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/idxd.h File Reference

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

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

Macros

- #define DIWIDTH 40

  Indexed double precision bin width.
- #define SIWIDTH 13

  Indexed single precision bin width.
- #define idxd_DIMAXINDEX (((DBL_MAX_EXP - DBL_MIN_EXP + DBL_MANT_DIG - 1)/DIWIDTH) - 1)

  Indexed double precision maximum index.
- #define idxd_SIMAXINDEX (((FLT_MAX_EXP - FLT_MIN_EXP + FLT_MANT_DIG - 1)/SIWIDTH) - 1)

  Indexed single precision maximum index.
- #define idxd_DIMAXFOLD (idxd_DIMAXINDEX + 1)

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

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

  Indexed double precision deposit endurance.
- #define idxd_SIENDURANCE (1 << (FLT_MANT_DIG - SIWIDTH - 2))

  Indexed single precision deposit endurance.
- #define idxd_DICAPACITY (idxd_DIENDURANCE*(1.0/DBL_EPSILON - 1.0))

  Indexed double precision capacity.
- #define idxd_SICAPACITY (idxd_SIENDURANCE*(1.0/FLT_EPSILON - 1.0))

  Indexed single precision capacity.
- #define idxd_DMCOMPRESSION (1.0/(1 << (DBL_MANT_DIG - DIWIDTH + 1)))

  Indexed double precision compression factor.
- #define idxd_SMCOMPRESSION (1.0/(1 << (FLT_MANT_DIG - SIWIDTH + 1)))

  Indexed single precision compression factor.
- #define idxd_DMEXPANSION (1.0*(1 << (DBL_MANT_DIG - DIWIDTH + 1)))

  Indexed double precision expansion factor.
- #define idxd_SMEXPANSION (1.0*(1 << (FLT_MANT_DIG - SIWIDTH + 1)))

  Indexed single precision expansion factor.

Typedefs

- typedef double double_indexed

  The indexed double datatype.
- typedef double double_complex_indexed

  The indexed complex double datatype.
- typedef float float_indexed

  The indexed float datatype.
- typedef float float_complex_indexed

  The indexed complex float datatype.

Functions

- size_t idxd_disize (const int fold)

  indexed double precision size
- size_t idxd_zisize (const int fold)

  indexed complex double precision size
- size_t idxd_sisize (const int fold)

  indexed single precision size
- size_t idxd_cisize (const int fold)

  indexed complex single precision size
- double_indexed ∗ idxd_dialloc (const int fold)

  indexed double precision allocation
- double_complex_indexed ∗ idxd_zialloc (const int fold)

  indexed complex double precision allocation
- float_indexed ∗ idxd_sialloc (const int fold)

  indexed single precision allocation
- float_complex_indexed ∗ idxd_cialloc (const int fold)

  indexed complex single precision allocation
- int idxd_dinum (const int fold)

  indexed double precision size
- int idxd_zinum (const int fold)

  indexed complex double precision size
- int idxd_sinum (const int fold)

  indexed single precision size
- int idxd_cinum (const int fold)

  indexed complex single precision size
- double idxd_dibound (const int fold, const int N, const double X, const double S)

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

  Get indexed single precision summation error bound.
- const double ∗ idxd_dmbins (const int X)

  Get indexed double precision reference bins.
- const float ∗ idxd_smbins (const int X)

  Get indexed single precision reference bins.
- int idxd_dindex (const double X)

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

  Get index of manually specified indexed double precision.
- int idxd_dmindex0 (const double ∗priX)

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

  Get index of single precision.
- int idxd_smindex (const float ∗priX)

  Get index of manually specified indexed single precision.
- int idxd_smindex0 (const float ∗priX)

  Check if index of manually specified indexed single precision is 0.
- int idxd_dmdenorm (const int fold, const double ∗priX)

  Check if indexed type has denormal bits.
- int idxd_zmdenorm (const int fold, const double ∗priX)

  Check if indexed type has denormal bits.
- int idxd_smdenorm (const int fold, const float ∗priX)

  Check if indexed type has denormal bits.
- int idxd_cmdenorm (const int fold, const float ∗priX)

  Check if indexed type has denormal bits.
- void idxd_diprint (const int fold, const double_indexed ∗X)

  Print indexed double precision.
- void idxd_dmprint (const int fold, const double ∗priX, const int incpriX, const double ∗carX, const int inccarX)

  Print manually specified indexed double precision.
- void idxd_ziprint (const int fold, const double_complex_indexed ∗X)

  Print indexed complex double precision.
- void idxd_zmprint (const int fold, const double ∗priX, const int incpriX, const double ∗carX, const int inccarX)

  Print manually specified indexed complex double precision.
- void idxd_siprint (const int fold, const float_indexed ∗X)

  Print indexed single precision.
- void idxd_smprint (const int fold, const float ∗priX, const int incpriX, const float ∗carX, const int inccarX)

  Print manually specified indexed single precision.
- void idxd_ciprint (const int fold, const float_complex_indexed ∗X)

  Print indexed complex single precision.
- void idxd_cmprint (const int fold, const float ∗priX, const int incpriX, const float ∗carX, const int inccarX)

  Print manually specified indexed complex single precision.
- void idxd_didiset (const int fold, const double_indexed ∗X, double_indexed ∗Y)

  Set indexed double precision (Y = X)
- void idxd_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 indexed double precision (Y = X)
- void idxd_ziziset (const int fold, const double_complex_indexed ∗X, double_complex_indexed ∗Y)

  Set indexed complex double precision (Y = X)
- void idxd_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 indexed complex double precision (Y = X)
- void idxd_zidiset (const int fold, const double_indexed ∗X, double_complex_indexed ∗Y)

  Set indexed complex double precision to indexed double precision (Y = X)
- void idxd_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 indexed complex double precision to manually specified indexed double precision (Y = X)
- void idxd_sisiset (const int fold, const float_indexed ∗X, float_indexed ∗Y)

  Set indexed single precision (Y = X)
- void idxd_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 indexed single precision (Y = X)

- void idxd_ciciset (const int fold, const float_complex_indexed ∗X, float_complex_indexed ∗Y)

    Set indexed complex single precision (Y = X)
- void idxd_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 indexed complex single precision (Y = X)
- void idxd_cisiset (const int fold, const float_indexed ∗X, float_complex_indexed ∗Y)

    Set indexed complex single precision to indexed single precision (Y = X)
- void idxd_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 indexed complex single precision to manually specified indexed single precision (Y = X)
- void idxd_disetzero (const int fold, double_indexed ∗X)

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

    Set manually specified indexed double precision to 0 (X = 0)
- void idxd_zisetzero (const int fold, double_complex_indexed ∗X)

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

    Set manually specified indexed complex double precision to 0 (X = 0)
- void idxd_sisetzero (const int fold, float_indexed ∗X)

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

    Set manually specified indexed single precision to 0 (X = 0)
- void idxd_cisetzero (const int fold, float_complex_indexed ∗X)

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

    Set manually specified indexed complex single precision to 0 (X = 0)
- void idxd_didiadd (const int fold, const double_indexed ∗X, double_indexed ∗Y)

    Add indexed double precision (Y += X)
- void idxd_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 indexed double precision (Y += X)
- void idxd_ziziadd (const int fold, const double_complex_indexed ∗X, double_complex_indexed ∗Y)

    Add indexed complex double precision (Y += X)
- void idxd_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 indexed complex double precision (Y += X)
- void idxd_sisiadd (const int fold, const float_indexed ∗X, float_indexed ∗Y)

    Add indexed single precision (Y += X)
- void idxd_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 indexed single precision (Y += X)
- void idxd_ciciadd (const int fold, const float_complex_indexed ∗X, float_complex_indexed ∗Y)

    Add indexed complex single precision (Y += X)
- void idxd_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 indexed complex single precision (Y += X)
- void idxd_didiaddv (const int fold, const int N, const double_indexed ∗X, const int incX, double_indexed ∗Y, const int incY)

    Add indexed double precision vectors (Y += X)
- void idxd_ziziaddv (const int fold, const int N, const double_complex_indexed ∗X, const int incX, double_↩ complex_indexed ∗Y, const int incY)

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

- void idxd_sisiaddv (const int fold, const int N, const float_indexed ∗X, const int incX, float_indexed ∗Y, const int incY)

    Add indexed single precision vectors (Y += X)
- void idxd_ciciaddv (const int fold, const int N, const float_complex_indexed ∗X, const int incX, float_complex↩_indexed ∗Y, const int incY)

    Add indexed complex single precision vectors (Y += X)
- void idxd_didadd (const int fold, const double X, double_indexed ∗Y)

