Sun™ S3L 4.0 Reference Manual

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900 U.S.A.
650-960-1300

Part No. 816-0653-10
August 2001, Revision A

Send comments about this document to: docfeedback@sun.com

Please
Recycle

Adobe PostScript™

Contents

Preface

This manual describes the Sun™ Scalable Scientific Subroutine Library (Sun S3L). It is directed to anyone developing message-passing C, C++, F77, or F90 programs.

Acknowledgments

The Sun S3L dense linear algebra routines make use of the ScaLAPACK library described in "ScaLAPACK: Linear Algebra Software for Distributed Memory Architectures," J. Demmel, J. Dongarra, R. van de Geijn, and D. Walker in *Parallel Computers: Theory and Practice*, Ed. by T. Casavant, P. Tvrdik, and F. Plasil. (IEEE Press, 1995, pp. 267-282.)

ScaLAPACK routines access the Sun MPI library through calls to the BLACS library described in "Two-dimensional Basic Linear Algebra Communications Subprograms," J. Dongarra and R. van de Geijn, in E*nvironments and Tools for Parallel Scientific Computing*, Ed. by J. Dongarra and B. Tourancheau (Elsevier Science Publisher B.V., 1993, pp. 31-40), in "Basic Linear Algebra Communication Subprograms: Analysis and Implementation Across Multiple Parallel Architectures," R.C. Whaley.

How This Book Is Organized

Chapter 1 contains a list of the routines in Sun S3L, organized into general classes, such as Dense Matrix Operations, Sparse Matrix Operations, and so forth.

Chapter 2 contains individual descriptions of the Sun S3L routines, presented in alphabetical order.

Appendix A describes the error codes that are returned when an array handle error is encountered.

Using UNIX Commands

This document may not contain information on basic UNIX® commands and procedures.

See one or both of the following for such information:

- AnswerBook™ online documentation for the Solaris™ software environment
- Other software documentation that you received with your system

Typographic Conventions

Typeface or Symbol	Meaning	Examples
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `% You have mail`.
`AaBbCc123`	What you type, when contrasted with on-screen computer output	`%` **`ls -a`**
AaBbCc123	Book titles, new words or terms, words to be emphasized	Read Chapter 6 in the *User's Guide*. These are called *class* options. You *must* be superuser to do this.
	Command-line variable; replace with a real name or value	To delete a file, type `rm` *filename*.

Shell Prompts

Shell	Prompt
C shell	%
C shell superuser	#
Bourne shell and Korn shell	$
Bourne shell and Korn shell superuser	#

Related Documentation

Application	Title	Part Number
All	*Sun HPC ClusterTools™ 4 Product Notes*	816-0647-10
All	*Sun HPC ClusterTools 4 Performance Guide*	816-0656-10
Sun S3L programming	*Sun S3L 4.0 Programming Guide*	816-0652-10
Sun MPI programming	*Sun MPI 5.0 Programming and Reference Guide*	816-0651-10
Sun MPI programming	*Sun HPC ClusterTools 4 User's Guide*	816-0650-10
Prism™	*Prism 6.2 User's Guide*	816-0654-10
Prism	*Prism 6.2 Reference Manual*	816-0655-10

Accessing Sun Documentation Online

The `docs.sun.com`℠ web site enables you to access a select group of Sun technical documentation on the Web. You can browse the `docs.sun.com` archive or search for a specific book title or subject at:

`http://docs.sun.com`

Ordering Sun Documentation

Fatbrain.com, an Internet professional bookstore, stocks select product documentation from Sun Microsystems, Inc.

For a list of documents and how to order them, visit the Sun Documentation Center on Fatbrain.com at:

`http://www.fatbrain.com/documentation/sun`

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. You can email your comments to Sun at:

`docfeedback@sun.com`

Please include the part number (816-0653-10) of your document in the subject line of your email.

CHAPTER 1

Introduction

This chapter contains a quick-reference index to the routines in the Sun Scalable Scientific Subroutine Library (Sun S3L). The routines are organized into two tables:

- TABLE 1-1 lists the parallel functions in Sun S3L that perform mathematical operations. These are referred to as the Sun S3L *core* routines.
- TABLE 1-2 lists the supplemental functions in Sun S3L that simplify many tasks involved in distributed memory computing. These are referred to as the Sun S3L *toolkit* routines.

Note – Many Sun S3L routines support the corresponding ScaLAPACK APIs. TABLE 1-3 lists the ScaLAPACK APIs that are supported.

Sun S3L Overview

TABLE 1-1 Sun S3L Core Mathematical Routines

Function	Description
Dense Matrix Operations	
`S3L_2_norm()`	Compute 2-norm of a vector.
`S3L_inner_prod()`	Compute inner product of two vectors.
`S3L_mat_mult()`	Compute product of two matrices.
`S3L_mat_vec_mult()`	Compute product of a matrix and vector.
`S3L_outer_prod()`	Compute outer product of two vectors.
Sparse Matrix Operations	
`S3L_declare_sparse()`	Create an S3L handle for an S3L sparse array.
`S3L_free_sparse()`	Free memory allocated to S3L sparse array.
`S3L_convert_sparse()`	Convert an S3L array from one sparse format to another.
`S3L_rand_sparse()`	Create an S3L array with random values and sparsity.
`S3L_matvec_sparse()`	Compute the product of a sparse matrix and dense vector.
`S3L_read_sparse()`	Read a sparse matrix from an ASCII file.
`S3L_write_sparse()`	Write a sparse matrix to a file.
`S3L_print_sparse()`	Print all nonzero values from a sparse matrix.
Gaussian Elimination for Dense Systems	
`S3L_lu_factor()`	Perform LU factorization of a matrix.
`S3L_lu_invert()`	Compute inverse of square matrix instances of S3L array using `S3l_lu_factor()` results.
`S3L_lu_solve()`	Solve system of linear equations (AX=B) for square matrix instances of S3L array.
`S3L_lu_deallocate()`	Deallocate `S3L_lu_factor()` resources.
Walsh Transform	
`S3L_walsh()`	Compute discrete Walsh/Hadamard transform of 1D and 2D S3L arrays.
`S3L_walsh_setup()`	Prepare internal data structure for discrete Walsh/Hadamard transform.
`S3L_walsh_free_setup()`	Free memory allocated to Walsh/Hadamard transform.
Iterative Eigenpairs Computation	
`S3L_eigen_iter()`	Compute selected eigenpairs of dense or sparse matrices.

TABLE 1-1 Sun S3L Core Mathematical Routines *(Continued)*

Function	Description
Finite-Difference Stock Option Pricing	
`S3L_fin_fd_1D()`	Solve 1D Black-Scholes PDE to compute prices of vanilla and several exotic stock options.
`S3L_fin_fd_2D()`	Solve 2D Black-Scholes PDE to compute prices of vanilla and several exotic stock options.
Discrete Cosine Transform	
`S3L_dct_iv()`	Compute DCT Type IV of 1D, 2D, and 3D S3L arrays.
`S3L_dct_iv_setup()`	Prepare internal data structures for DCT Type IV operation.
`S3L_dct_iv_free_setup()`	Free memory allocated to DCT setup.
Discrete Sine Transform	
`S3L_dst()`	Compute DST of 1D, 2D, and 3D S3L arrays.
`S3L_dst_setup()`	Prepare internal data structures for DST.
`S3L_dst_free_setup()`	Free memory allocated to DST setup.
QR Array Factoring/Solving	
`S3L_qr_factor()`	Compute QR decomposition of a real or complex S3L array.
`S3L_get_qr()`	Extract Q and R arrays from a QR-decomposed S3L array.
`S3L_qr_solve()`	Compute the least-squares solution to an over-determined system of the form a*x=b.
`S3L_qr_free()`	Free memory allocated to QR decomposition.
Quadratic Programming Optimization	
`S3L_qp_attr_init()`	Initialize a set of QP attributes with default values.
`S3L_qp_attr_destroy()`	Destroy a specified set of QP attributes.
`S3L_qp_attr_set()`	Specify the type of solver to be used and amount of error output.
`S3L_qp()`	Solve linear/quadratic optimization problem.
Cholesky Solver	
`S3L_cholesky_factor()`	Perform Cholesky factorization for each square matrix in an S3L array.
`S3L_cholesky_solve()`	Solve a system of distributed linear equations of the form AX = B for each square matrix in an S3L array.
`S3L_cholesky_invert()`	Compute the inverse of each square matrix in an S3L array.

TABLE 1-1 Sun S3L Core Mathematical Routines *(Continued)*

Function	Description
Sparse Linear System Solver	
Direct Method	
`S3L_sparse_solve()`	A direct solver for solving sparse linear systems of equations of the form A*x = y.
`S3L_sparse_solve_free()`	Free memory allocated to the direct solver.
Iterative Method	
`S3L_gen_iter_solve()`	An iterative solver for solving sparse linear systems of equations of the form A*x =b.
Sparse Linear Problem Solver	
`S3L_lp_sparse()`	Solve a linear/quadric optimization problem of the form `min c'*x`.
Fast Fourier Transforms	
`S3L_fft()`	Perform simple FFT on an S3L array.
`S3L_fft_detailed()`	Perform in-place forward or inverse FFT along a specified axis of an S3L array.
`S3L_ifft()`	Perform the inverse FFT on an S3L axis.
`S3L_rc_fft()`	Perform forward FFT of a real S3L array.
`S3L_cr_fft()`	Perform inverse FFT of a complex S3L array.
`S3L_fft_setup()`	Prepare internal data structure for FFT operation.
`S3L_rc_fft_setup()`	Prepare internal data structure for real-to-complex and complex-to-real FFTs.
`S3L_fft_free_setup()`	Free memory allocated to FFT setup.
`S3L_rc_fft_free_setup()`	Free memory allocated to real-to-complex or complex-to-real FFT setup.
Structured Solvers	
`S3L_gen_band_factor()`	Perform LU factorization of an n x n general banded S3L array.
`S3L_gen_band_solve()`	Solve a banded system.
`S3L_gen_band_free_factors()`	Free resources allocated to factorization of a general banded S3L array.
`S3L_gen_trid_factor()`	Compute factorization of a tridiagonal matrix.
`S3L_gen_trid_solve()`	Solve a tridiagonal system.
`S3L_gen_trid_free_factors()`	Free memory allocated to factorization of a tridiagonal matrix.
Dense Symmetric Eigenvalue Solver	
`S3L_sym_eigen()`	Find eigenvalues and, optionally, eigenvectors in Hermitian matrices.

TABLE 1-1 Sun S3L Core Mathematical Routines *(Continued)*

Function	Description
Condition Numbers	
`S3L_condition_number()`	Compute the condition numbers of one or more instances of a square S3L array.
Parallel Random Number Generators	
`S3L_setup_rand_fib()`	Initialize state table for the Lagged-Fibonacci random number generator (LFG).
`S3L_rand_fib()`	Initialize an S3L array with an LFG.
`S3L_rand_lcg()`	Initialize an S3L array with a linear congruential random number generator.
`S3L_free_rand_fib()`	Free memory allocated to the random number generator state table.
Least-Squares Solver	
`S3L_gen_lsq()`	Find the least-squares solution of an overdetermined system or the minimum norm solution of an underdetermined system.
Dense Singular Value Decomposition	
`S3L_gen_svd()`	Compute the singular value of an S3L array and, optionally, the right singular vector or left singular vector.
Autocorrelation	
`S3L_acorr_setup()`	Set up initial conditions for computing the autocorrelation of a signal.
`S3L_acorr()`	Compute 1D or 2D autocorrelation of a signal.
`S3L_acorr_free_setup()`	Free memory allocated to a particular autocorrelation setup.
Convolution	
`S3L_conv_setup()`	Set up conditions for computing the convolution of a signal.
`S3L_conv()`	Compute 1D or 2D convolution of a signal.
`S3L_conv_free_setup()`	Free memory allocated to a particular convolution setup.

TABLE 1-1 Sun S3L Core Mathematical Routines *(Continued)*

Function	Description
Deconvolution	
`S3L_deconv_setup()`	Set up initial conditions for computing the deconvolution of an S3L array.
`S3L_deconv()`	Compute 1D or 2D deconvolution of an S3L array.
`S3L_deconv_free_setup()`	Free memory allocated to a particular deconvolution setup.
Grade Elements of an Array	
`S3L_grade_up()`	Grade all elements of an S3L array in ascending order.
`S3L_grade_down()`	Grade all elements of an S3L array in descending order.
`S3L_grade_detailed_up()`	Grade elements along one axis of an S3L array in ascending order.
`S3L_grade_detailed_down()`	Grade elements along one axis of an S3L array in descending order.
Sort Elements of an Array	
`S3L_sort()`	Sort all elements of a one-dimensional array in ascending order.
`S3L_sort_up()`	Sort all elements of a one-dimensional or multidimensional array in ascending order.
`S3L_sort_down()`	Sort all elements of a one-dimensional or multidimensional array in descending order.
`S3L_sort_detailed()`	Sort elements along one axis of an S3L array in either ascending or descending order using quicksort or radixsort algorithm.
`S3L_sort_detailed_up()`	Sort elements along one axis of an S3L array in ascending order.
`S3L_sort_detailed_down()`	Sort elements along one axis of an S3L array in descending order.
Parallel Transpose	
`S3L_trans()`	Perform generalized transposition of an S3L array.

TABLE 1-2 Sun S3L Toolkit Routines

Function	Description
Create/Exit Sun S3L Environment	
`S3L_init()`	Set up Sun S3L environment.
`S3L_exit()`	Leave Sun S3L environment.
Create S3L Array Handles	
`S3L_declare()`	Declare an S3L array (basic method).
`S3L_declare_detailed()`	Declare S3L array (control more parameters).
Release S3L Array Handles	
`S3L_free()`	Release an S3L array.
Control S3L Process Grids	
`S3L_set_process_grid()`	Define an S3L process grid.
`S3L_free_process_grid()`	Release resources allocated to a process grid.
Perform Operations on S3L Arrays	
`S3L_array_op1()`	Perform operation on one array.
`S3L_array_op2()`	Perform operation on two arrays.
`S3L_array_scalar_op2()`	Perform operation on array and scalar value.
`S3L_cshift()`	Perform circular shift along a specified axis.
`S3L_forall()`	Apply a user-defined function to some or all elements in an array.
`S3L_reduce()`	Perform a reduction function across an array.
`S3L_reduce_axis()`	Perform a reduction function along one axis of an array.
`S3L_set_array_element()`	Set the value of an element of an S3L array.
`S3L_set_array_element_on_proc()`	Set the value of an element of an S3L array, using the value supplied on a specific process.
`S3L_get_array_element()`	Retrieve the value of an element of an S3L array.
`S3L_get_array_element_on_proc()`	Retrieve the value of an element of an S3L array, as supplied by a specified process.
`S3L_zero_elements()`	Set all elements in an S3L array to zero.

TABLE 1-2 Sun S3L Toolkit Routines *(Continued)*

Function	Description
Get Information About S3L Arrays	
`S3L_describe()`	Get information about an S3L array or process grid.
`S3L_get_attribute()`	Get the value of an S3L array attribute.
`S3L_read_array()`	Read an S3L array from a file.
`S3L_read_sub_array()`	Read part of an S3L array from a file.
`S3L_print_array()`	Print an S3L array to standard output.
`S3L_print_sub_array()`	Print part of an S3L array to standard output.
`S3L_write_array()`	Write an S3L array to a specified file.
`S3L_write_sub_array()`	Write part of an S3L array to a specified file.
Miscellaneous Tools	
`S3L_copy_array()`	Copy an S3L array into another S3L array.
`S3L_copy_array_detailed()`	Copy a section of an S3L array into another S3L array.
`S3L_from_ScaLAPACK_desc()`	Convert ScaLAPACK descriptor to S3L handle.
`S3L_to_ScaLAPACK_desc()`	Convert S3L handle to ScaLAPACK descriptor.
`S3L_thread_comm_setup()`	Prepare S3L environment for thread-safe operation.
`S3L_set_safety()`	Set error-checking level for S3L operations.
`S3L_get_safety()`	Get S3L error-checking level.

TABLE 1-3 Supported ScaLAPACK APIs

Category	Routines
PBLAS 1,2,3	`p{s,d}dot`, `p{c,z}dotu`, `p{s,d}nrm2`, `p{sc,dz}nrm2`, `p{s,d}ger`, `p{c,z}geru`, `p{s,d,c,z}gemv`, `p{s,d,c,z}gemm`
LU factor, solve, inverse	`p{s,d,c,z}getrf,p{s,d,c,z}getrs,p{s,d,c,z}getri`
Tridiagonal solvers	`p{s,d,c,z}dttrf, p{s,d,c,z}dttrs`
Banded solvers	`p{s,d,c,z}gbsv, p{s,d,c,z}gbtrf, p{s,d,c,z}gbtrs`
Symmetric eigensolver	`p{s,d}syevx, p{c,z}heevx`
Singular value decomposition	`p{s,d,c,z}geqrf`
Least-squares solver	`p{s,d,c,z}gels`
Condition number	`p{s,d,c,z}gecon`

CHAPTER 2

Sun S3L Functions

This chapter describes the full set of functions in Sun S3L 4.0. The functions are listed in alphabetical order, with core and toolkit routines intermixed.

S3L_2_norm and S3L_gbl_2_norm

Description

Multiple-Instance 2-norm – The multiple-instance 2-norm routine, S3L_2_norm, computes one or more instances of the 2-norm of a vector. The single-instance 2-norm routine, S3L_gbl_2_norm, computes the global 2-norm of a parallel array.

For each instance z of z, the multiple-instance routine S3L_2_norm performs the operation shown in TABLE 2-1.

TABLE 2-1 S3L Multiple-Instance 2-norm Operations

Operation	Data Type
$z = (x^T x)^{1/2} = \|\|x\|\|(2)$	Real
$z = (x^H x)^{1/2} = \|\|x\|\|(2)$	Complex

Upon successful completion, S3L_2_norm overwrites each element of z with the 2-norm of the corresponding vector in x.

Single-Instance 2-norm – The single-instance routine S3L_gbl_2_norm routine performs the operations shown in TABLE 2-2.

TABLE 2-2 S3L Single-Instance 2-norm Operations

Operation	Data Type
$a = (x^T x)^{1/2} = ||x||(2)$	Real
$a = (x^H x)^{1/2} = ||x||(2)$	Complex

Upon successful completion, a is overwritten with the global 2-norm of x.

Syntax

The C and Fortran syntax for S3L_2_norm and S3L_gbl_2_norm is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_2_norm(z, x, x_vector_axis)
S3L_gbl_2_norm(a, x)
    S3L_array_t        a
    S3L_array_t        z
    S3L_array_t        x
    int                x_vector_axis
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_2_norm(z, x, ier)
S3L_gbl_2_norm(a, x, x_vector_axis, ier)
    integer*8          a
    integer*8          z
    integer*8          x
    integer*4          x_vector_axis
    integer*4          ier
```

Input

`S3L_2_norm` accepts the following arguments as input:

- `x` – Array handle for an S3L parallel array. For calls to `S3L_2_norm` (multiple-instance routine), `x` must represent a parallel array of rank >= 2, with at least one nonlocal instance axis. The variable `x` contains one or more instances of the vector `x` whose 2-norm will be computed.

  For calls to `S3L_gbl_2_norm` (single-instance routine), `x` must represent a parallel array of rank >= 1, with any instance axes declared to have length 1.

- `x_vector_axis` – Scalar variable. Identifies the axis of `x` along which the vectors lie.

Output

`S3L_2_norm` uses the following arguments as output:

- `z` – Array handle for the S3L parallel array that will contain the results of the multiple-instance 2-norm routine. The rank of `z` must be one less than that of `x`. The axes of `z` must match the instance axes of `x` in length and order of declaration. Thus, each vector `x` in `x` corresponds to a single destination value `z` in `z`.
- `a` – Pointer to a scalar variable. Destination for the single-instance 2-norm routine.
- `ier` (Fortran only) – When called from a Fortran program, these functions return error status in `ier`.

Error Handling

On success, `S3L_2_norm` and `S3L_gbl_2_norm` return `S3L_SUCCESS`.

`S3L_2_norm` and `S3L_gbl_2_norm` perform generic checking of the validity of the arrays they accept as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause the functions to terminate and return the associated error code.

- `S3L_ERR_ARG_RANK` – `x` has invalid rank.
- `S3L_ERR_ARG_AXISNUM` – (`S3L_2_norm` only) `x_vector_axis` is a bad axis number. For C program calls, this parameter must be >= 0 and less than the rank of `x`. For Fortran program calls, it must be >= 1 and not exceed the rank of `x`.

- S3L_ERR_MATCH_RANK – z does not have a rank of one less than that of x.

Examples

/opt/SUNWhpc/examples/s3l/dense_matrix_ops/norm2.c

/opt/SUNWhpc/examples/s3l/dense_matrix_ops-f/norm2.f

Related Functions

S3L_inner_prod(3)

S3L_outer_prod(3)

S3L_mat_vec_mult(3)

S3L_mat_mult(3)

S3L_acorr

Description

S3L_acorr computes the 1D or 2D autocorrelation of a signal represented by the parallel array described by S3L array handle A. The result is stored in the parallel array described by the S3L array handle C.

A and C are S3L array handles of the same real or complex type.

For the 1D case, if A is of length ma, the result of the autocorrelation will be of length 2*ma-1. In the 2D case, if A is of size [ma,na], the result of the autocorrelation is of size [2*ma-1,2*na-1].

The size of C has to be at least equal to the size of the autocorrelation for each case, as described above. If it is larger, the excess elements of C will contain zero or non-significant entries.

The result of the autocorrelation of A is stored in wraparound order along each dimension. If the extent of C along a given axis is lc, the autocorrelation at zero lag is stored in C(0), the autocorrelation at lag 1 in C(1), and so forth. The autocorrelation at lag -1 is stored in C(lc-1), the autocorrelation at lag -2 is stored in C(lc-2), and so forth.

Side Effect

Following calculation of the autocorrelation of `A`, `A` may be destroyed, since it is used internally as auxiliary storage. If its contents will be reused after autocorrelation is performed, first copy it to a temporary array.

Note – `S3L_acorr` is most efficient when all arrays have the same length and when this length is one that can be computed efficiently by means of `S3L_fft` or `S3L_rc_fft`. See "S3L_fft" on page 101 and "S3L_rc_fft and S3L_cr_fft" on page 253 for more information about execution efficiency.

Restriction

The dimensions of array `C` must be such that a 1D or 2D complex-to-complex FFT or real-to-complex FFT can be computed.

Syntax

The C and Fortran syntax for `S3L_acorr` is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_acorr(A, C, setup_id)
    S3L_array_t       A
    S3L_array_t       C
    int               setup_id
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_acorr(A, C, setup_id, ier)
    integer*8         A
    integer*8         C
    integer*4         setup_id
    integer*4         ier
```

Input

S3L_acorr accepts the following arguments as input:

- A – S3L internal array handle for the parallel array upon which the autocorrelation will be performed. A is of size ma (1D case) or ma x na (2D case).
- setup_id – Integer value returned by a previous call to S3L_acorr_setup.

Output

S3L_acorr uses the following arguments as output:

- C – S3L internal array handle for the parallel array that contains the results of the autocorrelation. Its length must be at least 2*ma-1 (1D case) or 2*ma-1 x 2*na-1 (2D case).
- ier (Fortran only) – When called from a Fortran program, this function returns error status in ier.

Error Handling

On success, S3L_acorr returns S3L_SUCCESS.

S3L_acorr performs generic checking of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions cause the function to terminate and return one of the following codes:

- S3L_ERR_ARG_DTYPE – The data type of one of the array arguments is invalid. It must be S3L_float, S3L_double, S3L_complex, or S3L_double_complex.
- S3L_ERR_ARG_EXTENTS – The extents of C are smaller than 2*ma-1 (1D case) or 2*ma-1 x 2*na-1 (2D case).
- S3L_ERR_ARG_RANK – The rank of one of the array arguments is not 1 or 2 as required.
- S3L_ERR_MATCH_DTYPE – A and C are not the same data type.
- S3L_ERR_MATCH_RANK – A and C do not have the same rank.

In addition, since S3L_fft or S3L_rc_fft is used internally to compute the autocorrelation, if the dimensions of C are not suitable for S3L_fft or S3L_rc_fft, an error code indicating this unsuitability is returned. For more details, refer to the man pages for S3L_fft and S3L_rc_fft.

Examples

/opt/SUNWhpc/examples/s3l/acorr/ex_acorr.c

/opt/SUNWhpc/examples/s3l/acorr-f/ex_acorr.f

Related Functions

S3L_acorr_setup(3)

S3L_acorr_free_setup(3)

S3L_acorr_free_setup

Description

S3L_acorr_free_setup invalidates the ID specified by the setup_id argument. This deallocates the internal memory that was reserved for the autocorrelation computation associated with that ID.

Syntax

The C and Fortran syntax for S3L_acorr_free_setup is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_acorr_free_setup(setup_id)
    int                setup_id
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_acorr_free_setup(setup_id, ier)
    integer*4          setup_id
    integer*4          ier
```

Input

S3L_acorr_free_setup accepts the following arguments as input:

- setup_id – Valid autocorrelation setup ID as returned from a previous call to S3L_acorr_setup.

Output

S3L_acorr_free_setup uses the following arguments as output:

- ier (Fortran only) – When called from a Fortran program, this function returns error status in ier.

Error Handling

On success, S3L_acorr_free_setup returns S3L_SUCCESS.

In addition, the following condition causes the function to terminate and return the associated code:

- S3L_ERR_ARG_SETUP – Invalid setup_id value.

Examples

/opt/SUNWhpc/examples/s3l/acorr/ex_acorr.c

/opt/SUNWhpc/examples/s3l/acorr-f/ex_acorr.f

Related Functions

S3L_acorr(3)

S3L_acorr_setup(3)

S3L_acorr_setup

Description

S3L_acorr_setup sets up the initial conditions necessary for computation of the autocorrelation C = acorr(A). It returns an integer setup value that can be used by subsequent calls to S3L_acorr and S3L_acorr_free_setup.

Syntax

The C and Fortran syntax for S3L_acorr_setup is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_acorr_setup(A, C, setup_id)
    S3L_array_t        A
    S3L_array_t        C
    int                *setup_id
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_acorr_setup(A, C, setup_id, ier)
    integer*8          A
    integer*8          C
    integer*4          setup_id
    integer*4          ier
```

Input

S3L_acorr_setup accepts the following arguments as input:

- A – S3L internal array handle for the parallel 1D or 2D array of real or complex type whose autocorrelation is to be computed.
- C – S3L internal array handle for the parallel 1D or 2D array of the same type as A, used to store the result of the autocorrelation. Its extents along each axis must be at least equal to two times the corresponding extent of A minus 1.

Output

S3L_acorr_setup uses the following arguments as output:

- setup – Integer value returned by this function. Use this value for the setup_id argument in subsequent calls to S3_acorr and S3L_acorr_free_setup.
- ier (Fortran only) – When called from a Fortran program, this function returns error status in ier.

Error Handling

On success, S3L_acorr_setup returns S3L_SUCCESS.

S3L_acorr_setup performs generic checking of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions cause the function to terminate and return one of the following codes:

- S3L_ERR_ARG_DTYPE – The data type of one of the array arguments is invalid. It must be S3L_float, S3L_double, S3L_complex, or S3L_double_complex.
- S3L_ERR_MATCH_DTYPE – The array arguments are not all of the same data type.
- S3L_ERR_MATCH_RANK – The array arguments are not all of the same rank.
- S3L_ERR_ARG_RANK – The rank of one of the array arguments is not 1 or 2 as required.
- S3L_ERR_ARG_EXTENTS – The extents of C are less than the extents of A.

Examples

/opt/SUNWhpc/examples/s3l/acorr/ex_acorr.c

/opt/SUNWhpc/examples/s3l/acorr-f/ex_acorr.f

Related Functions

S3L_acorr(3)

S3L_acorr_free_setup(3)

S3L_array_op1

Description

S3L_array_op1 applies a predefined unary (single operand) operation to each element of an S3L parallel array. The S3L array handle argument, a, identifies the parallel array to be operated on and the op argument specifies the operation to be performed. The value of op must be:

- S3L_OP_ABS – Replaces each element in a with its absolute value.
- S3L_OP_MINUS – Replaces each element in a with its negative value.
- S3L_OP_EXP – Replaces each element in the real or complex array a with its exponential.

Syntax

The C and Fortran syntax for `S3L_array_op1` is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_array_op1(a, op)
    S3L_array_t        a
    S3L_op_type        op
```

F77/F90

```
include ‘s3l/s3l-f.h’
include ‘s3l/s3l_errno-f.h’
subroutine
S3L_array_op1(a, op, ier)
    integer*8          a
    integer*4          op
    integer*4          ier
```

Input

`S3L_array_op1` accepts the following arguments as input:

- `a` – S3L array handle for the parallel array on which the operation will be performed.
- `op` – Predefined constant specifying the operation to be applied. See the Description section for details.

Output

`S3L_array_op1` uses the following argument for output:

- `ier` (Fortran only) – When called from a Fortran program, `S3L_array_op1` returns error status in `ier`.

Error Handling

On success, S3L_array_op1 returns S3L_SUCCESS.

S3L_array_op1 performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following condition will cause the function to terminate and return the associated error code.

- S3L_ERR_ARG_DTYPE – op is equal to S3L_OP_EXP, but a is of integer type.

Examples

/opt/SUNWhpc/examples/s3l/fft/ex_fft1.c

/opt/SUNWhpc/examples/s3l/deconv-f/ex_deconv.f

Related Functions

S3L_array_op2(3)

S3L_array_scalar_op2(3)

S3L_reduce(3)

S3L_array_op2

Description

S3L_array_op2 applies the operation specified by op to elements of parallel arrays a and b, which must be of the same type and have the same distribution. The parameter op can be one of the following:

- S3L_OP_MUL – a equals a .* b
- S3L_OP_CMUL – a equals a .* conjg(b)

- S3L_OP_DIV – a equals a ./ b
- S3L_OP_MINUS – a equals a – b
- S3L_OP_PLUS – a equals a + b

Note – The operators ".*" and "./" denote pointwise multiplication and division of the elements in arrays a and b.

S3L_OP_MUL replaces each element in a with the elementwise product of multiplying a and b.

S3L_OP_CMUL performs the same operation as S3L_OP_MUL, except it multiplies each element in a by the conjugate of the corresponding element in b.

S3L_OP_DIV performs elementwise division of a by b, overwriting a with the integer (truncated quotient) results.

S3L_OP_MINUS performs elementwise subtraction of b from a, overwriting a with the difference.

S3L_OP_PLUS performs elementwise addition of a with b, overwriting a with the sum.

Syntax

The C and Fortran syntax for S3L_array_op2 is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_array_op2(a, b, op)
    S3L_array_t       a
    S3L_array_t       b
    S3L_op_type       op
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_array_op2(a, b, op, ier)
    integer*8          a
    integer*8          b
    integer*4          op
    integer*4          ier
```

Input

S3L_array_op2 accepts the following arguments as input:

- a – S3L array handle for one of two parallel arrays to which the operation will be applied.
- b – S3L array handle for the second of two parallel arrays to which the operation will be applied.
- op – Predefined constant specifying the operation to be applied. See the Description section for details.

