carY, const int inccarY)

  Add to scaled manually specified binned single precision Y the scaled sum of squares of elements of single precision vector X.
- void binnedBLAS_sbsdot (const int fold, const int N, const float ∗X, const int incX, const float ∗Y, const int incY, float_binned ∗Z)

  Add to binned single precision Z the dot product of single precision vectors X and Y.
- void binnedBLAS_smsdot (const int fold, const int N, const float ∗X, const int incX, const float ∗Y, const int incY, float ∗manZ, const int incmanZ, float ∗carZ, const int inccarZ)

  Add to manually specified binned single precision Z the dot product of single precision vectors X and Y.
- void binnedBLAS_cbcsum (const int fold, const int N, const void ∗X, const int incX, float_binned ∗Y)

  Add to binned complex single precision Y the sum of complex single precision vector X.
- void binnedBLAS_cmcsum (const int fold, const int N, const void ∗X, const int incX, float ∗priY, const int incpriY, float ∗carY, const int inccarY)

  Add to manually specified binned complex single precision Y the sum of complex single precision vector X.
- void binnedBLAS_sbcasum (const int fold, const int N, const void ∗X, const int incX, float_binned ∗Y)

  Add to binned single precision Y the absolute sum of complex single precision vector X.

- void binnedBLAS_smcasum (const int fold, const int N, const void ∗X, const int incX, float ∗priY, const int incpriY, float ∗carY, const int inccarY)

  Add to manually specified binned single precision Y the absolute sum of complex single precision vector X.
- float binnedBLAS_sbcssq (const int fold, const int N, const void ∗X, const int incX, const float scaleY, float_binned ∗Y)

  Add to scaled binned single precision Y the scaled sum of squares of elements of complex single precision vector X.
- float binnedBLAS_smcssq (const int fold, const int N, const void ∗X, const int incX, const float scaleY, float ∗priY, const int incpriY, float ∗carY, const int inccarY)

  Add to scaled manually specified binned single precision Y the scaled sum of squares of elements of complex single precision vector X.
- void binnedBLAS_cbcdotu (const int fold, const int N, const void ∗X, const int incX, const void ∗Y, const int incY, float_binned ∗Z)

  Add to binned complex single precision Z the unconjugated dot product of complex single precision vectors X and Y.
- void binnedBLAS_cmcdotu (const int fold, const int N, const void ∗X, const int incX, const void ∗Y, const int incY, float ∗manZ, const int incmanZ, float ∗carZ, const int inccarZ)

  Add to manually specified binned complex single precision Z the unconjugated dot product of complex single precision vectors X and Y.
- void binnedBLAS_cbcdotc (const int fold, const int N, const void ∗X, const int incX, const void ∗Y, const int incY, float_binned ∗Z)

  Add to binned complex single precision Z the conjugated dot product of complex single precision vectors X and Y.
- void binnedBLAS_cmcdotc (const int fold, const int N, const void ∗X, const int incX, const void ∗Y, const int incY, float ∗manZ, const int incmanZ, float ∗carZ, const int inccarZ)

  Add to manually specified binned complex single precision Z the conjugated dot product of complex single precision vectors X and Y.
- void binnedBLAS_dbdgemv (const int fold, const char Order, const char TransA, const int M, const int N, const double alpha, const double ∗A, const int lda, const double ∗X, const int incX, double_binned ∗Y, const int incY)

  Add to binned double precision vector Y the matrix-vector product of double precision matrix A and double precision vector X.
- void binnedBLAS_dbdgemm (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const double alpha, const double ∗A, const int lda, const double ∗B, const int ldb, double_binned ∗C, const int ldc)

  Add to binned double precision matrix C the matrix-matrix product of double precision matrices A and B.
- void binnedBLAS_sbsgemv (const int fold, const char Order, const char TransA, const int M, const int N, const float alpha, const float ∗A, const int lda, const float ∗X, const int incX, float_binned ∗Y, const int incY)

  Add to binned single precision vector Y the matrix-vector product of single precision matrix A and single precision vector X.
- void binnedBLAS_sbsgemm (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const float alpha, const float ∗A, const int lda, const float ∗B, const int ldb, float_binned ∗C, const int ldc)

  Add to binned single precision matrix C the matrix-matrix product of single precision matrices A and B.
- void binnedBLAS_zbzgemv (const int fold, const char Order, const char TransA, const int M, const int N, const void ∗alpha, const void ∗A, const int lda, const void ∗X, const int incX, double_complex_binned ∗Y, const int incY)

  Add to binned complex double precision vector Y the matrix-vector product of complex double precision matrix A and complex double precision vector X.
- void binnedBLAS_zbzgemm (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void ∗alpha, const void ∗A, const int lda, const void ∗B, const int ldb, double_complex_binned ∗C, const int ldc)

  Add to binned complex double precision matrix C the matrix-matrix product of complex double precision matrices A and B.
- void binnedBLAS_cbcgemv (const int fold, const char Order, const char TransA, const int M, const int N, const void ∗alpha, const void ∗A, const int lda, const void ∗X, const int incX, float_complex_binned ∗Y, const int incY)

Add to binned complex single precision vector Y the matrix-vector product of complex single precision matrix A and complex single precision vector X.

- void binnedBLAS_cbcgemm (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void ∗alpha, const void ∗A, const int lda, const void ∗B, const int ldb, float_complex_binned ∗C, const int ldc)

  Add to binned complex single precision matrix C the matrix-matrix product of complex single precision matrices A and B.

2.2.1 Detailed Description

binnedBLAS.h defines BLAS Methods that operate on binned types.

This header is modeled after cblas.h, and as such functions are prefixed with character sets describing the data types they operate upon. For example, the function `dfoo` would perform the function `foo` on `double` possibly returning a `double`.

If two character sets are prefixed, the first set of characters describes the output and the second the input type. For example, the function `dzbar` would perform the function `bar` on `double complex` and return a `double`.

Such character sets are listed as follows:

- d - double (`double`)
- z - complex double (`*void`)
- s - float (`float`)
- c - complex float (`*void`)
- db - binned double (double_binned)
- zb - binned complex double (double_complex_binned)
- sb - binned float (float_binned)
- cb - binned complex float (float_complex_binned)
- dm - manually specified binned double (`double`, `double`)
- zm - manually specified binned complex double (`double`, `double`)
- sm - manually specified binned float (`float`, `float`)
- cm - manually specified binned complex float (`float`, `float`)

Throughout the library, complex types are specified via `*void` pointers. These routines will sometimes be suffixed by sub, to represent that a function has been made into a subroutine. This allows programmers to use whatever complex types they are already using, as long as the memory pointed to is of the form of two adjacent floating point types, the first and second representing real and imaginary components of the complex number.

The goal of using binned types is to obtain either more accurate or reproducible summation of floating point numbers. In reproducible summation, floating point numbers are split into several slices along predefined boundaries in the exponent range. The space between two boundaries is called a bin. Binned types are composed of several accumulators, each accumulating the slices in a particular bin. The accumulators correspond to the largest consecutive nonzero bins seen so far.

The parameter `fold` describes how many accumulators are used in the binned types supplied to a subroutine (an binned type with `k` accumulators is `k-fold`). The default value for this parameter can be set in config.h. If you are unsure of what value to use for `fold`, we recommend 3. Note that the `fold` of binned types must be the same for all binned types that interact with each other. Operations on more than one binned type assume all binned types being operated upon have the same `fold`. Note that the `fold` of an binned type may not be changed once the type has been allocated. A common use case would be to set the value of `fold` as a global macro in your code and supply it to all binned functions that you use.Power users of the library may find themselves wanting to manually specify the underlying primary and carry vectors of an binned type themselves. If you do not know what these are, don't worry about the manually specified binned types.

2.2.2 Function Documentation

2.2.2.1 void binnedBLAS_camax_sub (const int *N*, const void * *X*, const int *incX*, void * *amax*)

Find maximum magnitude in vector of complex single precision.

Returns the magnitude of the element of maximum magnitude in an array.

Parameters

N	vector length
X	complex single precision vector
incX	X vector stride (use every incX'th element)
amax	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.2 void binnedBLAS_camaxm_sub (const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, void * *amaxm*)

Find maximum magnitude pairwise product between vectors of complex single precision.

Returns the magnitude of the pairwise product of maximum magnitude between X and Y.

Parameters

N	vector length
X	complex single precision vector
incX	X vector stride (use every incX'th element)
Y	complex single precision vector
incY	Y vector stride (use every incY'th element)
amaxm	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.3 void binnedBLAS_cbcdotc (const int *fold*, const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, float_complex_binned * *Z*)

Add to binned complex single precision Z the conjugated dot product of complex single precision vectors X and Y.

Add to Z the binned sum of the pairwise products of X and conjugated Y.

Parameters

fold	the fold of the binned types
N	vector length
X	complex single precision vector
incX	X vector stride (use every incX'th element)
Y	complex single precision vector
incY	Y vector stride (use every incY'th element)
Z	binned scalar Z

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.4 void binnedBLAS_cbcdotu (const int *fold*, const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, float_complex_binned * *Z*)

Add to binned complex single precision Z the unconjugated dot product of complex single precision vectors X and Y.

Add to Z the binned sum of the pairwise products of X and Y.

Parameters

fold	the fold of the binned types
N	vector length
X	complex single precision vector
incX	X vector stride (use every incX'th element)
Y	complex single precision vector
incY	Y vector stride (use every incY'th element)
Z	binned scalar Z

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.5 void binnedBLAS_cbcgemm (const int *fold*, const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const void ∗ *alpha*, const void ∗ *A*, const int *lda*, const void ∗ *B*, const int *ldb*, float_complex_binned ∗ *C*, const int *ldc*)

Add to binned complex single precision matrix C the matrix-matrix product of complex single precision matrices A and B.

Performs one of the matrix-matrix operations

C := alpha∗op(A)∗op(B) + C,

where op(X) is one of

op(X) = X or op(X) = X∗∗T or op(X) = X∗∗H,

alpha is a scalar, A and B are matrices with op(A) an M by K matrix and op(B) a K by N matrix, and C is an binned M by N matrix.

Parameters

fold	the fold of the binned types
Order	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
TransA	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
TransB	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
M	number of rows of matrix op(A) and of the matrix C.
N	number of columns of matrix op(B) and of the matrix C.
K	number of columns of matrix op(A) and columns of the matrix op(B).
alpha	scalar alpha
A	complex single precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
lda	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
B	complex single precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
ldb	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
C	binned complex single precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
ldc	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.6 void binnedBLAS_cbcgemv (const int *fold*, const char *Order*, const char *TransA*, const int *M*, const int *N*, const void * *alpha*, const void * *A*, const int *lda*, const void * *X*, const int *incX*, float_complex_binned * *Y*, const int *incY*)

Add to binned complex single precision vector Y the matrix-vector product of complex single precision matrix A and complex single precision vector X.

Performs one of the matrix-vector operations

y := alpha*A*x + y or y := alpha*A**T*x + y or y := alpha*A**H*x + y,

where alpha is a scalar, x is a vector, y is an binned vector, and A is an M by N matrix.

Parameters

fold	the fold of the binned types
Order	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
TransA	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
M	number of rows of matrix A
N	number of columns of matrix A
alpha	scalar alpha
A	complex single precision matrix of dimension (M, lda) in row-major or (lda, N) in column-major
lda	the first dimension of A as declared in the calling program
X	complex single precision vector of at least size N if not transposed or size M otherwise
incX	X vector stride (use every incX'th element)
Y	binned complex single precision vector Y of at least size M if not transposed or size N otherwise
incY	Y vector stride (use every incY'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.7 void binnedBLAS_cbcsum (const int *fold*, const int *N*, const void * *X*, const int *incX*, float_complex_binned * *Y*)

Add to binned complex single precision Y the sum of complex single precision vector X.

Add to Y the binned sum of X.

Parameters

fold	the fold of the binned types
N	vector length
X	complex single precision vector
incX	X vector stride (use every incX'th element)
Y	binned scalar Y

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.8 void binnedBLAS_cmcdotc (const int *fold*, const int *N*, const void ∗ *X*, const int *incX*, const void ∗ *Y*, const int *incY*, float ∗ *priZ*, const int *incpriZ*, float ∗ *carZ*, const int *inccarZ*)

Add to manually specified binned complex single precision Z the conjugated dot product of complex single precision vectors X and Y.