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

    Add double precision to manually specified indexed double precision (Y += X)
- void idxd_zizadd (const int fold, const void ∗X, double_complex_indexed ∗Y)

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

    Add complex double precision to manually specified indexed complex double precision (Y += X)
- void idxd_sisadd (const int fold, const float X, float_indexed ∗Y)

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

    Add single precision to manually specified indexed single precision (Y += X)
- void idxd_cicadd (const int fold, const void ∗X, float_complex_indexed ∗Y)

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

    Add complex single precision to manually specified indexed complex single precision (Y += X)
- void idxd_didupdate (const int fold, const double X, double_indexed ∗Y)

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

    Update manually specified indexed double precision with double precision (X -> Y)
- void idxd_zizupdate (const int fold, const void ∗X, double_complex_indexed ∗Y)

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

    Update manually specified indexed complex double precision with complex double precision (X -> Y)
- void idxd_zidupdate (const int fold, const double X, double_complex_indexed ∗Y)

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

    Update manually specified indexed complex double precision with double precision (X -> Y)
- void idxd_sisupdate (const int fold, const float X, float_indexed ∗Y)

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

    Update manually specified indexed single precision with single precision (X -> Y)
- void idxd_cicupdate (const int fold, const void ∗X, float_complex_indexed ∗Y)

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

    Update manually specified indexed complex single precision with complex single precision (X -> Y)
- void idxd_cisupdate (const int fold, const float X, float_complex_indexed ∗Y)

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

    Update manually specified indexed complex single precision with single precision (X -> Y)
- void idxd_diddeposit (const int fold, const double X, double_indexed ∗Y)

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

  Add double precision to suitably indexed manually specified indexed double precision (Y += X)
- void idxd_zizdeposit (const int fold, const void ∗X, double_complex_indexed ∗Y)

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

  Add complex double precision to suitably indexed manually specified indexed complex double precision (Y += X)
- void idxd_sisdeposit (const int fold, const float X, float_indexed ∗Y)

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

  Add single precision to suitably indexed manually specified indexed single precision (Y += X)
- void idxd_cicdeposit (const int fold, const void ∗X, float_complex_indexed ∗Y)

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

  Add complex single precision to suitably indexed manually specified indexed complex single precision (Y += X)
- void idxd_direnorm (const int fold, double_indexed ∗X)

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

  Renormalize manually specified indexed double precision.
- void idxd_zirenorm (const int fold, double_complex_indexed ∗X)

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

  Renormalize manually specified indexed complex double precision.
- void idxd_sirenorm (const int fold, float_indexed ∗X)

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

  Renormalize manually specified indexed single precision.
- void idxd_cirenorm (const int fold, float_complex_indexed ∗X)

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

  Renormalize manually specified indexed complex single precision.
- void idxd_didconv (const int fold, const double X, double_indexed ∗Y)

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

  Convert double precision to manually specified indexed double precision (X -> Y)
- void idxd_zizconv (const int fold, const void ∗X, double_complex_indexed ∗Y)

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

  Convert complex double precision to manually specified indexed complex double precision (X -> Y)
- void idxd_sisconv (const int fold, const float X, float_indexed ∗Y)

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

  Convert single precision to manually specified indexed single precision (X -> Y)
- void idxd_cicconv (const int fold, const void ∗X, float_complex_indexed ∗Y)

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

  Convert complex single precision to manually specified indexed complex single precision (X -> Y)
- double idxd_ddiconv (const int fold, const double_indexed ∗X)

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

- double idxd_ddmconv (const int fold, const double ∗priX, const int incpriX, const double ∗carX, const int inccarX)

  Convert manually specified indexed double precision to double precision (X -> Y)
- void idxd_zziconv_sub (const int fold, const double_complex_indexed ∗X, void ∗conv)

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

  Convert manually specified indexed complex double precision to complex double precision (X -> Y)
- float idxd_ssiconv (const int fold, const float_indexed ∗X)

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

  Convert manually specified indexed single precision to single precision (X -> Y)
- void idxd_cciconv_sub (const int fold, const float_complex_indexed ∗X, void ∗conv)

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

  Convert manually specified indexed complex single precision to complex single precision (X -> Y)
- void idxd_dinegate (const int fold, double_indexed ∗X)

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

  Negate manually specified indexed double precision (X = -X)
- void idxd_zinegate (const int fold, double_complex_indexed ∗X)

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

  Negate manually specified indexed complex double precision (X = -X)
- void idxd_sinegate (const int fold, float_indexed ∗X)

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

  Negate manually specified indexed single precision (X = -X)
- void idxd_cinegate (const int fold, float_complex_indexed ∗X)

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

  Negate manually specified indexed complex single precision (X = -X)
- double idxd_dscale (const double X)

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

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

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

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

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

  rescale manually specified indexed complex single precision sum of squares
- double idxd_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 indexed double precision scaled sums of squares (Y += X)

- double idxd_didiaddsq (const int fold, const double scaleX, const double_indexed ∗X, const double scaleY, double_indexed ∗Y)

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

- float idxd_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 indexed single precision scaled sums of squares (Y += X)

- float idxd_sisiaddsq (const int fold, const float scaleX, const float_indexed ∗X, const float scaleY, float_indexed ∗Y)

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

- double idxd_ufp (const double X)

  unit in the first place

- float idxd_ufpf (const float X)

  unit in the first place

2.1.1 Detailed Description

idxd.h defines the indexed 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`)
- di - indexed double (double_indexed)
- zi - indexed complex double (double_complex_indexed)
- si - indexed float (float_indexed)
- ci - indexed complex float (float_complex_indexed)
- dm - manually specified indexed double (`double`, `double`)
- zm - manually specified indexed complex double (`double`, `double`)
- sm - manually specified indexed float (`float`, `float`)
- cm - manually specified indexed 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 indexed 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. Indexed 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 indexed types supplied to a subroutine. The maximum 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 indexed types must be the same for all indexed types that interact with each other. Operations on more than one indexed type assume all indexed types being operated upon have the same `fold`. Note that the `fold` of an indexed 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 indexed functions that you use.Power users of the library may find themselves wanting to manually specify the underlying primary and carry vectors of an indexed type themselves. If you do not know what these are, don't worry about the manually specified indexed types.

2.1.2 Macro Definition Documentation

2.1.2.1 #define DIWIDTH 40

Indexed double precision bin width.

bin width (in bits)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.2.2 #define idxd_DICAPACITY (idxd_DIENDURANCE*(1.0/DBL_EPSILON - 1.0))

Indexed double precision capacity.

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

Author

Peter Ahrens

Date

27 Apr 2015

2.1.2.3 #define idxd_DIENDURANCE (1 << (DBL_MANT_DIG - DIWIDTH - 2))

Indexed double precision deposit endurance.

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

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.2.4 #define idxd_DIMAXFOLD (idxd_DIMAXINDEX + 1)

The maximum double precision fold supported by the library.

Author

Peter Ahrens

Date

14 Jan 2016

2.1.2.5 #define idxd_DIMAXINDEX (((DBL_MAX_EXP - DBL_MIN_EXP + DBL_MANT_DIG - 1)/DIWIDTH) - 1)

Indexed double precision maximum index.

maximum index (inclusive)

Author

Peter Ahrens

Date

24 Jun 2015

2.1.2.6 #define idxd_DMCOMPRESSION (1.0/(1 << (DBL_MANT_DIG - DIWIDTH + 1)))

Indexed 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

2.1.2.7 #define idxd_DMEXPANSION (1.0*(1 << (DBL_MANT_DIG - DIWIDTH + 1)))

Indexed 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

2.1.2.8 #define idxd_SICAPACITY (idxd_SIENDURANCE*(1.0/FLT_EPSILON - 1.0))

Indexed single precision capacity.

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

Author

Peter Ahrens

Date

27 Apr 2015

2.1.2.9 #define idxd_SIENDURANCE (1 << (FLT_MANT_DIG - SIWIDTH - 2))

Indexed single precision deposit endurance.

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

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.2.10 #define idxd_SIMAXFOLD (idxd_SIMAXINDEX + 1)

The maximum single precision fold supported by the library.

Author

Peter Ahrens

Date

14 Jan 2016

2.1.2.11 #define idxd_SIMAXINDEX (((FLT_MAX_EXP - FLT_MIN_EXP + FLT_MANT_DIG - 1)/SIWIDTH) - 1)

Indexed single precision maximum index.

maximum index (inclusive)

Author

Peter Ahrens

Date

24 Jun 2015

2.1.2.12 #define idxd_SMCOMPRESSION (1.0/(1 << (FLT_MANT_DIG - SIWIDTH + 1)))

Indexed 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

2.1.2.13 #define idxd_SMEXPANSION (1.0*(1 << (FLT_MANT_DIG - SIWIDTH + 1)))

Indexed 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

2.1.2.14 #define SIWIDTH 13

Indexed 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_complex_indexed**

The indexed complex double datatype.