Output

S3L_array_op2 uses the following argument for output:

- ier (Fortran only) – When called from a Fortran program, S3L_array_op2 returns error status in ier.

Error Handling

On success, S3L_array_op2 returns S3L_SUCCESS.

S3L_array_op2 performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause the function to terminate and return the associated error code:

- S3L_ERR_MATCH_HOME – Both arrays are local but not on the same process.

- S3L_ERR_MATCH_ALIGN – The arrays do not have the same subgrid sizes.
- S3L_ERR_ARG_OP – An illegal operation was requested.

Examples

/opt/SUNWhpc/examples/s3l/fft/ex_fft1.c

/opt/SUNWhpc/examples/s3l/fft-f/ex_fft1.f

Related Functions

S3L_array_op1(3)

S3L_array_scalar_op2(3)

S3L_array_scalar_op2

Description

S3L_array_scalar_op2 applies a binary operation to each element of an S3L array that involves the element and a scalar.

op determines which operation will be performed. It can be one of:

- S3L_OP_MULT – Pointwise multiplication.
- S3L_OP_DIV – Pointwise division.
- S3L_OP_PLUS – Pointwise addition.
- S3L_OP_MINUS – Pointwise subtraction.
- S3L_OP_ASSIGN – Assignment.

Syntax

The C and Fortran syntax for S3L_array_scalar_op2 is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_array_scalar_op2(a, scalar, op)
    S3L_array_t         a
    void                *scalar
    S3L_op_type         op
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_array_scalar_op2(a, scalar, op, ier)
    integer*8          a
    <type>             scalar
    integer*4          op
    integer*4          ier
```

where <type> is one of: integer*4, integer*8, real*4, real*8, complex*8, or complex*16.

Input

S3L_array_scalar_op2 accepts the following arguments as input:

- a – S3L array handle for the parallel array to which the operation will be applied.
- scalar – Scalar value used as an operand in the operation applied to each element of a.
- op – Predefined constant specifying the operation to be applied. See the Description section for details.

Output

S3L_array_scalar_op2 uses the following argument for output:

- ier (Fortran only) – When called from a Fortran program, S3L_array_scalar_op2 returns error status in ier.

Error Handling

On success, S3L_array_scalar_op2 returns S3L_SUCCESS.

S3L_array_scalar_op2 performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following condition will cause the function to terminate and return the associated error code:

- S3L_ERR_ARG_OP – op is not one of: S3L_OP_MUL, S3L_OP_DIV, S3L_OP_PLUS, S3L_OP_MINUS, or S3L_OP_ASSIGN

Examples

/opt/SUNWhpc/examples/s3l/fft/ex_fft1.c

/opt/SUNWhpc/examples/s3l/fft-f/ex_fft1.f

Related Functions

S3L_array_op1(3)

S3L_array_op2(3)

S3L_cholesky_factor

Description

For each square A in a, S3L_cholesky_factor computes the Cholesky factorization. The factorization has the form A = U' x U, where U is an upper triangular matrix.

Syntax

The C and Fortran syntax for `S3L_cholesky_factor` is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_cholesky_factor(a, row_axis, col_axis)
    S3L_array_t         a
    int                 row_axis
    int                 col_axis
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_cholesky_factor(a, row_axis, col_axis, ier)
    integer*8           a
    integer*4           row_axis
    integer*4           col_axis
    integer*4           ier
```

Input

`S3L_cholesky_factor` accepts the following arguments as input:

- `a` – S3L array of rank 2 or greater. This array contains one or more instances of a square matrix, `A`, which is to be factored. Each `A` is assumed to be dense, with rows counted by axis `row_axis` and columns counted by axis `col_axis`.

  Upon successful completion, each matrix instance `A` is overwritten with the upper triangular matrix `U`.

- `row_axis` – Scalar integer variable. Identifies the axis of `a` that counts the rows of each matrix `A`.

  For C program calls, `row_axis` must be >= 0 and less than the rank of `a`. For Fortran program calls, `row_axis` must be >= 1 and not exceed the rank of `a`. In addition, `row_axis` must be less than `col_axis`.

- `col_axis` – Scalar integer variable. Identifies the axis of `a` that counts the columns of each matrix `A`.

For C program calls, col_axis must be >= 0 and less than the rank of a. For Fortran program calls, col_axis must be >= 1 and not exceed the rank of a. In addition, col_axis must be greater than row_axis.

Output

S3L_cholesky_factor uses the following arguments for output:

- a – On exit, each matrix instance A is overwritten with the upper triangular matrix U.
- ier (Fortran only) – When called from a Fortran program, this function returns error status in ier.

Error Handling

On success, S3L_cholesky_factor returns S3L_SUCCESS.

S3L_cholesky_factor performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause S3L_cholesky_factor to terminate and return the associated error code:

- S3L_ERR_ARG_RANK – Invalid rank. The rank of a must be >= 2.
- S3L_ERR_ARG_DTYPE – Invalid data type. The data type of a must be real or complex (single- or double-precision).
- S3L_ERR_ARG_AXISNUM – row_axis or col_axis is invalid. This condition can be caused by either an out-of-range axis value or row_axis = col_axis. See the row_axis or col_axis argument description for allowed axis index ranges.
- S3L_ERR_ARRNOTSQ – Arrays A in a are not square.
- S3L_ERR_FACTOR_FAIL – Factorization could not be completed.

Examples

/opt/SUNWhpc/examples/s3l/cholesky/cholesky.c

/opt/SUNWhpc/examples/s3l/cholesky-f/cholesky.f

Related Functions

S3L_cholesky_solve(3)

S3L_cholesky_invert(3)

S3L_cholesky_invert

Description

For each square matrix A in a, S3L_cholesky_invert uses the result from S3L_cholesky_factor to compute the inverse of each square matrix instance A of the S3L array a. It does this by inverting the Cholesky factor U and then computing inverse(U) * inverse(U)'.

Syntax

The C and Fortran syntax for S3L_cholesky_invert is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_cholesky_invert(a, row_axis, col_axis)
    S3L_array_t         a
    int                 row_axis
    int                 col_axis
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_cholesky_invert(a, row_axis, col_axis, ier)
    integer*8          a
    integer*4          row_axis
    integer*4          col_axis
    integer*4          ier
```

Input

S3L_cholesky_invert accepts the following arguments as input:

- a – S3L array that was factored by S3L_cholesky_factor, where each matrix instance A is a dense square matrix. Supply the same value a that was used in S3L_cholesky_factor.
- row_axis – Scalar integer variable. Identifies the axis of a that counts the rows of each matrix A.

  For C program calls, row_axis must be >= 0 and less than the rank of a. For Fortran program calls, row_axis must be >= 1 and not exceed the rank of a. In addition, row_axis must be less than col_axis.
- col_axis – Scalar integer variable. Identifies the axis of a that counts the columns of each matrix A.

  For C program calls, col_axis must be >= 0 and less than the rank of a. For Fortran program calls, col_axis must be >= 1 and not exceed the rank of a. In addition, col_axis must be greater than row_axis.

Output

S3L_cholesky_invert uses the following argument for output:

- ier (Fortran only) – When called from a Fortran program, this function returns error status in ier.

Error Handling

On success, S3L_cholesky_invert returns S3L_SUCCESS.

S3L_cholesky_invert performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause S3L_cholesky_solve to terminate and return the associated error code:

- S3L_ERR_ARG_RANK – Invalid rank. The rank of a must be >= 2.
- S3L_ERR_ARG_DTYPE – Invalid data type. The data type of a must be real or complex (single- or double-precision).
- S3L_ERR_ARG_AXISNUM – row_axis or col_axis is invalid. This condition can be caused by either an out-of-range axis value or row_axis = col_axis. See the row_axis or col_axis argument description for allowed axis index ranges.
- S3L_ERR_ARRNOTSQ – The arrays A in a are not square.
- S3L_ERR_FACTOR_FAIL – A diagonal element in U (the array containing factorization of a from a previous call to S3L_cholesky_factor) is zero; therefore, inversion could not be performed.

Examples

/opt/SUNWhpc/examples/s3l/cholesky/cholesky.c

/opt/SUNWhpc/examples/s3l/cholesky-f/cholesky.f

Related Functions

S3L_cholesky_factor(3)

S3L_cholesky_solve(3)

S3L_cholesky_solve

Description

For each square matrix A in a, S3L_cholesky_solve solves a system of distributed linear equations of the form AX = B, using Cholesky factors computed by S3L_cholesky_factor.

A and B are corresponding instances within a and b, respectively. To solve AX = B, S3L_cholesky_solve performs the following by means of back substitution:

1. Solve U′ * X = B, overwriting B with X

2. Solve U * X = B, overwriting B with X

Syntax

The C and Fortran syntax for S3L_cholesky_solve is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_cholesky_solve(a, row_axis, col_axis, b)
    S3L_array_t        a
    int                row_axis
    int                col_axis
    S3L_array_t        b
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_cholesky_solve(a, row_axis, col_axis, b, ier)
    integer*8          a
    integer*4          row_axis
    integer*4          col_axis
    integer*8          b
    integer*4          ier
```

Input

S3L_cholesky_solve accepts the following arguments as input:

- a – S3L array that was factored by S3L_cholesky_factor, where each matrix instance A is a dense square matrix. Supply the same value a that was used in S3L_cholesky_factor.

- `row_axis` – Scalar integer variable. Identifies the axis of `a` that counts the rows of each matrix `A`.

  For C program calls, `row_axis` must be >= 0 and less than the rank of `a`. For Fortran program calls, `row_axis` must be >= 1 and not exceed the rank of `a`. In addition, `row_axis` must be less than `col_axis`.

- `col_axis` – Scalar integer variable. Identifies the axis of `a` that counts the columns of each matrix `A`.

  For C program calls, `col_axis` must be >= 0 and less than the rank of `a`. For Fortran program calls, `col_axis` must be >= 1 and not exceed the rank of `a`. In addition, `col_axis` must be greater than `row_axis`.

- `b` – S3L array of the same type (real or complex) and precision as `a`. Array `b` must be distinct from `a`.

  The instance axes of `b` must match those of `a` in order of declaration and extents. The rows and columns of `b` must be counted by axes `row_axis` and `col_axis`, respectively (from the `S3L_cholesky_factor` call). If `b` consists of only one right-hand side vector, it must be represented as an array of rank 2 with the number of columns set to 1 and the elements counted by axis `row_axis`.

Output

`S3L_cholesky_solve` uses the following argument for output:

- `ier` (Fortran only) – When called from a Fortran program, this function returns error status in `ier`.

Error Handling

On success, `S3L_cholesky_solve` returns `S3L_SUCCESS`.

`S3L_cholesky_solve` performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause `S3L_cholesky_solve` to terminate and return the associated error code:

- `S3L_ERR_MATCH_RANK` – Invalid rank. For cases where rank >= 3, rank(`b`) must equal rank(`a`). For the two-dimensional case, rank(`b`) must be either 1 or 2.
- `S3L_ERR_ARG_DTYPE` – Invalid data type. The data type of `a` must be real or complex (single- or double-precision).

- S3L_ERR_MATCH_EXTENTS – The extents of a and b are mismatched along the row or instance axis.
- S3L_ERR_MATCH_DTYPE – The data types of a and b do not match.
- S3L_ERR_ARRNOTSQ – Invalid matrix size. Each coefficient matrix A must be square.
- S3L_ERR_ARG_AXISNUM – row_axis or col_axis is invalid. This condition can be caused by either an out-of-range axis value or row_axis = col_axis. See the row_axis or col_axis argument description for allowed axis index ranges.

Examples

/opt/SUNWhpc/examples/s3l/cholesky/cholesky.c

/opt/SUNWhpc/examples/s3l/cholesky-f/cholesky.f

Related Functions

S3L_cholesky_factor(3)

S3L_cholesky_invert(3)

S3L_condition_number, S3L_gbl_condition_number

Description

S3L_condition_number and S3L_gbl_condition_number compute the condition numbers of square arrays. LU factorization is used internally in combination with a norm as specified by the argument norm_type.

Note – Array variables must not overlap.

Performance Note

The condition number functions perform LU factorization and compute the norm internally. If these operations are already performed elsewhere in the calling program, you can achieve better performance by calling one of the following ScaLAPACK functions directly: psgecon, pdgecon, pcgecon, or pzgecon. To use any of these ScaLAPACK functions, you will need a ScaLAPACK descriptor, which you can obtain from the corresponding S3L array descriptor with the routine S3L_to_ScaLAPACK_desc(3).

Syntax

The C and Fortran syntax for S3L_condition_number and S3L_gbl_condition_number is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_condition_number(rcn_array, a, row_axis, col_axis,
norm_type)
S3L_gbl_condition_number(rcn_array, a, row_axis, col_axis,
norm_type)
    void              *rcn
    S3L_array_t       rcn_array
    S3L_array_t       a
    int               row_axis
    int               col_axis
    int               norm_type
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_condition_number(rcn_array, a, row_axis, col_axis,
norm_type, ier)
S3L_gbl_condition_number(rcn_array, a, row_axis, col_axis,
norm_type, ier)
    <type>            rcn
    integer*8         rcn_array
    integer*8         a
    integer*4         row_axis
    integer*4         col_axis
    integer*4         norm_type
    integer*4         ier
```

Input

S3L_condition_number accepts the following arguments as input:

- a – S3L array of rank 2 or greater. a contains one or more instances of a square matrix A whose condition number is to be computed.
- row_axis – Scalar integer variable. Identifies the axis of a that counts the rows of each matrix A.

  For C program calls, row_axis must be >= 0 and less than the rank of a. For Fortran program calls, row_axis must be >= 1 and not exceed the rank of a. In addition, row_axis must be less than col_axis.

- col_axis – Scalar integer variable. Identifies the axis of a that counts the columns of each matrix A.

  For C program calls, col_axis must be >= 0 and less than the rank of a. For Fortran program calls, col_axis must be >= 1 and not exceed the rank of a. In addition, col_axis must be greater than row_axis .

- norm_type – Specifies the type of norm to be used in calculating the condition number. Allowed values are:

S3L_ONENORM_CONDITION_NO	Use the 1-norm.
S3L_INFNORM_CONDITION_NO	Use the infinity norm.

Output

S3L_condition_number uses the following arguments for output:

- rcn_array – S3L array whose rank is two less than the rank a. It should be of data type real with the same precision as a. On exit, each element in rcn_array will hold the reciprocal condition number of the corresponding array A.
- rcn – Pointer to a scalar variable of data type real with the same precision as a. Upon exit, rcn will hold the reciprocal condition number of a.
- ier (Fortran only) – When called from a Fortran program, this function returns error status in ier.

Error Handling

On success, both S3L_condition_number and S3L_gbl_condition_number return S3L_SUCCESS.

S3L_condition_number and S3L_gbl_condition_number perform generic checking of the arrays they accept as arguments. If an array argument contains an invalid or corrupted value, these functions will terminate and an error code indicating which value of the array handle was invalid will be returned. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause these functions to terminate and return the associated error code:

- S3L_ERR_ARG_RANK – The rank of a is 1.
- S3L_ERR_MATCH_RANK – The rank of rcn_array is not valid. It must be equal to two less than the rank of a.
- S3L_ERR_ARG_AXISNUM – row_axis or col_axis is invalid. This condition can be caused by either an out-of-range axis value or row_axis = col_axis. See the row_axis or col_axis argument description for allowed axis index ranges.
- S3L_ERR_ARRNOTSQ – The arrays A in a are not square.
- S3L_ERR_MATCH_EXTENTS – The instance axes of rcn_array and a do not have the same extents.
- S3L_ERR_ARG_DTYPE – Invalid data type. The data type of a must be real (single- or double-precision).
- S3L_ERR_ARG_OP – The value supplied for norm_type is not either S3L_ONENORM_CONDITION_NO or S3L_INFNORM_CONDITION_NO.

Examples

```
/opt/SUNWhpc/examples/s3l/condition_number/gbl_condition_number.c
/opt/SUNWhpc/examples/s3l/condition_number/condition_number.c
/opt/SUNWhpc/examples/s3l/condition_number-f/gbl_condition_number.f
/opt/SUNWhpc/examples/s3l/condition_number-f/condition_number.f
```

Related Function

`S3L_lu_factor(3)`

S3L_conv

Description

`S3L_conv` computes the 1D or 2D convolution of a signal represented by a parallel array using a filter contained in a second parallel array. The result is stored in a third parallel array. These parallel arrays are described by the S3L array handles: `a` (signal), `b` (filter), and `c` (result). All three arrays are of the same real or complex type.

For the 1D case, if the signal `a` is of length ma and the filter `b` of length mb, the result of the convolution, `c`, will be of length ma + mb - 1. In the 2D case, if the signal is of size [ma,na] and the filter is of size [mb,nb], the result of the convolution is of size [ma+mb-1,na+nb-1].

Side Effect

Because `a` and `b` are used internally for auxiliary storage, they may be destroyed after the convolution calculation is complete. If the contents of `a` and `b` must be used after the convolution, they should first be copied to temporary arrays.

Note – `S3L_conv` is most efficient when all arrays have the same length and when this length can be computed efficiently by means of `S3L_fft` or `S3L_rc_fft`. See "S3L_fft" on page 101 and "S3L_rc_fft and S3L_cr_fft" on page 253 for additional information.

Restriction

The dimensions of the array `c` must be such that the 1D or 2D complex-to-complex FFT or real-to-complex FFT can be computed.

Syntax

The C and Fortran syntax for `S3L_conv` is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_conv(a, b, c, setup_id)
    S3L_array_t       a
    S3L_array_t       b
    S3L_array_t       c
    int               *setup_id
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_conv(a, b, c, setup_id, ier)
    integer*8         a
    integer*8         b
    integer*8         c
    integer*4         setup_id
    integer*4         ier
```

Input

`S3L_conv` accepts the following arguments as input:

- `a` – S3L array handle describing a parallel array of size ma (1D case) or ma x na (2D) case. `a` is the input signal that will be convolved.
- `b` – S3L array handle describing the parallel array that contains the filter.
- `setup_id` – Valid convolution setup ID as returned from a previous call to `S3L_conv_setup`.

Output

S3L_conv uses the following arguments for output:

- c – S3L array handle describing a parallel array containing the convolved signal. Its length must be at least ma+mb-1 (1D case) or ma+mb-1 x na+nb-1 (2D case).
- ier (Fortran only) – When called from a Fortran program, this function returns error status in ier.

Error Handling

On success, S3L_conv returns S3L_SUCCESS.

S3L_conv performs generic checking of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions cause the function to terminate and return one of the following error codes:

- S3L_ERR_MATCH_DTYPE – a, b, and c do not have the same data type.
- S3L_ERR_MATCH_RANK – a, b, and c do not have the same rank.
- S3L_ERR_ARG_RANK – The rank of an array argument is larger than 2.
- S3L_ERR_ARG_DTYPE – The data type of one of the array arguments is invalid. It must be one of:
  - S3L_float
  - S3L_double
  - S3L_complex
  - S3L_double_complex
- S3L_ERR_ARG_EXTENTS – The extents of c are smaller than two times the sum of the corresponding extents of a and b minus 1.

Examples

/opt/SUNWhpc/examples/s3l/conv/ex_conv.c

/opt/SUNWhpc/examples/s3l/conv-f/ex_conv.f

Related Functions

S3L_conv_setup(3)

S3L_conv_free_setup(3)

S3L_conv_free_setup

Description

S3L_conv_free_setup invalidates the ID specified by the setup_id argument. This deallocates the internal memory that was reserved for the convolution computation represented by that ID.

Syntax

The C and Fortran syntax for S3L_conv_free_setup is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_conv_free_setup(setup_id)
    int                 setup_id
```

F77/F90 Syntax

```
include ‘s3l/s3l-f.h’
include ‘s3l/s3l_errno-f.h’
subroutine
S3L_conv_free_setup(setup_id, ier)
    integer*4         setup_id
    integer*4         ier
```

Input

S3L_conv_free_setup accepts the following arguments as input:

- setup_id – Integer value returned by a previous call to S3L_conv_setup.

Output

S3L_conv_free_setup uses the following argument for output:

- ier (Fortran only) – When called from a Fortran program, this function returns error status in ier.

Error Handling

On success, S3L_conv_free_setup returns S3L_SUCCESS.

In addition, the following condition causes the function to terminate and return the associated code:

- S3L_ERR_ARG_SETUP – Invalid setup_id value.

Examples

/opt/SUNWhpc/examples/s3l/conv/ex_conv.c

/opt/SUNWhpc/examples/s3l/conv-f/ex_conv.f

Related Functions

S3L_conv(3)

S3L_conv_setup(3)

S3L_conv_setup

Description

S3L_conv_setup sets up the initial conditions necessary for computation of the convolution C = A conv B. It returns an integer setup value that can be used by a subsequent call to S3L_conv.

S3L array handles A, B, and C each describe a parallel array that can be either one- or two-dimensional. The extents of C along each axis i must be such that they are greater than or equal to two times the sum of the corresponding extents of A and B minus 1.

Syntax

The C and Fortran syntax for S3L_conv_setup is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_conv_setup(A, B, C, setup_id)
    S3L_array_t       A
    S3L_array_t       B
    S3L_array_t       C
    int               *setup_id
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_conv_setup(A, B, C, setup_id, ier)
    integer*8        A
    integer*8        B
    integer*8        C
    integer*4        setup_id
    integer*4        ier
```

Input

S3L_conv_setup accepts the following arguments as input:

- A – S3L array handle describing a parallel array of size ma (1D case) or ma x na (2D) case. A contains the input signal that will be convolved.
- B – S3L array handle describing a parallel array that contains the convolution filter.
- C – S3L array handle describing a parallel array in which the convolved signal is stored. Its length must be at least ma+mb-1 (1D case) or ma+mb-1 x na+nb-1 (2D case).

Output

S3L_conv_setup uses the following arguments for output:

- setup_id – Integer value returned by this function. Use this value for the setup_id argument in subsequent calls to S3_conv and S3L_conv_free_setup.
- ier (Fortran only) – When called from a Fortran program, this function returns error status in ier.

Error Handling

On success, S3L_conv_setup returns S3L_SUCCESS.

S3L_conv_setup performs generic checking of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions cause the function to terminate and return one of the following error codes:

- S3L_ERR_ARG_RANK – The rank of one of the array arguments is not 1 or 2.
- S3L_ERR_MATCH_RANK – The array arguments are not all of the same rank.
- S3L_ERR_MATCH_DTYPE – The array arguments are not all of the same data type.
- S3L_ERR_ARG_EXTENTS – The extents of C are less two times the sum of the corresponding extents of A and B minus 1.

Examples

/opt/SUNWhpc/examples/s3l/conv/ex_conv.c

/opt/SUNWhpc/examples/s3l/conv-f/ex_conv.f

Related Functions

S3L_conv(3)

S3L_conv_free_setup(3)

S3L_convert_sparse

Description

S3L_convert_sparse converts an S3L sparse matrix that is represented in one sparse format to a different sparse format. It supports the following sparse matrix storage formats:

S3L_SPARSE_COO	Coordinate format
S3L_SPARSE_CSR	Compressed Sparse Row format
S3L_SPARSE_CSC	Compressed Sparse Column format
S3L_SPARSE_VBR	Variable Block Row format

Detailed descriptions of the first three sparse formats are provided in "S3L_declare_sparse" on page 73. They are also described in the S3L_declare_sparse man page.

The Variable Block Row (VBR) format can be viewed as a generalization of the Compressed Sparse Row format, where

- The block entries of a matrix are stored block row by block row.
- Each block entry is stored as a dense matrix in standard column-major form.

More specifically, the data structure of S3L_SPARSE_VBR consists of the following six arrays:

rptr	Integer array. It contains the block row partitionings—that is, the first row number of each block row.
cptr	Integer array. It contains the block column partitionings—that is, the first column number of each block column.
val	Scalar array. It contains the block entries of the matrix.
indx	Integer array. It contains the pointers to the beginning of each block entry stored in val.
bindx	Integer array. It contains the block column indices of block entries of the matrix.
bptr	Integer array. It contains pointers to the beginning of each block row in bindx and val.

To illustrate the VBR data structure, consider the following 5x8 matrix with a variable block partitioning.

```
     0  1   2  3  4    5    6   7   8
   +------+---------+----+--------+
 0 | 1  3  5 | 0  0 |  9 |  0   0 |
 1 | 2  4  6 | 0  0 | 10 |  0   0 |
   +---------+------+----+--------+
 2 | 0  0  0 | 7  8 | 11 |  0   0 |
   +------+---------+----+--------+
 3 | 0  0  0 | 0  0 | 12 | 14  16 |
 4 | 0  0  0 | 0  0 | 13 | 15  17 |
   +------+---------+----+--------+
 5
```

The sparsity pattern for this matrix is:

```
     0      1     2     3    4
   +-----+-----+-----+-----+
 0 |  x  |  o  |  x  |  o  |
   +-----+-----+-----+-----+
 1 |  o  |  x  |  x  |  o  |
   +-----+-----+-----+-----+
 2 |  o  |  o  |  x  |  x  |
   +-----+-----+-----+-----+
 3
```

where `x` and `o` , respectively, are the nonzero and zero block entries of the matrix.

The matrix could be stored in VBR format as follows (using zero-based indexing):

```
rptr  = ( 0, 2, 3, 6 ),

cptr  = ( 0, 2, 5, 6, 8 ),

bptr  = ( 0, 2, 4, 6 ),

bindx = ( 0, 2, 1, 2, 2, 3 ),

indx  = ( 0, 6, 8, 10, 11, 13, 17 ),

val   = ( 1.0, 2.0, 3.0, 4.0, 6.0, 9.0, 10.0. 7.0,
          8.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0, 17.0 )
```

Syntax

The C and Fortran syntax for `S3L_convert_sparse` is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_convert_sparse(A, B, spfmt, ...)
    S3L_array_t              A
    S3L_array_t              *B
    S3L_sparse_storage_t     spfmt
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_convert_sparse(A, B, spfmt, ..., ier)
    integer*8               A
    integer*8               B
    integer*4               spfmt
    integer*4               ier
```

Input

`S3L_convert_sparse` accepts the following arguments as input:

- `A` – S3L internal array handle an m x n S3L sparse matrix to be converted. This handle is produced by a prior call to one of the following sparse routines:
  - `S3L_declare_sparse`
  - `S3L_read_sparse`
  - `S3L_rand_sparse`
  - `S3L_convert_sparse`
- `spfmt` – Specifies the sparse format into which `A` is to be converted.

  If the value of `spfmt` is `S3L_SPARSE_VBR`, the following four arguments should also be supplied:

  - `bm` – Scalar integer. Indicates the total number of block rows in the block sparse matrix.
  - `bn` – Scalar integer. Indicates the total number of block columns in the block sparse matrix.
  - `rptr` – Integer array of length bm+1 such that `rptr[i]` is the row index of the first point row in the i-th block row.
  - `cptr` – Integer array of length bn+1 such that `cptr[j]` is the column index of the first point column in the j-th block row.

  If used, `bm`, `bn`, `rptr`, and `cptr` follow the `spfmt` argument, as indicated by the "`...`" in the Syntax section.

Note – The four VBR-specific arguments give the user explicit control over the block partitioning structure. To use the S3L internal blocking algorithm instead, specify these arguments as NULL pointers (for C/C++) or set to –1 (for F77/F90).

Output

S3L_convert_sparse uses the following arguments for output:

- A – On output, this is the S3L internal array handle for the global general sparse matrix that resulted from the format conversion.
- B – Contains the converted S3L sparse matrix.
- ier (Fortran only) – When called from a Fortran program, this function returns error status in ier.

Error Handling

On success, the status of S3L_convert_sparse is S3L_SUCCESS.

S3L_convert_sparse performs generic checking of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function will terminate and an error code indicating which value of the array handle was invalid will be returned. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause this function to terminate and return the associated error code:

- S3L_ERR_ARG_NULL – Value specified for A is invalid. A must be a predefined sparse matrix. Otherwise, a NULL array is encountered for rptr or cptr. When specifying spfmt = S3L_SPARSE_VBR, bm and bn should be nonzero and rptr and cptr should be predefined.
- S3L_ERR_SPARSE_FORMAT – Invalid sparse format. It must be one of: S3L_SPARSE_COO, S3L_SPARSE_CSR, S3L_SPARSE_CSC, or S3L_SPARSE_VBR.

Examples

/opt/SUNWhpc/examples/s3l/sparse/ex_sparse1.c

/opt/SUNWhpc/examples/s3l/sparse-f/ex_sparse1.f

Related Function

S3L_declare_sparse(3)

S3L_copy_array

Description

S3L_copy_array copies the contents of array A into array B, which must have the same rank, extents, and data type as A.

Syntax

The C and Fortran syntax for S3L_copy_array is as follows.

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_copy_array(A, B)
    S3L_array_t             A
    S3L_array_t             B
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_copy_array(A, B, ier)
    integer*8               A
    integer*8               B
    integer*4               ier
```

Input

S3L_copy_array accepts the following arguments as input:

- A – S3L array handle for the parallel array to be copied.

Output

S3L_copy_array uses the following arguments for output:

- B – S3L array handle for a parallel array of the same rank, extents, and data type as A. On successful completion, B contains a copy of the contents of A.
- ier (Fortran only) – When called from a Fortran program, this function returns error status in ier.

Error Handling

On success, S3L_copy_array returns S3L_SUCCESS.