Add to Z the binned sum of the pairwise products of X and conjugated Y.

Parameters

fold	the fold of the binned types
N	vector length
X	complex single precision vector
incX	X vector stride (use every incX'th element)
Y	complex single precision vector
incY	Y vector stride (use every incY'th element)
priZ	Z's primary vector
incpriZ	stride within Z's primary vector (use every incpriZ'th element)
carZ	Z's carry vector
inccarZ	stride within Z's carry vector (use every inccarZ'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.9 void binnedBLAS_cmcdotu (const int *fold*, const int *N*, const void ∗ *X*, const int *incX*, const void ∗ *Y*, const int *incY*, float ∗ *priZ*, const int *incpriZ*, float ∗ *carZ*, const int *inccarZ*)

Add to manually specified binned complex single precision Z the unconjugated dot product of complex single precision vectors X and Y.

Add to Z the binned sum of the pairwise products of X and Y.

Parameters

fold	the fold of the binned types
N	vector length

Parameters

X	complex single precision vector
incX	X vector stride (use every incX'th element)
Y	complex single precision vector
incY	Y vector stride (use every incY'th element)
priZ	Z's primary vector
incpriZ	stride within Z's primary vector (use every incpriZ'th element)
carZ	Z's carry vector
inccarZ	stride within Z's carry vector (use every inccarZ'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.10 void binnedBLAS_cmcsum (const int *fold*, const int *N*, const void * *X*, const int *incX*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Add to manually specified binned complex single precision Y the sum of complex single precision vector X.

Add to Y the binned sum of X.

Parameters

fold	the fold of the binned types
fold	the fold of the binned types
N	vector length
X	complex single precision vector
incX	X vector stride (use every incX'th element)
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.11 double binnedBLAS_damax (const int *N*, const double * *X*, const int *incX*)

Find maximum absolute value in vector of double precision.

Returns the absolute value of the element of maximum absolute value in an array.

Parameters

N	vector length
X	double precision vector
incX	X vector stride (use every incX'th element)

Returns

absolute maximum value of X

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.12 double binnedBLAS_damaxm (const int *N*, const double * *X*, const int *incX*, const double * *Y*, const int *incY*)

Find maximum absolute value pairwise product between vectors of double precision.

Returns the absolute value of the pairwise product of maximum absolute value between X and Y.

Parameters

N	vector length
X	double precision vector
incX	X vector stride (use every incX'th element)
Y	double precision vector
incY	Y vector stride (use every incY'th element)

Returns

absolute maximum value multiple of X and Y

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.13 void binnedBLAS_dbdasum (const int *fold*, const int *N*, const double * *X*, const int *incX*, double_binned * *Y*)

Add to binned double precision Y the absolute sum of double precision vector X.

Add to Y the binned sum of absolute values of elements in X.

Parameters

fold	the fold of the binned types
N	vector length
X	double precision vector
incX	X vector stride (use every incX'th element)
Y	binned scalar Y

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.14 void binnedBLAS_dbddot (const int *fold*, const int *N*, const double * *X*, const int *incX*, const double * *Y*, const int *incY*, double_binned * *Z*)

Add to binned double precision Z the dot product of double precision vectors X and Y.

Add to Z the binned sum of the pairwise products of X and Y.

Parameters

fold	the fold of the binned types
N	vector length
X	double precision vector
incX	X vector stride (use every incX'th element)
Y	double precision vector
incY	Y vector stride (use every incY'th element)
Z	binned scalar Z

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.15 void binnedBLAS_dbdgemm (const int *fold*, const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const double *alpha*, const double * *A*, const int *lda*, const double * *B*, const int *ldb*, double_binned * *C*, const int *ldc*)

Add to binned double precision matrix C the matrix-matrix product of double precision matrices A and B.

Performs one of the matrix-matrix operations

C := alpha*op(A)*op(B) + C,

where op(X) is one of

op(X) = X or op(X) = X**T,

alpha is a scalar, A and B are matrices with op(A) an M by K matrix and op(B) a K by N matrix, and C is an binned M by N matrix.

Parameters

fold	the fold of the binned types
Order	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
TransA	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
TransB	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
M	number of rows of matrix op(A) and of the matrix C.
N	number of columns of matrix op(B) and of the matrix C.
K	number of columns of matrix op(A) and columns of the matrix op(B).
alpha	scalar alpha
A	double precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
lda	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
B	double precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
ldb	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
C	binned double precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
ldc	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.16 void binnedBLAS_dbdgemv (const int *fold,* const char *Order,* const char *TransA,* const int *M,* const int *N,* const double *alpha,* const double ∗ *A,* const int *lda,* const double ∗ *X,* const int *incX,* double_binned ∗ *Y,* const int *incY*)

Add to binned double precision vector Y the matrix-vector product of double precision matrix A and double precision vector X.

Performs one of the matrix-vector operations

y := alpha∗A∗x + y or y := alpha∗A∗∗T∗x + y,

where alpha is a scalar, x is a vector, y is an binned vector, and A is an M by N matrix.

Parameters

fold	the fold of the binned types
Order	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
TransA	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
M	number of rows of matrix A
N	number of columns of matrix A
alpha	scalar alpha
A	double precision matrix of dimension (M, lda) in row-major or (lda, N) in column-major
lda	the first dimension of A as declared in the calling program
X	double precision vector of at least size N if not transposed or size M otherwise
incX	X vector stride (use every incX'th element)
Y	binned double precision vector Y of at least size M if not transposed or size N otherwise
incY	Y vector stride (use every incY'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.17 double binnedBLAS_dbdssq (const int *fold,* const int *N,* const double ∗ *X,* const int *incX,* const double *scaleY,* double_binned ∗ *Y*)

Add to scaled binned double precision Y the scaled sum of squares of elements of double precision vector X.

Add to Y the scaled binned sum of the squares of each element of X. The scaling of each square is performed using binned_dscale()

Parameters

fold	the fold of the binned types
N	vector length
X	double precision vector
incX	X vector stride (use every incX'th element)
scaleY	the scaling factor of Y
Y	binned scalar Y

Returns

the new scaling factor of Y

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.18 void binnedBLAS_dbdsum (const int *fold*, const int *N*, const double * *X*, const int *incX*, double_binned * *Y*)

Add to binned double precision Y the sum of double precision vector X.

Add to Y the binned sum of X.

Parameters

fold	the fold of the binned types
N	vector length
X	double precision vector
incX	X vector stride (use every incX'th element)
Y	binned scalar Y

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.19 void binnedBLAS_dbzasum (const int *fold*, const int *N*, const void * *X*, const int *incX*, double_binned * *Y*)

Add to binned double precision Y the absolute sum of complex double precision vector X.

Add to Y the binned sum of magnitudes of elements of X.

Parameters

fold	the fold of the binned types
N	vector length
X	complex double precision vector
incX	X vector stride (use every incX'th element)
Y	binned scalar Y

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.20 double binnedBLAS_dbzssq (const int *fold*, const int *N*, const void ∗ *X*, const int *incX*, const double *scaleY*, double_binned ∗ *Y*)

Add to scaled binned double precision Y the scaled sum of squares of elements of complex double precision vector X.

Add to Y the scaled binned sum of the squares of each element of X. The scaling of each square is performed using binned_dscale()

Parameters

fold	the fold of the binned types
N	vector length
X	complex double precision vector
incX	X vector stride (use every incX'th element)
scaleY	the scaling factor of Y
Y	binned scalar Y

Returns

the new scaling factor of Y

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.21 void binnedBLAS_dmdasum (const int *fold*, const int *N*, const double ∗ *X*, const int *incX*, double ∗ *priY*, const int *incpriY*, double ∗ *carY*, const int *inccarY*)

Add to manually specified binned double precision Y the absolute sum of double precision vector X.

Add to Y the binned sum of absolute values of elements in X.

Parameters

fold	the fold of the binned types
N	vector length

Parameters

X	double precision vector
incX	X vector stride (use every incX'th element)
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.22 void binnedBLAS_dmddot (const int *fold*, const int *N*, const double ∗ *X*, const int *incX*, const double ∗ *Y*, const int *incY*, double ∗ *priZ*, const int *incpriZ*, double ∗ *carZ*, const int *inccarZ*)

Add to manually specified binned double precision Z the dot product of double precision vectors X and Y.

Add to Z the binned sum of the pairwise products of X and Y.

Parameters

fold	the fold of the binned types
N	vector length
X	double precision vector
incX	X vector stride (use every incX'th element)
Y	double precision vector
incY	Y vector stride (use every incY'th element)
priZ	Z's primary vector
incpriZ	stride within Z's primary vector (use every incpriZ'th element)
carZ	Z's carry vector
inccarZ	stride within Z's carry vector (use every inccarZ'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.23 double binnedBLAS_dmdssq (const int *fold,* const int *N,* const double ∗ *X,* const int *incX,* const double *scaleY,* double ∗ *priY,* const int *incpriY,* double ∗ *carY,* const int *inccarY*)

Add to scaled manually specified binned double precision Y the scaled sum of squares of elements of double precision vector X.

Add to Y the scaled binned sum of the squares of each element of X. The scaling of each square is performed using binned_dscale()

Parameters

fold	the fold of the binned types
N	vector length
X	double precision vector
incX	X vector stride (use every incX'th element)
scaleY	the scaling factor of Y
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Returns

the new scaling factor of Y

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.24 void binnedBLAS_dmdsum (const int *fold,* const int *N,* const double ∗ *X,* const int *incX,* double ∗ *priY,* const int *incpriY,* double ∗ *carY,* const int *inccarY*)

Add to manually specified binned double precision Y the sum of double precision vector X.

Set Y to the binned sum of X.

Parameters

fold	the fold of the binned types
N	vector length
X	double precision vector
incX	X vector stride (use every incX'th element)
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.25 void binnedBLAS_dmzasum (const int *fold*, const int *N*, const void ∗ *X*, const int *incX*, double ∗ *priY*, const int *incpriY*, double ∗ *carY*, const int *inccarY*)

Add to manually specified binned double precision Y the absolute sum of complex double precision vector X.

Add to Y the binned sum of magnitudes of elements of X.

Parameters

fold	the fold of the binned types
N	vector length
X	complex double precision vector
incX	X vector stride (use every incX'th element)
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.26 double binnedBLAS_dmzssq (const int *fold*, const int *N*, const void ∗ *X*, const int *incX*, const double *scaleY*, double ∗ *priY*, const int *incpriY*, double ∗ *carY*, const int *inccarY*)

Add to scaled manually specified binned double precision Y the scaled sum of squares of elements of complex double precision vector X.

Add to Y the scaled binned sum of the squares of each element of X. The scaling of each square is performed using binned_dscale()

Parameters

fold	the fold of the binned types
N	vector length
X	complex double precision vector
incX	X vector stride (use every incX'th element)

Parameters

scaleY	the scaling factor of Y
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Returns

the new scaling factor of Y

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.27 float binnedBLAS_samax (const int *N*, const float $*$ *X*, const int *incX*)

Find maximum absolute value in vector of single precision.

Returns the absolute value of the element of maximum absolute value in an array.

Parameters

N	vector length
X	single precision vector
incX	X vector stride (use every incX'th element)

Returns

absolute maximum value of X

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.28 float binnedBLAS_samaxm (const int *N*, const float $*$ *X*, const int *incX*, const float $*$ *Y*, const int *incY*)

Find maximum absolute value pairwise product between vectors of single precision.

Returns the absolute value of the pairwise product of maximum absolute value between X and Y.

Parameters

N	vector length
X	single precision vector
incX	X vector stride (use every incX'th element)
Y	single precision vector
incY	Y vector stride (use every incY'th element)

Returns

absolute maximum value multiple of X and Y

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.29 void binnedBLAS_sbcasum (const int *fold*, const int *N*, const void * *X*, const int *incX*, float_binned * *Y*)

Add to binned single precision Y the absolute sum of complex single precision vector X.

Add to Y the binned sum of magnitudes of elements of X.