To allocate a double_complex_indexed, call idxd_zialloc()

Warning

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

2.1.3.2 typedef double **double_indexed**

The indexed double datatype.

To allocate a double_indexed, call idxd_dialloc()

Warning

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

2.1.3.3 typedef float **float_complex_indexed**

The indexed complex float datatype.

To allocate a float_complex_indexed, call idxd_cialloc()

Warning

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

2.1.3.4 typedef float **float_indexed**

The indexed float datatype.

To allocate a float_indexed, call idxd_sialloc()

Warning

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

2.1.4 Function Documentation

2.1.4.1 void idxd_cciconv_sub (const int *fold*, const **float_complex_indexed** ∗ *X*, void ∗ *conv*)

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

Parameters

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

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.2 void idxd_ccmconv_sub (const int *fold*, const float ∗ *priX*, const int *incpriX*, const float ∗ *carX*, const int *inccarX*, void ∗ *conv*)

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

Parameters

fold	the fold of the indexed 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

2.1.4.3 float_complex_indexed∗ idxd_cialloc (const int *fold*)

indexed complex single precision allocation

Parameters

fold	the fold of the indexed type

Returns

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

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.4 void idxd_cicadd (const int *fold*, const void ∗ *X*, float_complex_indexed ∗ *Y*)

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

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

Parameters

fold	the fold of the indexed types
X	scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.5 void idxd_cicconv (const int *fold*, const void ∗ *X*, float_complex_indexed ∗ *Y*)

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

Parameters

fold	the fold of the indexed types
X	scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.6 void idxd_cicdeposit (const int *fold*, const void ∗ *X*, float_complex_indexed ∗ *Y*)

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

Performs the operation Y += X on an indexed 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 idxd_cicupdate() on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call idxd_cicdeposit() to deposit a maximum of idxd_SIENDURANCE elements into Y before renormalizing Y with idxd_cirenorm(). After any number of successive calls of idxd_cicdeposit() on Y, you must renormalize Y with idxd_cirenorm() before using any other function on Y.

Parameters

fold	the fold of the indexed types
X	scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.7 void idxd_ciciadd (const int *fold*, const float_complex_indexed ∗ *X*, float_complex_indexed ∗ *Y*)

Add indexed complex single precision (Y += X)

Performs the operation Y += X

Parameters

fold	the fold of the indexed types
X	indexed scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.8 void idxd_ciciaddv (const int *fold*, const int *N*, const float_complex_indexed * *X*, const int *incX*, float_complex_indexed * *Y*, const int *incY*)

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

Performs the operation Y += X

Parameters

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

Author

Peter Ahrens

Date

25 Jun 2015

2.1.4.9 void idxd_ciciset (const int *fold*, const float_complex_indexed * *X*, float_complex_indexed * *Y*)

Set indexed complex single precision (Y = X)

Performs the operation Y = X

Parameters

fold	the fold of the indexed types
X	indexed scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.10 void idxd_cicupdate (const int *fold*, const void * *X*, float_complex_indexed * *Y*)

Update indexed 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 indexed types
X	scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.11 void idxd_cinegate (const int *fold*, float_complex_indexed * *X*)

Negate indexed complex single precision (X = -X)

Performs the operation X = -X

Parameters

fold	the fold of the indexed types
X	indexed scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.12 int idxd_cinum (const int *fold*)

indexed complex single precision size

Parameters

fold	the fold of the indexed type

Returns

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

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.13 void idxd_ciprint (const int *fold*, const float_complex_indexed * *X*)

Print indexed complex single precision.

Parameters

fold	the fold of the indexed types
X	indexed scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.14 void idxd_cirenorm (const int *fold*, float_complex_indexed * *X*)

Renormalize indexed 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 indexed types
X	indexed scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.15 void idxd_cisetzero (const int *fold*, float_complex_indexed ∗ *X*)

Set indexed single precision to 0 (X = 0)

Performs the operation X = 0

Parameters

fold	the fold of the indexed types
X	indexed scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.16 void idxd_cisiset (const int *fold*, const float_indexed ∗ *X*, float_complex_indexed ∗ *Y*)

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

Performs the operation Y = X

Parameters

fold	the fold of the indexed types
X	indexed scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.17 size_t idxd_cisize (const int *fold*)

indexed complex single precision size

Parameters

fold	the fold of the indexed type

Returns

the size (in bytes) of the indexed type

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.18 void idxd_cisupdate (const int *fold*, const float *X*, float_complex_indexed ∗ *Y*)

Update indexed 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 indexed types
X	scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

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

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

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

Parameters

fold	the fold of the indexed 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 idxd_cmcconv (const int *fold,* const void ∗ *X,* float ∗ *priY,* const int *incpriY,* float ∗ *carY,* const int *inccarY*)

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

Parameters

fold	the fold of the indexed 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 idxd_cmcdeposit (const int *fold,* const void ∗ *X,* float ∗ *priY,* const int *incpriY*)

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

Performs the operation Y += X on an indexed 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 idxd_cmcupdate() on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call idxd_cmcdeposit() to deposit a maximum of idxd_SIENDURANCE elements into Y before renormalizing Y with idxd_cmrenorm(). After any number of successive calls of idxd_cmcdeposit() on Y, you must renormalize Y with idxd_cmrenorm() before using any other function on Y.

Parameters

fold	the fold of the indexed 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 idxd_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 indexed complex single precision (Y += X)

Performs the operation Y += X

Parameters

fold	the fold of the indexed 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 idxd_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 indexed complex single precision (Y = X)

Performs the operation Y = X

Parameters

fold	the fold of the indexed 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

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

Update manually specified indexed 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 indexed 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 idxd_cmdenorm (const int *fold*, const float $*$ *priX*)

Check if indexed 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 indexed 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 idxd_cmnegate (const int *fold*, float $*$ *priX*, const int *incpriX*, float $*$ *carX*, const int *inccarX*)

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

Performs the operation X = -X

Parameters

fold	the fold of the indexed 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 idxd_cmprint (const int *fold*, const float ∗ *priX*, const int *incpriX*, const float ∗ *carX*, const int *inccarX*)

Print manually specified indexed complex single precision.

Parameters

fold	the fold of the indexed 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

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

Renormalize manually specified indexed 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 indexed 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 idxd_cmsetzero (const int *fold*, float * *priX*, const int *incpriX*, float * *carX*, const int *inccarX*)

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

Performs the operation X = 0

Parameters

fold	the fold of the indexed 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 idxd_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 indexed complex single precision to manually specified indexed single precision (Y = X)

Performs the operation Y = X

Parameters

fold	the fold of the indexed 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 idxd_cmsrescale (const int *fold*, const float *X*, const float *scaleY*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

rescale manually specified indexed complex single precision sum of squares

Rescale an indexed complex single precision sum of squares Y

Parameters

fold	the fold of the indexed types
X	Y's new scaleY (X == idxd_sscale (f) for some `float` f) (X >= scaleY)
scaleY	Y's current scaleY (scaleY == idxd_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 idxd_cmsupdate (const int *fold*, const float *X*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Update manually specified indexed 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 indexed 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 idxd_ddiconv (const int *fold*, const double_indexed * *X*)

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

Parameters

fold	the fold of the indexed types
X	indexed scalar X

Returns

scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.34 double idxd_ddmconv (const int *fold*, const double * *priX*, const int *incpriX*, const double * *carX*, const int *inccarX*)

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

Parameters

fold	the fold of the indexed 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

2.1.4.35 double_indexed* idxd_dialloc (const int *fold*)

indexed double precision allocation

Parameters

fold	the fold of the indexed type

Returns

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

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.36 double idxd_dibound (const int *fold*, const int *N*, const double *X*, const double *S*)

Get indexed double precision summation error bound.

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

Parameters

fold	the fold of the indexed 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 indexed types

Returns

error bound

Author

Peter Ahrens

Date

31 Jul 2015

2.1.4.37 void idxd_didadd (const int *fold*, const double *X*, double_indexed * *Y*)

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

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

Parameters

fold	the fold of the indexed types
X	scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.38 void idxd_didconv (const int *fold*, const double *X*, double_indexed * *Y*)

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

Parameters

fold	the fold of the indexed types
X	scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.39 void idxd_diddeposit (const int *fold*, const double *X*, double_indexed ∗ *Y*)

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

Performs the operation Y += X on an indexed 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 idxd_didupdate() on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call idxd_diddeposit() to deposit a maximum of idxd_DIENDURANCE elements into Y before renormalizing Y with idxd_direnorm(). After any number of successive calls of idxd_diddeposit() on Y, you must renormalize Y with idxd_direnorm() before using any other function on Y.