S3L_copy_array checks the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause the function to terminate and return the associated code:

- S3L_ERR_MATCH_RANK – The ranks of A and B do not match.
- S3L_ERR_MATCH_EXTENTS – The extents of A and B do not match.
- S3L_ERR_MATCH_DTYPE – The data types of A and B do not match.
- S3L_ERR_ARG_DTYPE – The data type of A and/or B is invalid.

Examples

/opt/SUNWhpc/examples/s3l/utils/copy_array.c

/opt/SUNWhpc/examples/s3l/utils-f/copy_array.f

S3L_copy_array_detailed

Description

S3L_copy_array_detailed copies an array section of array a to an array section of array b. The array section of a is defined along each axis by indices:

```
lba(i) <= j <= uba(i), with strides sta(i), i=0, rank -1
```

The array section of array b is defined along each axis by indices:

```
lbb(i) <= j <= ubb(i), with strides stb(i), i=0, rank -1
```

If perm is NULL (C/C++) or its first element is negative (F77/F90), it is ignored. Otherwise, the axes of b are permuted similarly to the permutation performed by S3L_trans.

Syntax

The C and Fortran syntax for S3L_copy_array_detailed is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_copy_array_detailed(a, b, lba, uba, sta, lbb, ubb, stb,
perm)
    S3L_array_t        a
    S3L_array_t        b
    int                *lba
    int                *uba
    int                *sta
    int                *lbb
    int                *ubb
    int                *stb
    int                *perm
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_copy_array_detailed(a, b, lba, uba, sta, lbb, ubb, stb,
perm, ier)
    integer*8          a
    integer*8          b
    integer*4          lba(*)
    integer*4          uba(*)
    integer*4          sta(*)
    integer*4          lbb(*)
    integer*4          ubb(*)
    integer*4          stb(*)
    integer*4          perm(*)
    integer*4          ier
```

Input

`S3L_copy_array_detailed` accepts the following arguments as input:

- `a` – S3L array whose elements will be copied into array `b`.
- `lba` – Lower bound of the array section of `a` to be copied.
- `uba` – Upper bound of the array section of `a` to be copied.
- `sta` – Stride used to define the elements of the array section of `a` to be copied.

- lbb – Lower bound of the array section of b into which the array section of a is to be copied.
- ubb – Upper bound of the array section of b into which the array section of a is to be copied.
- stb – Stride used to define the elements of the array section of b into which the array section of a is to be copied.
- perm – Axes permutation vector.

Output

S3L_copy_array_detailed uses the following argument for output:

- b – S3L array which, on exit, will contain elements copied from array a.
- ier (Fortran only) – When called from a Fortran program, this function returns error status in ier.

Error Handling

On success, S3L_copy_array_detailed returns S3L_SUCCESS.

S3L_copy_array_detailed performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause S3L_copy_array_detailed to terminate and return the associated error code:

- S3L_ERR_MATCH_RANK – a and b do not have the same number of dimensions (rank).
- S3L_ERR_MATCH_DTYPE – a and b do not have the same data type.
- S3L_ERR_INDX_INVALID – One or more of the lower bound, upper bound, stride, or permutation axis parameters is invalid.
- S3L_ERR_TRANS_PERMAX – The permutation axis argument contains invalid entries.

Examples

/opt/SUNWhpc/examples/s3l/utils/copy_array_det.c

/opt/SUNWhpc/examples/s3l/utils-f/copy_array_det.f

Related Functions

S3L_copy_array(3)

S3L_trans(3)

S3L_cshift

Description

S3L_cshift performs a circular shift of a specified amount along a specified axis of the parallel array associated with array handle A. The argument axis indicates the dimension to be shifted, and index prescribes the shift distance.

Shift direction is upward for positive index values and downward for negative index values.

For example, if A denotes a one-dimensional array of length n before the cshift, B denotes the same array after the cshift, and index is equal to 1, the array A will be circularly shifted upward, as follows :

For C Programs:

```
B[1:n-1]=A[0:n-2], B[0]=A[n-1]
```

For Fortran Programs:

```
B(2:n)=A(1:n-1), B(1)=A(n)
```

Syntax

The C and Fortran syntax for S3L_cshift is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_cshift(A, axis, index)
    S3L_array_t        A
    void               axis
    int                index
```

F77/F90 Syntax

```
include ‘s3l/s3l-f.h’
include ‘s3l/s3l_errno-f.h’
subroutine
S3L_cshift(A, axis, index, ier)
    integer*8          A
    integer*4          axis
    integer*4          index
    integer*4          ier
```

Input

S3L_cshift accepts the following arguments as input:

- A – Array handle for the parallel array to be shifted.
- axis – Specifies the axis along which the shift is to take place. This value must assume zero-based axis indexing when S3L_cshift is called from a C or C++ application and one-based indexing when called from an F77 or F90 application.
- index – Specifies the shift distance. If the extent of the axis being shifted is N, legal values for index are –N < index < N.

Output

S3L_cshift uses the following argument for output:

- ier (Fortran only) – When called from a Fortran program, S3L_cshift returns error status in ier.

Error Handling

On success, S3L_cshift returns S3L_SUCCESS.

S3L_cshift performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause the function to terminate and return the associated error codes:

- S3L_ERR_ARG_AXISNUM – Invalid axis value.
- S3L_ERR_INDX_INVALID – index value is out of range.

Examples

/opt/SUNWhpc/examples/s3l/utils/cshift_reduce.c

/opt/SUNWhpc/examples/s3l/utils-f/cshift_reduce.f

Related Functions

S3L_reduce(3)

S3L_reduce_axis(3)

S3L_dct_iv

Description

S3L_dct_iv computes the Discrete Cosine Transform Type IV (DCT) of 1D, 2D, and 3D S3L arrays. The arrays have to be real (S3L_float or S3L_double). Depending on the rank of the input array a, the following array size constraints apply:

- 1D – The array size must be divisible by 4 x p^2, where p is the number of processors.
- 2D – Each of the array lengths must be divisible by 2 x p, where p is the number of processors.

- 3D – The first dimension must be even and must have a length of at least 4. The second and third dimensions must be divisible by 2 x p, where p is the number of processors.

Note – When the input array `a` is 1D, the number of processes must be either an even number or 1.

Syntax

The C and Fortran syntax for `S3L_dct_iv` is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_dct_iv(a, setup, direction)
    S3L_array_t         a
    int                 setup
    int                 direction
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_dct_iv(a, setup, direction, ier)
    integer*8           a
    integer*4           setup
    integer*4           direction
    integer*4           ier
```

Input

`S3L_dct_iv` accepts the following arguments as input:

- `a` – Input array whose DCT is to be computed. Also used for output, as described in the Output section.

- setup – Integer corresponding to a DCT setup value that was returned by a previous call to S3L_dct_iv_setup.
- direction – Integer, which must be one of:

S3L_DCT_FORWARD	compute the forward DCT
S3L_DCT_INVERSE	compute the inverse DCT

Output

S3L_dct_iv uses the following arguments for output:

- a – On exit, a contains the values of the DCT.
- ier (Fortran only) – When called from a Fortran program, S3L_dct_iv returns error status in ier.

Error Handling

On success, S3L_dct_iv returns S3L_SUCCESS.

S3L_dct_iv performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause this function to terminate and return the associated error code:

- S3L_ERR_ARG_SETUP – Invallid setup value.
- S3L_ERR_PARAM_INVALID – The first element of the options vector is not one of: S3L_DCT_FORWARD or S3L_DCT_INVERSE.

Examples

/opt/SUNWhpc/examples/s3l/dct/ex_dct1.c

/opt/SUNWhpc/examples/s3l/dct/ex_dct2.c

/opt/SUNWhpc/examples/s3l/dct-f/ex_dct1.f

/opt/SUNWhpc/examples/s3l/dct-f/ex_dct2.f

/opt/SUNWhpc/examples/s3l/dct-f/ex_dct3.f

Related Functions

S3L_dct_iv_setup(3)

S3L_dct_iv_free_setup(3)

S3L_rc_fft(3)

S3L_dct_iv_free_setup

Description

S3L_dct_iv_free_setup frees all internal data structures that are used for the computation of a parallel Discrete Cosine Transform, Type IV (DCT).

Syntax

The C and Fortran syntax for S3L_dct_iv_free_setup is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_dct_iv_free_setup(setup)
    int                 *setup
```

F77/F90 Syntax

```
include ‘s3l/s3l-f.h’
include ‘s3l/s3l_errno-f.h’
subroutine
S3L_dct_iv_free_setup(setup, ier)
    integer*4           setup
    integer*4           ier
```

Input

`S3L_dct_iv_free_setup` accepts the following argument as input:

- `setup` – Integer corresponding to a DCT setup.

Output

`S3L_dct_iv_free_setup` uses the following argument for output:

- `ier` (Fortran only) – When called from a Fortran program, `S3L_dct_iv_free_setup` returns error status in `ier`.

Error Handling

On success, `S3L_dct_iv_free_setup` returns `S3L_SUCCESS`.

On error, `S3L_dct_iv_free_setup` returns the following error code:

- `S3L_ERR_ARG_SETUP` – Invalid `setup` value.

Examples

/opt/SUNWhpc/examples/s3l/dct/ex_dct1.c

/opt/SUNWhpc/examples/s3l/dct/ex_dct2.c

/opt/SUNWhpc/examples/s3l/dct-f/ex_dct1.f

/opt/SUNWhpc/examples/s3l/dct-f/ex_dct2.f

/opt/SUNWhpc/examples/s3l/dct-f/ex_dct3.f

Related Functions

S3L_dct_iv(3)

S3L_dct_iv_setup(3)

S3L_rc_fft(3)

S3L_dct_iv_setup

Description

S3L_dct_iv_setup initializes internal data structures required for the computation of a parallel Discrete Cosine Transform, Type IV (DCT).

Note

If DCT transforms will be performed on multiple arrays that all have the same data type and extents, only one call to S3L_dct_iv_setup would be needed to support those multiple DCT transformations. In other words, the setup performed by a single call to S3L_dct_iv_setup could be referenced by any number of subsequent calls to S3L_dct_iv, so long as their arrays all matched the data type and extents of the array prescribed for the setup.

Syntax

The C and Fortran syntax for S3L_dct_iv_setup is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_dct_iv_setup(a, setup)
    S3L_array_t         a
    int                 *setup
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_dct_iv_setup(a, setup, ier)
    integer*8           a
    integer*4           setup
    integer*4           ier
```

Input

S3L_dct_iv_setup accepts the following argument as input:

- a – Input array whose DCT is to be computed. The data contained in the array are not modified.

Output

S3L_dct_iv_setup uses the following arguments for output:

- setup – Integer corresponding to a DCT setup. This parameter can be used in any subsequent call(s) to S3L_dct_iv to perform the DCT of an array whose data type and extents are the same as those of array a.
- ier (Fortran only) – When called from a Fortran program, S3L_dct_iv_setup returns error status in ier.

Error Handling

On success, S3L_dct_iv_setup returns S3L_SUCCESS.

S3L_dct_iv_setup performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause this function to terminate and return the associated error code:

- S3L_ERR_ARG_RANK – The rank of a is not 1, 2, or 3.
- S3L_ERR_NREAL – The data type of a is not real.
- S3L_ERR_NEVEN – Some of the extents of a are not even or 1.

- S3L_ERR_ARG_EXTENTS – The extents of a are not valid for the rank of a and the number of processes over which a is distributed. The following summarizes the rules for extents when a is 1D, 2D, or 3D:

1D	Its length must be divisible by 4*sqr(np), where np is the number of processes over which a is distributed.
2D	Its extents must both be divisible by 2*np.
3D	Its first extent must be even and its last two extents must both be divisible by 2*np.

- S3L_ERR_NP_NEVEN – The rank of a is 1 but the total number of processes is not even or equal to 1.

Examples

/opt/SUNWhpc/examples/s3l/dct/ex_dct1.c

/opt/SUNWhpc/examples/s3l/dct/ex_dct2.c

/opt/SUNWhpc/examples/s3l/dct-f/ex_dct1.f

/opt/SUNWhpc/examples/s3l/dct-f/ex_dct2.f

/opt/SUNWhpc/examples/s3l/dct-f/ex_dct3.f

Related Functions

S3L_dct_iv(3)

S3L_dct_iv_free_setup(3)

S3L_rc_fft(3)

S3L_declare

Description

S3L_declare creates an S3L array handle that describes an S3L parallel array. It supports calling arguments that enable the user to specify:

- The array's rank (number of dimensions)

- The extent of each axis
- The array's data type
- Which axes, if any, will be distributed locally
- How memory will be allocated for the array

Based on the argument-supplied specifications, a process grid size is internally determined to distribute the array as evenly as possible.

Note – An array subgrid is the set of array elements that is allocated to a particular process.

The `axis_is_local` argument specifies which array axes (if any) will be local to the process. It consists of an integer vector whose length is at least equal to the rank (number of dimensions) of the array. Each element of the vector indicates whether the corresponding axis is local or not: `1` = local, `0` = not local.

When `axis_is_local` is ignored, all array axes except the last will be local. The last axis will be block-distributed.

For greater control over array distribution, use `S3L_declare_detailed()`.

Upon successful completion, `S3L_declare` returns an S3L array handle, which subsequent S3L calls can use as an argument to gain access to that array.

Syntax

The C and Fortran syntax for `S3L_declare` is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_declare(A, rank, extents, type, axis_is_local, atype)
    S3L_array_t        *A
    int                rank
    int                *extents
    S3L_data_type      type
    S3L_boolean_t      *axis_is_local
    S3L_alloc_type     atype
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_declare(A, rank, extents, type, axis_is_local, atype, ier)
    integer*8          A
    integer*4          rank
    integer*4          extents(*)
    integer*4          type
    integer*4          axis_is_local(*)
    integer*4          atype
    integer*4          ier
```

Input

`S3L_declare` accepts the following argument as input:

- `rank` – Specifies the number of dimensions the array will have. The range of legal values for `rank` is 1 <= `rank` <= 31.
- `extents` – An integer vector whose length is equal to the number of dimensions in the array. Each element in `extents` specifies the extent of the corresponding array axis. Note that axis indexing is zero-based for the C interface and one-based for the Fortran interface, as follows:
  - When called from a C or C++ application, the first element of `extents` corresponds to axis 0, the second element to axis 1, and so forth.
  - When called from an F77 or F90 application, the first element corresponds to axis 1, the second to axis 2, and so forth.
- `type` – Specifies the array's data type; this must be a type supported by Sun S3L. See the *Sun S3L Programming Guide* for a complete list of supported data types.
- `axis_is_local` – An integer vector whose length equals the array's rank. Each element of `axis_is_local` controls the distribution of the corresponding array axis as follows:
  - If `axis_is_local[i]`= 0, axis[`i`] of the array will be block-distributed along axis [`i`] of the process grid.
  - If `axis_is_local[i]`= 1, axis[`i`] will not be distributed.

    If `axis_is_local` is `NULL` (C/C++) or if its first integer value is negative (F77/F90), this argument will be ignored.
- `atype` – Use one of the following predefined values to specify how the array will be allocated:
  - `S3L_USE_MALLOC` – Uses `malloc()` to allocate the array subgrids.

- S3L_USE_MEMALIGN64 – Uses memalign() to allocate the array subgrids and to align them on 64-byte boundaries.
- S3L_USE_MMAP – Uses mmap() to allocate the array subgrids. Array subgrids on the same node will be in shared memory.
- S3L_USE_SHMGET – Uses shmget() to allocate the array subgrids. Array subgrids on the same node will be in intimate shared memory.

Output

S3L_declare uses the following arguments for output:

- A – S3L_declare returns the array handle in A.
- ier (Fortran only) – When called from a Fortran program, S3L_declare returns error status in ier.

Error Handling

On successful completion, S3L_declare returns S3L_SUCCESS.

S3L_declare applies various checks to the arrays it accepts as arguments. If an array argument fails any of these checks, the function returns an error code indicating the kind of error that was detected and terminates. See Appendix A of this manual for a list of these error codes.

In addition, the following conditions will cause S3L_declare to terminate and return the associated error code:

- S3L_ERR_ARG_RANK – The rank specified is invalid. The range of legal values for rank is 1 <= rank <= 31.
- S3L_ERR_ARG_EXTENTS – One or more of the array extents is less than 1.
- S3L_ERR_ARG_BLKSIZE – One or more block sizes is less than 1.
- S3L_ERR_ARG_DISTTYPE – axis_is_local has one or more invalid values. See the description of axis_is_local in the Input section for details.

Notes

When S3L_USE_MMAP or S3L_USE_SHMGET is used on a 32-bit platform, the part of an S3L array owned by a single SMP cannot exceed 2 gigabytes.

When S3L_USE_MALLOC or S3L_USE_MEMALIGN64 is used, the part of the array owned by any single process cannot exceed 2 gigabytes.

If these size restrictions are violated, an S3L_ERR_MEMALLOC will be returned. On 64-bit platforms, the upper bound is equal to the system's maximum available memory.

Examples

/opt/SUNWhpc/examples/s3l/transpose/ex_trans1.c

/opt/SUNWhpc/examples/s3l/grade-f/ex_grade.f

Related Functions

S3L_declare_detailed(3)

S3L_free(3)

S3L_declare_detailed

Description

S3L_declare_detailed offers the same functionality as S3L_declare, but supports the additional input argument, addr_a, which gives the user additional control over array distribution.

If you do not need the level of control provided by S3L_declare_detailed, S3L_declare offers essentially the same functionality, but has a simpler interface.

Syntax

The C and Fortran syntax for S3L_declare_detailed is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_declare_detailed(A, addr_a, rank, extents, type, blocksizes,
proc_src, axis_is_local, pgrid, atype)
    S3L_array_t         *A
    void                *addr_a
    int                 rank
    int                 *extents
    S3L_data_type       type
    int                 *blocksizes
    int                 *proc_src
    S3L_boolean_t       *axis_is_local
    S3L_pgrid_t         pgrid
    S3L_alloc_type      atype
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_declare_detailed(A, addr_a, rank, extents, type, blocksizes,
proc_src, axis_is_local, pgrid, atype, ier)
    integer*8           A
    <type>              array(1)
    pointer             (addr_a,array)
    integer*4           rank
    integer*4           extents(*)
    integer*4           blocksizes(*)
    integer*4           proc_src(*)
    integer*4           axis_is_local(*)
    integer*8           pgrid
    integer*4           atype
    integer*4           ier
```

where `<type>` is one of: `integer*4`, `integer*8`, `real*4`, `real*8`, `complex*8`, or `complex*16`.

Input

`S3L_declare_detailed` accepts the following arguments as input:

- addr_a – If the atype argument is set to S3L_DONOT_ALLOCATE, addr_a is taken as the starting address of the local (per process) portion of the parallel array A. If atype is not equal to S3L_DONOT_ALLOCATE, addr_a will be ignored.
- rank – Specifies the number of dimensions the array will have. The range of legal values for rank is 1 <= rank <= 31.
- extents – An integer vector whose length is at least equal to the array rank. Each element in extents specifies the extent of the corresponding array axis. Note that axis indexing is zero-based for the C interface and one-based for the Fortran interface, as follows:
  - When called from a C or C++ application, the first element of extents corresponds to axis 0, the second element to axis 1, and so forth.
  - When called from an F77 or F90 application, the first vector element corresponds to axis 1, the second to axis 2, and so forth.
- type – Specifies the array's data type; this must be a type supported by Sun S3L.
- blocksizes – Specifies the blocksize to be used in block-cyclic distribution along each axis. If blocksizes is a NULL pointer (C/C++) or if its first element is negative (F77/F90), default blocksizes will be chosen by the system.
- proc_src – Vector of length at least equal to the rank. The indices of the processes contain the start of the array—that is, the first element along the particular axis. If this argument is a NULL pointer (C/C++) or if its first element is negative (F77/F90), default values will be used. Along each axis, the process whose process grid coordinate along that axis is equal to zero contains the first array element. If proc_src is present, the pgrid (process grid) argument should also be present. Otherwise an error code will be returned.
- axis_is_local – An integer vector whose length is at least equal to the array rank. Each element of axis_is_local controls the distribution of the corresponding array axis as follows:
  - If axis_is_local[i]= 0 and blocksizes is not specified, axis[i] of the array will be block-distributed along axis [i] of the process grid.
  - If axis_is_local[i]= 1, the corresponding array axis will not be distributed.

  The axis_is_local argument is used only if a pgrid is not specified. If it is NULL (C/C++) or if its first integer value is negative (F77/F90), axis_is_local will be ignored.
- pgrid – An S3L process grid handle that was obtained by calling either S3L_set_process_grid or S3L_get_attribute. If this argument is NULL (C/C++) or is equal to zero (F77/F90), S3L will choose an appropriate pgrid for the array.
- atype – This argument specifies how the array will be allocated, as follows:
  - S3L_USE_MALLOC – Uses malloc() to allocate the array subgrids.

- S3L_USE_MEMALIGN64 – Uses memalign() to allocate the array subgrids and to align them on 64-byte boundaries.
- S3L_USE_MMAP – Uses mmap() to allocate the array subgrids. Array subgrids on the same SMP will be in shared memory.
- S3L_USE_SHMGET – Uses shmget() to allocate the array subgrids. Array subgrids on the same SMP will be in shared memory.
- S3L_DONOT_ALLOCATE – No memory is allocated for the parallel array, and addr_a is taken to be the starting address of the per-process portion of the parallel array.

Note – A process grid is the array of processes onto which the data is distributed.

Output

S3L_declare_detailed uses the following arguments for output:

- A – On return, A points to an S3L array handle for the declared array. This handle can be used later when calling other S3L functions that will use this array.
- ier (Fortran only) – When called from a Fortran program, S3L_declare_detailed returns error status in ier.

Error Handling

On successful completion, S3L_declare_detailed returns S3L_SUCCESS.

S3L_declare_detailed applies various checks to the arrays it accepts as arguments. If an array argument fails any of these checks, the function returns an error code indicating the kind of error that was detected and terminates. See Appendix A of this manual for a list of these error codes.

In addition, the following conditions will cause S3L_declare_detailed to terminate and return the associated error codes:

- S3L_ERR_ARG_RANK – The rank specified is invalid. The range of legal values for rank is 1 <= rank <= 31.
- S3L_ERR_ARG_EXTENTS – One or more of the array extents are less than 1.
- S3L_ERR_ARG_BLKSIZE – One or more blocksizes are less than 1.
- S3L_ERR_ARG_DISTTYPE – axis_is_local has one or more invalid values.
- S3L_ERR_ARG_ALLOCTYPE – atype has an invalid value.

Notes

When S3L_USE_MMAP or S3L_USE_SHMGET is used on a 32-bit platform, the part of an S3L array owned by a single SMP cannot exceed 2 gigabytes.

When S3L_USE_MALLOC or S3L_USE_MEMALIGN64 is used, the part of the array owned by any single process cannot exceed 2 gigabytes.

An S3L_ERR_MEMALLOC will be returned if these size restrictions are violated. On 64-bit platforms, the upper bound is equal to the system's maximum available memory.

Examples

/opt/SUNWhpc/examples/s3l/utils/copy_array.c

/opt/SUNWhpc/examples/s3l/utils-f/copy_array.f

/opt/SUNWhpc/examples/s3l/utils/get_attribute.c

/opt/SUNWhpc/examples/s3l/utils-f/get_attribute.f

/opt/SUNWhpc/examples/s3l/utils/scalapack_conv.c

/opt/SUNWhpc/examples/s3l/utils-f/scalapack_conv.f

Related Functions

S3L_declare(3)

S3L_free(3)

S3L_set_process_grid(3)

S3L_get_attribute(3)

S3L_declare_sparse

Description

S3L_declare_sparse creates an internal S3L array handle that describes a sparse matrix. The sparse matrix A may be represented in one of three sparse formats: the Coordinate format, the Compressed Sparse Row format, or the Compressed Sparse Column format. Upon successful completion, S3L_declare_sparse returns an S3L array handle in A that describes the distributed sparse matrix.

The Coordinate format consists of the following three arrays:

- indx – Integer array that contains the row indices of the matrix A. indx receives its contents from the argument row.
- jndx – Integer array that contains the column indices of the matrix A. jndx receives its contents from the argument col.
- val – Floating-point array that stores the nonzero elements of sparse matrix A in any order. val receives its contents from the argument val.

The Compressed Sparse Row format stores the sparse matrix A in the following three arrays:

- ptr – Integer array that contains pointers to the beginning of each row in indx and val. ptr receives its contents from the argument row.
- indx – Integer array that contains the column indices of the nonzero elements in val. indx receives its contents from the argument col.
- val – Floating-point array that stores the nonzero elements of the sparse matrix A. val receives its contents from the argument val.

The Compressed Sparse Column format also stores the sparse matrix A in three arrays, but the pointer and index references swap axes. In other words, the Compressed Sparse Column format can be viewed as the Compressed Sparse Row format for the transpose of matrix A. In the Compressed Sparse Column format, the three internal arrays are:

- ptr – Integer array that contains pointers to the beginning of each column in indx and val. ptr receives its contents from the argument row.
- indx – Integer array that contains the row indices of the nonzero elements in val. indx receives its contents from the argument col.
- val – Floating-point array that stores the nonzero elements of sparse matrix A. val receives its contents from the argument val.

Note – `S3L_declare_sparse` follows different indexing conventions, depending on which language the calling program is written in, Fortran or C. Its Fortran interface uses a one-based convention to index elements of the matrix, while the C interface assumes that the elements are indexed with zero-based values. The zero-based convention is employed in the examples that follow.

To illustrate these three sparse formats, consider the following 4x6 sparse matrix:

```
 3.14      0          0         20.04      0          0
 0        27          0          0        -0.6        0
 0         0         -0.01       0         0          0
-0.031     0          0          0.08      0        314.0
```

Representations of this sample 4x6 matrix are as follows in each of the supported formats.

In `S3L_SPARSE_COO`:

```
indx = (    3,    1,    0,    3,     2,    0,     1,    3 ),
jndx = (    5,    1,    3,    3,     2,    0,     4,    0 ),
val  = ( 314.0, 27.0, 20.04, 0.08, -0.01, 3.14, -0.6, -0.031 )
```

In `S3L_SPARSE_CSR`:

```
ptr  = ( 0, 2, 4, 5, 8 ),
indx = (  0,     3,     1,    4,    2,     0,     3,      5 ),
val  = ( 3.14, 20.04, 27.0, -0.6, -0.01, -0.031, 0.08, 314.0 )
```

In `S3L_SPARSE_CSC`:

```
ptr  = ( 0, 2, 3, 4, 6, 7, 8 ),
indx = (  0,     3,      1,    2,     0,    3,     1,     3 ),
val  = ( 3.14, -0.031, 27.0, -0.01, 20.04, 0.08, -0.6, 314.0 )
```

Syntax

The C and Fortran syntax for `S3L_declare_sparse` is as follows.

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_declare_sparse(A, spfmt, m, n, row, col, val)
    S3L_array_t             *A
    S3L_sparse_storage_t    spfmt
    int                     m
    int                     n
    S3L_array_t             row
    S3L_array_t             col
    S3L_array_t             val
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_declare_sparse(A, spfmt, m, n, row, col, val, ier)
    integer*8               A
    integer*8               spfmt
    integer*4               m
    integer*4               n
    integer*8               row
    integer*8               col
    integer*8               val
    integer*4               ier
```

Input

S3L_declare_sparse accepts the following arguments as input:

- spfmt – Indicates the sparse format used for representing the sparse matrix. Use S3L_SPARSE_COO to specify the Coordinate format, S3L_SPARSE_CSR for the Compressed Sparse Row format, and S3L_SPARSE_CSC for the Compressed Sparse Column format.
- m – Indicates the total number of rows in the sparse matrix.
- n – Indicates the total number of columns in the sparse matrix.
- row – Integer parallel array of rank 1. Its length and content can vary, depending on which sparse format is used.
  - S3L_SPARSE_COO – row is of the same size as arrays col and val and contains row indices of the nonzero elements in array val.

- S3L_SPARSE_CSR – row is of size m+1 and contains pointers to the beginning of each row in arrays col and val.
- S3L_SPARSE_CSC – row is of size n+1 and contains pointers to the beginning of each column in arrays col and val.

- col – Integer parallel array of rank 1 with the same length as array val. For both S3L_SPARSE_COO and S3L_SPARSE_CSR, col contains column indices of the corresponding elements stored in array val. For S3L_SPARSE_CSC, it contains row indices of the corresponding elements in S3L array val.
- val – Parallel array of rank 1, containing the nonzero elements of the sparse matrix. For S3L_SPARSE_COO, nonzero elements can be stored in any order. For S3L_SPARSE_CSR, nonzero elements should be stored row by row, from row 1 to m. For S3L_SPARSE_CSC, nonzero elements should be stored column by column, from column 1 to n.

  The length of val is nnz for all three formats, which is the total number of nonzero elements in the sparse matrix. The data type of array elements can be real or complex (single- or double-precision).

Note – Because row, col, and val are copied to working arrays, they can be deallocated immediately following the S3L_declare_sparse call.

Output

S3L_declare_sparse uses the following arguments for output:

- A – Upon return, A contains an S3L internal array handle for the global general sparse matrix. This handle can be used in subsequent calls to other S3L sparse array functions.
- ier (Fortran only) – When called from a Fortran program, this function returns error status in ier.

Error Handling

On success, S3L_declare_sparse returns S3L_SUCCESS.

The S3L_declare_sparse routine performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause these functions to terminate and return the associated error code:

- S3L_ERR_SPARSE_FORMAT – Invalid storage format. It must be one of: S3L_SPARSE_COO, S3L_SPARSE_CSR, or S3L_SPARSE_CSR.
- S3L_ERR_ARG_EXTENTS – Invalid m or n. Each must be > 0.
- S3L_ERR_ARG_NULL – One or more of the arguments row, col, or val are invalid. All must be preallocated S3L arrays.
- S3L_ERR_MATCH_RANK – Ranks of arrays row, col, and val are mismatched. They all must be rank 1 arrays.
- S3L_ERR_MATCH_DTYPE – Arrays row and col data types do not match. They must be of type S3L_integer.
- S3L_ERR_MATCH_EXTENTS – The lengths of arrays row, col, and val are mismatched. Array extents must match as follows:
  - For S3L_SPARSE_COO, they must all be the same size.
  - For both S3L_SPARSE_CSR and S3L_SPARSE_CSC, the length of array col must equal that of array val.
  - For S3L_SPARSE_CSR, array row must be of size m+1.
  - For S3L_SPARSE_CSC, array row must be of size n+1.