Parameters

fold	the fold of the binned types
N	vector length
X	complex single precision vector
incX	X vector stride (use every incX'th element)
Y	binned scalar Y

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.30 float binnedBLAS_sbcssq (const int *fold*, const int *N*, const void * *X*, const int *incX*, const float *scaleY*, float_binned * *Y*)

Add to scaled binned single precision Y the scaled sum of squares of elements of complex single precision vector X.

Add to Y the scaled binned sum of the squares of each element of X. The scaling of each square is performed using binned_sscale()

Parameters

fold	the fold of the binned types
N	vector length
X	complex single precision vector
incX	X vector stride (use every incX'th element)
scaleY	the scaling factor of Y
Y	binned scalar Y

Returns

the new scaling factor of Y

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.31 void binnedBLAS_sbsasum (const int *fold*, const int *N*, const float * *X*, const int *incX*, float_binned * *Y*)

Add to binned single precision Y the absolute sum of single precision vector X.

Add to Y the binned sum of absolute values of elements in X.

Parameters

fold	the fold of the binned types
N	vector length
X	single precision vector
incX	X vector stride (use every incX'th element)
Y	binned scalar Y

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.32 void binnedBLAS_sbsdot (const int *fold*, const int *N*, const float * *X*, const int *incX*, const float * *Y*, const int *incY*, float_binned * *Z*)

Add to binned single precision Z the dot product of single precision vectors X and Y.

Add to Z the binned sum of the pairwise products of X and Y.

Parameters

fold	the fold of the binned types
N	vector length
X	single precision vector
incX	X vector stride (use every incX'th element)
Y	single precision vector
incY	Y vector stride (use every incY'th element)
Z	binned scalar Z

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.33 void binnedBLAS_sbsgemm (const int *fold,* const char *Order,* const char *TransA,* const char *TransB,* const int *M,* const int *N,* const int *K,* const float *alpha,* const float * *A,* const int *lda,* const float * *B,* const int *ldb,* float_binned * *C,* const int *ldc*)

Add to binned single precision matrix C the matrix-matrix product of single precision matrices A and B.

Performs one of the matrix-matrix operations

C := alpha*op(A)*op(B) + C,

where op(X) is one of

op(X) = X or op(X) = X**T,

alpha is a scalar, A and B are matrices with op(A) an M by K matrix and op(B) a K by N matrix, and C is an binned M by N matrix.

Parameters

fold	the fold of the binned types
Order	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
TransA	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
TransB	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
M	number of rows of matrix op(A) and of the matrix C.
N	number of columns of matrix op(B) and of the matrix C.
K	number of columns of matrix op(A) and columns of the matrix op(B).
alpha	scalar alpha
A	single precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
lda	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
B	single precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
ldb	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
C	binned single precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.34 void binnedBLAS_sbsgemv (const int *fold*, const char *Order*, const char *TransA*, const int *M*, const int *N*, const float *alpha*, const float * *A*, const int *lda*, const float * *X*, const int *incX*, float_binned * *Y*, const int *incY*)

Add to binned single precision vector Y the matrix-vector product of single precision matrix A and single precision vector X.

Performs one of the matrix-vector operations

y := alpha*A*x + y or y := alpha*A**T*x + y,

where alpha is a scalar, x is a vector, y is an binned vector, and A is an M by N matrix.

Parameters

fold	the fold of the binned types
Order	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
TransA	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
M	number of rows of matrix A
N	number of columns of matrix A
alpha	scalar alpha
A	single precision matrix of dimension (M, lda) in row-major or (lda, N) in column-major
lda	the first dimension of A as declared in the calling program
X	single precision vector of at least size N if not transposed or size M otherwise
incX	X vector stride (use every incX'th element)
Y	binned single precision vector Y of at least size M if not transposed or size N otherwise
incY	Y vector stride (use every incY'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.35 float binnedBLAS_sbsssq (const int *fold*, const int *N*, const float * *X*, const int *incX*, const float *scaleY*, float_binned * *Y*)

Add to scaled binned single precision Y the scaled sum of squares of elements of single precision vector X.

Add to Y the scaled binned sum of the squares of each element of X. The scaling of each square is performed using binned_sscale()

Parameters

fold	the fold of the binned types
N	vector length
X	single precision vector
incX	X vector stride (use every incX'th element)
scaleY	the scaling factor of Y
Y	binned scalar Y

Returns

the new scaling factor of Y

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.36 void binnedBLAS_sbssum (const int *fold*, const int *N*, const float * *X*, const int *incX*, float_binned * *Y*)

Add to binned single precision Y the sum of single precision vector X.

Add to Y the binned sum of X.

Parameters

fold	the fold of the binned types
N	vector length
X	single precision vector
incX	X vector stride (use every incX'th element)
Y	binned scalar Y

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.37 void binnedBLAS_smcasum (const int *fold*, const int *N*, const void * *X*, const int *incX*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Add to manually specified binned single precision Y the absolute sum of complex single precision vector X.

Add to Y the binned sum of magnitudes of elements of X.

Parameters

fold	the fold of the binned types
N	vector length
X	complex single precision vector
incX	X vector stride (use every incX'th element)
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.38 float binnedBLAS_smcssq (const int *fold*, const int *N*, const void ∗ *X*, const int *incX*, const float *scaleY*, float ∗ *priY*, const int *incpriY*, float ∗ *carY*, const int *inccarY*)

Add to scaled manually specified binned single precision Y the scaled sum of squares of elements of complex single precision vector X.

Add to Y the scaled binned sum of the squares of each element of X. The scaling of each square is performed using binned_sscale()

Parameters

fold	the fold of the binned types
N	vector length
X	complex single precision vector
incX	X vector stride (use every incX'th element)
scaleY	the scaling factor of Y
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Returns

the new scaling factor of Y

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.39 void binnedBLAS_smsasum (const int *fold*, const int *N*, const float * *X*, const int *incX*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Add to manually specified binned single precision Y the absolute sum of double precision vector X.

Add to Y to the binned sum of absolute values of elements in X.

Parameters

fold	the fold of the binned types
N	vector length
X	single precision vector
incX	X vector stride (use every incX'th element)
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.40 void binnedBLAS_smsdot (const int *fold*, const int *N*, const float * *X*, const int *incX*, const float * *Y*, const int *incY*, float * *priZ*, const int *incpriZ*, float * *carZ*, const int *inccarZ*)

Add to manually specified binned single precision Z the dot product of single precision vectors X and Y.

Add to Z the binned sum of the pairwise products of X and Y.

Parameters

fold	the fold of the binned types
N	vector length
X	single precision vector
incX	X vector stride (use every incX'th element)
Y	single precision vector
incY	Y vector stride (use every incY'th element)
priZ	Z's primary vector
incpriZ	stride within Z's primary vector (use every incpriZ'th element)
carZ	Z's carry vector
inccarZ	stride within Z's carry vector (use every inccarZ'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.41 float binnedBLAS_smsssq (const int *fold*, const int *N*, const float * *X*, const int *incX*, const float *scaleY*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Add to scaled manually specified binned single precision Y the scaled sum of squares of elements of single precision vector X.

Add to Y the scaled binned sum of the squares of each element of X. The scaling of each square is performed using binned_sscale()

Parameters

fold	the fold of the binned types
N	vector length
X	single precision vector
incX	X vector stride (use every incX'th element)
scaleY	the scaling factor of Y
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Returns

the new scaling factor of Y

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.42 void binnedBLAS_smssum (const int *fold*, const int *N*, const float * *X*, const int *incX*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Add to manually specified binned single precision Y the sum of single precision vector X.

Add to Y the binned sum of X.

Parameters

fold	the fold of the binned types
N	vector length
X	single precision vector
incX	X vector stride (use every incX'th element)
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.43 void binnedBLAS_zamax_sub (const int *N*, const void ∗ *X*, const int *incX*, void ∗ *amax*)

Find maximum magnitude in vector of complex double precision.

Returns the magnitude of the element of maximum magnitude in an array.

Parameters

N	vector length
X	complex double precision vector
incX	X vector stride (use every incX'th element)
amax	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.44 void binnedBLAS_zamaxm_sub (const int *N*, const void ∗ *X*, const int *incX*, const void ∗ *Y*, const int *incY*, void ∗ *amaxm*)

Find maximum magnitude pairwise product between vectors of complex double precision.

Returns the magnitude of the pairwise product of maximum magnitude between X and Y.

Parameters

N	vector length
X	complex double precision vector
incX	X vector stride (use every incX'th element)
Y	complex double precision vector
incY	Y vector stride (use every incY'th element)
amaxm	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.45 void binnedBLAS_zbzdotc (const int *fold*, const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, double_complex_binned * *Z*)

Add to binned complex double precision Z the conjugated dot product of complex double precision vectors X and Y.

Add to Z the binned sum of the pairwise products of X and conjugated Y.

Parameters

fold	the fold of the binned types
N	vector length
X	complex double precision vector
incX	X vector stride (use every incX'th element)
Y	complex double precision vector
incY	Y vector stride (use every incY'th element)
Z	scalar return Z

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.46 void binnedBLAS_zbzdotu (const int *fold*, const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, double_complex_binned * *Z*)

Add to binned complex double precision Z the unconjugated dot product of complex double precision vectors X and Y.

Add to Z the binned sum of the pairwise products of X and Y.

Parameters

fold	the fold of the binned types
N	vector length
X	complex double precision vector
incX	X vector stride (use every incX'th element)
Y	complex double precision vector
incY	Y vector stride (use every incY'th element)
Z	binned scalar Z

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.47 void binnedBLAS_zbzgemm (const int *fold*, const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const void * *alpha*, const void * *A*, const int *lda*, const void * *B*, const int *ldb*, double_complex_binned * *C*, const int *ldc*)

Add to binned complex double precision matrix C the matrix-matrix product of complex double precision matrices A and B.

Performs one of the matrix-matrix operations

C := alpha*op(A)*op(B) + C,

where op(X) is one of

op(X) = X or op(X) = X**T or op(X) = X**H,

alpha is a scalar, A and B are matrices with op(A) an M by K matrix and op(B) a K by N matrix, and C is an binned M by N matrix.

Parameters

fold	the fold of the binned types
Order	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
TransA	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
TransB	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
M	number of rows of matrix op(A) and of the matrix C.
N	number of columns of matrix op(B) and of the matrix C.
K	number of columns of matrix op(A) and columns of the matrix op(B).
alpha	scalar alpha
A	complex double precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.

Parameters

lda	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
B	complex double precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
ldb	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
C	binned complex double precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
ldc	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.48 void binnedBLAS_zbzgemv (const int *fold*, const char *Order*, const char *TransA*, const int *M*, const int *N*, const void ∗ *alpha*, const void ∗ *A*, const int *lda*, const void ∗ *X*, const int *incX*, double_complex_binned ∗ *Y*, const int *incY*)

Add to binned complex double precision vector Y the matrix-vector product of complex double precision matrix A and complex double precision vector X.

Performs one of the matrix-vector operations

y := alpha∗A∗x + y or y := alpha∗A∗∗T∗x + y or y := alpha∗A∗∗H∗x + y,

where alpha is a scalar, x is a vector, y is an binned vector, and A is an M by N matrix.

Parameters

fold	the fold of the binned types
Order	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
TransA	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
M	number of rows of matrix A
N	number of columns of matrix A
alpha	scalar alpha
A	complex double precision matrix of dimension (M, lda) in row-major or (lda, N) in column-major
lda	the first dimension of A as declared in the calling program
X	complex double precision vector of at least size N if not transposed or size M otherwise
incX	X vector stride (use every incX'th element)
Y	binned complex double precision vector Y of at least size M if not transposed or size N otherwise
incY	Y vector stride (use every incY'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.49 void binnedBLAS_zbzsum (const int *fold*, const int *N*, const void * *X*, const int *incX*, double_complex_binned * *Y*)

Add to binned complex double precision Y the sum of complex double precision vector X.

Add to Y the binned sum of X.

Parameters

fold	the fold of the binned types
N	vector length
X	complex double precision vector
incX	X vector stride (use every incX'th element)
Y	binned scalar Y

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.50 void binnedBLAS_zmzdotc (const int *fold*, const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, double * *priZ*, const int *incpriZ*, double * *carZ*, const int *inccarZ*)

Add to manually specified binned complex double precision Z the conjugated dot product of complex double precision vectors X and Y.