Parameters

fold	the fold of the indexed types
X	scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.40 void idxd_didiadd (const int *fold*, const double_indexed ∗ *X*, double_indexed ∗ *Y*)

Add indexed double precision (Y += X)

Performs the operation Y += X

Parameters

fold	the fold of the indexed types
X	indexed scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.41 double idxd_didiaddsq (const int *fold*, const double *scaleX*, const double_indexed * *X*, const double *scaleY*, double_indexed * *Y*)

Add indexed 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 indexed types
scaleX	scale of X (scaleX == idxd_dscale (Z) for some `double` Z)
X	indexed scalar X
scaleY	scale of Y (scaleY == idxd_dscale (Z) for some `double` Z)
Y	indexed scalar Y

Returns

updated scale of Y

Author

Peter Ahrens

Date

2 Dec 2015

2.1.4.42 void idxd_didiaddv (const int *fold*, const int *N*, const double_indexed * *X*, const int *incX*, double_indexed * *Y*, const int *incY*)

Add indexed double precision vectors (Y += X)

Performs the operation Y += X

Parameters

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

Author

Peter Ahrens

Date

25 Jun 2015

2.1.4.43 void idxd_didiset (const int *fold*, const double_indexed ∗ *X*, double_indexed ∗ *Y*)

Set indexed double precision (Y = X)

Performs the operation Y = X

Parameters

fold	the fold of the indexed types
X	indexed scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.44 void idxd_didupdate (const int *fold*, const double *X*, double_indexed ∗ *Y*)

Update indexed 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 indexed types
X	scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.45 int idxd_dindex (const double *X*)

Get index of double precision.

The index of a non-indexed type is the smallest index an indexed 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

2.1.4.46 void idxd_dinegate (const int *fold*, double_indexed * *X*)

Negate indexed double precision (X = -X)

Performs the operation X = -X

Parameters

fold	the fold of the indexed types
X	indexed scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.47 int idxd_dinum (const int *fold*)

indexed double precision size

Parameters

fold	the fold of the indexed type

Returns

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

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.48 void idxd_diprint (const int *fold*, const **double_indexed** ∗ *X*)

Print indexed double precision.

Parameters

fold	the fold of the indexed types
X	indexed scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.49 void idxd_direnorm (const int *fold*, **double_indexed** ∗ *X*)

Renormalize indexed double precision.

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

Parameters

fold	the fold of the indexed types
X	indexed scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.50 void idxd_disetzero (const int *fold*, double_indexed * *X*)

Set indexed double precision to 0 (X = 0)

Performs the operation X = 0

Parameters

fold	the fold of the indexed types
X	indexed scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.51 size_t idxd_disize (const int *fold*)

indexed double precision size

Parameters

fold	the fold of the indexed type

Returns

the size (in bytes) of the indexed type

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.52 const double∗ idxd_dmbins (const int *X*)

Get indexed 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 idxd_dmdadd (const int *fold*, const double *X*, double ∗ *priY*, const int *incpriY*, double ∗ *carY*, const int *inccarY*)

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

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

Parameters

fold	the fold of the indexed 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 idxd_dmdconv (const int *fold*, const double *X*, double * *priY*, const int *incpriY*, double * *carY*, const int *inccarY*)

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

Parameters

fold	the fold of the indexed 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 idxd_dmddeposit (const int *fold*, const double *X*, double * *priY*, const int *incpriY*)

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

Performs the operation Y += X on an indexed 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 idxd_dmdupdate() on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call idxd_dmddeposit() to deposit a maximum of idxd_DIENDURANCE elements into Y before renormalizing Y with idxd_dmrenorm(). After any number of successive calls of idxd_dmddeposit() on Y, you must renormalize Y with idxd_dmrenorm() before using any other function on Y.

Parameters

fold	the fold of the indexed 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 idxd_dmdenorm (const int *fold,* const double $*$ *priX*)

Check if indexed 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 indexed 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 idxd_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 indexed double precision (Y += X)

Performs the operation Y += X

Parameters

fold	the fold of the indexed 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 idxd_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 indexed 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 indexed types
scaleX	scale of X (scaleX == idxd_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 == idxd_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 idxd_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 indexed double precision (Y = X)

Performs the operation Y = X

Parameters

fold	the fold of the indexed 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 idxd_dmdrescale (const int *fold*, const double *X*, const double *scaleY*, double ∗ *priY*, const int *incpriY*, double ∗ *carY*, const int *inccarY*)

rescale manually specified indexed double precision sum of squares

Rescale an indexed double precision sum of squares Y

Parameters

fold	the fold of the indexed types
X	Y's new scaleY (X == idxd_dscale (f) for some `double` f) (X >= scaleY)
scaleY	Y's current scaleY (scaleY == idxd_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 idxd_dmdupdate (const int *fold*, const double *X*, double $*$ *priY*, const int *incpriY*, double $*$ *carY*, const int *inccarY*)

Update manually specified indexed 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 indexed 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 idxd_dmindex (const double $*$ *priX*)

Get index of manually specified indexed double precision.

The index of an indexed 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 idxd_dmindex0 (const double * *priX*)

Check if index of manually specified indexed 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 idxd_dmnegate (const int *fold*, double * *priX*, const int *incpriX*, double * *carX*, const int *inccarX*)

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

Performs the operation X = -X

Parameters

fold	the fold of the indexed 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 idxd_dmprint (const int *fold*, const double * *priX*, const int *incpriX*, const double * *carX*, const int *inccarX*)

Print manually specified indexed double precision.

Parameters

fold	the fold of the indexed 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 idxd_dmrenorm (const int *fold*, double ∗ *priX*, const int *incpriX*, double ∗ *carX*, const int *inccarX*)

Renormalize manually specified indexed double precision.

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

Parameters

fold	the fold of the indexed 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 idxd_dmsetzero (const int *fold*, double ∗ *priX*, const int *incpriX*, double ∗ *carX*, const int *inccarX*)

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

Performs the operation X = 0

Parameters

fold	the fold of the indexed 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 idxd_dscale (const double *X*)

Get a reproducible double precision scale.

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

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

such that

Y * 2^(-DBL_MANT_DIG - DIWIDTH - 1) < X < Y * 2\^(DIWIDTH + 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_indexed* idxd_sialloc (const int *fold*)

indexed single precision allocation

Parameters

fold	the fold of the indexed type

Returns

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

Author

Peter Ahrens

Date

27 Apr 2015

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

Get indexed single precision summation error bound.

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

Parameters

fold	the fold of the indexed 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 indexed 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 int idxd_sindex (const float *X*)

Get index of single precision.

The index of a non-indexed type is the smallest index an indexed 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

2.1.4.72 void idxd_sinegate (const int *fold*, float_indexed * *X*)

Negate indexed single precision (X = -X)

Performs the operation X = -X

Parameters

fold	the fold of the indexed types
X	indexed scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.73 int idxd_sinum (const int *fold*)

indexed single precision size

Parameters

fold	the fold of the indexed type

Returns

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

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.74 void idxd_siprint (const int *fold*, const float_indexed * *X*)

Print indexed single precision.

Parameters

fold	the fold of the indexed types
X	indexed scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.75 void idxd_sirenorm (const int *fold*, float_indexed * *X*)

Renormalize indexed single precision.

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

Parameters

fold	the fold of the indexed types
X	indexed scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.76 void idxd_sisadd (const int *fold*, const float *X*, float_indexed * *Y*)

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

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

Parameters

fold	the fold of the indexed types
X	scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.77 void idxd_sisconv (const int *fold*, const float *X*, float_indexed * *Y*)

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

Parameters

fold	the fold of the indexed types
X	scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.78 void idxd_sisdeposit (const int *fold*, const float *X*, float_indexed * *Y*)

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

Performs the operation Y += X on an indexed 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 idxd←_sisupdate() on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call idxd_sisdeposit() to deposit a maximum of idxd_SIENDURANCE elements into Y before renormalizing Y with idxd_sirenorm(). After any number of successive calls of idxd_sisdeposit() on Y, you must renormalize Y with idxd_sirenorm() before using any other function on Y.