Example

/opt/SUNWhpc/examples/s3l/sparse/ex_sparse2.c

/opt/SUNWhpc/examples/s3l/sparse-f/ex_sparse2.f

Related Functions

S3L_convert_sparse(3)

S3L_matvec_sparse(3)

S3L_rand_sparse(3)

S3L_read_sparse(3)

S3L_deconv

Description

If a can be expressed as the convolution of an unknown vector c with b, S3L_deconv deconvolves the vector b out of a. The result, which is returned in c, is such that conv(c,b)=a.

In the general case, c will only represent the quotient of the polynomial division of a by b.

The remainder of that division can be obtained by explicitly convolving with b and subtracting the result from a.

If ma, mb, and mc are the lengths of a, b, and c, respectively, ma must be at least equal to mb. The length of mc will be such that mc +mb-1=ma or, equivalently, mc= ma –mb+1.

Note – S3L_deconv is most efficient when all arrays have the same length and when this length is such that it can be computed efficiently by S3L_fft or S3L_rc_fft. See “S3L_fft” on page 101 and “S3L_rc_fft and S3L_cr_fft” on page 253 for additional information.

Restriction

The dimensions of the array c must be such that the 1D or 2D complex-to-complex FFT or real-to-complex FFT can be computed.

Scaling

The results of the deconvolution are scaled according to the underlying FFT that is used. In particular, for multiple processes, if a and b are real 1D, the result is scaled by n/2, where n is the length of c. For single processes, it is scaled by n. In all other cases, the result is scaled by the product of the extents of c.

Side Effect

Because a and b are used internally for auxiliary storage, they may be destroyed after the deconvolution calculation is complete. If a and b must be used after the deconvolution, they should first be copied to temporary arrays.

Syntax

The C and Fortran syntax for S3L_deconv is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_deconv(a, b, c, setup_id)
    S3L_array_t       a
    S3L_array_t       b
    S3L_array_t       c
    int               *setup_id
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_deconv(a, b, c, setup_id, ier)
    integer*8         a
    integer*8         b
    integer*8         c
    integer*4         setup_id
    integer*4         ier
```

Input

S3L_deconv accepts the following arguments as input:

- a – S3L array handle describing a parallel array that contains the convolution of an unknown vector c with b. Its length must be at least ma+mb-1 (1D case) or ma+mb-1 x na+nb-1 (2D case).

- b – S3L array handle describing the parallel array that contains the vector.
- setup_id – Valid convolution setup ID as returned from a previous call to S3L_deconv_setup.

Output

S3L_deconv uses the following arguments for output:

- c – S3L array handle describing a parallel array. Its length must be at least ma+mb-1 (1D case) or ma+mb-1 x na+nb-1 (2D case). Upon successful completion, the results of deconvolving a will be stored in c.
- ier (Fortran only) – When called from a Fortran program, this function returns error status in ier.

Error Handling

On success, S3L_deconv returns S3L_SUCCESS.

S3L_deconv performs generic checking of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions cause the function to terminate and return one of the following error codes:

- S3L_ERR_MATCH_DTYPE – a, b, and c do not have the same data type.
- S3L_ERR_MATCH_RANK – a, b, and c do not have the same rank.
- S3L_ERR_ARG_RANK – The rank of an array argument is larger than 2.
- S3L_ERR_ARG_DTYPE – The data type of one of the array arguments is invalid. It must be one of:
  - S3L_float
  - S3L_double
  - S3L_complex
  - S3L_double_complex
- S3L_ERR_ARG_EXTENTS – The extents of c are smaller than two times the sum of the corresponding extents of a and b minus 1.

In addition, since S3L_fft or S3L_rc_fft is used internally to compute the deconvolution, if the dimensions of c are not appropriate for using S3L_fft or S3L_rc_fft, an error code indicating the unsuitability is returned. See “S3L_fft” on page 101 and “S3L_rc_fft and S3L_cr_fft” on page 253 for more details.

Examples

/opt/SUNWhpc/examples/s3l/deconv/ex_deconv.c

/opt/SUNWhpc/examples/s3l/deconv-f/ex_deconv.f

Related Functions

S3L_deconv_setup(3)

S3L_deconv_free_setup(3)

S3L_deconv_free_setup

Description

S3L_deconv_free_setup invalidates the ID specified by the setup_id argument. This deallocates internal memory that was reserved for the deconvolution computation represented by that ID.

Syntax

The C and Fortran syntax for S3L_deconv_free_setup is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_deconv_free_setup(setup_id)
    int               setup_id
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_deconv_free_setup(setup_id, ier)
    integer*4         setup_id
    integer*4         ier
```

Input

S3L_deconv_free_setup accepts the following arguments as input:

- setup_id – Integer value returned by a previous call to S3L_deconv_setup.

Output

S3L_deconv_free_setup uses the following argument for output:

- ier (Fortran only) – When called from a Fortran program, this function returns error status in ier.

Error Handling

On success, S3L_deconv_free_setup returns S3L_SUCCESS.

In addition, the following condition causes the function to terminate and return the associated code:

- S3L_ERR_ARG_SETUP – Invalid setup_id value.

Examples

/opt/SUNWhpc/examples/s3l/deconv/ex_deconv.c

/opt/SUNWhpc/examples/s3l/deconv-f/ex_deconv.f

Related Functions

S3L_deconv(3)

S3L_deconv_setup(3)

S3L_deconv_setup

Description

S3L_deconv_setup sets up the initial conditions required for computing the deconvolution of A with B. It returns an integer setup value that can be used by subsequent calls to S3L_deconv or S3L_deconv_free_setup.

Syntax

The C and Fortran syntax for S3L_deconv_setup is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_deconv_setup(A, B, C, setup_id)
    S3L_array_t       A
    S3L_array_t       B
    S3L_array_t       C
    int               *setup_id
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_deconv_setup(A, B, C, setup_id, ier)
    integer*8        A
    integer*8        B
    integer*8        C
    integer*4        setup_id
    integer*4        ier
```

Input

S3L_deconv_setup accepts the following arguments as input:

- A – S3L internal array handle for the parallel array that contains the input signal to be deconvolved.
- B – S3L internal array handle for the parallel array that contains the vector.
- C – S3L internal array handle for the parallel array that will store the deconvolved signal.

Output

S3L_deconv_setup uses the following arguments for output:

- setup_id – Integer value returned by this function. Use this value for the setup_id argument in subsequent calls to S3L_deconv and S3L_deconv_free_setup.
- ier (Fortran only) – When called from a Fortran program, this function returns error status in ier.

Error Handling

On success, S3L_deconv_setup returns S3L_SUCCESS.

S3L_deconv_setup performs generic checking of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions cause the function to terminate and return one of the following error codes:

- S3L_ERR_ARG_RANK – The rank of one of the array arguments is not 1 or 2.
- S3L_ERR_MATCH_RANK – The array arguments are not all of the same rank.
- S3L_ERR_MATCH_DTYPE – The array arguments are not all of the same data type.
- S3L_ERR_ARG_EXTENTS – The extents of C are less than the corresponding extents ext(A) - ext(B) + 1, or the extents of A are less than the corresponding extents of B.

Examples

/opt/SUNWhpc/examples/s3l/deconv/ex_deconv.c

/opt/SUNWhpc/examples/s3l/deconv-f/ex_deconv.f

Related Functions

S3L_deconv(3)

S3L_deconv_free_setup(3)

S3L_describe

Description

S3L_describe prints information about a parallel array or a process grid to standard output. If an array handle is supplied in argument A, the parallel array is described. If a process grid is supplied in A, the associated process grid is described. The info_node argument specifies the MPI rank of the process on which the subgrid of interest is located.

If A is an S3L array handle, the following are provided:

- Information on the rank extents and the data type of the array, as well as the starting address in memory of its subgrid.
- A description of the underlying grid of processes to which data is mapped.

If the entire array fits on the process specified by info_node, all parts of the S3L_describe output apply to the full array. Otherwise, some parts of the output, such as subgrid size, will apply only to the portion of the array that is on process info_node.

If A is a process grid handle, S3L_describe provides only a description of the underlying grid of processes to which data is mapped.

To determine what value to enter for info_node, run MPI_Comm_rank on the process of interest.

Syntax

The C and Fortran syntax for S3L_describe is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_describe(A, info_node)
    S3L_array_t       A
    int               info_node
```

F77/F90 Syntax

```
include ‘s3l/s3l-f.h’
include ‘s3l/s3l_errno-f.h’
subroutine
S3L_describe(A, info_node, ier)
    integer*8          A
    integer*4          info_node
    integer*4          ier
```

Input

S3L_describe accepts the following arguments as input:

- A – May be a parallel array handle or a process grid handle.

- info_node – Scalar integer variable that specifies the index or rank of the process from which the information will be gathered. Note that certain array parameters, such as the subgrid size and addresses, will vary from process to process.

Output

S3L_describe uses the following argument for output:

- ier (Fortran only) – When called from a Fortran program, S3L_describe returns error status in ier.

Error Handling

On success, S3L_describe returns S3L_SUCCESS.

S3L_describe performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following condition will cause the function to terminate and return the associated error code:

- S3L_ERR_ARG_ARRAY – A is not a valid parallel array or process grid handle.

Examples

/opt/SUNWhpc/examples/s3l/utils/scalapack_conv.c

/opt/SUNWhpc/examples/s3l/utils-f/scalapack_conv.f

Related Functions

MPI_Comm_rank(3)

S3L_declare(3)

S3L_declare_detailed(3)

S3L_set_process_grid(3)

S3L_dst

Description

S3L_dst computes the Discrete Sine Transform (DST) of 1D, 2D, and 3D S3L arrays. The data type of the arrays must be real (S3L_float or S3L_double). Depending on the rank of the input array a, the following array size constraints apply:

- 1D – The array size must be divisible by 4 x p^2, where p is the number of processors.
- 2D – Each of the array lengths must be divisible by 2 x p, where p is the number of processors.
- 3D – The first dimension must be even and must have a length of at least 4. The second and third dimensions must be divisible by 2 x p, where p is the number of processors.

Note – When the input array a is 1D, the number of processes must be either an even number or 1.

Notes

Efficient distribution: The S3L_dst function is more efficient when the arrays are block-distributed along their last dimension. In all other cases, S3L performs an internal redistribution of the arrays, which may result in additional overhead.

Forward/inverse DST: The inverse DST is the same as the forward one.

First element: The DST does not take into account the first element of an input array (the element with index 0 in C or index 1 in F77). This means that, when performing a forward DST followed by an inverse DST, the first element must be zero to ensure perfect reconstruction. Otherwise, only the elements with nonzero index (C) or non-one index (F77) will be reconstructed. This extends to multidimensional DST transforms—elements whose index contains 0 (C) or 1 (F77) along any dimension do not contribute to the DST and are therefore ignored in the reconstruction.

Scaling: When the forward DST of an array is followed by the inverse DST of the array, the original array is scaled by a factor that is determined in the following manner:

1D	reconstructed array is scaled by n/2, where n is the length of the original array
2D	reconstructed array is scaled by (m*n)/4, where m and n are the array extents
3D	reconstructed array is scaled by (m*n*k)/4, where m, n, and k are the array extents

Syntax

The C and Fortran syntax for `S3L_dst` is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_dst(a, setup)
    S3L_array_t            a
    int                    setup
```

F77/F90 Syntax

```
include ‘s3l/s3l-f.h’
include ‘s3l/s3l_errno-f.h’
subroutine
S3L_dst(a, setup, ier)
    integer*8              a
    integer*4              setup
    integer*4              ier
```

Input

`S3L_dst` accepts the following arguments as input:

- `a` – Input array whose DST is to be computed.
- `setup` – Integer corresponding to DST setup as returned by `S3L_dst_setup`.

Output

S3L_dst uses the following argument for output:

- ier (Fortran only) – When called from a Fortran program, S3L_dst returns error status in ier.

Error Handling

On success, S3L_dst returns S3L_SUCCESS.

S3L_dst performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following condition will cause S3L_dst to terminate and return the associated error code:

- S3L_ERR_ARG_SETUP – Invalid setup value.

Examples

/opt/SUNWhpc/examples/s3l/dst/ex_dst1.c

/opt/SUNWhpc/examples/s3l/dst/ex_dst2.c

/opt/SUNWhpc/examples/s3l/dst-f/ex_dst1.f

/opt/SUNWhpc/examples/s3l/dst-f/ex_dst2.f

/opt/SUNWhpc/examples/s3l/dst-f/ex_dst3.f

Related Functions

S3L_dst_setup(3)

S3L_dst_free_setup(3)

S3L_rc_fft(3)

S3L_dst_free_setup

Description

S3L_dst_free_setup frees all internal data structures required for the computation of a parallel Discrete Sine Transform (DST).

Syntax

The C and Fortran syntax for S3L_dst_free_setup is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_dst_free_setup(setup)
    int                     *setup
```

F77/F90 Syntax

```
include ‘s3l/s3l-f.h’
include ‘s3l/s3l_errno-f.h’
subroutine
S3L_dst_free_setup(setup, ier)
    integer*4               setup
    integer*4               ier
```

Input

S3L_dst_free_setup accepts the following argument as input:

- setup – Integer corresponding to a DST setup.

Output

S3L_dst_free_setup uses the following argument for output:

- ier (Fortran only) – When called from a Fortran program, S3L_dst_free_setup returns error status in ier.

Error Handling

On success, S3L_dst_free_setup returns S3L_SUCCESS.

On error, S3L_dst_free_setup returns the following error code:

- S3L_ERR_ARG_SETUP – Invalid setup value.

Examples

/opt/SUNWhpc/examples/s3l/dst/ex_dst1.c

/opt/SUNWhpc/examples/s3l/dst/ex_dst2.c

/opt/SUNWhpc/examples/s3l/dst-f/ex_dst1.f

/opt/SUNWhpc/examples/s3l/dst-f/ex_dst2.f

/opt/SUNWhpc/examples/s3l/dst-f/ex_dst3.f

Related Functions

S3L_dst(3)

S3L_dst_setup(3)

S3L_rc_fft(3)

S3L_dst_setup

Description

S3L_dst_setup initializes internal data structures required for the computation of a parallel Discrete Sine Transform (DST).

Note

If DST transforms will be performed on multiple arrays that all have the same data type and extents, only one call to `S3l_dst_setup` is needed to support those multiple DST transformations. In other words, the setup performed by a single call to `S3l_dst_setup` could be referenced by any number of subsequent calls to `S3L_dst` so long as their arrays all match the data type and extents of the array prescribed for the setup.

Syntax

The C and Fortran syntax for `S3L_dst_setup` is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_dst_setup(a, setup)
    S3L_array_t            a
    int                    *setup
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_dst_setup(a, setup, ier)
    integer*8              a
    integer*4              setup
    integer*4              ier
```

Input

`S3L_dst_setup` accepts the following argument as input:

- `a` – Input array whose DST is to be computed. The data contained in the array are not modified.

Output

S3L_dst_setup uses the following arguments for output:

- setup – Integer corresponding to a DST setup. This parameter can be used in any subsequent calls to S3L_dst to perform the DST of an array whose data type and extents are the same as those of array a.
- ier (Fortran only) – When called from a Fortran program, S3L_dst_setup returns error status in ier.

Error Handling

On success, S3L_dst_setup returns S3L_SUCCESS.

S3L_dst_setup performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause S3L_dst_setup to terminate and return the associated error code:

- S3L_ERR_ARG_RANK – The rank of a is not 1, 2, or 3.
- S3L_ERR_ARG_NREAL – The data type of a is not real.
- S3L_ERR_ARG_NEVEN – Some of the extents of array a are not even.
- S3L_ERR_ARG_EXTENTS – The extents of a are not valid for the rank of a and the number of processes over which a is distributed. The following summarizes the rules for extents when a is 1D, 2D, or 3D:

1D	Its length must be divisible by 4*sqr(np), where np is the number of processes over which a is distributed.
2D	Its extents must both be divisible by 2*np.
3D	Its first extent must be even and its last two extents must both be divisible by 2*np.

- S3L_ERR_NP_NEVEN – The rank of a is 1 but the total number of processes is not even or equal to 1.

Examples

/opt/SUNWhpc/examples/s3l/dst/ex_dst1.c

/opt/SUNWhpc/examples/s3l/dst/ex_dst2.c

```
/opt/SUNWhpc/examples/s3l/dst-f/ex_dst1.f
/opt/SUNWhpc/examples/s3l/dst-f/ex_dst2.f
/opt/SUNWhpc/examples/s3l/dst-f/ex_dst3.f
```

Related Functions

S3L_dst(3)

S3L_dst_free_setup(3)

S3L_rc_fft(3)

S3L_eigen_iter

Description

S3L_eigen_iter is an iterative eigensolver that computes selected eigenpairs of dense or sparse matrices. Users may specify eigenpairs with certain properties, such as largest magnitude. For dense arrays, users can process multiple instances of matrices.

Syntax

The C and Fortran syntax for S3L_eigen_iter is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_eigen_iter(a, nev, ncv, matcode, which, vec, eig, maxitr,
tol, row_axis, col_axis, vec_axis, nev_axis, eig_axis)
    S3L_array_t             a
    int                     nev
    int                     ncv
    S3L_eigen_iter_type     matcode
    char                    *which
    S3L_array_t             vec
    S3L_array_t             eig
    int                     maxitr
    void                    *tol
    int                     row_axis
    int                     col_axis
    int                     vec_axis
    int                     nev_axis
    int                     eig_axis
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_eigen_iter(a, nev, ncv, matcode, which, vec, eig, maxitr,
tol, row_axis, col_axis, vec_axis, nev_axis, eig_axis, ier)
    integer*8               a
    integer*4               nev
    integer*4               ncv
    integer*4               matcode
    character*2             which
    integer*8               vec
    integer*8               eig
    integer*4               maxitr
    <type_tol>              tol
    integer*4               row_axis
    integer*4               col_axis
    integer*4               vec_axis
    integer*4               nev_axis
    integer*4               eig_axis
    integer*4               ier
```

Input

`S3L_eigen_iter` accepts the following arguments as input:

- `a` – A square S3L array; it may be sparse or dense.
- `nev` – Specifies the number of eigenpairs requested.
- `ncv` – Specifies the number of columns in the array `vec`. This indicates the number of Lanczos vectors generated at each update iteration. Increasing `ncv` increases the amount of work done in each iteration and decreases the number of iterations.
- `matcode` – Specifies which solver algorithm to use, as follows:
  - When array `a` is symmetric, `matcode` must be set to `S3L_EIGEN_SYM`.

    If the data type of `a` is `S3L_float` or `S3L_double`, the Lanczos solver will be used. If `a` is either `S3L_complex` or `S3L_double_complex`, the Arnoldi (general) solver will be used. In such cases, the type of `eig` returned will be correspondingly `S3L_complex` or `S3L_double_complex`.

    Note: When a complex Hermitian problem is being solved, the imaginary part of the returned eigenvalues may contain small, nonzero round-off errors. These errors should be ignored unless they are significant when compared with eigenvalues of the largest magnitude computed.
  - When array `a` is asymmetric, `matcode` must be set to `S3L_EIGEN_GEN`, which will force use of the Arnoldi solver.
  - To compute singular value decomposition, set `matcode` to `S3L_EIGEN_SVD`.
- `which` – An array of two characters denoting the Ritz values to be computed. The allowed values of `which` and their uses are described below:
  - When `matcode` is set to `S3L_EIGEN_SYM` or to `S3L_EIGEN_SVD` and the data type of array `a` is either `S3L_float` or `S3L_double`, the following values can be used:

    `LA` – computes the `nev` largest (algebraic) eigenvalues.

    `SA` – computes the `nev` smallest (algebraic) eigenvalues.

    `LM` – computes the `nev` largest (in magnitude) eigenvalues.

    `SM` – computes the `nev` smallest (in magnitude) eigenvalues.

    `BE` – computes `nev` eigenvalues, half from each end of the spectrum. When `nev` is odd, computes one more from the high end than from the low end.
  - When `matcode` is set to `S3L_EIGEN_GEN` or to `S3L_EIGEN_SVD` and the data type of array `a` is either `S3L_complex` or `S3L_double_complex`, the following values can be used:

    `LR` – computes the `nev` eigenvalues with the largest real part.

    `SR` – computes the `nev` eigenvalues with the smallest real part.

    `LI` – computes the `nev` eigenvalues with the largest imaginary part.

    `SI` – computes the `nev` eigenvalues with the smallest imaginary part.

LM – computes the nev largest (in magnitude) eigenvalues.

SM – computes the nev smallest (in magnitude) eigenvalues.

- maxitr – Specifies the maximum number of iterations.
- tol – Specifies the tolerance value to be used in determining when convergence has been reached. Convergence is reached when

```
|| Ax - abs( Ritz(i) x ) || <= tol * abs( Ritz(i) )
```

  where Ritz(i) is the approximation of the i-th eigenvalue. If tol <= 0.0, the machine precision is used.

- row_axis – Specifies the axis of a that counts the rows of the embedded matrix or matrices (in the multiple-instance case). This argument is ignored for sparse matrices.
- col_axis – Specifies the axis of a that counts the columns of the embedded matrix or matrices (in the multiple-instance case). This argument is ignored for sparse matrices.
- vec_axis – Specifies the axis of vec along which the elements of the embedded eigenvectors lie. This argument is ignored for sparse matrices.
- nev_axis – Specifies the axis of vec along which the embedded requested eigenvectors lie. This argument is ignored for sparse matrices.
- eig_axis – Specifies the axis of eig along which the elements of the embedded eigenvalues lie. This argument is ignored for sparse matrices.

Output

S3L_eigen_iter uses the following arguments for output:

- vec – S3L array. On exit, vec contains nev eigenvectors. vec must have the same number of rows as a and at least nev columns.
- eig – S3L array. When matcode is set to S3L_EIGEN_SYM or S3L_EIGEN_SVD, eig contains, on exit, nev eigenvalues. When matcode is set to S3L_EIGEN_SVD, eig contains singular values on exit.
- ier (Fortran only) – When called from a Fortran program, S3L_eigen_iter returns error status in ier.

Error Handling

On success, S3L_eigen_iter returns S3L_SUCCESS.

S3L_eigen_iter performs generic checking of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause the function to terminate and return the associated error code:

- S3L_ERR_MAXITER – The maximum number of iterations was exceeded.
- S3L_ERR_EIGITER_MATCODE – The matcode argument has an invalid value.
- S3L_ERR_EIGITER_WHICH – The which argument has an invalid value.
- S3L_ERR_PARAM_INVALID – This error indicates one or more of the following:

```
- nev and/or ncv have invalid values
- matcode = S3L_EIGEN_SVD and m x n
- matrix a has m < n.
```

Examples

/opt/SUNWhpc/examples/s3l/eigen_iter/ex_gen_sparse_z.c

/opt/SUNWhpc/examples/s3l/eigen_iter/ex_svd_dense_z.c

/opt/SUNWhpc/examples/s3l/eigen_iter/ex_sym_sparse_f.c

/opt/SUNWhpc/examples/s3l/eigen_iter-f/ex_complex.f

/opt/SUNWhpc/examples/s3l/eigen_iter-f/ex_gen.f

/opt/SUNWhpc/examples/s3l/eigen_iter-f/ex_svd_sparse.f

/opt/SUNWhpc/examples/s3l/eigen_iter-f/ex_sym.f

S3L_exit

Description

When an application is finished using Sun S3L functions, it must call S3L_exit to perform various cleanup tasks associated with the current S3L environment.

S3L_exit checks to see if the S3L environment is in the initialized state, that is, to see if S3L_init has been called more recently than S3L_exit. If not, S3L_exit returns the error message S3L_ERR_NOT_INIT and exits.

Syntax

The C and Fortran syntax for S3L_exit is as follows.

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_exit()
```

F77/F90 Syntax

```
include ‘s3l/s3l-f.h’
include ‘s3l/s3l_errno-f.h’
subroutine
S3L_exit(ier)
    integer*4          ier
```

Input

S3L_exit takes no input arguments.

Output

When called from a Fortran program, S3L_exit returns error status in ier.

Error Handling

On successful completion, S3L_exit returns S3L_SUCCESS.

The following condition will cause S3L_exit to terminate and return the associated error code:

- S3L_ERR_NOT_INIT – S3L has not been initialized.

Examples

/opt/SUNWhpc/examples/s3l/dense_matrix_ops/inner_prod.c

/opt/SUNWhpc/examples/s3l/dense_matrix_ops-f/inner_prod.f

/opt/SUNWhpc/examples/s3l/utils/copy_array.f

Related Function

S3L_init(3)

S3L_fft

Description

S3L_fft performs a simple Fast Fourier Transform (FFT) on the complex parallel array a. The same FFT operation is performed along all axes of the array.

Both power-of-two and arbitrary radix FFTs are supported. The 1D parallel FFT can be used for sizes that are a multiple of the square of the number of processes. The 2D and 3D FFTs can be used for arbitrary sizes and distributions.

The S3L_fft routine computes a multidimensional transform by performing a one-dimensional transform along each axis in turn.

The sign of the twiddle factor exponents determines the direction of an FFT. Twiddle factors with a negative exponent imply a forward transform, and twiddle factors with positive exponents are used for an inverse transform.

For the 2D FFT, a more efficient transpose algorithm will be used if the block sizes along each dimension are equal to the extents divided by the number of processes, resulting in significant performance improvements.

S3L_fft (and S3L_ifft) can only be used for complex and double-complex data types. To compute a real-data forward FFT, use S3L_rc_fft. This performs a forward FFT on the real data, yielding packed representation of the complex results. To compute the corresponding inverse FFT, use S3L_cr_fft, which will perform an inverse FFT on the complex data, overwriting the original real array with real-valued results of the inverse FFT.

The floating-point precision of the result always matches that of the input.

Note – S3L_fft_detailed, S3L_fft_detailed, and S3L_ifft do not perform any scaling. Consequently, when a forward FFT is followed by an inverse FFT, the original data will be scaled by the product of the extents of the array.

Syntax

The C and Fortran syntax for S3L_fft is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_fft(a, setup_id)
    S3L_array_t        a
    int                setup_id
```

F77/F90 Syntax

```
include ‘s3l/s3l-f.h’
include ‘s3l/s3l_errno-f.h’
subroutine
S3L_fft(a, setup_id, ier)
    integer*8          a
    integer*4          setup_id
    integer*4          ier
```

Input

S3L_fft accepts the following arguments as input:

- a – Parallel array that is to be transformed. Its rank, extents, and type must be the same as the parallel array a supplied in the S3L_fft_setup call.
- setup_id – Scalar integer variable. Use the value returned by the S3L_fft_setup call for this argument.

Output

S3L_fft uses the following arguments for output:

- a – The input array a is overwritten with the result of the FFT.
- ier (Fortran only) – When called from a Fortran program, S3L_fft returns error status in ier.

Error Handling

On success, S3L_fft returns S3L_SUCCESS.

S3L_fft performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

The following conditions will cause the function to terminate and return the associated error code:

- S3L_ERR_FFT_RANKGT3 – The rank of the array a is larger than 3.
- S3L_ERR_ARG_NCOMPLEX – Array a is not of type S3L_complex or S3L_double_complex.
- S3L_ERR_FFT_EXTSQPROCS – Array a is 1D, but its extent is not divisible by the square of the number of processes.
- S3L_ERR_ARG_SETUP – Invalid setup_id value.

Examples

/opt/SUNWhpc/examples/s3l/fft/fft.c

/opt/SUNWhpc/examples/s3l/fft/ex_fft1.c

/opt/SUNWhpc/examples/s3l/fft/ex_fft2.c

/opt/SUNWhpc/examples/s3l/fft-f/fft.f

Related Functions

S3L_fft_setup(3)

S3L_fft_free_setup(3)

S3L_ifft(3)

S3L_fft_detailed(3)

S3L_cr_fft(3)

S3L_rc_fft(3)

S3L_rc_fft_setup(3)

S3L_fft_detailed

Description

S3L_fft_detailed computes the in-place forward or inverse FFT along a specified axis of a complex or double-complex parallel array, a. FFT direction and axis are specified by the arguments iflag and axis, respectively. Both power-of-two and arbitrary radix FFTs are supported. Upon completion, a is overwritten with the FFT result.

A 1D parallel FFT can be used for array sizes that are a multiple of the square of the number of processes. Higher-dimensionality FFTs can be used for arbitrary sizes and distributions.

For the 2D FFT, a more efficient transpose algorithm is employed when the blocksizes along each dimension are equal to the extents divided by the number of processes. This yields significant performance benefits.

S3L_fft_detailed can only be used for complex and double-complex data types. To compute a real-data forward FFT, use S3L_rc_fft. This performs a forward FFT on the real data, yielding packed representation of the complex results. To compute the corresponding inverse FFT, use S3L_cr_fft, which will perform an inverse FFT on the complex data, overwriting the original real array with real-valued results of the inverse FFT.

The floating-point precision of the result always matches that of the input.

Note – S3L_fft_detailed, S3L_fft_detailed, and S3L_ifft do not perform any scaling. Consequently, when a forward FFT is followed by an inverse FFT, the original data will be scaled by the product of the extents of the array.