Add to Z the binned sum of the pairwise products of X and conjugated Y.

Parameters

fold	the fold of the binned types
N	vector length
X	complex double precision vector
incX	X vector stride (use every incX'th element)
Y	complex double precision vector
incY	Y vector stride (use every incY'th element)
priZ	Z's primary vector
incpriZ	stride within Z's primary vector (use every incpriZ'th element)
carZ	Z's carry vector
inccarZ	stride within Z's carry vector (use every inccarZ'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.51 void binnedBLAS_zmzdotu (const int *fold*, const int *N*, const void ∗ *X*, const int *incX*, const void ∗ *Y*, const int *incY*, double ∗ *priZ*, const int *incpriZ*, double ∗ *carZ*, const int *inccarZ*)

Add to manually specified binned complex double precision Z the unconjugated dot product of complex double precision vectors X and Y.

Add to Z to the binned sum of the pairwise products of X and Y.

Parameters

fold	the fold of the binned types
N	vector length
X	complex double precision vector
incX	X vector stride (use every incX'th element)
Y	complex double precision vector
incY	Y vector stride (use every incY'th element)
priZ	Z's primary vector
incpriZ	stride within Z's primary vector (use every incpriZ'th element)
carZ	Z's carry vector
inccarZ	stride within Z's carry vector (use every inccarZ'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.52 void binnedBLAS_zmzsum (const int *fold*, const int *N*, const void ∗ *X*, const int *incX*, double ∗ *priY*, const int *incpriY*, double ∗ *carY*, const int *inccarY*)

Add to manually specified binned complex double precision Y the sum of complex double precision vector X.

Add to Y the binned sum of X.

Parameters

fold	the fold of the binned types
N	vector length
X	complex double precision vector

Parameters

incX	X vector stride (use every incX'th element)
priY	Y's primary vector
incpriY	stride within Y's primary vector (use every incpriY'th element)
carY	Y's carry vector
inccarY	stride within Y's carry vector (use every inccarY'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.3 include/binnedMPI.h File Reference

binnedMPI.h defines MPI wrapper functions for binned types and the necessary functions to perform reproducible reductions.

```
#include <mpi.h>
#include "binned.h"
```

Functions

- MPI_Op binnedMPI_DBDBADD (const int fold)
  Get an MPI_OP to add binned double precision (Y += X)
- MPI_Op binnedMPI_ZBZBADD (const int fold)
  Get an MPI_OP to add binned complex double precision (Y += X)
- MPI_Op binnedMPI_SBSBADD (const int fold)
  Get an MPI_OP to add binned double precision (Y += X)
- MPI_Op binnedMPI_CBCBADD (const int fold)
  Get an MPI_OP to add binned complex single precision (Y += X)
- MPI_Op binnedMPI_DBDBADDSQ (const int fold)
  Get an MPI_OP to add binned double precision scaled sums of squares (Y += X)
- MPI_Op binnedMPI_SBSBADDSQ (const int fold)
  Get an MPI_OP to add binned single precision scaled sums of squares (Y += X)
- MPI_Datatype binnedMPI_DOUBLE_BINNED (const int fold)
  Get an MPI_DATATYPE representing binned double precision.
- MPI_Datatype binnedMPI_DOUBLE_COMPLEX_BINNED (const int fold)
  Get an MPI_DATATYPE representing binned complex double precision.
- MPI_Datatype binnedMPI_FLOAT_BINNED (const int fold)
  Get an MPI_DATATYPE representing binned single precision.
- MPI_Datatype binnedMPI_FLOAT_COMPLEX_BINNED (const int fold)
  Get an MPI_DATATYPE representing binned complex single precision.
- MPI_Datatype binnedMPI_DOUBLE_BINNED_SCALED (const int fold)
  Get an MPI_DATATYPE representing scaled binned double precision.
- MPI_Datatype binnedMPI_FLOAT_BINNED_SCALED (const int fold)
  Get an MPI_DATATYPE representing scaled binned single precision.

2.3.1 Detailed Description

binnedMPI.h defines MPI wrapper functions for binned types and the necessary functions to perform reproducible reductions.

This header is modeled after cblas.h, and as such functions are prefixed with character sets describing the data types they operate upon. For example, the function `dfoo` would perform the function `foo` on `double` possibly returning a `double`.

If two character sets are prefixed, the first set of characters describes the output and the second the input type. For example, the function `dzbar` would perform the function `bar` on `double complex` and return a `double`.

Such character sets are listed as follows:

- d - double (`double`)
- z - complex double (`*void`)
- s - float (`float`)
- c - complex float (`*void`)
- db - binned double (double_binned)
- zb - binned complex double (double_complex_binned)
- sb - binned float (float_binned)
- cb - binned complex float (float_complex_binned)

The goal of using binned types is to obtain either more accurate or reproducible summation of floating point numbers. In reproducible summation, floating point numbers are split into several slices along predefined boundaries in the exponent range. The space between two boundaries is called a bin. Binned types are composed of several accumulators, each accumulating the slices in a particular bin. The accumulators correspond to the largest consecutive nonzero bins seen so far.

The parameter `fold` describes how many accumulators are used in the binned types supplied to a subroutine (an binned type with `k` accumulators is `k-fold`). The default value for this parameter can be set in config.h. If you are unsure of what value to use for `fold`, we recommend 3. Note that the `fold` of binned types must be the same for all binned types that interact with each other. Operations on more than one binned type assume all binned types being operated upon have the same `fold`. Note that the `fold` of an binned type may not be changed once the type has been allocated. A common use case would be to set the value of `fold` as a global macro in your code and supply it to all binned functions that you use.

2.3.2 Function Documentation

2.3.2.1 MPI_Op binnedMPI_CBCBADD (const int *fold*)

Get an MPI_OP to add binned complex single precision (Y += X)

Creates (if it has not already been created) and returns a function handle for an MPI reduction operation that performs the operation Y += X on two arrays of binned complex single precision datatypes of the specified fold. An MPI datatype handle can be created for such a datatype with binnedMPI_FLOAT_COMPLEX_BINNED.

This method may call `MPI_Op_create()`. If there is an error, this method will call `MPI_Abort()`.

Parameters

fold	the fold of the binned types

Author

Peter Ahrens

Date

18 Jun 2016

2.3.2.2 MPI_Op binnedMPI_DBDBADD (const int *fold*)

Get an MPI_OP to add binned double precision (Y += X)

Creates (if it has not already been created) and returns a function handle for an MPI reduction operation that performs the operation Y += X on two arrays of binned double precision datatypes of the specified fold. An MPI datatype handle can be created for such a datatype with binnedMPI_DOUBLE_BINNED.

This method may call `MPI_Op_create()`. If there is an error, this method will call `MPI_Abort()`.

Parameters

fold	the fold of the binned types

Author

Peter Ahrens

Date

18 Jun 2016

2.3.2.3 MPI_Op binnedMPI_DBDBADDSQ (const int *fold*)

Get an MPI_OP to add binned double precision scaled sums of squares (Y += X)

Creates (if it has not already been created) and returns a function handle for an MPI reduction operation that performs the operation Y += X where X and Y represent scaled sums of squares on two arrays of scaled binned double precision datatypes of the specified fold. An MPI datatype handle can be created for such a datatype with binnedMPI_DOUBLE_BINNED_SCALED.

This method may call `MPI_Op_create()`. If there is an error, this method will call `MPI_Abort()`.

Parameters

fold	the fold of the binned types

Author

Peter Ahrens

Date

18 Jun 2016

2.3.2.4 MPI_Datatype binnedMPI_DOUBLE_BINNED (const int *fold*)

Get an MPI_DATATYPE representing binned double precision.

Creates (if it has not already been created) and returns a datatype handle for an MPI datatype that represents an binned double precision type.

This method may call `MPI_Type_contiguous()` and `MPI_Type_commit()`. If there is an error, this method will call `MPI_Abort()`.

Parameters

fold	the fold of the binned types

Author

Peter Ahrens

Date

18 Jun 2016

2.3.2.5 MPI_Datatype binnedMPI_DOUBLE_BINNED_SCALED (const int *fold*)

Get an MPI_DATATYPE representing scaled binned double precision.

Creates (if it has not already been created) and returns a datatype handle for an MPI datatype that represents a scaled binned double precision type.

This method may call `MPI_Type_contiguous()` and `MPI_Type_commit()`. If there is an error, this method will call `MPI_Abort()`.

Parameters

fold	the fold of the binned types

Author

Peter Ahrens

Date

18 Jun 2016

2.3.2.6 MPI_Datatype binnedMPI_DOUBLE_COMPLEX_BINNED (const int *fold*)

Get an MPI_DATATYPE representing binned complex double precision.

Creates (if it has not already been created) and returns a datatype handle for an MPI datatype that represents an binned complex double precision type.

This method may call `MPI_Type_contiguous()` and `MPI_Type_commit()`. If there is an error, this method will call `MPI_Abort()`.

Parameters

fold	the fold of the binned types

Author

Peter Ahrens

Date

18 Jun 2016

2.3.2.7 MPI_Datatype binnedMPI_FLOAT_BINNED (const int *fold*)

Get an MPI_DATATYPE representing binned single precision.

Creates (if it has not already been created) and returns a datatype handle for an MPI datatype that represents a binned single precision type.

This method may call `MPI_Type_contiguous()` and `MPI_Type_commit()`. If there is an error, this method will call `MPI_Abort()`.

Parameters

fold	the fold of the binned types

Author

Peter Ahrens

Date

18 Jun 2016

2.3.2.8 MPI_Datatype binnedMPI_FLOAT_BINNED_SCALED (const int *fold*)

Get an MPI_DATATYPE representing scaled binned single precision.

Creates (if it has not already been created) and returns a datatype handle for an MPI datatype that represents a scaled binned single precision type.

This method may call `MPI_Type_contiguous()` and `MPI_Type_commit()`. If there is an error, this method will call `MPI_Abort()`.

Parameters

fold	the fold of the binned types

Author

Peter Ahrens

Date

18 Jun 2016

2.3.2.9 MPI_Datatype binnedMPI_FLOAT_COMPLEX_BINNED (const int *fold*)

Get an MPI_DATATYPE representing binned complex single precision.

Creates (if it has not already been created) and returns a datatype handle for an MPI datatype that represents a binned complex single precision type.

This method may call `MPI_Type_contiguous()` and `MPI_Type_commit()`. If there is an error, this method will call `MPI_Abort()`.

Parameters

fold	the fold of the binned types

Author

Peter Ahrens

Date

18 Jun 2016

2.3.2.10 MPI_Op binnedMPI_SBSBADD (const int *fold*)

Get an MPI_OP to add binned double precision (Y += X)

Creates (if it has not already been created) and returns a function handle for an MPI reduction operation that performs the operation Y += X on two arrays of binned double precision datatypes of the specified fold. An MPI datatype handle can be created for such a datatype with binnedMPI_FLOAT_BINNED.

This method may call `MPI_Op_create()`. If there is an error, this method will call `MPI_Abort()`.

Parameters

fold	the fold of the binned types

Author

Peter Ahrens

Date

18 Jun 2016

2.3.2.11 MPI_Op binnedMPI_SBSBADDSQ (const int *fold*)

Get an MPI_OP to add binned single precision scaled sums of squares (Y += X)

Creates (if it has not already been created) and returns a function handle for an MPI reduction operation that performs the operation Y += X where X and Y represent scaled sums of squares on two arrays of scaled binned single precision datatypes of the specified fold. An MPI datatype handle can be created for such a datatype with binnedMPI_FLOAT_BINNED_SCALED.

This method may call `MPI_Op_create()`. If there is an error, this method will call `MPI_Abort()`.

Parameters

fold	the fold of the binned types

Author

Peter Ahrens

Date

18 Jun 2016

2.3.2.12 MPI_Op binnedMPI_ZBZBADD (const int *fold*)

Get an MPI_OP to add binned complex double precision (Y += X)

Creates (if it has not already been created) and returns a function handle for an MPI reduction operation that performs the operation Y += X on two arrays of binned complex double precision datatypes of the specified fold. An MPI datatype handle can be created for such a datatype with binnedMPI_DOUBLE_COMPLEX_BINNED.