Parameters

fold	the fold of the indexed types
X	scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.79 void idxd_sisetzero (const int *fold*, float_indexed * *X*)

Set indexed single precision to 0 (X = 0)

Performs the operation X = 0

Parameters

fold	the fold of the indexed types
X	indexed scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.80 void idxd_sisiadd (const int *fold*, const **float_indexed** ∗ *X*, **float_indexed** ∗ *Y*)

Add indexed single precision (Y += X)

Performs the operation Y += X

Parameters

fold	the fold of the indexed types
X	indexed scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.81 float idxd_sisiaddsq (const int *fold*, const float *scaleX*, const **float_indexed** ∗ *X*, const float *scaleY*, **float_indexed** ∗ *Y*)

Add indexed 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 indexed types
scaleX	scale of X (scaleX == idxd_sscale (Z) for some `float` Z)
X	indexed scalar X
scaleY	scale of Y (scaleY == idxd_sscale (Z) for some `float` Z)
Y	indexed scalar Y

Returns

updated scale of Y

Author

Peter Ahrens

Date

2 Dec 2015

2.1.4.82 void idxd_sisiaddv (const int *fold*, const int *N*, const float_indexed ∗ *X*, const int *incX*, float_indexed ∗ *Y*, const int *incY*)

Add indexed single precision vectors (Y += X)

Performs the operation Y += X

Parameters

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

Author

Peter Ahrens

Date

25 Jun 2015

2.1.4.83 void idxd_sisiset (const int *fold*, const float_indexed ∗ *X*, float_indexed ∗ *Y*)

Set indexed single precision (Y = X)

Performs the operation Y = X

Parameters

fold	the fold of the indexed types
X	indexed scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.84 size_t idxd_sisize (const int *fold*)

indexed single precision size

Parameters

fold	the fold of the indexed type

Returns

the size (in bytes) of the indexed type

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.85 void idxd_sisupdate (const int *fold*, const float *X*, float_indexed ∗ *Y*)

Update indexed 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 indexed types
X	scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.86 const float∗ idxd_smbins (const int *X*)

Get indexed 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 idxd_smdenorm (const int *fold*, const float * *priX*)

Check if indexed 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 indexed 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 idxd_smindex (const float * *priX*)

Get index of manually specified indexed single precision.

The index of an indexed 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 idxd_smindex0 (const float * *priX*)

Check if index of manually specified indexed 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 idxd_smnegate (const int *fold,* float * *priX,* const int *incpriX,* float * *carX,* const int *inccarX*)

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

Performs the operation X = -X

Parameters

fold	the fold of the indexed 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 idxd_smprint (const int *fold*, const float * *priX*, const int *incpriX*, const float * *carX*, const int *inccarX*)

Print manually specified indexed single precision.

Parameters

fold	the fold of the indexed 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 idxd_smrenorm (const int *fold*, float * *priX*, const int *incpriX*, float * *carX*, const int *inccarX*)

Renormalize manually specified indexed single precision.

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

Parameters

fold	the fold of the indexed 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 idxd_smsadd (const int *fold*, const float *X*, float ∗ *priY*, const int *incpriY*, float ∗ *carY*, const int *inccarY*)

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

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

Parameters

fold	the fold of the indexed 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 idxd_smsconv (const int *fold*, const float *X*, float ∗ *priY*, const int *incpriY*, float ∗ *carY*, const int *inccarY*)

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

Parameters

fold	the fold of the indexed 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 idxd_smsdeposit (const int *fold*, const float *X*, float * *priY*, const int *incpriY*)

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

Performs the operation Y += X on an indexed 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 idxd_smsupdate() on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call idxd_smsdeposit() to deposit a maximum of idxd_SIENDURANCE elements into Y before renormalizing Y with idxd_smrenorm(). After any number of successive calls of idxd_smsdeposit() on Y, you must renormalize Y with idxd_smrenorm() before using any other function on Y.

Parameters

fold	the fold of the indexed 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 idxd_smsetzero (const int *fold*, float * *priX*, const int *incpriX*, float * *carX*, const int *inccarX*)

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

Performs the operation X = 0

Parameters

fold	the fold of the indexed 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 idxd_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 indexed single precision (Y += X)

Performs the operation Y += X

Parameters

fold	the fold of the indexed 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 idxd_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 indexed 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 indexed types
scaleX	scale of X (scaleX == idxd_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 == idxd_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 idxd_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 indexed single precision (Y = X)

Performs the operation Y = X

Parameters

fold	the fold of the indexed 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 idxd_smsrescale (const int *fold*, const float *X*, const float *scaleY*, float ∗ *priY*, const int *incpriY*, float ∗ *carY*, const int *inccarY*)

rescale manually specified indexed single precision sum of squares

Rescale an indexed single precision sum of squares Y

Parameters

fold	the fold of the indexed types
X	Y's new scaleY (X == idxd_sscale (f) for some `float` f) (X >= scaleY)
scaleY	Y's current scaleY (scaleY == idxd_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 idxd_smsupdate (const int *fold*, const float *X*, float ∗ *priY*, const int *incpriY*, float ∗ *carY*, const int *inccarY*)

Update manually specified indexed 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 indexed 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 idxd_sscale (const float *X*)

Get a reproducible single precision scale.

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

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

such that

Y * 2^(-FLT_MANT_DIG - SIWIDTH - 1) < X < Y * 2\^(SIWIDTH + 2)

Parameters

X	single precision number to be scaled

Returns

reproducible scaling factor

Author

Peter Ahrens

Date

19 Jun 2015

2.1.4.103 float idxd_ssiconv (const int *fold*, const float_indexed * *X*)

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

Parameters

fold	the fold of the indexed types
X	indexed scalar X

Returns

scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

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

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

Parameters

fold	the fold of the indexed 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 idxd_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 idxd_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_indexed* idxd_zialloc (const int *fold*)

indexed complex double precision allocation

Parameters

fold	the fold of the indexed type

Returns

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

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.108 void idxd_zidiset (const int *fold*, const double_indexed * *X*, double_complex_indexed * *Y*)

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

Performs the operation Y = X

Parameters

fold	the fold of the indexed types
X	indexed scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.109 void idxd_zidupdate (const int *fold*, const double *X*, double_complex_indexed ∗ *Y*)

Update indexed 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 indexed types
X	scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.110 void idxd_zinegate (const int *fold*, double_complex_indexed ∗ *X*)

Negate indexed complex double precision (X = -X)

Performs the operation X = -X

Parameters

fold	the fold of the indexed types
X	indexed scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.111 int idxd_zinum (const int *fold*)

indexed complex double precision size

Parameters

fold	the fold of the indexed type

Returns

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

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.112 void idxd_ziprint (const int *fold,* const **double_complex_indexed** $*$ *X*)

Print indexed complex double precision.

Parameters

fold	the fold of the indexed types
X	indexed scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.113 void idxd_zirenorm (const int *fold,* **double_complex_indexed** $*$ *X*)

Renormalize indexed 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 indexed types
X	indexed scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.114 void idxd_zisetzero (const int *fold*, double_complex_indexed ∗ *X*)

Set indexed double precision to 0 (X = 0)

Performs the operation X = 0

Parameters

fold	the fold of the indexed types
X	indexed scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.115 size_t idxd_zisize (const int *fold*)

indexed complex double precision size

Parameters

fold	the fold of the indexed type

Returns

the size (in bytes) of the indexed type

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.116 void idxd_zizadd (const int *fold*, const void * *X*, double_complex_indexed * *Y*)

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

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

Parameters

fold	the fold of the indexed types
X	scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.117 void idxd_zizconv (const int *fold*, const void * *X*, double_complex_indexed * *Y*)

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

Parameters

fold	the fold of the indexed types
X	scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.118 void idxd_zizdeposit (const int *fold*, const void * *X*, double_complex_indexed * *Y*)

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

Performs the operation Y += X on an indexed 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 idxd_zizupdate() on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call idxd_zizdeposit() to deposit a maximum of idxd_DIENDURANCE elements into Y before renormalizing Y with idxd_zirenorm(). After any number of successive calls of idxd_zizdeposit() on Y, you must renormalize Y with idxd_zirenorm() before using any other function on Y.