Syntax

The C and Fortran syntax for S3L_fft_detailed is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_fft_detailed(a, setup_id, iflag, axis)
    S3L_array_t        a
    int                setup_id
    int                iflag
    int                axis
```

F77/F90 Syntax

```
include ‘s3l/s3l-f.h’
include ‘s3l/s3l_errno-f.h’
subroutine
S3L_fft_detailed(a, setup_id, iflag, axis, ier)
    integer*8          a
    integer*4          setup_id
    integer*4          iflag
    integer*4          axis
    integer*4          ier
```

Input

`S3L_fft_detailed` accepts the following arguments as input:

- `a` – Parallel array that is to be transformed. Its rank, extents, and type must be the same as the parallel array `a` supplied in the `S3L_fft_setup` call.
- `setup_id` – Scalar integer variable. Use the value returned by the `S3L_fft_setup` call for this argument.
- `iflag` – Determines the transform direction. Set `iflag` to 1 for forward FFT; set to –1 for inverse FFT.
- `axis` – Determines the axis along which the FFT is to be computed.

Output

`S3L_fft_detailed` uses the following arguments for output:

- `a` – The input array `a` is overwritten with the result of the FFT.
- `ier` (Fortran only) – When called from a Fortran program, `S3L_fft_detailed` returns error status in `ier`.

Error Handling

On success, S3L_fft_detailed returns S3L_SUCCESS.

S3L_fft_detailed performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and returns an error code indicating which value was invalid. See Appendix A of this manual for a detailed list of these error codes.

The following conditions will cause the function to terminate and return the associated error code:

- S3L_ERR_ARG_NCOMPLEX – Array a is not complex.
- S3L_ERR_FFT_EXTSQPROCS – Array a is 1D, but its extent is not divisible by the square of the number of processes.
- S3L_ERR_ARG_SETUP – Invalid setup_id value.
- S3L_ERR_FFT_INVIFLAG – The iflag argument is invalid.

Examples

/opt/SUNWhpc/examples/s3l/fft/fft.c

/opt/SUNWhpc/examples/s3l/fft/ex_fft1.c

/opt/SUNWhpc/examples/s3l/fft/ex_fft2.c

/opt/SUNWhpc/examples/s3l/fft-f/fft.f

Related Functions

S3L_fft_setup(3)

S3L_fft_free_setup(3)

S3L_ifft(3)

S3L_fft(3)

S3L_cr_fft(3)

S3L_rc_fft(3)

S3L_rc_fft_setup(3)

S3L_fft_free_setup

Description

S3L_fft_free_setup deallocates internal memory associated with setup_id by a previous call to S3L_fft_setup.

Syntax

The C and Fortran syntax for S3L_fft_free_setup is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_fft_free_setup(setup_id)
    int                setup_id
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_fft_free_setup(setup_id, ier)
    integer*4          setup_id
    integer*4          ier
```

Input

S3L_fft_free_setup accepts the following argument as input:

- setup_id – Scalar integer variable. Use the value returned by the S3L_fft_setup call for this argument.

Output

S3L_fft_free_setup uses the following argument for output:

- ier (Fortran only) – When called from a Fortran program, S3L_fft_free_setup returns error status in ier.

Error Handling

On success, S3L_fft_free_setup returns S3L_SUCCESS.

The following condition will cause S3L_fft_free_setup to terminate and return the associated error code:

- S3L_ERR_ARG_SETUP – Invalid setup_id value

Examples

/opt/SUNWhpc/examples/s3l/fft/fft.c

/opt/SUNWhpc/examples/s3l/fft/ex_fft1.c

/opt/SUNWhpc/examples/s3l/fft/ex_fft2.c

/opt/SUNWhpc/examples/s3l/fft-f/fft.f

/opt/SUNWhpc/examples/s3l/fft-f/ex_fft1.f

Related Functions

S3L_fft_setup(3)

S3L_fft(3)

S3L_ifft(3)

S3L_fft_detailed(3)

S3L_fft_setup

Description

A call to S3L_fft_setup is the first step in executing Sun S3L Fast Fourier Transforms. It taskes as an argument the S3L handle of the parallel array a that is to be transformed. It returns a setup value in setup_id, which is used in subsequent calls to other S3L FFT routines.

When S3L_fft_setup is called, the contents of array a can be arbitrary. The setup routine neither examines nor modifies the contents of this parallel array. It simply uses its size and type to create the setup object.

The setup ID computed by the S3L_fft_setup call can be used for any parallel arrays that have the same rank, extents, and type as the a argument supplied in the S3L_fft_setup call—but only for such parallel arrays. If a transform is to be performed on two parallel arrays, a and b, identical in rank, extents, and type, then one call to the setup routine suffices, even if transforms are performed on different axes of the two parallel arrays. But if a and b differ in rank, extents, or type, a separate setup call is required for each.

More than one setup ID can be active at a time; that is, the setup routine can be called more than once before deallocating any setup IDs. Consequently, special care must be taken to specify the correct setup ID for calls to S3L_fft, S3L_ifft, S3L_fft_detailed, and S3L_fft_free_setup.

The time required to compute the contents of an FFT setup_id structure is substantially longer than the time required to actually perform an FFT. For this reason, and because it is common to perform FFTs on many parallel variables with the same rank, extents, and type, Sun S3L keeps the setup and transform phases distinct.

When a is no longer needed, S3L_fft_free_setup should be called to deallocate the FFT setup_id.

Syntax

The C and Fortran syntax for S3L_fft_setup is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_fft_setup(a, setup_id)
    S3L_array_t         a
    int                 *setup_id
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_fft_setup(a, setup_id, ier)
    integer*8           a
    integer*4           setup_id
    integer*4           ier
```

Input

`S3L_fft_setup` accepts the following argument as input:

- `a` – S3L array handle for a parallel array that will be the subject of subsequent transform operations.

Output

`S3L_fft_setup` uses the following arguments for output:

- `setup_id` – On output, it contains an integer value that can be used in subsequent calls to `S3L_fft`, `S3L_ifft`, `S3L_fft_detailed`, and `S3L_fft_free_setup`.
- `ier` (Fortran only) – When called from a Fortran program, `S3L_fft_setup` returns error status in `ier`.

Error Handling

On success, `S3L_fft_setup` returns `S3L_SUCCESS`.

S3L_fft_setup performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

The following conditions will cause S3L_fft_setup to terminate and return the associated error code:

- S3L_ERR_FFT_RANKGT3 – The rank of array a is larger than 3.
- S3L_ERR_ARG_NCOMPLEX – Array a is not of type S3L_complex or S3L_double_complex.
- S3L_ERR_FFT_EXTSQPROCS – Array a is a 1D array, but its extent is not a multiple of the square of the number of processes over which it was defined.

Examples

/opt/SUNWhpc/examples/s3l/fft/fft.c

/opt/SUNWhpc/examples/s3l/fft/ex_fft1.c

/opt/SUNWhpc/examples/s3l/fft/ex_fft2.c

/opt/SUNWhpc/examples/s3l/fft-f/fft.f

/opt/SUNWhpc/examples/s3l/fft-f/ex_fft1.f

Related Functions

S3L_fft(3)

S3L_fft_free_setup(3)

S3L_ifft(3)

S3L_fft_detailed(3)

S3L_fin_fd_1D

Description

S3L_fin_fd_1D uses the fourth-order, unconditionally stable, oscillation-free finite-difference (FD) method to solve a one-dimensional (1D) Black-Scholes partial differential equation (PDE) in the user-specified region. It computes prices of vanilla and several exotic stock options. It also provides optional support for hedge statistics ("Greeks"). The types of supported exotic options are described in the list of arguments.

Syntax

The C and Fortran syntax for S3L_fin_fd_1D is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_fin_fd_1D(strike_price, interest_rate, dvdnd_yield,
volatility, exercise_schedule, n_ex, dividend_schedule, n_ds,
dividends, option_charm, option_type, exercise_type, hedge_stat,
s_min, s_max, n_s, n_time, error_tol, num_iterations,
option_price, stock_price, delta, gamma, theta, vega, rho)
    <type>              strike_price
    <type>              interest_rate
    <type>              dvdnd_yield
    <type>              volatility
    <type>              *exercise_schedule
    int                 n_ex
    <type>              *dividend_schedule
    int                 n_ds
    <type>              *dividends
    int                 option_charm
    int                 option_type
    int                 exercise_type
    int                 hedge_stat
    <type>              s_min
    <type>              s_max
    int                 n_s
    int                 n_time
    <type>              *error_tol
    int                 *num_iterations
    S3L_array_t         *option_price
    S3L_array_t         *stock_price
    S3L_array_t         *delta
    S3L_array_t         *gamma
    S3L_array_t         *theta
    S3L_array_t         *vega
    S3L_array_t         *rho
```

where `<type>` is either `float` or `double`.

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_fin_fd_1D(strike_price, interest_rate, dvdnd_yield,
volatility, exercise_schedule, n_ex, dividend_schedule, n_ds,
dividends, option_charm, option_type, exercise_type, hedge_stat,
s_min, s_max, n_s, n_time, error_tol, num_iterations,
option_price, stock_price, delta, gamma, theta, vega, rho, ier)
    <type>          strike_price
    <type>          interest_rate
    <type>          dvdnd_yield
    <type>          volatility
    <type>          exercise_schedule
    integer*4       n_ex
    <type>          dividend_schedule
    integer*4       n_ds
    <type>          dividends
    integer*4       option_charm
    integer*4       option_type
    integer*4       exercise_type
    integer*4       hedge_stat
    <type>          s_min
    <type>          s_max
    integer*4       n_s
    integer*4       n_time
    <type>          error_tol
    integer*4       num_iterations
    integer*8       option_price
    integer*8       stock_price
    integer*8       delta
    integer*8       gamma
    integer*8       theta
    integer*8       vega
    integer*8       rho
    integer*4       ier
```

where `<type>` is either `real*4` or `real*8`.

Input

`S3L_fin_fd_1D` accepts the following arguments as input:

- `strike_price` – Input parameter specifying strike price. Must be greater than 0.

- `interest_rate` – Input parameter specifying interest rate. Must be greater than 0.
- `dvdnd_yield` – Input parameter specifying continuous dividend yield. Must be greater than or equal to 0.
- `volatility` – Input parameter specifying stock volatility. Must be greater than 0.
- `exercise_schedule` – Input parameter specifying expiration dates. All entries in `exercise_schedule` must be greater than 0. The largest entry in the array is used to set the time to maturity of the contract. If European or American options are specified, the other entries are not used. If Bermudan options are specified, all entries in `exercise_schedule` are used. Entries in `exercise_schedule` do not have to be sorted.
- `n_ex` – Input parameter specifying the size of `exercise_schedule`. Must be greater than 0.
- `dividend_schedule` – Input parameter specifying dividend dates. All entries in `dividend_schedule` must be greater than 0 and less than the maximum value in `exercise_schedule`. For discrete dividends, `dividend_schedule` should be an array of type <type_data>. Entries in `dividend_schedule` do not need to be sorted. However, the i-th element in `dividend_schedule` is always associated with the i-th element in `dividends`.
- `n_ds` – Input parameter specifying size of `dividend_schedule`. Must be greater than 0.
- `dividends` – Input parameter specifying dividends. All entries must be greater than or equal to 0. For discrete dividends, `dividends` should be an array of type <type_data>.
- `option_charm` – Input parameter specifying option version. The allowed values for `option_charm` are:

`S3L_VANILLA`	For standard (vanilla) option
`S3L_BINARY_CON`	For binary cash-or-nothing option
`S3L_BINARY_AON`	For binary asset-or-nothing option

- `option_type` – Input parameter specifying option type. The allowed values for `option_type` are:

`S3L_CALL`	For call option
`S3L_PUT`	For put option

- exercise_type – Input parameter specifying option exercise type. The allowed values for exercise_type are:

S3L_EUROPEAN	For European option
S3L_BERMUDAN	For Bermudan option
S3L_AMERICAN	For American option

- hedge_stat – Input parameter specifying computation of hedge statistics (Greeks). The allowed values for hedge_stat are:

0	Do not compute Greeks
nonzero	Compute Greeks

- s_min – Input parameter specifying the minimum stock price for the range in which the option price is computed. Must be greater than 0 and less than s_max.
- s_max – Input parameter specifying the maximum stock price for the range in which the option price is computed. Must be greater than 0.
- n_s – Input parameter specifying stock price discretization. This is the number of grid points between s_min and s_max. n_s must be even and greater than 0.
- n_time – Input parameter specifying time discretization. This is the number of grid points between 0 and the expiration date. Must be greater than 0.
- error_tol – Input parameter specifying error tolerance. If a negative value is given for error_tol, 1.0e–08 will be used in its place.
- num_iterations – Input parameter specifying the maximum number of iterations. If a negative value is given for num_iterations, 10000 will be used in its place.

Output

S3L_fin_fd_1D uses the following arguments for output:

- option_price – S3L array. On exit, option_price holds option prices for the corresponding stock prices in stock_price. option_price should have a length of at least n_s.
- stock_price – S3L array. On exit, stock_price holds stock prices that fall between s_min and s_max with nonuniform discretization. stock_price should have a length of at least n_s.
- delta – S3L array. On exit, delta holds values of delta (the first derivative of option price with respect to stock price) for the corresponding stock prices in stock_price. If hedge_stat is not 0, delta should have a length of at least n_s. delta is not used if hedge_stat is 0.

- gamma – S3L array. On exit, gamma holds values of gamma (the second derivative of option price with respect to stock price) for the corresponding stock prices in stock_price. If hedge_stat is not zero, gamma should have a length of at least n_s. If hedge_stat is zero, gamma is not used.
- theta – S3L array. On exit, theta holds values of theta (the first derivative of option price with respect to time) for the corresponding stock prices in stock_price. theta should have a length of at least n_s.
- vega – S3L array. On exit, vega holds values of vega (the first derivative of option price with respect to volatility) for the corresponding stock prices in stock_price. If hedge_stat is not zero, vega should have a length of at least n_s. If hedge_stat is zero, vega is not used.
- rho – S3L array. On exit, rho holds values of rho (the first derivative of option price with respect to interest rate) for the corresponding stock prices in stock_price. If hedge_stat is not zero, rho should have a length of at least n_s. If hedge_stat is zero, rho is not used.
- ier (Fortran only) – When called from a Fortran program, S3L_fin_fd_1D returns error status in ier.

Error Handling

On success, S3L_fin_fd_1D returns S3L_SUCCESS.

S3L_fin_fd_1D performs generic checking of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

Examples

/opt/SUNWhpc/examples/s3l/financial/ex_fin_fd_1D.c

/opt/SUNWhpc/examples/s3l/financial-f/ex_fin_fd_1D.f

Related Function

S3L_fin_fd_2D(3)

S3L_fin_fd_2D

Description

S3L_fin_fd_2D uses the fourth-order, unconditionally stable, oscillation-free finite-difference (FD) method to solve a two-dimensional (2D) Black-Scholes partial differential equation (PDE) in the user-specified region. It computes prices of certain exotic stock options. It also provides optional support for hedge statistics ("Greeks"). The types of supported exotic options are described in the list of arguments.

Syntax

The C and Fortran syntax for S3L_fin_fd_2D is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_fin_fd_2D(strike_price, interest_rate, dvdnd_yield,
volatility, exercise_schedule, n_ex, dividend_schedule, n_ds,
dividends, observation_schedule, n_os, option_charm,
option_type, exercise_type, hedge_stat, x_min, x_max,
n_discretization, n_time, error_tol, num_iterations,
option_price, stock_price, delta, gamma, theta, vega, rho)
    <type>            strike_price
    <type>            interest_rate
    <type>            dvdnd_yield
    <type>            volatility
    <type>            *exercise_schedule
    int               n_ex
    <type>            *dividend_schedule
    int               n_ds
    <type>            *dividends
    <type>            *observation_schedule
    int               n_os
    int               option_charm
    int               option_type
    int               exercise_type
    int               hedge_stat
    <type>            x_min[2]
    <type>            x_max[2]
    int               n_discretization[2]
    int               n_time
    <type>            *error_tol
    int               *num_iterations
    S3L_array_t       *option_price
    S3L_array_t       *stock_price
    S3L_array_t       *delta
    S3L_array_t       *gamma
    S3L_array_t       *theta
    S3L_array_t       *vega
    S3L_array_t       *rho
```

where `<type>` is either `float` or `double`.

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_fin_fd_2D(strike_price, interest_rate, dvdnd_yield,
volatility, exercise_schedule, n_ex, dividend_schedule, n_ds,
dividends, observation_schedule, n_os, option_charm,
option_type, exercise_type, hedge_stat, x_min, x_max,
n_discretization, n_time, error_tol, num_iterations,
option_price, stock_price, delta, gamma, theta, vega, rho, ier)
    <type>            strike_price
    <type>            interest_rate
    <type>            dvdnd_yield
    <type>            volatility
    <type>            exercise_schedule
    integer*4         n_ex
    <type>            dividend_schedule
    integer*4         n_ds
    <type>            dividends
    <type>            observation_schedule
    integer*4         n_os
    integer*4         option_charm
    integer*4         option_type
    integer*4         exercise_type
    integer*4         hedge_stat
    <type>            x_min(2)
    <type>            x_max(2)
    integer*4         n_discretization
    integer*4         n_time
    <type>            error_tol
    integer*4         num_iterations
    integer*8         option_price
    integer*8         stock_price
    integer*8         delta
    integer*8         gamma
    integer*8         theta
    integer*8         vega
    integer*8         rho
    integer*4         ier
```

where `<type>` is either `real*4` or `real*8`.

Input

S3L_fin_fd_2D accepts the following arguments as input:

- strike_price – Input parameter specifying strike price. Must be greater than 0.
- interest_rate – Input parameter specifying interest rate. Must be greater than 0.
- dvdnd_yield – Input parameter specifying continuous dividend yield. Must be greater than or equal to 0.
- volatility – Input parameter specifying stock volatility. Must be greater than 0.
- exercise_schedule – Input parameter specifying expiration dates. All entries in exercise_schedule must be greater than 0. The largest entry in the array is used to set the time to maturity of the contract. If European or American options are specified, the other entries are not used. If Bermudan options are specified, all entries in exercise_schedule are used. Entries in exercise_schedule do not have to be sorted.
- n_ex – Input parameter specifying the size of exercise_schedule. Must be greater than 0.
- dividend_schedule – Input parameter specifying dividend dates. All entries in dividend_schedule must be greater than 0 and less than the maximum value in exercise_schedule. For discrete dividends, dividend_schedule should be an array of type <type_data>. Entries in dividend_schedule do not need to be sorted. However, the i-th element in dividend_schedule is always associated with the i-th element in dividends.
- n_ds – Input parameter specifying size of dividend_schedule. Must be greater than 0.
- dividends – Input parameter specifying dividends. All entries must be greater than or equal to 0. For discrete dividends, dividends should be an array of type <type_data>.
- observation_schedule – Input parameter specifying dates for exotic options—that is, dates when the average or minimum/maximum value of a stock price is sampled. All entries in observation_schedule must be greater than 0 and less than the largest value in exercise_schedule. observation_schedule should be an array of type <type_data>. Entries in observation_schedule do not have to be sorted.
- n_os – Input parameter specifying the size of observation_schedule. n_os must be greater than 0.
- option_charm – Input parameter specifying option version. The allowed value for option_charm is:

S3L_ASIAN_A_RT	For arithmetic average rate option (also known as fixed strike option)

- option_type – Input parameter specifying option type. The allowed values for option_type are:

S3L_CALL	For call option
S3L_PUT	For put option

- exercise_type – Input parameter specifying option exercise type. The allowed values for exercise_type are:

S3L_EUROPEAN	For European option
S3L_BERMUDAN	For Bermudan option
S3L_AMERICAN	For American option

- hedge_stat – Input parameter specifying computation of hedge statistics (Greeks). The allowed values for hedge_stat are:

0	Do not compute Greeks
nonzero	Compute Greeks

- x_min – Input parameter specifying the minimum stock price for the range in which the option price is computed. x_min should be a two-element array of type <type_data>. Each value in the array must be greater than 0 and less than the corresponding x_max.
- x_max – Input parameter specifying the maximum stock price for the range in which the option price is computed. x_max should be a two-element array of type <type_data>. Each value in the array must be greater than 0.
- n_discretization – Input parameter specifying variable discretization. This is the number of grid points between x_min and x_max. n_discretization should be a two-element array of integers. Each value in the array must be even and greater than 0.

  n_discretization[0] specifies stock price discretization and n_discretization[1] specifies discretization of the second parameter—in the case of the Asian option, for example, it specifies discretization of the average stock price.

- n_time – Input parameter specifying time discretization. This is the number of grid points between 0 and the expiration date. Must be greater than 0.
- error_tol – Input parameter specifying error tolerance. If a negative value is given for error_tol, 1.0e–08 will be used in its place.
- num_iterations – Input parameter specifying the maximum number of iterations. If a negative value is given for num_iterations, 10000 will be used in its place.

Output

S3L_fin_fd_2D uses the following arguments for output:

- option_price – S3L array. On exit, option_price holds option prices for the corresponding stock prices in stock_price. option_price should have a length of at least n_discretization[0].
- stock_price – S3L array. On exit, stock_price holds stock prices that fall between x_min and x_max with nonuniform discretization. stock_price should have a length of at least n_discretization[0].
- delta – S3L array. On exit, delta holds values of delta (the first derivative of option price with respect to stock price) for the corresponding stock prices in stock_price. If hedge_stat is not zero, delta should have a length of at least n_discretization[0]. delta is not used if hedge_stat is zero.
- gamma – S3L array. On exit, gamma holds values of gamma (the second derivative of option price with respect to stock price) for the corresponding stock prices in stock_price. If hedge_stat is not zero, gamma should have a length of at least n_discretization[0]. If hedge_stat is zero, gamma is not used.
- theta – S3L array. On exit, theta holds values of theta (the first derivative of option price with respect to time) for the corresponding stock prices in stock_price. theta should have a length of at least n_discretization[0].
- vega – S3L array. On exit, vega holds values of vega (the first derivative of option price with respect to volatility) for the corresponding stock prices in stock_price. If hedge_stat is not zero, vega should have a length of at least n_discretization[0]. If hedge_stat is zero, vega is not used.
- rho – S3L array. On exit, rho holds values of rho (the first derivative of option price with respect to interest rate) for the corresponding stock prices in stock_price. If hedge_stat is not zero, rho should have a length of at least n_discretization[0]. If hedge_stat is zero, rho is not used.
- ier (Fortran only) – When called from a Fortran program, S3L_fin_fd_2D returns error status in ier.

Error Handling

On success, S3L_fin_fd_2D returns S3L_SUCCESS.

S3L_fin_fd_2D performs generic checking of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

Examples

/opt/SUNWhpc/examples/s3l/financial/ex_fin_fd_2D.c

/opt/SUNWhpc/examples/s3l/financial-f/ex_fin_fd_2D.f

Related Function

S3L_fin_fd_1D(3)

S3L_forall

Description

S3L_forall applies a user-defined function to elements of a parallel Sun S3L array and sets its values accordingly. Three different function types are supported. These types are described in TABLE 2-3.

TABLE 2-3 User-Defined Function Types for S3L_forall

fn_type	C Prototype	Fortran Interface
S3L_ELEM_FN1	void user_fn(void *elem_addr);	subroutine user_fn(a) <type> a end user_fn
S3L_ELEM_FNN	void user_fn(void *elem_addr, int n);	subroutine user_fn(a,n) <type> a integer*4 n end user_fn
S3L_INDEX_FN	void user_fn(void *elem_addr, int *coord);	subroutine user_fn(a,v coord) <type> a

Here, <type> is one of integer*4, integer*8, real*4, real*8, complex*8, or complex*16, and rank is the rank of the array.

For S3L_ELEM_FN1, the user function is applied to each element in the array.

For S3L_ELEM_FNN, the user function is supplied the local subgrid address and subgrid size and iterates over subgrid elements. This form delivers the highest performance because the looping over the elements is contained within the function call.

For S3L_INDEX_FN, the user function is applied to each element in the subarray specified by the triplets argument to S3L_forall. If the triplets argument is NULL in C/C++ or has a leading value of 0 in F77/F90, the whole array is implied. The user function may involve the global coordinates of the array element; these are contained in the coord argument. Global coordinates of array elements are 0-based for C programs and 1-based for Fortran programs.

Note – When a Fortran program uses triplets, the length of the first axis of the triplets must equal the rank of the array. Failure to meet this requirement can produce wrong results or a segmentation violation.

Note – A subgrid is the portion of the parallel array that is owned by a process. A subarray is the portion of the parallel array that is described by a lower bound, an upper bound, and a stride in each dimension.

Syntax

The C and Fortran syntax for S3L_forall is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_forall(a, user_fn, fn_type, triplets)
    S3L_array_t        a
    void               (*user_fn)()
    int                fn_type
    int                triplets[rank][3]
```

where rank is the rank of the array.

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_forall(a, user_fn, fn_type, triplets, ier)
    integer*8          a
    <external>         user_fn
    integer*4          fn_type
    integer*4          triplets(rank,3)
    integer*4          ier
```

where `rank` is the rank of the array.

Input

`S3L_forall` accepts the following arguments as input:

- `a` – Parallel array to which the function will be applied.
- `user_fn` – Pointer to the user-defined function.
- `fn_type` – Predefined value specifying the class of functions to which the function belongs. See the Description section for a list of valid `fn_type` entries.
- `triplets` – An integer vector that is used to restrict the function to a range of elements. For each axis of the array, a triplet takes the form:

  inclusive lower bound

  inclusive upper bound

  stride

  The stride must be positive. To apply the function to all the elements in the array, set `triplets` to `NULL` (C/C++) or to `<= 0` (F77/F90).

Output

`S3L_forall` uses the following argument for output:

- `ier` (Fortran only) – When called from a Fortran program, `S3L_forall` returns error status in `ier`.

Error Handling

On success, S3L_forall returns S3L_SUCCESS.

S3L_forall performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause the function to terminate and return the associated error code:

- S3L_ERR_FORALL_INVFN – User-specified function is invalid. fn_type is not one of:
  - S3L_ELEM_FN1
  - S3L_ELEM_FNN
  - S3L_INDEX_FN
- S3L_ERR_INDX_INVALID – fn_type is S3L_INDEX_FN and one or more of the elements in the triplets argument has an invalid value.

Examples

/opt/SUNWhpc/examples/s3l/forall/ex_forall.c

/opt/SUNWhpc/examples/s3l/forall/ex_forall2.cc

/opt/SUNWhpc/examples/s3l/forall-f/ex_forall.f

S3L_free

Description

S3L_free deallocates the memory reserved for a parallel S3L array and undefines the associated array handle.

Note – If memory was allocated for the array by the user rather than by S3L, `S3L_free` destroys the array handle but does not deallocate the memory. This situation can arise when `S3L_declare_detailed()` is invoked with the `atype` argument set to `S3L_DONOT_ALLOCATE`.

Syntax

The C and Fortran syntax for `S3L_free` is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_free(a)
    S3L_pgrid_t        *a
```

F77/F90 Syntax

```
include ‘s3l/s3l-f.h’
include ‘s3l/s3l_errno-f.h’
subroutine
S3L_free(a, ier)
    integer*8          a
    integer*4          ier
```

Input

`S3L_free` accepts the following argument as input:

- `a` – Handle for the parallel S3L array that is to be deallocated. This handle was returned by a previous call to `S3L_declare`, `S3L_declare_detailed`.

Output

S3L_free uses the following argument for output:

- ier (Fortran only) – When called from a Fortran program, S3L_free returns error status in ier.

Error Handling

On success, S3L_free returns S3L_SUCCESS.

On error, S3L_free returns the following error code:

- S3L_ERR_ARG_ARRAY – a is a NULL pointer (C/C++) or 0 (F77/F90).

Examples

/opt/SUNWhpc/examples/s3l/io/ex_print1.c

/opt/SUNWhpc/examples/s3l/io-f/ex_print1.f

Related Functions

S3L_declare(3)

S3L_declare_detailed(3)

S3L_free_process_grid

Description

S3L_free_process_grid frees the process grid handle returned by a previous call to S3L_set_process_grid.

Syntax

The C and Fortran syntax for S3L_free_process_grid is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_free_process_grid(pgrid)
    S3L_pgrid_t        *pgrid
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_free_process_grid(pgrid, ier)
    integer*8          pgrid
    integer*4          ier
```

Input

S3L_free_process_grid accepts the following argument as input:

- pgrid – The process grid handle returned by a previous call to S3L_set_process_grid.

Output

S3L_free_process_grid uses the following argument for output:

- ier (Fortran only) – When called from a Fortran program, S3L_free_process_grid returns error status in ier.

Error Handling

On success, S3L_free_process_grid returns S3L_SUCCESS.

On error, S3L_free returns the following error code:

- S3L_ERR_PGRID_NULL – An invalid process grid argument was supplied.

Examples

/opt/SUNWhpc/examples/s3l/utils/scalapack_conv.c

/opt/SUNWhpc/examples/s3l/utils-f/scalapack_conv.f

Related Function

S3L_set_process_grid(3)

S3L_free_rand_fib

Description

S3L_free_rand_fib frees memory allocated to a random number generator state table associated with a particular setup ID value.

Syntax

The C and Fortran syntax for S3L_free_rand_fib is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_free_rand_fib(setup_id)
    int                setup_id
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_free_rand_fib(setup_id, ier)
    integer*4          setup_id
    integer*4          ier
```

Input

S3L_free_rand_fib accepts the following argument as input:

- setup_id – Integer index that has been initialized by a call to S3L_setup_rand_fib and is used to identify a particular state table setup.

Output

S3L_free_rand_fib uses the following argument for output:

- ier (Fortran only) – When called from a Fortran program, S3L_free_rand_fib returns error status in ier.

Error Handling

On success, S3L_free_rand_fib returns S3L_SUCCESS.

On error, S3L_free returns the following error code:

- S3L_ERR_ARG_SETUP – Invalid setup_id value.

Examples

/opt/SUNWhpc/examples/s3l/rand_fib/rand_fib.c

/opt/SUNWhpc/examples/s3l/rand_fib-f/rand_fib.f

Related Functions

S3L_rand_fib(3)

S3L_setup_rand_fib(3)

S3L_free_sparse

Description

S3L_free_sparse deallocates the memory reserved for a sparse matrix and the associated array handle.

Syntax

The C and Fortran syntax for S3L_free_sparse is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_free_sparse(A)
    S3L_array_t         *A
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_free_sparse(A, ier)
    integer*8           A
    integer*4           ier
```

Input

S3L_free_sparse accepts the following argument as input:

- A – Handle for the parallel S3L array that was allocated through a previous call to S3L_declare_sparse, S3L_read_sparse, or S3L_rand_sparse.

Output

S3L_free_sparse uses the following argument for output:

- ier (Fortran only) – When called from a Fortran program, S3L_free_sparse returns error status in ier.

Error Handling

On success, S3L_free_sparse returns S3L_SUCCESS.

On error, S3L_free returns the following error code:

- S3L_ERR_ARG_ARRAY – A is a NULL pointer (C/C++) or 0 (F77/F90).

Examples

/opt/SUNWhpc/examples/s3l/sparse/ex_sparse.c

/opt/SUNWhpc/examples/s3l/sparse/ex_sparse2.c

/opt/SUNWhpc/examples/s3l/iter/ex_iter.c

/opt/SUNWhpc/examples/s3l/sparse-f/ex_sparse.f

/opt/SUNWhpc/examples/s3l/iter-f/ex_iter.f

Related Functions

S3L_declare_sparse(3)

S3L_read_sparse(3)

S3L_rand_sparse(3)

S3L_from_ScaLAPACK_desc

Description

S3L_from_ScaLAPACK_desc converts the ScaLAPACK descriptor and subgrid address specified by scdesc and address into an S3L array handle, which is returned in s3ldesc.

Syntax

The C and Fortran syntax for S3L_from_ScaLAPACK_desc is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_from_ScaLAPACK_desc(s3ldesc, scdesc, data_type, address)
    S3L_array_t         *s3ldesc
    int                 *scdesc
    S3L_data_type       data_type
    void                *address
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_from_ScaLAPACK_desc(s3ldesc, scdesc, data_type, address, ier)
    integer*8           s3ldesc
    integer*4           scdesc(*)
    integer*4           data_type
    pointer             address
    integer*4           ier
```

Input

S3L_from_ScaLAPACK_desc accepts the following arguments as input:

- scdesc – ScaLAPACK descriptor for a parallel array.
- data_type – Specifies the data type of the S3L array. It must specify a data type supported by Sun S3L.
- address – This input argument holds the starting address of an existing array subgrid.