This method may call `MPI_Op_create()`. If there is an error, this method will call `MPI_Abort()`.

Parameters

fold	the fold of the binned types

Author

Peter Ahrens

Date

18 Jun 2016

2.4 include/reproBLAS.h File Reference

reproBLAS.h defines reproducible BLAS Methods.

```
#include <complex.h>
```

Functions

- double reproBLAS_rdsum (const int fold, const int N, const double ∗X, const int incX)

  Compute the reproducible sum of double precision vector X.
- double reproBLAS_rdasum (const int fold, const int N, const double ∗X, const int incX)

  Compute the reproducible absolute sum of double precision vector X.
- double reproBLAS_rdnrm2 (const int fold, const int N, const double ∗X, const int incX)

  Compute the reproducible Euclidian norm of double precision vector X.
- double reproBLAS_rddot (const int fold, const int N, const double ∗X, const int incX, const double ∗Y, const int incY)

  Compute the reproducible dot product of double precision vectors X and Y.
- float reproBLAS_rsdot (const int fold, const int N, const float ∗X, const int incX, const float ∗Y, const int incY)

  Compute the reproducible dot product of single precision vectors X and Y.
- float reproBLAS_rsasum (const int fold, const int N, const float ∗X, const int incX)

  Compute the reproducible absolute sum of single precision vector X.
- float reproBLAS_rssum (const int fold, const int N, const float ∗X, const int incX)

  Compute the reproducible sum of single precision vector X.
- float reproBLAS_rsnrm2 (const int fold, const int N, const float ∗X, const int incX)

  Compute the reproducible Euclidian norm of single precision vector X.
- void reproBLAS_rzsum_sub (const int fold, const int N, const void ∗X, int incX, void ∗sum)

  Compute the reproducible sum of complex double precision vector X.
- double reproBLAS_rdzasum (const int fold, const int N, const void ∗X, const int incX)

  Compute the reproducible absolute sum of complex double precision vector X.
- double reproBLAS_rdznrm2 (const int fold, const int N, const void ∗X, int incX)

  Compute the reproducible Euclidian norm of complex double precision vector X.
- void reproBLAS_rzdotc_sub (const int fold, const int N, const void ∗X, const int incX, const void ∗Y, const int incY, void ∗dotc)

  Compute the reproducible conjugated dot product of complex double precision vectors X and Y.
- void reproBLAS_rzdotu_sub (const int fold, const int N, const void ∗X, const int incX, const void ∗Y, const int incY, void ∗dotu)

  Compute the reproducible unconjugated dot product of complex double precision vectors X and Y.
- void reproBLAS_rcsum_sub (const int fold, const int N, const void ∗X, const int incX, void ∗sum)

  Compute the reproducible sum of complex single precision vector X.
- float reproBLAS_rscasum (const int fold, const int N, const void ∗X, const int incX)

  Compute the reproducible absolute sum of complex single precision vector X.

- float reproBLAS_rscnrm2 (const int fold, const int N, const void ∗X, const int incX)

    Compute the reproducible Euclidian norm of complex single precision vector X.
- void reproBLAS_rcdotc_sub (const int fold, const int N, const void ∗X, const int incX, const void ∗Y, const int incY, void ∗dotc)

    Compute the reproducible conjugated dot product of complex single precision vectors X and Y.
- void reproBLAS_rcdotu_sub (const int fold, const int N, const void ∗X, const int incX, const void ∗Y, const int incY, void ∗dotu)

    Compute the reproducible unconjugated dot product of complex single precision vectors X and Y.
- void reproBLAS_rdgemv (const int fold, const char Order, const char TransA, const int M, const int N, const double alpha, const double ∗A, const int lda, const double ∗X, const int incX, const double beta, double ∗Y, const int incY)

    Add to double precision vector Y the reproducible matrix-vector product of double precision matrix A and double precision vector X.
- void reproBLAS_rdgemm (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const double alpha, const double ∗A, const int lda, const double ∗B, const int ldb, const double beta, double ∗C, const int ldc)

    Add to double precision matrix C the reproducible matrix-matrix product of double precision matrices A and B.
- void reproBLAS_rsgemv (const int fold, const char Order, const char TransA, const int M, const int N, const float alpha, const float ∗A, const int lda, const float ∗X, const int incX, const float beta, float ∗Y, const int incY)

    Add to single precision vector Y the reproducible matrix-vector product of single precision matrix A and single precision vector X.
- void reproBLAS_rsgemm (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const float alpha, const float ∗A, const int lda, const float ∗B, const int ldb, const float beta, float ∗C, const int ldc)

    Add to single precision matrix C the reproducible matrix-matrix product of single precision matrices A and B.
- void reproBLAS_rzgemv (const int fold, const char Order, const char TransA, const int M, const int N, const void ∗alpha, const void ∗A, const int lda, const void ∗X, const int incX, const void ∗beta, void ∗Y, const int incY)

    Add to complex double precision vector Y the reproducible matrix-vector product of complex double precision matrix A and complex double precision vector X.
- void reproBLAS_rzgemm (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void ∗alpha, const void ∗A, const int lda, const void ∗B, const int ldb, const void ∗beta, void ∗C, const int ldc)

    Add to complex double precision matrix C the reproducible matrix-matrix product of complex double precision matrices A and B.
- void reproBLAS_rcgemv (const int fold, const char Order, const char TransA, const int M, const int N, const void ∗alpha, const void ∗A, const int lda, const void ∗X, const int incX, const void ∗beta, void ∗Y, const int incY)

    Add to complex single precision vector Y the reproducible matrix-vector product of complex single precision matrix A and complex single precision vector X.
- void reproBLAS_rcgemm (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void ∗alpha, const void ∗A, const int lda, const void ∗B, const int ldb, const void ∗beta, void ∗C, const int ldc)

    Add to complex single precision matrix C the reproducible matrix-matrix product of complex single precision matrices A and B.
- double reproBLAS_dsum (const int N, const double ∗X, const int incX)

    Compute the reproducible sum of double precision vector X.
- double reproBLAS_dasum (const int N, const double ∗X, const int incX)

    Compute the reproducible absolute sum of double precision vector X.
- double reproBLAS_dnrm2 (const int N, const double ∗X, const int incX)

    Compute the reproducible Euclidian norm of double precision vector X.
- double reproBLAS_ddot (const int N, const double ∗X, const int incX, const double ∗Y, const int incY)

    Compute the reproducible dot product of double precision vectors X and Y.
- float reproBLAS_sdot (const int N, const float ∗X, const int incX, const float ∗Y, const int incY)

Compute the reproducible dot product of single precision vectors X and Y.
- float reproBLAS_sasum (const int N, const float ∗X, const int incX)

  Compute the reproducible absolute sum of single precision vector X.
- float reproBLAS_ssum (const int N, const float ∗X, const int incX)

  Compute the reproducible sum of single precision vector X.
- float reproBLAS_snrm2 (const int N, const float ∗X, const int incX)

  Compute the reproducible Euclidian norm of single precision vector X.
- void reproBLAS_zsum_sub (const int N, const void ∗X, int incX, void ∗sum)

  Compute the reproducible sum of complex double precision vector X.
- double reproBLAS_dzasum (const int N, const void ∗X, const int incX)

  Compute the reproducible absolute sum of complex double precision vector X.
- double reproBLAS_dznrm2 (const int N, const void ∗X, int incX)

  Compute the reproducible Euclidian norm of complex double precision vector X.
- void reproBLAS_zdotc_sub (const int N, const void ∗X, const int incX, const void ∗Y, const int incY, void ∗dotc)

  Compute the reproducible conjugated dot product of complex double precision vectors X and Y.
- void reproBLAS_zdotu_sub (const int N, const void ∗X, const int incX, const void ∗Y, const int incY, void ∗dotu)

  Compute the reproducible unconjugated dot product of complex double precision vectors X and Y.
- void reproBLAS_csum_sub (const int N, const void ∗X, const int incX, void ∗sum)

  Compute the reproducible sum of complex single precision vector X.
- float reproBLAS_scasum (const int N, const void ∗X, const int incX)

  Compute the reproducible absolute sum of complex single precision vector X.
- float reproBLAS_scnrm2 (const int N, const void ∗X, const int incX)

  Compute the reproducible Euclidian norm of complex single precision vector X.
- void reproBLAS_cdotc_sub (const int N, const void ∗X, const int incX, const void ∗Y, const int incY, void ∗dotc)

  Compute the reproducible conjugated dot product of complex single precision vectors X and Y.
- void reproBLAS_cdotu_sub (const int N, const void ∗X, const int incX, const void ∗Y, const int incY, void ∗dotu)

  Compute the reproducible unconjugated dot product of complex single precision vectors X and Y.
- void reproBLAS_dgemv (const char Order, const char TransA, const int M, const int N, const double alpha, const double ∗A, const int lda, const double ∗X, const int incX, const double beta, double ∗Y, const int incY)

  Add to double precision vector Y the reproducible matrix-vector product of double precision matrix A and double precision vector X.
- void reproBLAS_dgemm (const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const double alpha, const double ∗A, const int lda, const double ∗B, const int ldb, const double beta, double ∗C, const int ldc)

  Add to double precision matrix C the reproducible matrix-matrix product of double precision matrices A and B.
- void reproBLAS_sgemv (const char Order, const char TransA, const int M, const int N, const float alpha, const float ∗A, const int lda, const float ∗X, const int incX, const float beta, float ∗Y, const int incY)

  Add to single precision vector Y the reproducible matrix-vector product of single precision matrix A and single precision vector X.
- void reproBLAS_sgemm (const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const float alpha, const float ∗A, const int lda, const float ∗B, const int ldb, const float beta, float ∗C, const int ldc)

  Add to single precision matrix C the reproducible matrix-matrix product of single precision matrices A and B.
- void reproBLAS_zgemv (const char Order, const char TransA, const int M, const int N, const void ∗alpha, const void ∗A, const int lda, const void ∗X, const int incX, const void ∗beta, void ∗Y, const int incY)

  Add to complex double precision vector Y the reproducible matrix-vector product of complex double precision matrix A and complex double precision vector X.
- void reproBLAS_zgemm (const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void ∗alpha, const void ∗A, const int lda, const void ∗B, const int ldb, const void ∗beta, void ∗C, const int ldc)

Add to complex double precision matrix C the reproducible matrix-matrix product of complex double precision matrices A and B.

- void reproBLAS_cgemv (const char Order, const char TransA, const int M, const int N, const void ∗alpha, const void ∗A, const int lda, const void ∗X, const int incX, const void ∗beta, void ∗Y, const int incY)

  Add to complex single precision vector Y the reproducible matrix-vector product of complex single precision matrix A and complex single precision vector X.
- void reproBLAS_cgemm (const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void ∗alpha, const void ∗A, const int lda, const void ∗B, const int ldb, const void ∗beta, void ∗C, const int ldc)

  Add to complex single precision matrix C the reproducible matrix-matrix product of complex single precision matrices A and B.

2.4.1 Detailed Description

reproBLAS.h defines reproducible BLAS Methods.

This header is modeled after cblas.h, and as such functions are prefixed with character sets describing the data types they operate upon. For example, the function `dfoo` would perform the function `foo` on `double` possibly returning a `double`.

If two character sets are prefixed, the first set of characters describes the output and the second the input type. For example, the function `dzbar` would perform the function `bar` on `double complex` and return a `double`.

Such character sets are listed as follows:

- d - double (`double`)
- z - complex double (∗`void`)
- s - float (`float`)
- c - complex float (∗`void`)

Throughout the library, complex types are specified via ∗`void` pointers. These routines will sometimes be suffixed by sub, to represent that a function has been made into a subroutine. This allows programmers to use whatever complex types they are already using, as long as the memory pointed to is of the form of two adjacent floating point types, the first and second representing real and imaginary components of the complex number.

The goal of using binned types is to obtain either more accurate or reproducible summation of floating point numbers. In reproducible summation, floating point numbers are split into several slices along predefined boundaries in the exponent range. The space between two boundaries is called a bin. Binned types are composed of several accumulators, each accumulating the slices in a particular bin. The accumulators correspond to the largest consecutive nonzero bins seen so far.