Parameters

fold	the fold of the indexed types
X	scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.119 void idxd_ziziadd (const int *fold*, const double_complex_indexed ∗ *X*, double_complex_indexed ∗ *Y*)

Add indexed complex double precision (Y += X)

Performs the operation Y += X

Parameters

fold	the fold of the indexed types
X	indexed scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.120 void idxd_ziziaddv (const int *fold*, const int *N*, const double_complex_indexed ∗ *X*, const int *incX*, double_complex_indexed ∗ *Y*, const int *incY*)

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

Performs the operation Y += X

Parameters

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

Author

Peter Ahrens

Date

25 Jun 2015

2.1.4.121 void idxd_ziziset (const int *fold*, const double_complex_indexed * *X*, double_complex_indexed * *Y*)

Set indexed complex double precision (Y = X)

Performs the operation Y = X

Parameters

fold	the fold of the indexed types
X	indexed scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.122 void idxd_zizupdate (const int *fold*, const void * *X*, double_complex_indexed * *Y*)

Update indexed 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 indexed types
X	scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

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

Check if indexed 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 indexed 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 idxd_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 indexed complex double precision to manually specified indexed double precision (Y = X)

Performs the operation Y = X

Parameters

fold	the fold of the indexed 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 idxd_zmdrescale (const int *fold*, const double *X*, const double *scaleY*, double $*$ *priY*, const int *incpriY*, double $*$ *carY*, const int *inccarY*)

rescale manually specified indexed complex double precision sum of squares

Rescale an indexed complex double precision sum of squares Y

Parameters

fold	the fold of the indexed types
X	Y's new scaleY (X == idxd_dscale (f) for some `double` f) (X >= scaleY)
scaleY	Y's current scaleY (scaleY == idxd_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 idxd_zmdupdate (const int *fold*, const double *X*, double $*$ *priY*, const int *incpriY*, double $*$ *carY*, const int *inccarY*)

Update manually specified indexed 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 indexed 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 idxd_zmnegate (const int *fold,* double ∗ *priX,* const int *incpriX,* double ∗ *carX,* const int *inccarX*)

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

Performs the operation X = -X

Parameters

fold	the fold of the indexed 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 idxd_zmprint (const int *fold,* const double ∗ *priX,* const int *incpriX,* const double ∗ *carX,* const int *inccarX*)

Print manually specified indexed complex double precision.

Parameters

fold	the fold of the indexed 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 idxd_zmrenorm (const int *fold*, double ∗ *priX*, const int *incpriX*, double ∗ *carX*, const int *inccarX*)

Renormalize manually specified indexed 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 indexed 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 idxd_zmsetzero (const int *fold*, double ∗ *priX*, const int *incpriX*, double ∗ *carX*, const int *inccarX*)

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

Performs the operation X = 0

Parameters

fold	the fold of the indexed 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 idxd_zmzadd (const int *fold*, const void ∗ *X*, double ∗ *priY*, const int *incpriY*, double ∗ *carY*, const int *inccarY*)

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

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

Parameters

fold	the fold of the indexed 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 idxd_zmzconv (const int *fold*, const void ∗ *X*, double ∗ *priY*, const int *incpriY*, double ∗ *carY*, const int *inccarY*)

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

Parameters

fold	the fold of the indexed 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 idxd_zmzdeposit (const int *fold*, const void * *X*, double * *priY*, const int *incpriY*)

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

Performs the operation Y += X on an indexed 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 idxd_zmzupdate() on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call idxd_zmzdeposit() to deposit a maximum of idxd_DIENDURANCE elements into Y before renormalizing Y with idxd_zmrenorm(). After any number of successive calls of idxd_zmzdeposit() on Y, you must renormalize Y with idxd_zmrenorm() before using any other function on Y.

Parameters

fold	the fold of the indexed 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 idxd_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 indexed complex double precision (Y += X)

Performs the operation Y += X

Parameters

fold	the fold of the indexed 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 idxd_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 indexed complex double precision (Y = X)

Performs the operation Y = X

Parameters

fold	the fold of the indexed 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

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

Update manually specified indexed 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 indexed 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 idxd_zziconv_sub (const int *fold*, const double_complex_indexed * *X*, void * *conv*)

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

Parameters

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

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

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

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

Parameters

fold	the fold of the indexed 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/idxdBLAS.h File Reference

idxdBLAS.h defines BLAS Methods that operate on indexed types.

```
#include "idxd.h"
#include "reproBLAS.h"
#include <complex.h>
```

Functions

- float idxdBLAS_samax (const int N, const float *X, const int incX)

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

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

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

  Find maximum magnitude in vector of complex double precision.
- float idxdBLAS_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 idxdBLAS_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 idxdBLAS_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 idxdBLAS_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 idxdBLAS_didsum (const int fold, const int N, const double *X, const int incX, double_indexed *Y)

  Add to indexed double precision Y the sum of double precision vector X.
- void idxdBLAS_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 indexed double precision Y the sum of double precision vector X.
- void idxdBLAS_didasum (const int fold, const int N, const double *X, const int incX, double_indexed *Y)

  Add to indexed double precision Y the absolute sum of double precision vector X.
- void idxdBLAS_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 indexed double precision Y the absolute sum of double precision vector X.
- double idxdBLAS_didssq (const int fold, const int N, const double *X, const int incX, const double scaleY, double_indexed *Y)

  Add to scaled indexed double precision Y the scaled sum of squares of elements of double precision vector X.
- double idxdBLAS_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 indexed double precision Y the scaled sum of squares of elements of double precision vector X.

- void idxdBLAS_diddot (const int fold, const int N, const double ∗X, const int incX, const double ∗Y, const int incY, double_indexed ∗Z)

    Add to indexed double precision Z the dot product of double precision vectors X and Y.
- void idxdBLAS_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 indexed double precision Z the dot product of double precision vectors X and Y.
- void idxdBLAS_zizsum (const int fold, const int N, const void ∗X, const int incX, double_indexed ∗Y)

    Add to indexed complex double precision Y the sum of complex double precision vector X.
- void idxdBLAS_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 indexed complex double precision Y the sum of complex double precision vector X.
- void idxdBLAS_dizasum (const int fold, const int N, const void ∗X, const int incX, double_indexed ∗Y)

    Add to indexed double precision Y the absolute sum of complex double precision vector X.
- void idxdBLAS_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 indexed double precision Y the absolute sum of complex double precision vector X.
- double idxdBLAS_dizssq (const int fold, const int N, const void ∗X, const int incX, const double scaleY, double_indexed ∗Y)

    Add to scaled indexed double precision Y the scaled sum of squares of elements of complex double precision vector X.
- double idxdBLAS_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 indexed double precision Y the scaled sum of squares of elements of complex double precision vector X.
- void idxdBLAS_zizdotu (const int fold, const int N, const void ∗X, const int incX, const void ∗Y, const int incY, double_indexed ∗Z)

    Add to indexed complex double precision Z the unconjugated dot product of complex double precision vectors X and Y.
- void idxdBLAS_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 indexed complex double precision Z the unconjugated dot product of complex double precision vectors X and Y.
- void idxdBLAS_zizdotc (const int fold, const int N, const void ∗X, const int incX, const void ∗Y, const int incY, double_indexed ∗Z)

    Add to indexed complex double precision Z the conjugated dot product of complex double precision vectors X and Y.
- void idxdBLAS_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 indexed complex double precision Z the conjugated dot product of complex double precision vectors X and Y.
- void idxdBLAS_sissum (const int fold, const int N, const float ∗X, const int incX, float_indexed ∗Y)

    Add to indexed single precision Y the sum of single precision vector X.
- void idxdBLAS_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 indexed single precision Y the sum of single precision vector X.
- void idxdBLAS_sisasum (const int fold, const int N, const float ∗X, const int incX, float_indexed ∗Y)

    Add to indexed single precision Y the absolute sum of single precision vector X.
- void idxdBLAS_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 indexed single precision Y the absolute sum of double precision vector X.
- float idxdBLAS_sisssq (const int fold, const int N, const float ∗X, const int incX, const float scaleY, float_indexed ∗Y)

    Add to scaled indexed single precision Y the scaled sum of squares of elements of single precision vector X.
- float idxdBLAS_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 indexed single precision Y the scaled sum of squares of elements of single precision vector X.

- void idxdBLAS_sisdot (const int fold, const int N, const float ∗X, const int incX, const float ∗Y, const int incY, float_indexed ∗Z)

  Add to indexed single precision Z the dot product of single precision vectors X and Y.
- void idxdBLAS_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 indexed single precision Z the dot product of single precision vectors X and Y.
- void idxdBLAS_cicsum (const int fold, const int N, const void ∗X, const int incX, float_indexed ∗Y)

  Add to indexed complex single precision Y the sum of complex single precision vector X.
- void idxdBLAS_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 indexed complex single precision Y the sum of complex single precision vector X.
- void idxdBLAS_sicasum (const int fold, const int N, const void ∗X, const int incX, float_indexed ∗Y)

  Add to indexed single precision Y the absolute sum of complex single precision vector X.
- void idxdBLAS_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 indexed single precision Y the absolute sum of complex single precision vector X.
- float idxdBLAS_sicssq (const int fold, const int N, const void ∗X, const int incX, const float scaleY, float_↩ indexed ∗Y)

  Add to scaled indexed single precision Y the scaled sum of squares of elements of complex single precision vector X.
- float idxdBLAS_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 indexed single precision Y the scaled sum of squares of elements of complex single precision vector X.
- void idxdBLAS_cicdotu (const int fold, const int N, const void ∗X, const int incX, const void ∗Y, const int incY, float_indexed ∗Z)