Note – In Fortran programs, address should be either a pointer (see the Fortran documentation for details) or the starting address of a local array, as determined by the loc(3F) function.

Output

S3L_from_ScaLAPACK_desc uses the following arguments for output:

- s3ldesc – S3L array handle that is the output of S3L_from_ScaLAPACK_desc.
- ier (Fortran only) – When called from a Fortran program, S3L_from_ScaLAPACK_desc returns error status in ier.

Error Handling

On success, S3L_from_ScaLAPACK_desc returns S3L_SUCCESS.

S3L_from_ScaLAPACK_desc performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause the function to terminate and return the associated error code:

- S3L_ERR_ARG_NULL – The scdesc argument is a NULL pointer (C/C++) or 0 (F77/F90).
- S3L_ERR_NOT_SUPPORT – The ScaLAPACK descriptor data type is not supported by Sun S3L.
- S3L_ERR_PGRID_NOPROCS – The ScaLAPACK descriptor has an invalid BLACS context.

Examples

/opt/SUNWhpc/examples/s3l/utils/scalapack_conv.c

/opt/SUNWhpc/examples/s3l/utils-f/scalapack_conv.f

Related Function

S3L_to_ScaLAPACK_desc(3)

S3L_gen_band_factor

Description

S3L_gen_band_factor performs the LU factorization of an n x n general banded array with lower bandwidth bl and upper bandwidth bu. The nonzero diagonals of the array should be stored in an S3L array a of size [2*bl+2*bu+1,n].

In the more general case, a can be a multidimensional array, where axis_r and axis_d denote the array axes whose extents are 2*bl+2*bu+1 and n, respectively. The format of the array a is described in the following example:

Example:

Consider a 7 x 7 (n=7) banded array with bl = 1, bu = 2. c is the main diagonal, b is the first superdiagonal, and a the second. d is the first subdiagonal. The contents of the composite array a used as input to S3L_gen_band_factor should have the following organization:

```
 *   *   *   *   *   *   *
 *   *   *   *   *   *   *
 *   *   *   *   *   *   *
 *   *  a0  a1  a2  a3  a4
 *  b0  b1  b2  b3  b4  b5
c0  c1  c2  c3  c4  c5  c6
d0  d1  d2  d3  d4  d5   *
```

Note that, items denoted by '*' are not referenced.

If `a` is two-dimensional, `S3L_gen_band_factor` is more efficient when `axis_r` is the first axis, `axis_d` is the second axis, and array `a` is block-distributed along the second axis. For C programs, the indices of the first and second axes are `0` and `1`, respectively. For Fortran programs, the corresponding indices are `1` and `2`.

If `a` has more than two dimensions, `S3L_gen_band_factor` is most efficient when axes `axis_r` and `axis_d` of `a` are local (that is, are not distributed).

Syntax

The C and Fortran syntax for `S3L_gen_band_factor` is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_gen_band_factor(a, bl, bu, factors, axis_r, axis_d)
    S3L_array_t       a
    int               bl
    int               bu
    int               *factors
    int               axis_r
    int               axis_d
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_gen_band_factor(a, bl, bu, factors, axis_r, axis_d, ier)
    integer*4         a
    integer*4         bl
    integer*4         bu
    integer*4         factors
    integer*4         axis_r
    integer*4         axis_d
    integer*4         ier
```

Input

S3L_gen_band_factor accepts the following arguments as input:

- a – S3L array handle for a real or complex parallel array of size [1+2*bl+2*bl,n].
- bl – Lower bandwidth of a.
- bu – Upper bandwidth of a.
- axis_r – Specifies the row axis along which factorization will occur.
- axis_d – Specifies the column axis along which factorization will occur.

Output

S3L_gen_band_factor uses the following arguments for output:

- a – Upon successful completion, S3L_gen_band_factor stores the factorization results in a.
- factors – Pointer to an internal structure that holds the factorization.
- ier (Fortran only) – When called from a Fortran program, S3L_gen_band_factor returns error status in ier.

Error Handling

On success, S3L_gen_band_factor returns S3L_SUCCESS.

S3L_gen_band_factor performs generic checking of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause the function to terminate and return the associated error code:

- S3L_ERR_ARG_DTYPE – The type of a is not real, double, complex or double complex.
- S3L_ERR_INDX_INVALID – bl or bu value is invalid for either of the following reasons:
  - It is less than 0 (C/C++) or less than 1 (F77/F90).
  - It is greater than the extent of a along axis_d.
- S3L_ERR_ARG_EXTENTS – The extent of a along axis axis_r is not equal to 2*bl+2*bu+1.
- S3L_ERR_ARRTOOSMALL – The extents of a along axis axis_d are such that the block size in a block distribution is less than bu + bl + 1.

- S3L_ERR_ARG_AXISNUM – An axis argument is invalid for one of the following reasons:
  - It is less than 0 (C/C++) or less than 1 (F77/F90).
  - It is greater than the rank of the referenced array.
  - axis_d is equal to axis_r.
- S3L_ERR_BAND_FFAIL – The factorization could not be completed.

Examples

/opt/SUNWhpc/examples/s3l/band/ex_band.c

/opt/SUNWhpc/examples/s3l/band-f/ex_band.f

Related Functions

S3L_gen_band_solve(3)

S3L_gen_band_free_factors(3)

S3L_gen_band_free_factors

Description

S3L_gen_band_free_factors frees internal memory associated with a banded matrix factorization.

Syntax

The C and Fortran syntax for S3L_gen_band_free_factors is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_gen_band_free_factors(factors)
    int                 *factors
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_gen_band_free_factors(factors, ier)
    integer*4           factors
    integer*4           ier
```

Input

`S3L_gen_band_free_factors` accepts the following argument as input:

- `factors` – Pointer to the internal structure that will be freed.

Output

`S3L_gen_band_free_factors` uses the following argument for output:

- `ier` (Fortran only) – When called from a Fortran program, `S3L_gen_band_free_factors` returns error status in `ier`.

Error Handling

On success, `S3L_gen_band_free_factors` returns `S3L_SUCCESS`.

The following condition will cause `S3L_gen_band_free_factors` to terminate and return the associated error code:

- `S3L_ERR_ARG_SETUP` – The value of the `factors` argument is invalid.

Examples

/opt/SUNWhpc/examples/s3l/band/ex_band.c

/opt/SUNWhpc/examples/s3l/band-f/ex_band.f

Related Functions

S3L_gen_band_solve(3)

S3l_gen_band_factor(3)

S3L_gen_band_solve

Description

S3L_gen_band_solve solves a banded system whose factorization has been computed by a prior call to S3L_gen_band_factor.

The factored banded matrix is stored in array a, whose dimensions are 2*bu + 2*bl + 1 x n. The right-hand side is stored in array b, whose dimensions are n x nrhs.

If a and b have more than two dimensions, axis_r and axis_d refer to those axes of a whose extents are 2*bu + 2*bl + 1 and n, respectively. Likewise, axis_row and axis_col refer to the axes of b with extents n and nrhs.

Array Layout Guidelines

Two-Dimensional Arrays: If a and b are two-dimensional, S3L_gen_band_solve is more efficient when axis_r = 0, axis_d = 1, array a is block-distributed along axis 1, axis_row = 0, axis_col = 1, and array b is block distributed along axis 0.

Note that the values cited in the previous paragraph apply to programs using the C/C++ interface—that is, they assume zero-based array indexing. When S3L_gen_band_solve is called from F77 or F90 applications, these values must be increased by one. Therefore, when a and b are two-dimensional and S3L_gen_band_solve is called by a Fortran program, the solver is more efficient when axis_r = 1, axis_d = 2, array a is block-distributed along axis 2, axis_row = 1, axis_col = 2 and array b is block-distributed along axis 1.

When a and b are two-dimensional and nrhs is greater than 1, the size of a must be such that n is divisible by the number of processors.

Arrays With More Than Two Dimensions: If a and b have more than two dimensions, S3L_gen_band_solve is more efficient when axis_r and axis_d of a and axis_row and axis_col of b are local (not distributed).

Syntax

The C and Fortran syntax for S3L_gen_band_solve is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_gen_band_solve(a, bl, bu, factors, axis_r, axis_d, b,
axis_row, axis_col)
    S3L_array_t       a
    int               bl
    int               bu
    int               factors
    int               axis_r
    int               axis_d
    S3L_array_t       b
    int               axis_row
    int               axis_col
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_gen_band_solve(a, bl, bu, factors, axis_r, axis_d, b,
axis_row, axis_col, ier)
    integer*4         a
    integer*4         bl
    integer*4         bu
    integer*4         factors
    integer*4         axis_r
    integer*4         axis_d
    integer*8         b
    integer*4         axis_row
    integer*4         axis_col
    integer*4         ier
```

Input

`S3L_gen_band_solve` accepts the following arguments as input:

- `a` – S3L array handle for a real or complex parallel array of size [1+2*`bl`+2*`bu`,n].
- `bl` – Lower bandwidth of `a`.
- `bu` – Upper bandwidth of `a`.
- `factors` – Pointer to an internal structure that holds the factorization results.
- `axis_r` – Specifies the axis of array `a` whose extent is 1+2*`bl`+2*`bu`+1.
- `axis_d` – Specifies the axis of array `a` whose extent is n.
- `axis_row` – Specifies the axis of array `b` whose extent is n.
- `axis_col` – Specifies the axis of array `b` whose extent is nhrs.
- `b` – S3L array handle containing the right-hand side of the matrix equation ax=b.

Output

`S3L_gen_band_solve` uses the following arguments for output:

- `b` – On output, `b` is overwritten by the solution to the matrix equation `ax=b`.
- `ier` (Fortran only) – When called from a Fortran program, `S3L_gen_band_solve` returns error status in `ier`.

Error Handling

On success, S3L_gen_band_solve returns S3L_SUCCESS.

S3L_gen_band_solve performs generic checking of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause the function to terminate and return the associated error code:

- S3L_ERR_ARG_DTYPE – The type of a is not one of: real, double, complex or double complex.
- S3L_ERR_INDX_INVALID – bl or bu value is invalid for either of the following reasons:
  - It is less than 0 (C/C++) or less than 1 (F77/F90).
  - It is greater than the extent of a along axis_d.
- S3L_ERR_ARG_EXTENTS – The extent of a along axis axis_r is not equal to 2*bl+2*bu+1.
- S3L_ERR_ARRTOOSMALL – The extents of a along axis axis_d are such that the block size in a block distribution is less than bu + bl + 1.
- S3L_ERR_ARG_AXISNUM – An axis argument is invalidfor one of the following reasons:
  - It is less than 0 (C/C++) or less than 1 (F77/F90).
  - It is greater than the rank of the referenced array
  - axis_d is equal to axis_r.
- S3L_ERR_MATCH_RANK – The rank of a is not the same as that of b.
- S3L_ERR_ARG_SETUP – The factors value does not correspond to a valid setup.
- S3L_ERR_MATCH_EXTENTS – The extents of a along axis_d do not equal the extents of b along axis_row, or some of the other extents of a and b do not match.

Examples

/opt/SUNWhpc/examples/s3l/band/ex_band.c

/opt/SUNWhpc/examples/s3l/band-f/ex_band.f

Related Functions

S3L_gen_band_factor(3)

S3L_gen_band_free_factors(3)

S3L_gen_iter_solve

Description

Given a general square sparse matrix A and a right-hand side vector b, S3L_gen_iter_solve solves the linear system of equations Ax = b, using an iterative algorithm, with or without preconditioning.

The first three arguments to S3L_gen_iter_solve are S3L internal array handles that describe the global general sparse matrix A, the rank 1 global array b, and the rank 1 global array x.

The sparse matrix A is produced by a prior call to one of the following sparse routines:

- S3L_declare_sparse
- S3L_read_sparse
- S3L_rand_sparse
- S3L_convert_sparse

The rank 1 global arrays, b and x, have the same data type and precision as the sparse matrix A, and both have a length equal to the order of A.

Two local rank 1 arrays, iparm and rparm, provide user control over various aspects of S3L_gen_iter_solve behavior, including:

- Choice of algorithm to be used.
- Type of preconditioner to use on A.
- Flags to select the initial guess to the solution.
- Maximum number of iterations to be taken by the solver.
- If restarted GMRES algorithm is chosen, selection of the size of the Krylov subspace.
- Tolerance values to be used by the stopping criterion.
- If the Richardson algorithm is chosen, selection of the scaling factor to be used.

iparm is an integer array and rparm is a real array. The options supported by these arguments are described in the subsections titled: "Algorithm," "Preconditioning," "Convergence/Divergence Criteria," "Initial Guess," "Maximum Iterations," "Krylov Subspace," "Stopping-Criterion Tolerance," and "Richardson Scaling Factor." The "Iteration Termination" subsection identifies the conditions under which S3L_gen_iter_solve will terminate an operation.

Note – iparm and rparm must be preallocated and initialized before S3L_gen_iter_solve is called. To enable the default condition for any parameter, set it to 0. Otherwise, initialize iparm and rparm with the appropriate parameter values, as described in the following subsections.

Algorithm

S3L_gen_iter_solve attempts to solve Ax = b using one of the following iterative solution algorithms. The choice of algorithm is determined by the value supplied for the parameter iparm[S3L_iter_solver]. The various options available for this parameter are listed and described in TABLE 2-4.

TABLE 2-4 iparm[S3L_iter_solver] Options

Option	Description
S3L_bcgs	BiConjugate Gradient Stabilized (Bi-CGSTAB)
S3L_cgs	Conjugate Gradient Squared (CGS)
S3L_cg	Conjugate Gradient (CG)
S3L_cr	Conjugate Residuals (CR)
S3L_gmres	Generalized Minimum Residual (GMRES) – default
S3L_qmr	Quasi-Minimal Residual (QMR)
S3L_richardson	Richardson method

Preconditioning

S3L_gen_iter_solve implements left preconditioning. That is, preconditioning is applied to the linear system Ax = b by:

$$Q^{-1} A = Q^{-1} b$$

where Q is the preconditioner and Q^{-1} denotes the inverse of Q. The supported preconditioners are listed in TABLE 2-5.

TABLE 2-5 iparm[S3L_iter_pc] Options

Option	Description
S3L_none	No preconditioning will be done (default).
S3L_jacobi	Point Jacobi preconditioner will be used. Note that this option is not supported when the sparse matrix A is represented under S3L_SPARSE_VBR format.
S3L_bjacobi	Block Jacobi preconditioner will be used. Note that this option is supported only when the sparse matrix A is represented under S3L_SPARSE_VBR format.
S3L_ilu	Use a simplified ILU(0); the Incomplete LU factorization of level- zero preconditioner. This preconditioner modifies only diagonal nonzero elements of the matrix. Note that this option is not supported when the sparse matrix A is represented under S3L_SPARSE_VBR format.

Convergence/Divergence Criteria

The iparm[S3L_iter_conv] parameter selects the criterion to be used for stopping computation. Currently, the single valid option for this parameter is S3L_r0, which selects the default criterion for both convergence and divergence. The convergence criterion is satisfied when:

err = ||rj||_2 / ||r0||_2 < epsilon

and the divergence criterion is met when:

err = ||rj||_2 / ||r0||_2 > 10000.0

where:

- rj and r0 are the residuals obtained at iterations j and 0.
- ||.||_2 is the 2-norm.
- epsilon is the desired convergence tolerance stored in rparm[S3L_iter_tol].
- 10000.0 is the divergence tolerance, which is set internally in the solver.

Initial Guess

The parameter iparm[S3L_iter_init] determines the contents of the initial guess for the solution of the linear system as follows:

- `0` – Applies zero as the initial guess. This is the default.
- `1` – Applies the value contained in array `x` as the initial guess. For this case, the user must initialize `x` before calling `S3L_gen_iter_solve`.

Maximum Iterations

On input, the `iparm[S3L_iter_maxiter]` parameter specifies the maximum number of iterations to be taken by the solver. Set to `0` to select the default, which is 10000.

On output, `iparm[S3L_iter_maxiter]` contains the total number of iterations taken by the solver at the time of termination.

Krylov Subspace

If the restarted GMRES algorithm is selected, `iparm[S3L_iter_kspace]` specifies the size of the Krylov subspace to be used. The default is 30.

Stopping-Criterion Tolerance

On input, `rparm[S3L_iter_tol]` specifies the tolerance values to be used by the stopping criterion. Its default is 10-8.

On output, `rparm[S3L_iter_tol]` contains the computed error, `err`, according to the convergence criteria. See the `iparm[S3L_iter_conv]` description for details.

Richardson Scaling Factor

If the Richardson method is selected, `rparm[S3L_rich_scale]` specifies the scaling factor to be used. The default value is 1.0.

Iteration Termination

`S3L_gen_iter_solve` terminates the iteration when one of the following conditions is met:

- The computation has satisfied the convergence criterion.
- The computation has diverged.
- An algorithmic breakdown has occurred.
- The number of iterations has exceeded the supplied value.

Syntax

The C and Fortran syntax for S3L_gen_iter_solve is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_gen_iter_solve(A, b, x, iparm, rparm)
    S3L_array_t        A
    S3L_array_t        b
    S3L_array_t        x
    int                *iparm
    <type>             *rparm
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_gen_iter_solve(A, b, x, iparm, rparm, ier)
    integer*8          A
    integer*8          b
    integer*8          x
    integer*4          iparm(*)
    <type>             rparm(*)
    integer*4          ier
```

where <type> is real*4 or real*8 for both C/C++ and F77/F90.

Input

S3L_gen_iter_solve accepts the following arguments as input:

- A – S3L internal array handle for the global general sparse matrix. The matrix data type can be real or complex (single- or double-precision).
- b – Global array of rank 1, with the same data type and precision as A and x and a length equal to the order of the sparse matrix. b contains the right-hand side vector of the linear problem.

- x – Global array of rank 1, with the same data type and precision as A and b and a length equal to the order of the sparse matrix. On input, x may contain the initial guess for the solution to the linear system. Upon completion, x contains the converged solution (see the Output section).
- iparm – Integer local array of rank 1 and length s3l_iter_iparm_size. On input, iparm options have the following uses:
  - iparm[S3l_iter_solver] – Specifies the iterative algorithm to be used. Set it to 0 to use the default solver GMRES. See the Description section for details.
  - iparm[S3l_iter_pc] – Specifies the preconditioner to be used. Set it to 0 to use the default option, S3L_none.
  - iparm[S3l_iter_conv] – Selects the criterion to be used for stopping the computation.
  - iparm[S3l_iter_init] – Specifies the contents of the initial guess to the solution of the linear system.
  - iparm[S3l_iter_maxiter] – Specifies the maximum number of iterations to be taken by the solver.
  - iparm[S3l_iter_kspace] – Specifies the size of the Krylov subspace for restarted GMRES.
- rparm – Real local array with the same precision as x and a length equal to S3L_iter_rparm_size. On input, it provides the following options for computing all or part of the matrix U.
  - rparm[S3l_iter_tol] – Specifies the tolerance values to be used by the stopping criterion. It has a default of 10-8.
  - rparm[S3l_rich_scale] – Specifies the scaling factor to be used in the Richardson method. The default is 1.0.

Output

S3L_gen_iter_solve uses the following arguments for output:

- x – Upon successful completion, x contains the converged solution. If the computation breaks down or diverges, x will contain the solution produced by the most recent iteration.
- iparm[S3L_iter_maxiter] – On output, contains the total number of iterations taken by the solver at the time of termination.
- rparm[S3L_iter_tol] – On output, contains the computed error, err, according to the convergence criteria. See the iparm[S3L_iter_conv] description for details.
- ier (Fortran only) – When called from a Fortran program, S3L_gen_iter_solve returns error status in ier.

Error Handling

On success, S3L_gen_iter_solve returns S3L_SUCCESS.

S3L_gen_iter_solve performs generic checking of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

On error, it returns one of the following codes, which are organized by error type.

Input Errors

- S3L_ERR_ARG_NULL – Invalid array x or b or sparse matrix A. They all must be preallocated S3L arrays or sparse matrix.
- S3L_ERR_ARRNOTSQ – Invalid matrix size. Matrix A must be square.
- S3L_ERR_ARG_DTYPE – Invalid data type. The data type of matrix A must be real or complex (single- or double-precision).
- S3L_ERR_ARG_RANK – Invalid rank for arrays x and b. Both must be rank 1 arrays.
- S3L_ERR_MATCH_DTYPE – x, b, and A do not have the same data type.
- S3L_ERR_MATCH_EXTENTS – The lengths of x and b do not match the size of sparse matrix A. Both must be equal to the order of A.
- S3L_ERR_PARAM_INVALID – Invalid input for iparm or rparm. Both must be preallocated and initialized with the predefined values described in the Description section or set to 0 for the default value.
- S3L_ERR_PC_INVALID – Invalid input for preconditioner. Option S3L_bjacobi is valid only if sparse matrix A is represented under S3L_SPARSE_VBR.

Computational Errors

- S3L_ERR_ILU_ZRPVT – A zero pivot was encountered during ILU preconditioning.
- S3L_ERR_JACOBI_ZRDIAG – A zero pivot was encountered during Jacobi preconditioning.
- S3L_ERR_DIVERGE – Computation has diverged.
- S3L_ERR_ITER_BRKDWN – A breakdown has occurred.
- S3L_ERR_MAXITER – The number of iterations has exceeded the value supplied in iparm[S3L_iter_maxiter].

Examples

/opt/SUNWhpc/examples/s3l/iter/ex_iter.c

/opt/SUNWhpc/examples/s3l/iter-f/ex_iter.f

Related Functions

S3L_declare_sparse(3)

S3L_read_sparse(3)

S3L_rand_sparse(3)

S3L_gen_lsq

Description

If m >= n, S3L_gen_lsq finds the least-squares solution to an overdetermined system. That is, it solves the least-squares problem:

```
minimize || B - A*X ||
```

On output, the first n rows of B hold the least-squares solution X.

If m < n, S3L_gen_lsq finds the minimum norm solution to an underdetermined system:

```
A * X = B(1:m,:)
```

On output, B holds the minimum norm solution X.

Syntax

The C and Fortran syntax for S3L_gen_lsq is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_gen_lsq(A, B, axis1, axis2)
    S3L_array_t         A
    S3L_array_t         B
    int                 axis1
    int                 axis2
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_gen_lsq(A, B, axis1, axis2, ier)
    integer*8           A
    integer*8           B
    integer*4           axis1
    integer*4           axis2
    integer*4           ier
```

Input

`S3L_gen_lsq` accepts the following arguments as input:

- `A` – S3L array handle that describes a parallel array of dimensions m x n. On output, its contents may be destroyed.
- `B` – S3L array handle that describes a parallel array of dimensions max(m,n) x nrhs. On output, its contents may be destroyed.
- `axis1` – If `A` and `B` have more than two dimensions, `axis1` denotes the dimension of `A` with extent m. Otherwise, it has to be 0 for C/C++ programs or 1 for F77/F90 programs.
- `axis2` – If `A` and `B` have more than two dimensions, `axis2` denotes the dimension of `A` with extent n. Otherwise, it has to be 0 for C/C++ programs or 1 for F77/F90 programs.

Output

S3L_gen_lsq uses the following arguments for output:

- B – On output, B is overwritten by the result of the least-squares problem.
- ier (Fortran only) – When called from a Fortran program, S3L_gen_lsq returns error status in ier.

Error Handling

On success, S3L_gen_lsq returns S3L_SUCCESS.

S3L_gen_lsq checks the validity of the array arguments. If an array argument is found to be corrupted or invalid, an error code is returned. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause the function to terminate and return the associated error code.

- S3L_ERR_ARG_AXISNUM – An axis argument is invalid for one of the following reasons:
  - It is less than 0 (C/C++) or less than 1 (F77/F90).
  - It is greater than the rank of the referenced array.
  - axis1 is equal to axis2.
- S3L_ERR_MATCH_DTYPE – The array arguments are not all of the same data type, as required.
- S3L_ERR_MATCH_RANK – Corresponding ranks of the array arguments do not match.
- S3L_ERR_MATCH_EXTENTS – The extents of the arrays are not compatible.
- S3L_ERR_ARG_DTYPE – The array arguments are not float or double, complex, or double-precision complex.

Examples

/opt/SUNWhpc/examples/s3l/lsq/ex_lsq.c

/opt/SUNWhpc/examples/s3l/lsq-f/ex_lsq.f

S3L_gen_svd

Description

S3L_gen_svd computes the singular value of a parallel array A and, optionally, the right singular vector and/or the left singular vector. On exit, S contains the singular values. If requested, U and V contain the left and right singular vectors, respectively.

If A, U, and V are two-dimensional arrays, S3L_gen_svd is more efficient when A, U, and V are allocated on the same process grid and the same block size is used along both axes. When A, U, and V have more than two dimensions, S3L_gen_svd is more efficient when axis_r, axis_c, and axis_s are local (that is, are not distributed).

Syntax

The C and Fortran syntax for S3L_gen_svd is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_gen_svd(A, U, S, V, jobu, jobv, axis_r, axis_c, axis_s)
    S3L_array_t       A
    S3L_array_t       U
    S3L_array_t       S
    S3L_array_t       V
    char              jobu
    char              jobv
    int               axis_r
    int               axis_c
    int               axis_s
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_gen_svd(A, U, S, V, jobu, jobv, axis_r, axis_c, axis_s, ier)
    integer*8          A
    integer*8          U
    integer*8          S
    integer*8          V
    character*1        jobu
    character*1        jobv
    integer*4          axis_r
    integer*4          axis_c
    integer*4          axis_s
    integer*4          ier
```

Input

S3L_gen_svd accepts the following arguments as input:

- A – S3L array handle describing a parallel array of type S3L_double or S3L_float. In the 2D case, A is an m x n array. If A has more than two dimensions, axis_r and axis_c correspond to the axes of A whose extents are m and n, respectively.
- U – If jobu = V, U is a parallel array of dimensions m x min(m,n). Otherwise, U is not referred to. If U has more than two dimensions, axis_r and axis_c correspond to the axes of U whose extents are m and n, respectively. On output, U is overwritten with the left singular vectors.
- S – S3L array handle describing a parallel array (vector) of length min(m,n). If S is multidimensional, axis_s corresponds to the axis of S whose extent is min(m,n).
- V – If jobu = V, this is an S3L array handle describing a parallel array of dimensions min(m,n) x n. Otherwise, V is not referenced. If V has more than two dimensions, axis_r and axis_c correspond to the axes of V whose extents are m and n, respectively. On output, V is overwritten with the right singular vectors.
- jobu – Specifies options for computing all or part of the matrix U, as follows:
  - jobu = V – The first min(m,n) columns of U (the left singular vectors) are returned in the array U.
  - jobu = N – No columns of U (no left singular vectors) are computed.
- jobv – Specifies options for computing all or part of the matrix V, as follows:
  - jobv = V – The first min(m,n) rows of V (the right singular vectors) are returned in the array V.
  - jobv = N – No rows of V (no right singular vectors) are computed.

- axis_r – This is the axis of arrays A, U, and V such that the extent of array A along axis_r is m, the extent of array U along axis_r is m, and the extent of array V along axis_r is min(m,n).
- axis_c – This is the axis of arrays A, U, and V such that the extent of array A along axis_c is n, the extent of array U along axis_c is min(m,n), and the extent of array V along axis_c is n.
- axis_s – This is the axis of array S along which the length is equal to min(m,n).

Output

S3L_gen_svd uses the following arguments for output:

- U – On output, U is overwritten with the left singular vectors.
- S – On output, S is overwritten with the singular values.
- V – On output, V is overwritten with the right singular vectors.
- ier (Fortran only) – When called from a Fortran program, S3L_gen_svd returns error status in ier.

Error Handling

On success, S3L_gen_svd returns S3L_SUCCESS.

S3L_gen_svd performs generic checking of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause the function to terminate and return the associated error code:

- S3L_ERR_ARG_AXISNUM – An axis argument is invalid for one of the following reasons:
  - It is less than 0 (C/C++) or less than 1 (F77/F90).
  - It is greater than the rank of the referenced array.
  - axis_r is equal to axis_c.
- S3L_ERR_MATCH_DTYPE – The array arguments are not all of the same data type, as required.
- S3L_ERR_MATCH_RANK – Corresponding ranks of the array arguments do not match.
- S3L_ERR_MATCH_EXTENTS – The extents of the arrays are not compatible.
- S3L_ERR_ARG_DTYPE – The data types of the array arguments are not float or double.

- S3L_ERR_ARG_OP – jobv is not one of V or N.
- S3L_ERR_SVD_FAIL – The svd algorithm failed to converge.

Examples

/opt/SUNWhpc/examples/s3l/svd/ex_svd.c

/opt/SUNWhpc/examples/s3l/svd-f/ex_svd.f

S3L_gen_trid_factor

Description

S3L_gen_trid_factor factors a tridiagonal matrix, whose diagonal is stored in vector D. The first upper subdiagonal is stored in U, and the first lower subdiagonal in L.

On return, the integer factors contains a pointer to an internal setup structure that holds the factorization. Subsequent calls to S3L_gen_trid_solve use the value in factors to access the factorization results.

The contents of the vectors D, U, and L may be altered. These altered vectors, along with the factors parameter, have to be passed to a subsequent call to S3L_gen_trid_solve to produce the solution to a tridiagonal system.

D, U, and L must have the same extents and type. If they are one-dimensional, all three must be of length n. The first n–1 entries of U contain the elements of the superdiagonal. The last n–1 entries of L contain the elements of the first subdiagonal. The last element of U and the first element of L are not referenced and can be initialized arbitrarily.