The parameter `fold` describes how many accumulators are used in the binned types supplied to a subroutine (an binned type with `k` accumulators is `k-fold`). The default value for this parameter can be set in config.h. If you are unsure of what value to use for `fold`, we recommend 3. Note that the `fold` of binned types must be the same for all binned types that interact with each other. Operations on more than one binned type assume all binned types being operated upon have the same `fold`. Note that the `fold` of an binned type may not be changed once the type has been allocated. A common use case would be to set the value of `fold` as a global macro in your code and supply it to all binned functions that you use.

In reproBLAS, two copies of the BLAS are provided. The functions that share the same name as their BLAS counterparts perform reproducible versions of their corresponding operations using the default fold value specified in config.h. The functions that are prefixed by the character 'r' allow the user to specify their own fold for the underlying binned types.

2.4.2 Function Documentation

2.4.2.1 void reproBLAS_cdotc_sub (const int *N*, const void ∗ *X*, const int *incX*, const void ∗ *Y*, const int *incY*, void ∗ *dotc*)

Compute the reproducible conjugated dot product of complex single precision vectors X and Y.

Return the sum of the pairwise products of X and conjugated Y.

The reproducible dot product is computed with binned types of default fold using binnedBLAS_cbcdotc()

Parameters

N	vector length
X	complex single precision vector
incX	X vector stride (use every incX'th element)
Y	complex single precision vector
incY	Y vector stride (use every incY'th element)
dotc	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.2 void reproBLAS_cdotu_sub (const int *N*, const void ∗ *X*, const int *incX*, const void ∗ *Y*, const int *incY*, void ∗ *dotu*)

Compute the reproducible unconjugated dot product of complex single precision vectors X and Y.

Return the sum of the pairwise products of X and Y.

The reproducible dot product is computed with binned types of default fold using binnedBLAS_cbcdotu()

Parameters

N	vector length
X	complex single precision vector
incX	X vector stride (use every incX'th element)
Y	complex single precision vector
incY	Y vector stride (use every incY'th element)
dotu	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.3 void reproBLAS_cgemm (const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const void ∗ *alpha*, const void ∗ *A*, const int *lda*, const void ∗ *B*, const int *ldb*, const void ∗ *beta*, void ∗ *C*, const int *ldc*)

Add to complex single precision matrix C the reproducible matrix-matrix product of complex single precision matrices A and B.

Performs one of the matrix-matrix operations

C := alpha∗op(A)∗op(B) + beta∗C,

where op(X) is one of

op(X) = X or op(X) = X∗∗T or op(X) = X∗∗H,

alpha and beta are scalars, A and B and C are matrices with op(A) an M by K matrix, op(B) a K by N matrix, and C is an M by N matrix.

The matrix-matrix product is computed using binned types of default fold with binnedBLAS_cbcgemm()

Parameters

Order	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
TransA	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
TransB	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
M	number of rows of matrix op(A) and of the matrix C.
N	number of columns of matrix op(B) and of the matrix C.
K	number of columns of matrix op(A) and columns of the matrix op(B).
alpha	scalar alpha
A	complex single precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
lda	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
B	complex single precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
ldb	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
beta	scalar beta
C	complex single precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
ldc	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.4 void reproBLAS_cgemv (const char *Order*, const char *TransA*, const int *M*, const int *N*, const void * *alpha*, const void * *A*, const int *lda*, const void * *X*, const int *incX*, const void * *beta*, void * *Y*, const int *incY*)

Add to complex single precision vector Y the reproducible matrix-vector product of complex single precision matrix A and complex single precision vector X.

Performs one of the matrix-vector operations

y := alpha*A*x + beta*y or y := alpha*A**T*x + beta*y or y := alpha*A**H*x + beta*y,

where alpha and beta are scalars, x and y are vectors, and A is an M by N matrix.

The matrix-vector product is computed using binned types of default fold with binnedBLAS_cbcgemv()

Parameters

Order	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
TransA	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
M	number of rows of matrix A
N	number of columns of matrix A
alpha	scalar alpha
A	complex single precision matrix of dimension (M, lda) in row-major or (lda, N) in column-major
lda	the first dimension of A as declared in the calling program
X	complex single precision vector of at least size N if not transposed or size M otherwise
incX	X vector stride (use every incX'th element)
beta	scalar beta
Y	complex single precision vector Y of at least size M if not transposed or size N otherwise
incY	Y vector stride (use every incY'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.5 void reproBLAS_csum_sub (const int *N*, const void * *X*, const int *incX*, void * *sum*)

Compute the reproducible sum of complex single precision vector X.

Return the sum of X.

The reproducible sum is computed with binned types of default fold using binnedBLAS_cbcsum()

Parameters

N	vector length
X	single precision vector
incX	X vector stride (use every incX'th element)
sum	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.6 double reproBLAS_dasum (const int *N*, const double ∗ *X*, const int *incX*)

Compute the reproducible absolute sum of double precision vector X.

Return the sum of absolute values of elements in X.

The reproducible absolute sum is computed with binned types of default fold using binnedBLAS_dbdasum()

Parameters

N	vector length
X	double precision vector
incX	X vector stride (use every incX'th element)

Returns

absolute sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.7 double reproBLAS_ddot (const int *N*, const double ∗ *X*, const int *incX*, const double ∗ *Y*, const int *incY*)

Compute the reproducible dot product of double precision vectors X and Y.

Return the sum of the pairwise products of X and Y.

The reproducible dot product is computed with binned types of default fold using binnedBLAS_dbddot()

Parameters

N	vector length
X	double precision vector
incX	X vector stride (use every incX'th element)
Y	double precision vector
incY	Y vector stride (use every incY'th element)

Returns

the dot product of X and Y

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.8 void reproBLAS_dgemm (const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const double *alpha*, const double ∗ *A*, const int *lda*, const double ∗ *B*, const int *ldb*, const double *beta*, double ∗ *C*, const int *ldc*)

Add to double precision matrix C the reproducible matrix-matrix product of double precision matrices A and B.

Performs one of the matrix-matrix operations

C := alpha∗op(A)∗op(B) + beta∗C,

where op(X) is one of

op(X) = X or op(X) = X∗∗T,

alpha and beta are scalars, A and B and C are matrices with op(A) an M by K matrix, op(B) a K by N matrix, and C is an M by N matrix.

The matrix-matrix product is computed using binned types of default fold with binnedBLAS_dbdgemm()

Parameters

Order	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
TransA	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
TransB	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
M	number of rows of matrix op(A) and of the matrix C.
N	number of columns of matrix op(B) and of the matrix C.
K	number of columns of matrix op(A) and columns of the matrix op(B).
alpha	scalar alpha

Parameters

A	double precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
lda	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
B	double precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
ldb	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
beta	scalar beta
C	double precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
ldc	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.9 void reproBLAS_dgemv (const char *Order,* const char *TransA,* const int *M,* const int *N,* const double *alpha,* const double ∗ *A,* const int *lda,* const double ∗ *X,* const int *incX,* const double *beta,* double ∗ *Y,* const int *incY*)

Add to double precision vector Y the reproducible matrix-vector product of double precision matrix A and double precision vector X.

Performs one of the matrix-vector operations

y := alpha∗A∗x + beta∗y or y := alpha∗A∗∗T∗x + beta∗y,

where alpha and beta are scalars, x and y are vectors, and A is an M by N matrix.

The matrix-vector product is computed using binned types of default fold with binnedBLAS_dbdgemv()

Parameters

Order	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
TransA	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
M	number of rows of matrix A
N	number of columns of matrix A
alpha	scalar alpha
A	double precision matrix of dimension (M, lda) in row-major or (lda, N) in column-major
lda	the first dimension of A as declared in the calling program
X	double precision vector of at least size N if not transposed or size M otherwise
incX	X vector stride (use every incX'th element)
beta	scalar beta
Y	double precision vector Y of at least size M if not transposed or size N otherwise
incY	Y vector stride (use every incY'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.10 double reproBLAS_dnrm2 (const int *N*, const double ∗ *X*, const int *incX*)

Compute the reproducible Euclidian norm of double precision vector X.

Return the square root of the sum of the squared elements of X.

The reproducible Euclidian norm is computed with scaled binned types of default fold using binnedBLAS_dbdssq()

Parameters

N	vector length
X	double precision vector
incX	X vector stride (use every incX'th element)

Returns

Euclidian norm of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.11 double reproBLAS_dsum (const int *N*, const double ∗ *X*, const int *incX*)

Compute the reproducible sum of double precision vector X.

Return the sum of X.

The reproducible sum is computed with binned types of default fold using binnedBLAS_dbdsum()

Parameters

N	vector length
X	double precision vector
incX	X vector stride (use every incX'th element)

Returns

sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.12 double reproBLAS_dzasum (const int *N*, const void ∗ *X*, const int *incX*)

Compute the reproducible absolute sum of complex double precision vector X.

Return the sum of magnitudes of elements of X.

The reproducible absolute sum is computed with binned types of default fold using binnedBLAS_dbzasum()

Parameters

N	vector length
X	complex double precision vector
incX	X vector stride (use every incX'th element)

Returns

absolute sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.13 double reproBLAS_dznrm2 (const int *N*, const void ∗ *X*, const int *incX*)

Compute the reproducible Euclidian norm of complex double precision vector X.

Return the square root of the sum of the squared elements of X.

The reproducible Euclidian norm is computed with scaled binned types of default fold using binnedBLAS_dbzssq()

Parameters

N	vector length
X	complex double precision vector
incX	X vector stride (use every incX'th element)

Returns

Euclidian norm of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.14 void reproBLAS_rcdotc_sub (const int *fold,* const int *N,* const void ∗ *X,* const int *incX,* const void ∗ *Y,* const int *incY,* void ∗ *dotc*)

Compute the reproducible conjugated dot product of complex single precision vectors X and Y.

Return the sum of the pairwise products of X and conjugated Y.

The reproducible dot product is computed with binned types using binnedBLAS_cbcdotc()

Parameters

fold	the fold of the binned types
N	vector length
X	complex single precision vector
incX	X vector stride (use every incX'th element)
Y	complex single precision vector
incY	Y vector stride (use every incY'th element)
dotc	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.15 void reproBLAS_rcdotu_sub (const int *fold,* const int *N,* const void ∗ *X,* const int *incX,* const void ∗ *Y,* const int *incY,* void ∗ *dotu*)

Compute the reproducible unconjugated dot product of complex single precision vectors X and Y.

Return the sum of the pairwise products of X and Y.

The reproducible dot product is computed with binned types using binnedBLAS_cbcdotu()

Parameters

fold	the fold of the binned types
N	vector length
X	complex single precision vector
incX	X vector stride (use every incX'th element)
Y	complex single precision vector
incY	Y vector stride (use every incY'th element)
dotu	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.16 void reproBLAS_rcgemm (const int *fold,* const char *Order,* const char *TransA,* const char *TransB,* const int *M,* const int *N,* const int *K,* const void ∗ *alpha,* const void ∗ *A,* const int *lda,* const void ∗ *B,* const int *ldb,* const void ∗ *beta,* void ∗ *C,* const int *ldc*)

Add to complex single precision matrix C the reproducible matrix-matrix product of complex single precision matrices A and B.

Performs one of the matrix-matrix operations

C := alpha∗op(A)∗op(B) + beta∗C,

where op(X) is one of

op(X) = X or op(X) = X∗∗T or op(X) = X∗∗H,

alpha and beta are scalars, A and B and C are matrices with op(A) an M by K matrix, op(B) a K by N matrix, and C is an M by N matrix.

The matrix-matrix product is computed using binned types with binnedBLAS_cbcgemm()

Parameters

fold	the fold of the binned types
Order	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
TransA	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
TransB	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
M	number of rows of matrix op(A) and of the matrix C.
N	number of columns of matrix op(B) and of the matrix C.
K	number of columns of matrix op(A) and columns of the matrix op(B).
alpha	scalar alpha

Parameters

A	complex single precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
lda	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
B	complex single precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
ldb	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
beta	scalar beta
C	complex single precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
ldc	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.17 void reproBLAS_rcgemv (const int *fold*, const char *Order*, const char *TransA*, const int *M*, const int *N*, const void ∗ *alpha*, const void ∗ *A*, const int *lda*, const void ∗ *X*, const int *incX*, const void ∗ *beta*, void ∗ *Y*, const int *incY*)

Add to complex single precision vector Y the reproducible matrix-vector product of complex single precision matrix A and complex single precision vector X.