  Add to indexed complex single precision Z the unconjugated dot product of complex single precision vectors X and Y.
- void idxdBLAS_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 indexed complex single precision Z the unconjugated dot product of complex single precision vectors X and Y.
- void idxdBLAS_cicdotc (const int fold, const int N, const void ∗X, const int incX, const void ∗Y, const int incY, float_indexed ∗Z)

  Add to indexed complex single precision Z the conjugated dot product of complex single precision vectors X and Y.
- void idxdBLAS_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 indexed complex single precision Z the conjugated dot product of complex single precision vectors X and Y.
- void idxdBLAS_didgemv (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_indexed ∗Y, const int incY)

  Add to indexed double precision vector Y the matrix-vector product of double precision matrix A and double precision vector X.
- void idxdBLAS_didgemm (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_indexed ∗C, const int ldc)

  Add to indexed double precision matrix C the matrix-matrix product of double precision matrices A and B.
- void idxdBLAS_sisgemv (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_indexed ∗Y, const int incY)

  Add to indexed single precision vector Y the matrix-vector product of single precision matrix A and single precision vector X.
- void idxdBLAS_sisgemm (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_↩ indexed ∗C, const int ldc)

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

- void idxdBLAS_zizgemv (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_indexed ∗Y, const int incY)

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

- void idxdBLAS_zizgemm (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_indexed ∗C, const int ldc)

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

- void idxdBLAS_cicgemv (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_indexed ∗Y, const int incY)

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

- void idxdBLAS_cicgemm (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_indexed ∗C, const int ldc)

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

2.2.1 Detailed Description

idxdBLAS.h defines BLAS Methods that operate on indexed 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`)
- di - indexed double (double_indexed)
- zi - indexed complex double (double_complex_indexed)
- si - indexed float (float_indexed)
- ci - indexed complex float (float_complex_indexed)
- dm - manually specified indexed double (`double`, `double`)
- zm - manually specified indexed complex double (`double`, `double`)
- sm - manually specified indexed float (`float`, `float`)
- cm - manually specified indexed 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 indexed 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. Indexed 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 bins are used in the indexed types supplied to a subroutine. The maximum 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 indexed types must be the same for all indexed types that interact with each other. Operations on more than one indexed type assume all indexed types being operated upon have the same `fold`. Note that the `fold` of an indexed 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 indexed functions that you use.Power users of the library may find themselves wanting to manually specify the underlying primary and carry vectors of an indexed type themselves. If you do not know what these are, don't worry about the manually specified indexed types.

2.2.2 Function Documentation

2.2.2.1 void idxdBLAS_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 idxdBLAS_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 idxdBLAS_cicdotc (const int *fold*, const int *N*, const void ∗ *X*, const int *incX*, const void ∗ *Y*, const int *incY*, float_complex_indexed ∗ *Z*)

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

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

Parameters

fold	the fold of the indexed 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	indexed scalar Z

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.4 void idxdBLAS_cicdotu (const int *fold*, const int *N*, const void ∗ *X*, const int *incX*, const void ∗ *Y*, const int *incY*, float_complex_indexed ∗ *Z*)

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

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

Parameters

fold	the fold of the indexed 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	indexed scalar Z

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.5 void idxdBLAS_cicgemm (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_indexed ∗ *C,* const int *ldc*)

Add to indexed 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 indexed M by N matrix.

Parameters

fold	the fold of the indexed 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.

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 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	indexed 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 idxdBLAS_cicgemv (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_indexed ∗ *Y,* const int *incY*)

Add to indexed 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 indexed vector, and A is an M by N matrix.

Parameters

fold	the fold of the indexed 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	indexed 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 idxdBLAS_cicsum (const int *fold*, const int *N*, const void * *X*, const int *incX*, float_complex_indexed * *Y*)

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

Add to Y the indexed sum of X.

Parameters

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

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.8 void idxdBLAS_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 indexed complex single precision Z the conjugated dot product of complex single precision vectors X and Y.

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

Parameters

fold	the fold of the indexed 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 idxdBLAS_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 indexed complex single precision Z the unconjugated dot product of complex single precision vectors X and Y.

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

Parameters

fold	the fold of the indexed 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

2.2.2.10 void idxdBLAS_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 indexed complex single precision Y the sum of complex single precision vector X.

Add to Y the indexed sum of X.

Parameters

fold	the fold of the indexed types
fold	the fold of the indexed types
N	vector length

Parameters

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 idxdBLAS_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 idxdBLAS_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 idxdBLAS_didasum (const int *fold*, const int *N*, const double ∗ *X*, const int *incX*, double_indexed ∗ *Y*)

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

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

Parameters

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

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.14 void idxdBLAS_diddot (const int *fold*, const int *N*, const double ∗ *X*, const int *incX*, const double ∗ *Y*, const int *incY*, double_indexed ∗ *Z*)

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

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

Parameters

fold	the fold of the indexed 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	indexed scalar Z

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.15 void idxdBLAS_didgemm (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_indexed ∗ *C,* const int *ldc*)

Add to indexed 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 indexed M by N matrix.

Parameters

fold	the fold of the indexed 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	indexed double precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.16 void idxdBLAS_didgemv (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_indexed ∗ *Y,* const int *incY*)

Add to indexed 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 indexed vector, and A is an M by N matrix.

Parameters

fold	the fold of the indexed 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	indexed 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 idxdBLAS_didssq (const int *fold,* const int *N,* const double ∗ *X,* const int *incX,* const double *scaleY,* double_indexed ∗ *Y*)

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

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

Parameters

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

Returns

the new scaling factor of Y

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.18 void idxdBLAS_didsum (const int *fold*, const int *N*, const double ∗ *X*, const int *incX*, double_indexed ∗ *Y*)

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

Add to Y the indexed sum of X.

Parameters

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

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.19 void idxdBLAS_dizasum (const int *fold*, const int *N*, const void ∗ *X*, const int *incX*, double_indexed ∗ *Y*)

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

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

Parameters

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

Author

Peter Ahrens

Date

15 Jan 2016

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

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

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

Parameters

fold	the fold of the indexed 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	indexed scalar Y

Returns

the new scaling factor of Y

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.21 void idxdBLAS_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 indexed double precision Y the absolute sum of double precision vector X.

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

Parameters

fold	the fold of the indexed 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

2.2.2.22 void idxdBLAS_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 indexed double precision Z the dot product of double precision vectors X and Y.

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

Parameters

fold	the fold of the indexed 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 idxdBLAS_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 indexed double precision Y the scaled sum of squares of elements of double precision vector X.

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

Parameters

fold	the fold of the indexed 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 idxdBLAS_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 indexed double precision Y the sum of double precision vector X.

Set Y to the indexed sum of X.

Parameters

fold	the fold of the indexed 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 idxdBLAS_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 indexed double precision Y the absolute sum of complex double precision vector X.

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

Parameters

fold	the fold of the indexed 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 idxdBLAS_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 indexed double precision Y the scaled sum of squares of elements of complex double precision vector X.

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

Parameters

fold	the fold of the indexed 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 idxdBLAS_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 idxdBLAS_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 idxdBLAS_sicasum (const int *fold*, const int *N*, const void ∗ *X*, const int *incX*, float_indexed ∗ *Y*)

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

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

Parameters

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

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.30 float idxdBLAS_sicssq (const int *fold*, const int *N*, const void ∗ *X*, const int *incX*, const float *scaleY*, float_indexed ∗ *Y*)

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

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

Parameters

fold	the fold of the indexed 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	indexed scalar Y

Returns

the new scaling factor of Y

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.31 void idxdBLAS_sisasum (const int *fold*, const int *N*, const float ∗ *X*, const int *incX*, float_indexed ∗ *Y*)

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

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

Parameters

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

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.32 void idxdBLAS_sisdot (const int *fold*, const int *N*, const float ∗ *X*, const int *incX*, const float ∗ *Y*, const int *incY*, float_indexed ∗ *Z*)

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

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

Parameters

fold	the fold of the indexed 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	indexed scalar Z

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.33 void idxdBLAS_sisgemm (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_indexed ∗ *C*, const int *ldc*)

Add to indexed 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 indexed M by N matrix.

Parameters

fold	the fold of the indexed 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	indexed 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 idxdBLAS_sisgemv (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_indexed ∗ *Y*, const int *incY*)

Add to indexed 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 indexed vector, and A is an M by N matrix.

Parameters

fold	the fold of the indexed 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	indexed 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 idxdBLAS_sisssq (const int *fold*, const int *N*, const float ∗ *X*, const int *incX*, const float *scaleY*, float_indexed ∗ *Y*)

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

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

Parameters

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

Returns

the new scaling factor of Y

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.36 void idxdBLAS_sissum (const int *fold*, const int *N*, const float * *X*, const int *incX*, float_indexed * *Y*)

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

Add to Y the indexed sum of X.