If D, U, and L have more than one dimension, axis_d is the axis along which the multidimensional arrays are factored. If they are one-dimensional, axis_d must be 0 in C/C++ programs and 1 in F77/F90 programs.

S3L_gen_trid_factor is based on the ScaLAPACK routines pxdttrf, where x is single, double, complex, or double complex. It does no pivoting; consequently, the matrix has to be positive definite for the factorization to be stable.

For one-dimensional arrays, the routine is more efficient when D, U, and L are block-distributed. For multiple dimensions, the routine is more efficient when axis_d is a local axis.

Syntax

The C and Fortran syntax for S3L_gen_trid_factor is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_gen_trid_factor(D, U, L, factors, axis_d)
    S3L_array_t      D
    S3L_array_t      U
    S3L_array_t      L
    int              *factors
    int              axis_d
```

F77/F90 Syntax

```
include ‘s3l/s3l-f.h’
include ‘s3l/s3l_errno-f.h’
subroutine
S3L_gen_trid_factor(D, U, L, factors, axis_d, ier)
    integer*8        D
    integer*8        U
    integer*8        L
    integer*4        factors
    integer*4        axis_d
    integer*4        ier
```

Input

S3L_gen_trid_factor accepts the following arguments as input:

- D – Vector containing the diagonal for the matrix being factored.
- U – Vector containing the first upper diagonal for the matrix being factored.
- L – Vector containing the first lower diagonal for the matrix being factored.

- axis_d – When D, U, and L are one-dimensional, axis_d must be 0 (C/C++ programs) or 1 (F77/F90 programs). For multidimensional arrays, axis_d specifies the axis along which the arrays are factored.

Output

S3L_gen_trid_factor uses the following arguments for output:

- D – On output, D is overwritten with the partial result of the factorization.
- U – On output, U is overwritten with the partial result of the factorization.
- L – On output, L is overwritten with the partial result of the factorization.
- factors – Upon completion, factors points to the internal data structure containing the factorization results.
- ier (Fortran only) – When called from a Fortran program, S3L_gen_trid_factor returns error status in ier.

Error Handling

On success, S3L_gen_trid_factor returns S3L_SUCCESS.

S3L_gen_trid_factor performs generic checking of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause the function to terminate and return the associated error code:

- S3L_ERR_MATCH_DTYPE – The arrays are not the same data type.
- S3L_ERR_MATCH_RANK – The arrays do not have the same rank.
- S3L_ERR_MATCH_EXTENTS – The arrays do not have the same extents.
- S3L_ERR_ARG_DTYPE – The array type cannot be operated on by the routine because it is either integer or long long.
- S3L_ERR_ARRTOOSMALL – The array extent is too small, making the length of the main diagonal less than two times the number of processes.
- S3L_ERR_ARG_AXISNUM – An axis argument is invalid; that is, it is either:
  - Less than 0 (C/C++) or less than 1 (F77/F90).
  - Greater than the rank of the referenced array.
- S3L_ERR_FACTOR_FAIL – The tridiagonal matrix could not be factored for some reason. For example, it might not be diagonally dominant.

Examples

/opt/SUNWhpc/examples/s3l/trid/ex_trid.c

/opt/SUNWhpc/examples/s3l/trid-f/ex_trid.f

Related Functions

S3L_gen_trid_solve(3)

S3L_gen_trid_free_factors(3)

S3L_gen_trid_free_factors

Description

S3L_gen_trid_free_factors frees the internal memory setup that was reserved by a prior call to S3L_gen_trid_factor. The factors argument contains the value returned by the earlier S3L_gen_trid_factor call.

Syntax

The C and Fortran syntax for S3L_gen_trid_free_factors is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_gen_trid_free_factors(factors)
    int               *factors
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_gen_trid_free_factors(factors, ier)
    integer*4         factors
    integer*4         ier
```

Input

S3L_gen_trid_free_factors accepts the following argument as input:

- factors – Pointer to the internal structure that will be freed.

Output

S3L_gen_trid_free_factors uses the following argument for output:

- ier (Fortran only) – When called from a Fortran program, S3L_gen_trid_free_factors returns error status in ier.

Error Handling

On success, S3L_gen_trid_free_factors returns S3L_SUCCESS.

The following condition will cause S3L_gen_trid_free_factors to terminate and return the associated error code:

- S3L_ERR_ARG_SETUP – The factors value is invalid.

Examples

/opt/SUNWhpc/examples/s3l/trid/ex_trid.c

/opt/SUNWhpc/examples/s3l/trid-f/ex_trid.f

Related Functions

S3L_gen_trid_solve(3)

S3L_gen_trid_factor(3)

S3L_gen_trid_solve

Description

S3L_gen_trid_solve solves a tridiagonal system that has been previously factored through a call to S3L_gen_trid_factor.

If D, U, and L are of length n, B (the right-hand side of the tridiagonal system) must be of size n x nrhs. If D, U, and L are multidimensional, axis_d is the axis along which the system is solved. The rank of B must be one greater than the rank of D, U, and L.

If the rank of B is greater than 2, row_b and col_b specify the axes whose dimensions are n and nrhs, respectively. The extents of all other axes must be the same as the corresponding axes of D, U, and L.

When computing multiple tridiagonal systems in which only the right-hand-side matrix changes, the factorization routine S3L_gen_trid_factor need only be called once, before the first call to S3l_gen_trid_solve. Then, S3L_gen_trid_solve can be called repeatedly without calling S3L_gen_trid_factor again.

Syntax

The C and Fortran syntax for S3L_gen_trid_solve is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_gen_trid_solve(D, U, L, factors, B, axis_d, row_b, col_b)
    S3L_array_t       D
    S3L_array_t       U
    S3L_array_t       L
    int               factors
    S3L_array_t       B
    int               axis_d
    int               row_b
    int               col_b
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_gen_trid_solve(D, U, L, factors, B, axis_d, row_b, col_b, ier)
    integer*8         D
    integer*8         U
    integer*8         L
    integer*4         factors
    integer*8         B
    integer*4         axis_d
    integer*4         row_b
    integer*4         col_b
    integer*4         ier
```

Input

`S3L_gen_trid_solve` accepts the following argument as input:

- `D` – Vector containing the diagonal for the matrix being factored.
- `U` – Vector containing the first upper subdiagonal for the matrix being factored.
- `L` – Vector containing the first lower subdiagonal for the matrix being factored.
- `factors` – Pointer to an internal structure that holds the factorization results.
- `B` – The right-hand side of the tridiagonal system to be solved.
- `axis_d` – When `D`, `U`, and `L` are one-dimensional, `axis_d` must be 0 (C/C++ programs) or 1 (F77/F90 programs). For multidimensional arrays, `axis_d` specifies the axis along which factorization was carried out.

- row_b – Indicates the row axis of the right-hand-side array, B. The value of row_b depends on the following:
  - When B is two-dimensional and its sides are n x nrhs, row_b is 0 (C/C++) or 1 (F77/F90).
  - When B is two-dimensional and its sides are nrhs x n, row_b is 1 (C/C++) or 2 (F77/F90).
  - When B has more than two dimensions, row_b identifies the side of B with an extent of n. For C/C++ programs, the row_b value is zero-based and for F77/F90 programs, it is one-based.
- col_b – Indicates the column axis of the right-hand-side array, B, that has an extent of nrhs. The value of col_b is determined as follows:
  - When B is two-dimensional and its sides are n x nrhs, col_b is 1 (C/C++) or 2 (F77/F90).
  - When B is two-dimensional and its sides are nrhs x n, col_b is 0 (C/C++) or 1 (F77/F90).
  - When B has more than two dimensions, col_b identifies the side of B with an extent of nhrs. For C/C++ programs, the col_b value is zero-based and for F77/F90 programs, it is one-based.

Output

S3L_gen_trid_solve uses the following arguments for output:

- B – On output, B is overwritten with the solution to the tridiagonal system.
- ier (Fortran only) – When called from a Fortran program, S3L_gen_trid_solve returns error status in ier.

Error Handling

On success, S3L_gen_trid_solve returns S3L_SUCCESS.

S3L_gen_trid_solve performs generic checking of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause the function to terminate and return the associated error code.

- S3L_ERR_MATCH_DTYPE – The arrays are not the same data type.
- S3L_ERR_MATCH_RANK – The arrays do not have compatible rank.

- S3L_ERR_MATCH_EXTENTS – The arrays do not have compatible extents.
- S3L_ERR_ARG_DTYPE – The array type cannot be operated on by the routine because it is either integer or long long.
- S3L_ERR_ARRTOOSMALL – The array extent is too small, making the length of the main diagonal less than two times the number of processes.
- S3L_ERR_ARG_AXISNUM – An axis argument is invalidfor one of the following reasons:
  - It is less than 0 (C/C++) or less than 1 (F77/F90).
  - It is greater than the rank of the referenced array.
  - row_b is equal to col_b.
- S3L_ERR_ARG_SETUP – The factors value does not correspond to a valid setup.

Examples

/opt/SUNWhpc/examples/s3l/trid/ex_trid.c

/opt/SUNWhpc/examples/s3l/trid-f/ex_trid.f

Related Functions

S3L_gen_trid_factor(3)

S3L_gen_trid_free_factors(3)

S3L_get_attribute

Description

S3L_get_attribute returns a requested attribute of an S3L dense array or sparse matrix. The user specifies one of a set of predefined req_attr values and, on return, the integer value of the requested attribute is stored in attr. For attributes associated with array axes—such as the extents or blocksizes of an array—the user specifies the axis as well.

The req_attr entry must be one of the following:

- S3L_ELEM_TYPE – Retrieves in attr the S3L type of the elements of an S3L dense array or sparse matrix as they are defined in s3l-c.h or s3l-f.h.

- S3L_ELEM_SIZE – Retrieves in attr the size (in bytes) of the elements of an S3L dense array or sparse matrix.
- S3L_RANK – Retrieves in attr the rank (number of dimensions) of an S3L dense array or sparse matrix.
- S3L_EXTENT – If a is an S3L array handle, S3L_EXTENT retrieves in attr the extent of an S3L dense array or sparse matrix along the dimension given in axis. If a is an S3L process grid handle, it returns in attr the number of processes over which a given axis of an array is distributed.
- S3L_BLOCK_SIZE – Retrieves in attr the block size of the block-cyclic distribution of an S3L dense array along the dimension given in axis.
- S3L_BLOCK_START – Retrieves in attr the index of the starting process of the block-cyclic distribution of an S3L dense array along the dimension given in axis.
- S3L_SGRID_SIZE – Retrieves in attr the subgrid size of the block-cyclic distribution of an S3L dense array along the dimension given in axis.
- S3L_AXIS_LOCAL – Assigns 0 to attr if the axis is not distributed and 1 if it is.
- S3L_SGRID_ADDRESS – Returns in attr the starting address of the local subgrid (local per-process part) of an S3L dense array.
- S3L_MAJOR – If a is an S3L dense array, S3L_MAJOR returns in attr the majorness of the elements in the local part of the array. It can be either S3L_MAJOR_ROW (C major) or S3L_MAJOR_COLUMN (F77 major). If a is an S3L process grid descriptor, it returns in attr the majorness (F77 or C) of the internal process grid associated with an S3L process grid.
- S3L_ALLOC_TYPE – Returns in attr one of the predefined allocation types for dense S3L arrays. The user can use this option to determine, for example, whether the array has been allocated in shared memory, whether the local (per-process) parts of the array are 64-byte aligned, and so forth.
- S3L_SHARED_ADDR – For dense S3L arrays that have been allocated in shared memory (single SMP case), S3L_SHARED_ADDR returns in attr the global starting address of the array. All processes can directly access all elements of such arrays without the need for explicit interprocess communication.
- S3L_PGRID_DESC – Returns in attr the process grid descriptor associated with an S3L dense array or sparse matrix.
- S3L_SCALAPACK_DESC – For 1D and 2D S3L dense arrays, S3L_SCALAPACK_DESC returns in attr the ScaLAPACK array descriptor associated with the distribution of that array.
- S3L_SPARSE_FORMAT – For an S3L sparse matrix, S3L_SPARSE_FORMAT returns in attr the sparse format in which the matrix is stored.
- S3L_NONZEROS – For an S3L sparse matrix, S3L_NONZEROS returns in attr the number of nonzero elements of that matrix.

Note – The following six attribute entries only work for matrices stored under the S3L_SPARSE_COO or S3L_SPARSE_CSR format. The internal distribution schemes for matrices stored under S3L_SPARSE_CSC and S3L_SPARSE_VBR formats may change in the future.

- S3L_RIDX_SGRID_ADDR – For an S3L sparse matrix stored in the S3L_SPARSE_COO format, S3L_RIDX_SGRID_ADDR returns in attr the starting address of an array of index sets containing the local row numbers that comprise each local submatrix (per-process).

  For an S3L sparse matrix stored in the S3L_SPARSE_CSR format, S3L_RIDX_SGRID_ADDR returns in attr the starting address of an array containing the pointers to the beginning of each row of the local submatrix (per-process).

  Note: Users must not change the data returned in attr. It is created for internal use only.

- S3L_CIDX_SGRID_ADDR – For an S3L sparse matrix stored in either the S3L_SPARSE_COO or S3L_SPARSE_CSR format, S3L_CIDX_SGRID_ADDR returns in attr the starting address of an array of index sets containing the global column numbers that comprise each local submatrix (per-process).

  Note: Users must not change the data returned in attr. It is created for internal use only.

- S3L_NZRS_SGRID_ADDR – For an S3L sparse matrix stored in either the S3L_SPARSE_COO or S3L_SPARSE_CSR format, S3L_NZRS_SGRID_ADDR returns in attr the starting address of an array containing nonzero elements of the local submatrix (per-process).
- S3L_RIDX_SGRID_SIZE – For an S3L sparse matrix stored in the S3L_SPARSE_COO format, S3L_RIDX_SGRID_SIZE returns in attr the size of an array of index sets containing the local row numbers that comprise each local submatrix (per-process).

  For an S3L sparse matrix stored in the S3L_SPARSE_CSR format, S3L_RIDX_SGRID_SIZE returns in attr the size of an array containing the pointers to the beginning of each row of the local submatrix (per-process).

- S3L_CIDX_SGRID_SIZE – For an S3L sparse matrix stored in either the S3L_SPARSE_COO or S3L_SPARSE_CSR format, S3L_CIDX_SGRID_SIZE returns in attr the size of an array of index sets containing the global column numbers that comprise each local submatrix (per-process).
- S3L_NZRS_SGRID_SIZE – For an S3L sparse matrix stored in either the S3L_SPARSE_COO or S3L_SPARSE_CSR format, S3L_NZRS_SGRID_SIZE returns in attr the size of an array containing nonzero elements of the local submatrix (per-process).
- S3L_COORD – It returns in attr the coordinate of the calling process in an S3L process grid, along the dimension given in axis.
- S3L_ON_SINGLE_SMP – It returns 1 in attr if an S3L process grid is defined on a single SMP and 0 if not.

Syntax

The C and Fortran syntax for `S3L_get_attribute` is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_get_attribute(a, req_attr, axis, attr)
    S3L_array_t        a
    S3L_attr_type      req_attr
    int                axis
    void               *attr
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_get_attribute(a, req_attr, axis, attr, ier)
    integer*8          a
    integer*4          req_attr
    integer*4          axis
    <type>             attr
    integer*4          ier
```

where `<type>` is either of `integer*4` type or of `pointer` type. When `attr` is an address, it should be of `pointer` type. In all other cases, it should be of `integer*4` type.

Input

`S3L_get_attribute` accepts the following arguments as input:

- `a` – Pointer to a descriptor of an unknown type.
- `req_attr` – A predefined value that specifies the attribute to be retrieved. See the Description section for a list of valid `req_attr` entries.
- `axis` – Scalar integer variable. To retrieve axis-specific attributes, such as extents or blocksizes, use this parameter to specify the axis of interest.

- `attr` – Pointer to a variable of the appropriate type that will hold the retrieved attribute value.

Output

`S3L_get_attribute` uses the following argument for output:

- `ier` (Fortran only) – When called from a Fortran program, `S3L_get_attribute` returns error status in `ier`.

Error Handling

On success, `S3L_get_attribute` returns `S3L_SUCCESS`.

`S3L_get_attribute` performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following condition will cause the function to terminate and return the associated error code:

- `S3L_ERR_ATTR_INVALID` – Invalid attribute; the supplied descriptor does not have the requested attribute type.

Examples

`/opt/SUNWhpc/examples/s3l/utils/get_attribute.c`

`/opt/SUNWhpc/examples/s3l/utils-f/get_attribute.f`

`/opt/SUNWhpc/examples/s3l/sparse/ex_sparse2.c`

Related Functions

`S3L_set_array_element(3)`

`S3L_set_array_element_on_proc(3)`

S3l_get_qr

Description

S3L_get_qr extracts the Q and R arrays from the packed representation of a QR-decomposed S3L array. If A is of size m x n, the array Q should be m x min(m,n) and R should be min(m,n) x n. If either Q or R is zero, it is assumed that the extraction of the corresponding array is not desired. Q and R should not both be zero.

The setup parameter, returned by a previous call to S3L_qr_factor, refers to an internal QR factorization setup.

a, q, and r should all be of the same rank (that is, have the same number of dimensions) and be of the same data type. If a has more than two dimensions, QR factorization will have been performed along the axes axis_r and axis_c (see S3L_qr_factor). These axis numbers are included in the internal QR setup information referred to by the setup parameter.

The dimensions of q and r should have the appropriate lengths along axis_r and axis_c, as described for the 2D case. In addition, all other dimensions should have the same lengths as those of a.

Syntax

The C and Fortran syntax for S3L_get_qr is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_get_qr(a, q, r, setup)
    S3L_array_t         a
    S3L_array_t         q
    S3L_array_t         r
    int                 *setup
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_get_qr(a, q, r, setup, ier)
    integer*8          a
    integer*8          q
    integer*8          r
    integer*4          setup
    integer*4          ier
```

Input

S3L_get_qr accepts the following arguments as input:

- a – Input array containing a QR decomposition computed by S3L_qr_factor.
- q – Input array of size m x min(m,n). Also used for output, as described below.
- r – Input array of size m x min(m,n) x n. Also used for output, as described below.
- setup – Integer returned by a previous call to S3L_qr_factor.

Output

S3L_get_qr uses the following arguments for output:

- q – On exit, q contains the orthonormal array produced by the QR decomposition.
- r – On exit, r contains an upper triangular array.
- ier (Fortran only) – When called from a Fortran program, S3L_get_qr returns error status in ier.

Error Handling

On success, S3L_get_qr returns S3L_SUCCESS.

S3L_get_qr performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and returns an error code indicating which value was invalid. See Appendix A of this manual for a detailed list of these error codes.

The following conditions will cause the function to terminate and return the associated error code:

- S3L_ERR_ARG_RANK – Invalid rank. The rank of a is 1.
- S3L_ERR_ARG_DTYPE – The data type of a is not S3L_float, S3L_double, S3L_complex, or S3L_double_complex.
- S3L_ERR_ARG_EXTENTS – The extents of a, q, and r are not compatible.
- S3L_ERR_ARG_SETUP – Invalid setup_id value.

Examples

/opt/SUNWhpc/examples/s3l/qr/ex_qr1.c

/opt/SUNWhpc/examples/s3l/qr-f/ex_qr1.f

Related Functions

S3L_qr_factor(3)

S3L_qr_solve(3)

S3L_qr_free(3)

S3L_get_safety

Description

When S3L_get_safety is called from within an application, the value it returns indicates the current setting of the S3L safety mechanism. The possible return values are listed and their meaning explained in TABLE 2-6.

TABLE 2-6 S3L Safety-Level Return Values

Safety Level	Description
0	The safety mechanism is off.
2	This level detects potential race conditions in multithreaded S3L operations on parallel arrays. To avoid race conditions, an S3L function locks all parallel array handles in its argument list before proceeding. This safety level causes warning messages to be generated if more than one S3L function attempts to use the same parallel array at the same time.
5	In addition to checking for and reporting level 2 errors, level 5 performs explicit synchronization before and after each call and locates each error with respect to the synchronization points. This safety level is appropriate during program development or during runs for which a small performance penalty can be tolerated.
9	This level checks for and reports all level 2 and level 5 errors, as well as errors generated by any lower levels of code called from within S3L. Level 9 performs explicit synchronization in these lower levels of code and locates each error with respect to the synchronization points. This level is appropriate for detailed debugging following the occurrence of a problem.

Syntax

The C and Fortran syntax for S3L_get_safety is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_get_safety()
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_get_safety(ier)
    integer*4          ier
```

Input

`S3L_get_safety` takes no input arguments.

Output

`S3L_get_safety` returns the S3L safety level. When called by a Fortran program, it uses the following argument for output:

- `ier` – When called from a Fortran program, `S3L_get_safety` returns error status in `ier`.

Error Handling

On success, `S3L_get_safety` returns `S3L_SUCCESS`.

Examples

`/opt/SUNWhpc/examples/s3l/utils/copy_array.c`

`/opt/SUNWhpc/examples/s3l/utils-f/copy_array.f`

Related Function

S3L_set_safety(3)

S3L_grade_down, S3L_grade_up, S3L_grade_detailed_down, S3L_grade_detailed_up

Description

The S3L_grade family of functions computes the grade of the elements of a parallel array A. Grading is done in either descending or ascending order and is done either across the whole array or along a specified axis. The graded elements are stored in array G, using zero-based indexing when called from a C or C++ program and one-based indexing when called from an F77 or F90 program.

S3L_grade_down and S3L_grade_up

These two functions grade the elements across the entire array A and store the indices of the elements in descending or ascending order (S3L_grade_down or S3L_grade_up, respectively).

If A is an array of rank n and the product of its extents is l, G is a two-dimensional array whose extents are n x l.

Upon return of the function, every j-th column of array G is set to the indices of the j-th smallest (S3L_grade_down) or largest (S3L_grade_up) element of array A.

For example, if A is the 3 x 3 array:

```
| 6   2   4 |
|           |
| 1   3   8 |
|           |
| 9   7   5 |
```

and S3L_grade_down is called from a C program, it will store the following values in G:

```
| 2  1  2  0  2  0  1  0  1 |
|                           |
| 0  2  1  0  2  2  1  1  0 |
```

For the same array A, S3L_grade_up would store the following values in G (again, using zero-based indexing).

```
| 1  0  1  0  2  0  2  1  2 |
|                           |
| 0  1  1  2  2  0  1  2  0 |
```

When called by a Fortran program (F77/F90) each value in G would be one greater. For example, S3L_grade_up would store the following set of values.

```
| 2  1  2  1  3  1  3  2  3 |
|                           |
| 1  2  2  3  3  1  2  3  1 |
```

S3L_grade_detailed_down and S3L_grade_detailed_up

The S3L_grade_detailed_down and S3L_grade_detailed_up functions differ from S3L_grade_down and S3L_grade_up in two respects:

- Both grade along a single axis of A, as specified by the axis argument.
- Both store a set of indices, but these indices do not indicate element positions directly. Instead, each stored index indicates the index of the corresponding element of A that has either:)
  - The j-th smallest value along the specified axis (for S3L_grade_detailed_down)
  - The j-th largest value along the specified axis (for S3L_grade_detailed_up)

This means G is an integer array whose rank and extents are the same as those of A.

Repeating the 3 x 3 sample array shown above:

```
| 6   2   4 |
|           |
| 1   3   8 |
|           |
| 9   7   5 |
```

if S3_grade_detailed_down is called from a C program with the axis argument = 0, upon completion, G will contain the following values:

```
| 1   2   2 |
|           |
| 2   1   0 |
|           |
| 0   0   1 |
```

If, instead, axis = 1, G will contain:

```
| 0   2   1 |
|           |
| 2   1   0 |
|           |
| 0   1   2 |
```

If S3L_grade_detailed_up is called from a C program with axis = 0, G will contain:

```
| 1   0   0 |
|           |
| 0   1   2 |
|           |
| 2   2   1 |
```

If S3L_grade_detailed_up is called from a C program with axis = 1, G will contain:

```
| 2   0   1 |
|           |
| 0   1   2 |
|           |
| 2   1   0 |
```

For F77 or F90 calls, each index value in these examples, including the axis argument, would be increased by 1.

Syntax

The C and Fortran syntax for these functions is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_grade_up(A, grade)
S3L_grade_down(A, grade)
S3L_grade_up_detailed(A, grade, axis)
S3L_grade_down_detailed(A, grade, axis)
    S3L_array_t       A
    S3L_array_t       grade
    int               axis
```

F77/F90 Syntax

```
include ‘s3l/s3l-f.h’
include ‘s3l/s3l_errno-f.h’
subroutine
S3L_grade_up(A, grade, ier)
S3L_grade_down(A, grade, ier)
S3L_grade_up_detailed(A, grade, axis, ier)
S3L_grade_down_detailed(A, grade, axis, ier)
    integer*8         A
    integer*8         grade
    integer*4         axis
    integer*4         ier
```

Input

The `S3L_grade_` functions accept the following arguments as input:

- `A` – S3L internal array handle for the array to be graded. Its type can be real, double, integer, or long integer.
- `axis` – The axis along which `S3L_grade_detailed_down` or `S3L_grade_detailed_up` is to be computed. It may not be used in `S3L_grade_down` or `S3L_grade_up` calls.

Output

The S3L_grade_ functions use the following arguments for output:

- grade – S3L internal array handle for an integer array. Upon successful completion, grade contains the indices of the order of the elements.
- ier (Fortran only) – When called from a Fortran program, thes functions return error status in ier.

Error Handling

On success, these functions return S3L_SUCCESS.

These functions perform generic checking of the arrays they accept as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following condition will cause the functions to terminate and return the associated code:

- S3L_ERR_ARG_AXISNUM – The axis argument has an invalid value. The correct values for axis are
  - 0 <= axis < rank of a (C/C++)
  - 0 < axis <= rank of a (F77/F90)

Examples

/opt/SUNWhpc/examples/s3l/grade/ex_grade.c

/opt/SUNWhpc/examples/s3l/grade-f/ex_grade.f

Related Functions

S3L_sort(3)

S3L_sort_detailed_up(3)

S3L_sort_detailed_down(3)

S3L_ifft

Description

Run S3L_ifft to compute the inverse FFT of the complex or double-complex parallel array a. Use the setup ID returned by S3L_fft_setup to specify the array of interest.

Both power-of-two and arbitrary radix FFT are supported. The 1D parallel FFT can be used for sizes that are a multiple of the square of the number of nodes; the 2D and 3D FFTs can be used for arbitrary sizes and distributions.

Upon completion, a is overwritten with the result. The floating-point precision of the result always matches that of the input.

For the 2D FFT, if the blocksizes along each dimension are equal to the extents divided by the number of processes, a more efficient transpose algorithm is employed, which yields significant performance improvements.

S3L_ifft can only be used for complex and double-complex data types. To compute a real-data forward FFT, use S3L_rc_fft. This performs a forward FFT on the real data, yielding packed representation of the complex results. To compute the corresponding inverse FFT, use S3L_cr_fft, which performs an inverse FFT on the complex data, overwriting the original real array with real-valued results of the inverse FFT.

Note – S3L_fft and S3L_ifft do not perform any scaling. Consequently, when a forward FFT is followed by an inverse FFT, the original data will be scaled by the product of the extents of the array.

Syntax

The C and Fortran syntax for S3L_ifft is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_ifft(a, setup_id)
    S3L_array_t       a
    int               setup_id
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_ifft(a, setup_id, ier)
    integer*8         a
    integer*4         setup_id
    integer*4         ier
```

Input

S3L_ifft accepts the following arguments as input:

- a – S3L array handle for a parallel array that will be transformed. Its rank, extents, and type must be the same as the parallel array a supplied in the S3L_fft_setup call.
- setup_id – Scalar integer variable. Use the value returned by the S3L_fft_setup call for this argument.

Output

S3L_ifft uses the following arguments for output:

- a – The input array a is overwritten with the result of the FFT.
- ier (Fortran only) – When called from a Fortran program, S3L_ifft returns error status in ier.

Error Handling

On success, S3L_ifft returns S3L_SUCCESS.

S3L_ifft performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and returns an error code indicating which value was invalid. See Appendix A of this manual for a detailed list of these error codes.

The following conditions will cause the function to terminate and return the associated error code:

- S3L_ERR_FFT_RANKGT3 – The rank of the array a is larger than 3.
- S3L_ERR_ARG_NCOMPLEX – Array a is not complex.
- S3L_ERR_FFT_EXTSQPROCS – Array a is 1D, but its extent is not divisible by the square of the number of processes.
- S3L_ERR_ARG_SETUP – Invalid setup_id value.

Examples

/opt/SUNWhpc/examples/s3l/fft/fft.c

/opt/SUNWhpc/examples/s3l/fft-f/fft.f

Related Functions

S3L_fft_setup(3)

S3L_fft_free_setup(3)

S3L_fft_detailed(3)

S3L_init

Description

Before an application can start using Sun S3L functions, every process involved in the application must call S3L_init to initialize the S3L environment. S3L_init initializes the BLACS environment as well.

S3L_init tests the MPI library to verify that it is Sun MPI. If not, it returns an error and terminates. See the Error Handling section for details.

If the MPI layer is Sun MPI, S3L_init proceeds to initialize the S3L environment, the BLACS environment, and if not already initialized, the Sun MPI environment. It also enables the Prism library to access Sun S3L operations.

If S3L_init calls MPI_Init internally, subsequent use of S3L_exit will also result in an internal call to MPI_Finalize.

Syntax

The C and Fortran syntax for S3L_init is as follows.