Performs one of the matrix-vector operations

y := alpha∗A∗x + beta∗y or y := alpha∗A∗∗T∗x + beta∗y or y := alpha∗A∗∗H∗x + beta∗y,

where alpha and beta are scalars, x and y are vectors, and A is an M by N matrix.

The matrix-vector product is computed using binned types with binnedBLAS_cbcgemv()

Parameters

fold	the fold of the binned types
Order	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
TransA	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
M	number of rows of matrix A
N	number of columns of matrix A
alpha	scalar alpha
A	complex single precision matrix of dimension (M, lda) in row-major or (lda, N) in column-major
lda	the first dimension of A as declared in the calling program
X	complex single precision vector of at least size N if not transposed or size M otherwise
incX	X vector stride (use every incX'th element)
beta	scalar beta
Y	complex single precision vector Y of at least size M if not transposed or size N otherwise
incY	Y vector stride (use every incY'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.18 void reproBLAS_rcsum_sub (const int *fold*, const int *N*, const void ∗ *X*, const int *incX*, void ∗ *sum*)

Compute the reproducible sum of complex single precision vector X.

Return the sum of X.

The reproducible sum is computed with binned types using binnedBLAS_cbcsum()

Parameters

fold	the fold of the binned types
N	vector length
X	single precision vector
incX	X vector stride (use every incX'th element)
sum	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.19 double reproBLAS_rdasum (const int *fold*, const int *N*, const double ∗ *X*, const int *incX*)

Compute the reproducible absolute sum of double precision vector X.

Return the sum of absolute values of elements in X.

The reproducible absolute sum is computed with binned types using binnedBLAS_dbdasum()

Parameters

fold	the fold of the binned types
N	vector length
X	double precision vector
incX	X vector stride (use every incX'th element)

Returns

absolute sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.20 double reproBLAS_rddot (const int *fold*, const int *N*, const double * *X*, const int *incX*, const double * *Y*, const int *incY*)

Compute the reproducible dot product of double precision vectors X and Y.

Return the sum of the pairwise products of X and Y.

The reproducible dot product is computed with binned types using binnedBLAS_dbddot()

Parameters

fold	the fold of the binned types
N	vector length
X	double precision vector
incX	X vector stride (use every incX'th element)
Y	double precision vector
incY	Y vector stride (use every incY'th element)

Returns

the dot product of X and Y

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.21 void reproBLAS_rdgemm (const int *fold*, const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const double *alpha*, const double * *A*, const int *lda*, const double * *B*, const int *ldb*, const double *beta*, double * *C*, const int *ldc*)

Add to double precision matrix C the reproducible matrix-matrix product of double precision matrices A and B.

Performs one of the matrix-matrix operations

C := alpha∗op(A)∗op(B) + beta∗C,

where op(X) is one of

op(X) = X or op(X) = X∗∗T,

alpha and beta are scalars, A and B and C are matrices with op(A) an M by K matrix, op(B) a K by N matrix, and C is an M by N matrix.

The matrix-matrix product is computed using binned types with binnedBLAS_dbdgemm()

Parameters

fold	the fold of the binned types
Order	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
TransA	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
TransB	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
M	number of rows of matrix op(A) and of the matrix C.
N	number of columns of matrix op(B) and of the matrix C.
K	number of columns of matrix op(A) and columns of the matrix op(B).
alpha	scalar alpha
A	double precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
lda	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
B	double precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
ldb	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
beta	scalar beta
C	double precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
ldc	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.22 void reproBLAS_rdgemv (const int *fold*, const char *Order*, const char *TransA*, const int *M*, const int *N*, const double *alpha*, const double ∗ *A*, const int *lda*, const double ∗ *X*, const int *incX*, const double *beta*, double ∗ *Y*, const int *incY*)

Add to double precision vector Y the reproducible matrix-vector product of double precision matrix A and double precision vector X.

Performs one of the matrix-vector operations

y := alpha∗A∗x + beta∗y or y := alpha∗A∗∗T∗x + beta∗y,

where alpha and beta are scalars, x and y are vectors, and A is an M by N matrix.

The matrix-vector product is computed using binned types with binnedBLAS_dbdgemv()

Parameters

fold	the fold of the binned types
Order	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
TransA	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
M	number of rows of matrix A
N	number of columns of matrix A
alpha	scalar alpha
A	double precision matrix of dimension (M, lda) in row-major or (lda, N) in column-major
lda	the first dimension of A as declared in the calling program
X	double precision vector of at least size N if not transposed or size M otherwise
incX	X vector stride (use every incX'th element)
beta	scalar beta
Y	double precision vector Y of at least size M if not transposed or size N otherwise
incY	Y vector stride (use every incY'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.23 double reproBLAS_rdnrm2 (const int *fold,* const int *N,* const double ∗ *X,* const int *incX*)

Compute the reproducible Euclidian norm of double precision vector X.

Return the square root of the sum of the squared elements of X.

The reproducible Euclidian norm is computed with scaled binned types using binnedBLAS_dbdssq()

Parameters

fold	the fold of the binned types
N	vector length
X	double precision vector
incX	X vector stride (use every incX'th element)

Returns

Euclidian norm of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.24 double reproBLAS_rdsum (const int *fold*, const int *N*, const double ∗ *X*, const int *incX*)

Compute the reproducible sum of double precision vector X.

Return the sum of X.

The reproducible sum is computed with binned types using binnedBLAS_dbdsum()

Parameters

fold	the fold of the binned types
N	vector length
X	double precision vector
incX	X vector stride (use every incX'th element)

Returns

sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.25 double reproBLAS_rdzasum (const int *fold*, const int *N*, const void ∗ *X*, const int *incX*)

Compute the reproducible absolute sum of complex double precision vector X.

Return the sum of magnitudes of elements of X.

The reproducible absolute sum is computed with binned types using binnedBLAS_dbzasum()

Parameters

fold	the fold of the binned types
N	vector length
X	complex double precision vector
incX	X vector stride (use every incX'th element)

Returns

absolute sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.26 double reproBLAS_rdznrm2 (const int *fold,* const int *N,* const void ∗ *X,* const int *incX*)

Compute the reproducible Euclidian norm of complex double precision vector X.

Return the square root of the sum of the squared elements of X.

The reproducible Euclidian norm is computed with scaled binned types using binnedBLAS_dbzssq()

Parameters

fold	the fold of the binned types
N	vector length
X	complex double precision vector
incX	X vector stride (use every incX'th element)

Returns

Euclidian norm of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.27 float reproBLAS_rsasum (const int *fold*, const int *N*, const float * *X*, const int *incX*)

Compute the reproducible absolute sum of single precision vector X.

Return the sum of absolute values of elements in X.

The reproducible absolute sum is computed with binned types using binnedBLAS_sbsasum()

Parameters

fold	the fold of the binned types
N	vector length
X	single precision vector
incX	X vector stride (use every incX'th element)

Returns

absolute sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.28 float reproBLAS_rscasum (const int *fold*, const int *N*, const void * *X*, const int *incX*)

Compute the reproducible absolute sum of complex single precision vector X.

Return the sum of magnitudes of elements of X.

The reproducible absolute sum is computed with binned types using binnedBLAS_sbcasum()

Parameters

fold	the fold of the binned types
N	vector length
X	complex single precision vector
incX	X vector stride (use every incX'th element)

Returns

absolute sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.29 float reproBLAS_rscnrm2 (const int *fold*, const int *N*, const void * *X*, const int *incX*)

Compute the reproducible Euclidian norm of complex single precision vector X.

Return the square root of the sum of the squared elements of X.

The reproducible Euclidian norm is computed with scaled binned types using binnedBLAS_sbcssq()

Parameters

fold	the fold of the binned types
N	vector length
X	complex single precision vector
incX	X vector stride (use every incX'th element)

Returns

Euclidian norm of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.30 float reproBLAS_rsdot (const int *fold*, const int *N*, const float ∗ *X*, const int *incX*, const float ∗ *Y*, const int *incY*)

Compute the reproducible dot product of single precision vectors X and Y.

Return the sum of the pairwise products of X and Y.

The reproducible dot product is computed with binned types using binnedBLAS_sbsdot()

Parameters

fold	the fold of the binned types
N	vector length
X	single precision vector
incX	X vector stride (use every incX'th element)
Y	single precision vector
incY	Y vector stride (use every incY'th element)

Returns

the dot product of X and Y

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.31 void reproBLAS_rsgemm (const int *fold*, const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const float *alpha*, const float * *A*, const int *lda*, const float * *B*, const int *ldb*, const float *beta*, float * *C*, const int *ldc*)

Add to single precision matrix C the reproducible matrix-matrix product of single precision matrices A and B.

Performs one of the matrix-matrix operations

C := alpha*op(A)*op(B) + beta*C,

where op(X) is one of

op(X) = X or op(X) = X**T,

alpha and beta are scalars, A and B and C are matrices with op(A) an M by K matrix, op(B) a K by N matrix, and C is an M by N matrix.

The matrix-matrix product is computed using binned types with binnedBLAS_sbsgemm()

Parameters

fold	the fold of the binned types
Order	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
TransA	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
TransB	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
M	number of rows of matrix op(A) and of the matrix C.
N	number of columns of matrix op(B) and of the matrix C.
K	number of columns of matrix op(A) and columns of the matrix op(B).
alpha	scalar alpha
A	single precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
lda	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
B	single precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
ldb	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
beta	scalar beta
C	single precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
ldc	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.32 void reproBLAS_rsgemv (const int *fold*, const char *Order*, const char *TransA*, const int *M*, const int *N*, const float *alpha*, const float * *A*, const int *lda*, const float * *X*, const int *incX*, const float *beta*, float * *Y*, const int *incY*)

Add to single precision vector Y the reproducible matrix-vector product of single precision matrix A and single precision vector X.

Performs one of the matrix-vector operations

y := alpha*A*x + beta*y or y := alpha*A**T*x + beta*y,

where alpha and beta are scalars, x and y are vectors, and A is an M by N matrix.

The matrix-vector product is computed using binned types with binnedBLAS_sbsgemv()

Parameters

fold	the fold of the binned types
Order	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
TransA	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
M	number of rows of matrix A
N	number of columns of matrix A
alpha	scalar alpha
A	single precision matrix of dimension (M, lda) in row-major or (lda, N) in column-major
lda	the first dimension of A as declared in the calling program
X	single precision vector of at least size N if not transposed or size M otherwise
incX	X vector stride (use every incX'th element)
beta	scalar beta
Y	single precision vector Y of at least size M if not transposed or size N otherwise
incY	Y vector stride (use every incY'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.33 float reproBLAS_rsnrm2 (const int *fold*, const int *N*, const float * *X*, const int *incX*)

Compute the reproducible Euclidian norm of single precision vector X.

Return the square root of the sum of the squared elements of X.

The reproducible Euclidian norm is computed with scaled binned types using binnedBLAS_sbsssq()

Parameters

fold	the fold of the binned types
N	vector length
X	single precision vector
incX	X vector stride (use every incX'th element)

Returns

Euclidian norm of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.34 float reproBLAS_rssum (const int *fold*, const int *N*, const float * *X*, const int *incX*)

Compute the reproducible sum of single precision vector X.

Return the sum of X.

The reproducible sum is computed with binned types using binnedBLAS_sbssum()

Parameters

fold	the fold of the binned types
N	vector length
X	single precision vector
incX	X vector stride (use every incX'th element)

Returns

sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.35 void reproBLAS_rzdotc_sub (const int *fold*, const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, void * *dotc*)

Compute the reproducible conjugated dot product of complex double precision vectors X and Y.

Return the sum of the pairwise products of X and conjugated Y.