Parameters

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

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.37 void idxdBLAS_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 indexed single precision Y the absolute sum of complex single precision vector X.

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

Parameters

fold	the fold of the indexed 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 idxdBLAS_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 indexed single precision Y the scaled sum of squares of elements of complex single precision vector X.

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

Parameters

fold	the fold of the indexed 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 idxdBLAS_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 indexed single precision Y the absolute sum of double precision vector X.

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

Parameters

fold	the fold of the indexed 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 idxdBLAS_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 indexed single precision Z the dot product of single precision vectors X and Y.

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

Parameters

fold	the fold of the indexed 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 idxdBLAS_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 indexed single precision Y the scaled sum of squares of elements of single precision vector X.

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

Parameters

fold	the fold of the indexed 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 idxdBLAS_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 indexed single precision Y the sum of single precision vector X.

Add to Y the indexed sum of X.

Parameters

fold	the fold of the indexed 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 idxdBLAS_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 idxdBLAS_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 idxdBLAS_zizdotc (const int *fold*, const int *N*, const void ∗ *X*, const int *incX*, const void ∗ *Y*, const int *incY*, double_complex_indexed ∗ *Z*)

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

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

Parameters

fold	the fold of the indexed 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 idxdBLAS_zizdotu (const int *fold*, const int *N*, const void ∗ *X*, const int *incX*, const void ∗ *Y*, const int *incY*, double_complex_indexed ∗ *Z*)

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

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

Parameters

fold	the fold of the indexed 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	indexed scalar Z

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.47 void idxdBLAS_zizgemm (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_indexed ∗ *C,* const int *ldc*)

Add to indexed 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 indexed M by N matrix.

Parameters

fold	the fold of the indexed 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	indexed 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 idxdBLAS_zizgemv (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_indexed ∗ *Y*, const int *incY*)

Add to indexed 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 indexed vector, and A is an M by N matrix.

Parameters

fold	the fold of the indexed 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	indexed 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 idxdBLAS_zizsum (const int *fold*, const int *N*, const void ∗ *X*, const int *incX*, double_complex_indexed ∗ *Y*)

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

Add to Y the indexed sum of X.

Parameters

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

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.50 void idxdBLAS_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 indexed complex double precision Z the conjugated dot product of complex double precision vectors X and Y.

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

Parameters

fold	the fold of the indexed 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 idxdBLAS_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 indexed complex double precision Z the unconjugated dot product of complex double precision vectors X and Y.

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

Parameters

fold	the fold of the indexed 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 idxdBLAS_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 indexed complex double precision Y the sum of complex double precision vector X.

Add to Y the indexed sum of X.

Parameters

fold	the fold of the indexed 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/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.

2.3.1 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 indexed 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. Indexed 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 bins are used in the indexed types supplied to a subroutine. The maximum 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 indexed types must be the same for all indexed types that interact with each other. Operations on more than one indexed type assume all indexed types being operated upon have the same `fold`. Note that the `fold` of an indexed 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 indexed 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 indexed types.

2.3.2 Function Documentation

2.3.2.1 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 indexed types of default fold using idxdBLAS_cicdotc()

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

2.3.2.2 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 indexed types of default fold using idxdBLAS_cicdotu()

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

2.3.2.3 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 indexed types of default fold with idxdBLAS_cicgemm()

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.

Parameters

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

2.3.2.4 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 indexed types of default fold with idxdBLAS_cicgemv()

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

2.3.2.5 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 indexed types of default fold using idxdBLAS_cicsum()

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

2.3.2.6 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 indexed types of default fold using idxdBLAS_didasum()

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

2.3.2.7 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 indexed types of default fold using idxdBLAS_diddot()

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

2.3.2.8 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 indexed types of default fold with idxdBLAS_didgemm()

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	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

2.3.2.9 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 indexed types of default fold with idxdBLAS_didgemv()

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)

Parameters

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

2.3.2.10 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 indexed types of default fold using idxdBLAS_didssq()

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

2.3.2.11 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 indexed types of default fold using idxdBLAS_didsum()

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

2.3.2.12 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 indexed types of default fold using idxdBLAS_dizasum()

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

2.3.2.13 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 indexed types of default fold using idxdBLAS_dizssq()

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

2.3.2.14 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 indexed types using idxdBLAS_cicdotc()

Parameters

fold	the fold of the indexed 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

2.3.2.15 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 indexed types using idxdBLAS_cicdotu()

Parameters

fold	the fold of the indexed 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

2.3.2.16 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 indexed types with idxdBLAS_cicgemm()

Parameters

fold	the fold of the indexed 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.
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

2.3.2.17 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 indexed types with idxdBLAS_cicgemv()

Parameters

fold	the fold of the indexed types
Order	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)

Parameters

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

2.3.2.18 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 indexed types using idxdBLAS_cicsum()

Parameters

fold	the fold of the indexed 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

2.3.2.19 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 indexed types using idxdBLAS_didasum()

Parameters

fold	the fold of the indexed 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

2.3.2.20 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 indexed types using idxdBLAS_diddot()

Parameters

fold	the fold of the indexed 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

2.3.2.21 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 indexed types with idxdBLAS_didgemm()

Parameters

fold	the fold of the indexed 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

2.3.2.22 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 indexed types with idxdBLAS_didgemv()

Parameters

fold	the fold of the indexed 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

2.3.2.23 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 indexed types using idxdBLAS_didssq()

Parameters

fold	the fold of the indexed 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

2.3.2.24 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 indexed types using idxdBLAS_didsum()

Parameters

fold	the fold of the indexed 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

2.3.2.25 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 indexed types using idxdBLAS_dizasum()

Parameters

fold	the fold of the indexed 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

2.3.2.26 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 indexed types using idxdBLAS_dizssq()

Parameters

fold	the fold of the indexed 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

2.3.2.27 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 indexed types using idxdBLAS_sisasum()

Parameters

fold	the fold of the indexed 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

2.3.2.28 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 indexed types using idxdBLAS_sicasum()

Parameters

fold	the fold of the indexed 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

2.3.2.29 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 indexed types using idxdBLAS_sicssq()

Parameters

fold	the fold of the indexed 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

2.3.2.30 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 indexed types using idxdBLAS_sisdot()

Parameters

fold	the fold of the indexed 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

2.3.2.31 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 indexed types with idxdBLAS_sisgemm()

Parameters

fold	the fold of the indexed 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).

Parameters

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

2.3.2.32 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 indexed types with idxdBLAS_sisgemv()

Parameters

fold	the fold of the indexed 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

2.3.2.33 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 indexed types using idxdBLAS_sisssq()

Parameters

fold	the fold of the indexed 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

2.3.2.34 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 indexed types using idxdBLAS_sissum()

Parameters

fold	the fold of the indexed 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

2.3.2.35 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 indexed types using idxdBLAS_zizdotc()

Parameters

fold	the fold of the indexed 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

2.3.2.36 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 indexed types using idxdBLAS_zizdotu()

Parameters

fold	the fold of the indexed 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

2.3.2.37 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 indexed types with idxdBLAS_zizgemm()

Parameters

fold	the fold of the indexed 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.
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

2.3.2.38 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 indexed types with idxdBLAS_zizgemv()

Parameters

fold	the fold of the indexed 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

2.3.2.39 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 indexed types using idxdBLAS_zizsum()

Parameters

fold	the fold of the indexed 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

2.3.2.40 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 indexed types of default fold using idxdBLAS_sisasum()

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

2.3.2.41 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 indexed types of default fold using idxdBLAS_sicasum()

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

2.3.2.42 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 indexed types of default fold using idxdBLAS_sicssq()

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

2.3.2.43 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 indexed types of default fold using idxdBLAS_sisdot()

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

2.3.2.44 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 indexed types of default fold with idxdBLAS_sisgemm()

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

2.3.2.45 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 indexed types of default fold with idxdBLAS_sisgemv()

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)

Parameters

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

2.3.2.46 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 indexed types of default fold using idxdBLAS_sisssq()

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

2.3.2.47 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 indexed types of default fold using idxdBLAS_sissum()

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

2.3.2.48 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 indexed types of default fold using idxdBLAS_zizdotc()

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

2.3.2.49 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 indexed types of default fold using idxdBLAS_zizdotu()

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

2.3.2.50 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 indexed types of default fold with idxdBLAS_zizgemm()

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)

Parameters

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

2.3.2.51 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 indexed types of default fold with idxdBLAS_zizgemv()

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

2.3.2.52 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 indexed types of default fold using idxdBLAS_zizsum()

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