The reproducible dot product is computed with binned types using binnedBLAS_zbzdotc()

Parameters

fold	the fold of the binned types
N	vector length
X	complex double precision vector
incX	X vector stride (use every incX'th element)
Y	complex double precision vector
incY	Y vector stride (use every incY'th element)
dotc	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.36 void reproBLAS_rzdotu_sub (const int *fold*, const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, void * *dotu*)

Compute the reproducible unconjugated dot product of complex double precision vectors X and Y.

Return the sum of the pairwise products of X and Y.

The reproducible dot product is computed with binned types using binnedBLAS_zbzdotu()

Parameters

fold	the fold of the binned types
N	vector length
X	complex double precision vector
incX	X vector stride (use every incX'th element)
Y	complex double precision vector
incY	Y vector stride (use every incY'th element)
dotu	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.37 void reproBLAS_rzgemm (const int *fold*, const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const void * *alpha*, const void * *A*, const int *lda*, const void * *B*, const int *ldb*, const void * *beta*, void * *C*, const int *ldc*)

Add to complex double precision matrix C the reproducible matrix-matrix product of complex double precision matrices A and B.

Performs one of the matrix-matrix operations

C := alpha*op(A)*op(B) + beta*C,

where op(X) is one of

op(X) = X or op(X) = X**T or op(X) = X**H,

alpha and beta are scalars, A and B and C are matrices with op(A) an M by K matrix, op(B) a K by N matrix, and C is an M by N matrix.

The matrix-matrix product is computed using binned types with binnedBLAS_zbzgemm()

Parameters

fold	the fold of the binned types
Order	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
TransA	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
TransB	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
M	number of rows of matrix op(A) and of the matrix C.
N	number of columns of matrix op(B) and of the matrix C.
K	number of columns of matrix op(A) and columns of the matrix op(B).
alpha	scalar alpha
A	complex double precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
lda	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
B	complex double precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
ldb	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
beta	scalar beta
C	complex double precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
ldc	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.38 void reproBLAS_rzgemv (const int *fold*, const char *Order*, const char *TransA*, const int *M*, const int *N*, const void * *alpha*, const void * *A*, const int *lda*, const void * *X*, const int *incX*, const void * *beta*, void * *Y*, const int *incY*)

Add to complex double precision vector Y the reproducible matrix-vector product of complex double precision matrix A and complex double precision vector X.

Performs one of the matrix-vector operations

y := alpha*A*x + beta*y or y := alpha*A**T*x + beta*y or y := alpha*A**H*x + beta*y,

where alpha and beta are scalars, x and y are vectors, and A is an M by N matrix.

The matrix-vector product is computed using binned types with binnedBLAS_zbzgemv()

Parameters

fold	the fold of the binned types
Order	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
TransA	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
M	number of rows of matrix A
N	number of columns of matrix A
alpha	scalar alpha
A	complex double precision matrix of dimension (M, lda) in row-major or (lda, N) in column-major
lda	the first dimension of A as declared in the calling program
X	complex double precision vector of at least size N if not transposed or size M otherwise
incX	X vector stride (use every incX'th element)
beta	scalar beta
Y	complex double precision vector Y of at least size M if not transposed or size N otherwise
incY	Y vector stride (use every incY'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.39 void reproBLAS_rzsum_sub (const int *fold*, const int *N*, const void * *X*, const int *incX*, void * *sum*)

Compute the reproducible sum of complex double precision vector X.

Return the sum of X.

The reproducible sum is computed with binned types using binnedBLAS_zbzsum()

Parameters

fold	the fold of the binned types
N	vector length
X	complex double precision vector
incX	X vector stride (use every incX'th element)
sum	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.40 float reproBLAS_sasum (const int *N*, const float * *X*, const int *incX*)

Compute the reproducible absolute sum of single precision vector X.

Return the sum of absolute values of elements in X.

The reproducible absolute sum is computed with binned types of default fold using binnedBLAS_sbsasum()

Parameters

N	vector length
X	single precision vector
incX	X vector stride (use every incX'th element)

Returns

absolute sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.41 float reproBLAS_scasum (const int *N*, const void * *X*, const int *incX*)

Compute the reproducible absolute sum of complex single precision vector X.

Return the sum of magnitudes of elements of X.

The reproducible absolute sum is computed with binned types of default fold using binnedBLAS_sbcasum()

Parameters

N	vector length
X	complex single precision vector
incX	X vector stride (use every incX'th element)

Returns

absolute sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.42 float reproBLAS_scnrm2 (const int *N*, const void ∗ *X*, const int *incX*)

Compute the reproducible Euclidian norm of complex single precision vector X.

Return the square root of the sum of the squared elements of X.

The reproducible Euclidian norm is computed with scaled binned types of default fold using binnedBLAS_sbcssq()

Parameters

N	vector length
X	complex single precision vector
incX	X vector stride (use every incX'th element)

Returns

Euclidian norm of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.43 float reproBLAS_sdot (const int *N*, const float ∗ *X*, const int *incX*, const float ∗ *Y*, const int *incY*)

Compute the reproducible dot product of single precision vectors X and Y.

Return the sum of the pairwise products of X and Y.

The reproducible dot product is computed with binned types of default fold using binnedBLAS_sbsdot()

Parameters

N	vector length
X	single precision vector
incX	X vector stride (use every incX'th element)
Y	single precision vector
incY	Y vector stride (use every incY'th element)

Returns

the dot product of X and Y

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.44 void reproBLAS_sgemm (const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const float *alpha*, const float * *A*, const int *lda*, const float * *B*, const int *ldb*, const float *beta*, float * *C*, const int *ldc*)

Add to single precision matrix C the reproducible matrix-matrix product of single precision matrices A and B.

Performs one of the matrix-matrix operations

C := alpha*op(A)*op(B) + beta*C,

where op(X) is one of

op(X) = X or op(X) = X**T,

alpha and beta are scalars, A and B and C are matrices with op(A) an M by K matrix, op(B) a K by N matrix, and C is an M by N matrix.

The matrix-matrix product is computed using binned types of default fold with binnedBLAS_sbsgemm()

Parameters

Order	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
TransA	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
TransB	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
M	number of rows of matrix op(A) and of the matrix C.
N	number of columns of matrix op(B) and of the matrix C.
K	number of columns of matrix op(A) and columns of the matrix op(B).
alpha	scalar alpha
A	single precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
lda	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
B	single precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
ldb	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
beta	scalar beta
C	single precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
ldc	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.45 void reproBLAS_sgemv (const char *Order*, const char *TransA*, const int *M*, const int *N*, const float *alpha*, const float * *A*, const int *lda*, const float * *X*, const int *incX*, const float *beta*, float * *Y*, const int *incY*)

Add to single precision vector Y the reproducible matrix-vector product of single precision matrix A and single precision vector X.

Performs one of the matrix-vector operations

y := alpha*A*x + beta*y or y := alpha*A**T*x + beta*y,

where alpha and beta are scalars, x and y are vectors, and A is an M by N matrix.

The matrix-vector product is computed using binned types of default fold with binnedBLAS_sbsgemv()

Parameters

Order	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
TransA	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
M	number of rows of matrix A
N	number of columns of matrix A
alpha	scalar alpha
A	single precision matrix of dimension (M, lda) in row-major or (lda, N) in column-major
lda	the first dimension of A as declared in the calling program
X	single precision vector of at least size N if not transposed or size M otherwise
incX	X vector stride (use every incX'th element)
beta	scalar beta
Y	single precision vector Y of at least size M if not transposed or size N otherwise
incY	Y vector stride (use every incY'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.46 float reproBLAS_snrm2 (const int *N*, const float * *X*, const int *incX*)

Compute the reproducible Euclidian norm of single precision vector X.

Return the square root of the sum of the squared elements of X.

The reproducible Euclidian norm is computed with scaled binned types of default fold using binnedBLAS_sbsssq()

Parameters

N	vector length
X	single precision vector
incX	X vector stride (use every incX'th element)

Returns

Euclidian norm of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.47 float reproBLAS_ssum (const int *N*, const float ∗ *X*, const int *incX*)

Compute the reproducible sum of single precision vector X.

Return the sum of X.

The reproducible sum is computed with binned types of default fold using binnedBLAS_sbssum()

Parameters

N	vector length
X	single precision vector
incX	X vector stride (use every incX'th element)

Returns

sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.48 void reproBLAS_zdotc_sub (const int *N*, const void ∗ *X*, const int *incX*, const void ∗ *Y*, const int *incY*, void ∗ *dotc*)

Compute the reproducible conjugated dot product of complex double precision vectors X and Y.

Return the sum of the pairwise products of X and conjugated Y.

The reproducible dot product is computed with binned types of default fold using binnedBLAS_zbzdotc()

Parameters

N	vector length
X	complex double precision vector
incX	X vector stride (use every incX'th element)
Y	complex double precision vector
incY	Y vector stride (use every incY'th element)
dotc	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.49 void reproBLAS_zdotu_sub (const int *N*, const void ∗ *X*, const int *incX*, const void ∗ *Y*, const int *incY*, void ∗ *dotu*)

Compute the reproducible unconjugated dot product of complex double precision vectors X and Y.

Return the sum of the pairwise products of X and Y.

The reproducible dot product is computed with binned types of default fold using binnedBLAS_zbzdotu()

Parameters

N	vector length
X	complex double precision vector
incX	X vector stride (use every incX'th element)
Y	complex double precision vector
incY	Y vector stride (use every incY'th element)
dotu	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.50 void reproBLAS_zgemm (const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const void ∗ *alpha*, const void ∗ *A*, const int *lda*, const void ∗ *B*, const int *ldb*, const void ∗ *beta*, void ∗ *C*, const int *ldc*)

Add to complex double precision matrix C the reproducible matrix-matrix product of complex double precision matrices A and B.

Performs one of the matrix-matrix operations

C := alpha∗op(A)∗op(B) + beta∗C,

where op(X) is one of

op(X) = X or op(X) = X∗∗T or op(X) = X∗∗H,

alpha and beta are scalars, A and B and C are matrices with op(A) an M by K matrix, op(B) a K by N matrix, and C is an M by N matrix.

The matrix-matrix product is computed using binned types of default fold with binnedBLAS_zbzgemm()

Parameters

Order	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
TransA	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
TransB	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
M	number of rows of matrix op(A) and of the matrix C.
N	number of columns of matrix op(B) and of the matrix C.
K	number of columns of matrix op(A) and columns of the matrix op(B).
alpha	scalar alpha
A	complex double precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
lda	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
B	complex double precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
ldb	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
beta	scalar beta
C	complex double precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
ldc	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.51 void reproBLAS_zgemv (const char *Order*, const char *TransA*, const int *M*, const int *N*, const void ∗ *alpha*, const void ∗ *A*, const int *lda*, const void ∗ *X*, const int *incX*, const void ∗ *beta*, void ∗ *Y*, const int *incY*)

Add to complex double precision vector Y the reproducible matrix-vector product of complex double precision matrix A and complex double precision vector X.

Performs one of the matrix-vector operations

y := alpha*A*x + beta*y or y := alpha*A**T*x + beta*y or y := alpha*A**H*x + beta*y,

where alpha and beta are scalars, x and y are vectors, and A is an M by N matrix.

The matrix-vector product is computed using binned types of default fold with binnedBLAS_zbzgemv()

Parameters

Order	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
TransA	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
M	number of rows of matrix A
N	number of columns of matrix A
alpha	scalar alpha
A	complex double precision matrix of dimension (M, lda) in row-major or (lda, N) in column-major
lda	the first dimension of A as declared in the calling program
X	complex double precision vector of at least size N if not transposed or size M otherwise
incX	X vector stride (use every incX'th element)
beta	scalar beta
Y	complex double precision vector Y of at least size M if not transposed or size N otherwise
incY	Y vector stride (use every incY'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.52 void reproBLAS_zsum_sub (const int *N*, const void ∗ *X*, const int *incX*, void ∗ *sum*)

Compute the reproducible sum of complex double precision vector X.

Return the sum of X.

The reproducible sum is computed with binned types of default fold using binnedBLAS_zbzsum()

Parameters

N	vector length
X	complex double precision vector
incX	X vector stride (use every incX'th element)
sum	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

Index