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_init()
```

F77/F90 Syntax

```
include ‘s3l/s3l-f.h’
include ‘s3l/s3l_errno-f.h’
subroutine
S3L_init(ier)
    integer*4          ier
```

Input

S3L_init takes no input arguments.

Output

When called from a Fortran program, S3L_init returns error status in ier.

Error Handling

On successful completion, S3L_init returns S3L_SUCCESS.

S3L_init tests to see if the MPI library is Sun MPI. If not, it returns the following error message and terminates:

```
S3L error: invalid MPI. Please use Sun HPC MPI.
```

Examples

/opt/SUNWhpc/examples/s3l/utils/copy_array.c

/opt/SUNWhpc/examples/s3l/utils/copy_array.f

Related Function

S3L_exit(3)

S3L_inner_prod and S3_gbl_inner_prod

Description

Multiple-Instance Inner Product – Sun S3L provides six multiple-instance inner-product routines, all of which compute one or more instances of the inner product of two vectors embedded in two parallel arrays. The operations performed by the multiple-instance inner-product routines are shown in TABLE 2-7.

TABLE 2-7 S3L Multiple-Instance Inner-Product Operations

Routine	Operation	Data Type
S3L_inner_prod	$z = z + x^T y$	Real or complex
S3L_inner_prod_noadd	$z = x^T y$	Real or complex
S3L_inner_prod_addto	$z = u + x^T y$	Real or complex

TABLE 2-7 S3L Multiple-Instance Inner-Product Operations *(Continued)*

Routine	Operation	Data Type
S3L_inner_prod_c1	$z = z + x^H y$	Complex only
S3L_inner_prod_c1_noadd	$z = x^H y$	Complex only
S3L_inner_prod_c1_addto	$z = u + x^H y$	Complex only

For these multiple-instance operations, array x contains one or more instances of the first vector in each inner-product pair x. Likewise, array y contains one or more instances of the second vector in each pair y.

Note – The array arguments x, y, and so forth, actually represent array handles that describe S3L parallel arrays. For convenience, however, this discussion ignores that distinction and refers to them as if they were the arrays themselves.

---

x and y must be at least rank 1 arrays, must be of the same rank, and their corresponding axes must have the same extents. Additionally, x and y must both be distributed arrays—that is, each must have at least one axis that is nonlocal.

Array z, which stores the results of the multiple-instance inner-product operations, must be of rank one less than that of x and y. Its axes must match the instance axes of x and y in length and order of declaration and array z must also have at least one axis that is nonlocal. This means each vector pair in x and y corresponds to a single destination value in z.

For S3L_inner_prod and S3L_inner_prod_c1, z is also used as the source for a set of values, which are added to the inner products of the corresponding x and y vector pairs.

Finally, x, y, and z must match in data type and precision.

Two scalar integer variables, x_vector_axis and y_vector_axis, specify the axes of x and y along which the constituent vectors in each vector pair lie.

Note – When specifying values for x_vector_axis and y_vector_axis, keep in mind that Sun S3L functions employ zero-based array indexing when they are called via the C/C++ interface and one-based indexing when called by means of the F77/F90 interface.

The array handle u describes an S3L parallel array that is used by S3L_inner_prod_addto and S3L_inner_prod_c1_addto. These routines add the values contained in u to the inner products of the corresponding x and y vector pairs.

Upon successful completion of `S3L_inner_prod` or `S3L_inner_prod_c1`, the inner product of each vector pair `x` and `y` in `x` and `y`, respectively, is added to the corresponding value in `z`.

Upon successful completion of `S3L_inner_prod_noadd` or `S3L_inner_prod_c1_noadd`, the inner product of each vector pair `x` and `y` in `x` and `y`, respectively, overwrites the corresponding value in `z`.

Upon successful completion of `S3L_inner_prod_addto` or `S3L_inner_prod_c1_addto`, the inner product of each vector pair `x` and `y` in `x` and `y`, respectively, is added to the corresponding value in `u`, and each resulting sum overwrites the corresponding value in `z`.

Note – If each of the instance axes of `x` and `y`—that is, the axes along which the inner product will be taken—contains only a single vector, either declare the axes to have an extent of 1 or use the comparable single-instance inner-product routine, as described below.

Single-Instance Inner Product – Sun S3L also provides six single-instance inner-product routines, all of which compute the inner product over all the axes of two parallel arrays. The operations performed by the single-instance inner-product routines are shown in TABLE 2-8.

TABLE 2-8 S3L Single-Instance Inner-Product Operations

Routine	Operation	Data Type
`S3L_gbl_inner_prod`	$a = a + x^T y$	Real or complex
`S3L_gbl_inner_prod_noadd`	$a = x^T y$	Real or complex
`S3L_gbl_inner_prod_addto`	$a = b + x^T y$	Real or complex
`S3L_gbl_inner_prod_c1`	$a = a + x^H y$	Complex only
`S3L_gbl_inner_prod_c1_noadd`	$a = x^H y$	Complex only
`S3L_gbl_inner_prod_c1_addto`	$a = b + x^H y$	Complex only

Note – In these descriptions, x^T and x^H denote `x` transpose and `x` Hermitian, respectively.

For these single-instance functions, `x` and `y` are S3L parallel arrays of rank 1 or greater and with the same data type and precision.

`a` is a pointer to a scalar variable of the same data type as `x` and `y`. This variable stores the results of the single-instance inner-product operations.

For S3L_gbl_inner_prod and S3L_gbl_inner_prod_c1, a is also used as the source for a set of values, which are added to the inner product of x and y.

b is also a pointer to a scalar variable of the same data type as x and y. It contains a set of values that S3L_gbl_inner_prod_addto and S3L_gbl_inner_prod_c1_addto add to the inner product of x and y.

Upon successful completion of S3L_gbl_inner_prod or S3L_gbl_inner_prod_c1, the global inner product of x and y is added to a.

Upon successful completion of S3L_gbl_inner_prod_noadd or S3L_gbl_inner_prod_c1_noadd, the global inner product of x and y overwrites a.

Upon successful completion of S3L_gbl_inner_prod_addto or S3L_gbl_inner_prod_c1_addto, the global inner product of x and y is added to b, and the resulting sum overwrites a.

Note – Array variables must not overlap.

Syntax

The C and Fortran syntax for S3L_inner_prod and S3L_gbl_inner_prod is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_inner_prod(z, x, y, x_vector_axis, y_vector_axis)
S3L_inner_prod_noadd(z, x, y, x_vector_axis, y_vector_axis)
S3L_inner_prod_addto(z, x, y, u, x_vector_axis, y_vector_axis)
S3L_inner_prod_c1(z, x, y, x_vector_axis, y_vector_axis)
S3L_inner_prod_c1_noadd(z, x, y, x_vector_axis, y_vector_axis)
S3L_inner_prod_c1_addto(z, x, y, u, x_vector_axis, y_vector_axis)
S3L_gbl_inner_prod(a, x, y)
S3L_gbl_inner_prod_noadd(a, x, y)
S3L_gbl_inner_prod_addto(a, x, y, b)
S3L_gbl_inner_prod_c1(a, x, y)
S3L_gbl_inner_prod_c1_noadd(a, x, y)
S3L_gbl_inner_prod_c1_addto(a, x, y, b)
    S3L_array_t         z
    S3L_array_t         x
    S3L_array_t         y
    S3L_array_t         u
    S3L_array_t         a
    S3L_array_t         b
    int                 x_vector_axis
    int                 y_vector_axis
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_inner_prod(z, x, y, x_vector_axis, y_vector_axis, ier)
S3L_inner_prod_noadd(z, x, y, x_vector_axis, y_vector_axis, ier)
S3L_inner_prod_addto(z, x, y, u, x_vector_axis, y_vector_axis,
ier)
S3L_inner_prod_c1(z, x, y, x_vector_axis, y_vector_axis, ier)
S3L_inner_prod_c1_noadd(z, x, y, x_vector_axis, y_vector_axis,
ier)
S3L_inner_prod_c1_addto(z, x, y, u, x_vector_axis,
y_vector_axis, ier)
S3L_gbl_inner_prod(a, x, y, ier)
S3L_gbl_inner_prod_noadd(a, x, y, ier)
S3L_gbl_inner_prod_addto(a, x, y, b, ier)
S3L_gbl_inner_prod_c1(a, x, y, ier)
S3L_gbl_inner_prod_c1_noadd(a, x, y, ier)
S3L_gbl_inner_prod_c1_addto(a, x, y, b, ier)
```

```
integer*8          z
integer*8          x
integer*8          y
integer*8          u
integer*8          a
integer*8          b
integer*4          x_vector_axis
integer*4          y_vector_axis
integer*4          ier
```

Input

The S3L_inner_prod_ functions accept the following arguments as input:

- z – Array handle for an S3L parallel array, which S3L_inner_prod and S3L_inner_prod_c1 use as a source of values to be added to the inner products of the corresponding x and y vector pairs. z is also used for output; see the Output section for details.
- x – Array handle for an S3L parallel array that contains the first vector in each vector pair for which an inner product will be computed.
- y – Array handle for an S3L parallel array that contains the second vector in each vector pair for which an inner product will be computed.
- u – Array handle for an S3L parallel array whose rank is one less than that of x and y. S3L_inner_prod_addto and S3L_inner_prod_c1_addto add the contents of u to the inner products of the corresponding vector pairs of x and y.
- a – Pointer to a scalar variable, which S3L_gbl_inner_prod and S3L_gbl_inner_prod_c1 use as a source of values to be added to the inner product of x and y. a is also used for output; see the Output section for details.
- b – Pointer to a scalar variable, which S3L_gbl_inner_prod_addto and S3L_gbl_inner_prod_c1_addto use as a source of values to be added to the inner product of x and y.
- x_vector_axis – Scalar variable. Identifies the axis of x along which the vectors lie.
- y_vector_axis – Scalar variable. Identifies the axis of y along which the vectors lie.

Output

The S3L_inner_prod_ functions use the following arguments for output:

- z – Array handle for the S3L parallel array that will contain the results of the multiple-instance 2-norm routine.
- a – Pointer to a scalar variable, which is the destination for the single-instance inner-product routines.

- ier (Fortran only) – When called from a Fortran program, these functions return error status in ier.

Error Handling

On success, S3L_inner_prod and S3L_gbl_inner_prod return S3L_SUCCESS.

S3L_inner_prod and S3L_gbl_inner_prod perform generic checking of the validity of the arrays they accept as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause the function to terminate and return the associated error code:

- S3L_ERR_MATCH_RANK – x and y do not have the same rank.
- S3L_ERR_MATCH_EXTENTS – Axes of x and y do not have the same extents.
- S3L_ERR_MATCH_DTYPE – The arguments are not all of the same data type and precision.
- S3L_ERR_CONJ_INVAL – Conjugation was requested, but data supplied was not of type S3L_complex or S3L_double_complex.

Examples

/opt/SUNWhpc/examples/s3l/dense_matrix_ops/inner_prod.c

/opt/SUNWhpc/examples/s3l/dense_matrix_ops-f/inner_prod.f

Related Functions

S3L_2_norm(3)

S3L_outer_prod(3)

S3L_mat_vec_mult(3)

S3L_mat_mult(3)

S3L_lp_sparse

Description

S3L_lp_sparse applies an interior point method to solve the following linear/quadratic optimization problem:

```
min c'*x
```

subject to:

```
ub >= x(iub) >= 0
A*x = b
```

The arrays must be either single- or double-precision real (S3L_float or S3L_double).

iub is an integer array containing indices of the upper bounded variables. A is a sparse S3L array, while all other arrays are dense.

If convergence is achieved, the result of the optimization will be returned in x.

Syntax

The C and Fortran syntax for S3L_lp_sparse is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_lp_sparse(c, A, b, x, ub, iub, iter, tol, attrib)
    S3L_array_t        c
    S3L_array_t        A
    S3L_array_t        b
    S3L_array_t        x
    S3L_array_t        ub
    S3L_array_t        iub
    int                *iter
    <type>             *tol
    S3L_qp_attr_t      attrib
```

where `<type>` is either float or double.

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_lp_sparse(c, A, b, x, ub, iub, iter, tol, attrib, ier)
    integer*8          c
    integer*8          A
    integer*8          b
    integer*8          x
    integer*8          ub
    integer*8          iub
    integer*4          iter
    <type>             tol
    integer*8          attrib
    integer*4          ier
```

where `<type>` is either `real*4` or `real*8`.

Input

`S3L_lp_sparse` accepts the following arguments as input:

- `c` – S3L vector of length n.
- `A` – S3L sparse array of dimensions ne x n.

- b – Dense S3L vector of length ne.
- ub – Dense S3L vector of length nu.
- iub – Dense integer S3L vector of length nu. It contains the indices of the upper bounded variable x.
- iter – On entry, iter specifies the maximum number of iterations. Also used for output, as described below.
- tol – On entry, tol specifies the level of tolerance to be achieved in the linear complementarity gap for the problem to be considered solved. Also used for output, as described below.
- attrib – Attribute handle supplied by S3L_qr_attr_init.

Output

S3L_lp_sparse uses the following arguments for output:

- x – Dense S3L vector of length n. On exit, x contains the solution to the optimization problem.
- iter – On exit, iter contains the actual number of iterations performed.
- tol – On exit, tol contains the actual level of tolerance achieved.
- ier (Fortran only) – When called from a Fortran program, S3L_lp_sparse returns error status in ier.

Error Handling

On success, S3L_lp_sparse returns S3L_SUCCESS.

S3L_lp_sparse performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause S3L_lp_sparse to terminate and return the associated error code:

- S3L_ERR_NOTSUPPORT – c and/or A are sparse, but sparse support has not been enabled via the attrib argument. For example, the Sun Performance Library™ direct solver has been chosen, but an appropriate version of that library has not been linked in.
- S3L_ERR_ARG_DTYPE – The data type of the supplied arrays is not S3L_double or S3L_float.

The following error codes indicate that the interior point algorithm failed to converge. This can happen if the problem is infeasible or is very badly conditioned. In such cases, `S3L_lp_sparse` will return in `x` the best solution achieved up to that point. This allows the user to post-process the results and decide whether or not to accept them.

- `S3L_ERR_SOLVE_ERR_TOO_LARGE` – During each iteration, `S3L_lp_sparse` solves a sparse Cholesky linear system and then verifies the solution by computing the error. If the error is too large, this error code is returned.
- `S3L_ERR_PROBLEM_SINGULAR` – This error code is returned if the linear system to be solved is found to be singular.
- `S3L_ERR_OBJ_ERR_TOO_LARGE` – The error in the objective function that is to be minimized is more than 100 times greater than the initial error.
- `S3L_ERR_FEASIBLE_REGION` – During each iteration, `S3L_lp_sparse` attempts to modify the values of the parameters in such a way that the solution stays in the feasible region. If it cannot move the solution into the feasible region, it returns this error code.

Examples

`/opt/SUNWhpc/examples/s3l/optim/ex_lp1.c`

`/opt/SUNWhpc/examples/s3l/optim/ex_qp1.c`

`/opt/SUNWhpc/examples/s3l/optim/ex_lp_sparse1.c`

`/opt/SUNWhpc/examples/s3l/optim/ex_qp_sparse1.c`

Related Functions

`S3L_qp(3)`

`S3L_qp_attr_init(3)`

`S3L_qp_attr_destroy(3)`

`S3L_qp_attr_set(3)`

S3l_lu_deallocate

Description

S3L_lu_deallocate invalidates the specified setup ID, which deallocates the memory that has been set aside for the S3L_lu_factor routine associated with that ID. Attempts to use a deallocated setup ID will result in errors.

When you finish working with a set of factors, be sure to use S3L_lu_deallocate to free the associated memory. Repeated calls to S3L_lu_factor without deallocation can cause you to run out of memory.

Syntax

The C and Fortran syntax for S3L_lu_deallocate is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_lu_deallocate(setup_id)
    int                *setup_id
```

F77/F90 Syntax

```
include ‘s3l/s3l-f.h’
include ‘s3l/s3l_errno-f.h’
subroutine
S3L_lu_deallocate(setup_id, ier)
    integer*4          setup_id
    integer*4          ier
```

Input

S3L_lu_deallocate accepts the following argument as input:

- setup_id – Scalar integer variable. Use the value returned by the corresponding S3L_lu_factor call for this argument.

Output

S3L_lu_deallocate uses the following argument for output:

- ier (Fortran only) – When called from a Fortran program, S3L_lu_deallocate returns error status in ier.

Error Handling

On success, S3L_lu_deallocate returns S3L_SUCCESS.

The following condition will cause the function to terminate and return the associated error code:

- S3L_ERR_ARG_SETUP – Invalid setup_id value.

Examples

/opt/SUNWhpc/examples/s3l/lu/lu.c

/opt/SUNWhpc/examples/s3l/lu/ex_lu1.c

/opt/SUNWhpc/examples/s3l/lu/ex_lu2.c

/opt/SUNWhpc/examples/s3l/lu-f/lu.f

/opt/SUNWhpc/examples/s3l/lu-f/ex_lu1.f

Related Functions

S3L_lu_factor(3)

S3L_lu_solve(3)

S3L_lu_invert(3)

S3l_lu_factor

Description

For each M x N coefficient matrix A of a, S3L_lu_factor computes the LU factorization using partial pivoting with row interchanges.

The factorization has the form A = P x L x U, where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if M > N), and U is upper triangular (upper trapezoidal if M < N). L and U are stored in A.

In general, S3L_lu_factor performs most efficiently when the array is distributed using the same block size along each axis.

S3L_lu_factor behaves somewhat differently for 3D arrays, however. In this case, it applies nodal LU factorization to each M x N coefficient matrix across the instance axis. This factorization is performed concurrently on all participating processes.

You must call S3L_lu_factor before calling any of the other LU routines. The S3L_lu_factor routine performs on the preallocated parallel array and returns a setup ID. You must supply this setup ID in subsequent LU calls, as long as you are working with the same set of factors.

Be sure to call S3L_lu_deallocate when you have finished working with a set of LU factors. See "S3l_lu_deallocate" on page 197 for details.

The internal variable setup_id is required for communicating information between the factorization routine and the other LU routines. The application must not modify the contents of this variable.

Syntax

The C and Fortran syntax for S3L_lu_factor is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_lu_factor(a, row_axis, col_axis, setup_id)
    S3L_array_t         a
    int                 row_axis
    int                 col_axis
    int                 *setup_id
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_lu_factor(a, row_axis, col_, setup_id, ier)
    integer*8           a
    integer*4           row_axis
    integer*4           col_axis
    integer*4           setup_id
    integer*4           ier
```

Input

`S3L_lu_factor` accepts the following arguments as input:

- `a` – Parallel array of rank greater than or equal to 2. This array contains one or more instances of a coefficient matrix `A` to be factored. Each `A` is assumed to be dense with dimensions M x N with rows counted by axis `row_axis` and columns counted by axis `col_axis`.
- `row_axis` – Scalar integer variable. Identifies the axis of `a` that counts the rows of each matrix `A`. For C program calls, `row_axis` must be >= 0 and less than the rank of `a`; for Fortran program calls, it must be >= 1 and not exceed the rank of `a`. In addition, `row_axis` and `col_axis` must not be equal.
- `col_axis` – Scalar integer variable. Identifies the axis of a that counts the columns of each matrix `A`. For C program calls, `col_axis` must be >= 0 and less than the rank of `a`; for Fortran program calls, it must be >= 1 and not exceed the rank of `a`. In addition, `row_axis` and `col_axis` must not be equal.

Output

S3L_lu_factor uses the following arguments for output:

- a – Upon successful completion, each matrix instance A is overwritten with data giving the corresponding LU factors.
- setup_id – Scalar integer variable returned by S3L_lu_factor. It can be used when calling other LU routines to reference the LU-factored array.
- ier (Fortran only) – When called from a Fortran program, S3L_lu_factor returns error status in ier.

Error Handling

On success, S3L_lu_factor returns S3L_SUCCESS.

S3L_lu_factor performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and returns an error code indicating which value was invalid. See Appendix A of this manual for a detailed list of these error codes.

The following conditions will cause the function to terminate and return the associated error code:

- S3L_ERR_ARG_RANK – Invalid rank; must be >= 2.
- S3L_ERR_ARG_BLKSIZE – Invalid blocksize; must be >= 1.
- S3L_ERR_ARG_DTYPE – Invalid data type. It must be real or complex (single- or double-precision).
- S3L_ERR_ARG_NULL – Invalid array. a must be preallocated.
- S3L_ERR_ARG_AXISNUM – row_axis or col_axis is invalid. This condition can be caused by either an out-of-range axis number (see row_axis and col_axis argument definitions) or row_axis equal to col_axis.
- S3L_ERR_FACTOR_SING – A singular factor U is returned. If it is used by S3L_lu_solve, division by zero will occur.

Examples

/opt/SUNWhpc/examples/s3l/lu/lu.c

/opt/SUNWhpc/examples/s3l/lu/ex_lu1.c

/opt/SUNWhpc/examples/s3l/lu/ex_lu2.c

/opt/SUNWhpc/examples/s3l/lu-f/lu.f

/opt/SUNWhpc/examples/s3l/lu-f/ex_lu1.f

Related Functions

S3L_lu_deallocate(3)

S3L_lu_invert(3)

S3L_lu_solve(3)

S3l_lu_invert

Description

S3L_lu_invert uses the LU factorization generated by S3L_lu_factor to compute the inverse of each square (M x M) matrix instance A of the parallel array a. This is done by inverting U and then solving the system $A^{-1}L = U^{-1}$ for A^{-1}, where A^{-1} and U^{-1} denote the inverse of A and U, respectively.

In general, S3L_lu_invert performs most efficiently when the array is distributed using the same block size along each axis.

For arrays with rank > 2, the nodal inversion is applied on each of the 2D slices of a across the instance axis and is performed concurrently on all participating processes.

The internal variable setup_id is required for communicating information between the factorization routine and the other LU routines. The application must not modify the contents of this variable.

Syntax

The C and Fortran syntax for S3L_lu_invert is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_lu_invert(a, setup_id)
    S3L_array_t         a
    int                 *setup_id
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_lu_invert(a, setup_id, ier)
    integer*8           a
    integer*4           setup_id
    integer*4           ier
```

Input

`S3L_lu_invert` accepts the following arguments as input:

- `a` – Parallel array that was factored by `S3L_lu_factor`, where each matrix instance `A` is a dense M x M square matrix. Supply the same value `a` that was used in `S3L_lu_factor`.
- `setup_id` – Scalar integer variable. Use the value returned by the corresponding `S3L_lu_factor` call for this argument.

Output

`S3L_lu_invert` uses the following arguments for output:

- `a` – Upon successful completion, each matrix instance `A` is overwritten with data giving the corresponding LU factors.
- `setup_id` – Scalar integer variable returned by `S3L_lu_factor`. It can be used when calling other LU routines to reference the LU-factored array.
- `ier` (Fortran only) – When called from a Fortran program, `S3L_lu_invert` returns error status in `ier`.

Error Handling

On success, S3L_lu_invert returns S3L_SUCCESS.

S3L_lu_invert performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and returns an error code indicating which value was invalid. See Appendix A of this manual for a detailed list of these error codes.

The following conditions will cause the function to terminate and return the associated error code:

- S3L_ERR_ARG_NULL – Invalid array; must be the same value returned by S3L_lu_factor.
- S3L_ERR_ARG_SETUP – Invalid setup_id value.
- S3L_ERR_FACTOR_SING – a contains singular factors; its inverse could not be computed.

Examples

/opt/SUNWhpc/examples/s3l/lu/lu.c

/opt/SUNWhpc/examples/s3l/lu/ex_lu1.c

/opt/SUNWhpc/examples/s3l/lu/ex_lu2.c

/opt/SUNWhpc/examples/s3l/lu-f/lu.f

/opt/SUNWhpc/examples/s3l/lu-f/ex_lu1.f

Related Functions

S3L_lu_factor(3)

S3L_lu_deallocate(3)

S3L_lu_solve(3)

S3l_lu_solve

Description

For each square coefficient matrix A of a, S3L_lu_solve solves a system of distributed linear equations AX = B, with a general M x M square matrix instance A, using the LU factorization computed by S3L_lu_factor.

Note – Throughout these descriptions, L^{-1} and U^{-1} denote the inverse of L and U, respectively.

A and B are corresponding instances within a and b, respectively. To solve AX = B, S3L_lu_solve performs forward elimination:

Let UX = C
A = LU implies that AX = B is equivalent to $C = L^{-1}B$

followed by back substitution:

$X = U^{-1}C = U^{-1}(L^{-1}B)$

To obtain this solution, the S3L_lu_solve routine performs the following steps:

1. Applies L^{-1} to B.
2. Applies U^{-1} to $L^{-11}B$.

Upon successful completion, each B is overwritten with the solution to AX = B.

In general, S3L_lu_solve performs most efficiently when the array is distributed using the same block size along each axis.

S3L_lu_solve behaves somewhat differently for 3D arrays, however. In this case, the nodal solve is applied on each of the 2D systems AX = B across the instance axis of a and is performed concurrently on all participating processes.

The input parallel arrays a and b must be distinct.

The internal variable setup_id is required for communicating information between the factorization routine and the other LU routines. The application must not modify the contents of this variable.

Syntax

The C and Fortran syntax for S3L_lu_solve is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_lu_solve(b, a, setup_id)
    S3L_array_t        b
    S3L_array_t        a
    int                *setup_id
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_lu_solve(b, a, setup_id, ier)
    integer*8          b
    integer*8          a
    integer*4          setup_id
    integer*4          ier
```

Input

S3L_lu_solve accepts the following arguments as input:

- b – Parallel array of the same type (real or complex) and precision as a. Must be distinct from a. The instance axes of b must match those of a in order of declaration and extents. The rows and columns of each B must be counted by axes row_axis and col_axis, respectively (from the S3L_lu_factor call). For the two-dimensional case, if b consists of only one right-hand-side vector, you can represent b as a vector (an array of rank 1) or as an array of rank 2 with the number of columns set to 1 and the elements counted by axis row_axis.
- a – Parallel array that was factored by S3L_lu_factor, where each matrix instance A is a dense M x M square matrix. Supply the same value a that was used in S3L_lu_factor.
- setup_id – Scalar integer variable. Use the value returned by the corresponding S3L_lu_factor call for this argument.

Output

S3L_lu_solve uses the following arguments for output:

- b – Upon successful completion, each matrix instance B is overwritten with the solution to AX = B.
- ier (Fortran only) – When called from a Fortran program, S3L_lu_solve returns error status in ier.

Error Handling

On success, S3L_lu_solve returns S3L_SUCCESS.

S3L_lu_solve performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and returns an error code indicating which value was invalid. See Appendix A of this manual for a detailed list of these error codes.

The following conditions will cause the function to terminate and return the associated error code:

- S3L_ERR_ARG_NULL – Invalid array. b must be preallocated, and the same value returned by S3L_lu_factor must be supplied in a.
- S3L_ERR_ARG_RANK – Invalid rank. For cases where rank >= 3, the rank of b must equal the rank of a. For the two-dimensional case, the rank of b must be either 1 or 2.
- S3L_ERR_ARG_DTYPE – Invalid data type; must be real or complex (single- or double-precision).
- S3L_ERR_ARG_BLKSIZE – Invalid block size; must be >= 1.
- S3L_ERR_MATCH_EXTENTS – Extents of a and b are mismatched along the row or instance axis.
- S3L_ERR_MATCH_DTYPE – Data types of a and b do not match.
- S3L_ERR_ARRNOTSQ – Invalid matrix size; each coefficient matrix must be square.
- S3L_ERR_ARG_SETUP – Invalid setup_id value.

Examples

```
/opt/SUNWhpc/examples/s3l/lu/lu.c
/opt/SUNWhpc/examples/s3l/lu/ex_lu1.c
/opt/SUNWhpc/examples/s3l/lu/ex_lu2.c
/opt/SUNWhpc/examples/s3l/lu-f/lu.f
/opt/SUNWhpc/examples/s3l/lu-f/ex_lu1.f
```

Related Functions

```
S3L_lu_deallocate(3)
S3L_lu_factor(3)
S3L_lu_invert(3)
```

S3L_mat_mult

Description

Sun S3L provides 18 matrix multiplication routines that compute one or more instances of matrix products. For each instance, these routines perform the operations listed in TABLE 2-9.

Note – In these descriptions, A^T and A^H denote A transpose and A Hermitian, respectively.

TABLE 2-9 S3L Matrix Multiplication Operations

Routine	Operation	Data Type
S3L_mat_mult	C = C + AB	real or complex
S3L_mat_mult_noadd	C = AB	real or complex
S3L_mat_mult_addto	C = D + AB	real or complex
S3L_mat_mult_t1	$C = C + A^{T}B$	real or complex
S3L_mat_mult_t1_noadd	$C = A^{T}B$	real or complex
S3L_mat_mult_t1_addto	$C = D + A^{T}B$	real or complex
S3L_mat_mult_h1	$C = C + A^{H}B$	complex only
S3L_mat_mult_h1_noadd	$C = A^{H}B$	complex only
S3L_mat_mult_h1_addto	$C = D + A^{H}B$	complex only
S3L_mat_mult_t2	$C = C + AB^{T}$	real or complex
S3L_mat_mult_t2_noadd	$C = AB^{T}$	real or complex
S3L_mat_mult_t2_addto	$C = D + AB^{T}$	real or complex
S3L_mat_mult_h2	$C = C + AB^{H}$	complex only
S3L_mat_mult_h2_noadd	$C = AB^{H}$	complex only
S3L_mat_mult_h2_addto	$C = D + AB^{H}$	complex only
S3L_mat_mult_t1_t2	$C = C + A^{T}B^{T}$	real or complex
S3L_mat_mult_t1_t2_noadd	$C = A^{T}B^{T}$	real or complex
S3L_mat_mult_t1_t2_addto	$C = D + A^{T}B^{T}$	real or complex

The algorithm used depends on the axis lengths of the variables supplied.

For calls that do not transpose either matrix A or B, the variables conform correctly with the axis lengths for row_axis and col_axis shown in TABLE 2-10.

TABLE 2-10 Recommended row_axis and col_axis Values When Matrix A and Matrix B Are Not Transposed

Variable	row_axis **Length**	col_axis **Length**
A	p	q
B	q	r
C	p	r
D	p	r

For calls that transpose the matrix A, the variables conform correctly with the axis lengths for row_axis and col_axis shown in TABLE 2-11.

TABLE 2-11 Recommended row_axis and col_axis Values When Matrix A Is Transposed

Variable	row_axis Length	col_axis Length
A	q	p
B	q	r
C	p	r
D	p	r

For calls that transpose the matrix B, the variables conform correctly with the axis lengths for row_axis and col_axis shown in TABLE 2-12.

TABLE 2-12 Recommended row_axis and col_axis Values When Matrix B Is Transposed

Variable	row_axis Length	col_axis Length
A	p	q
B	r	q
C	p	r
D	p	r

For calls that transpose both A and B, the variables conform correctly with the axis lengths for row_axis and col_axis shown in TABLE 2-13.

TABLE 2-13 Recommended row_axis and col_axis Values When Both Matrix A and Matrix B Are Transposed

Variable	row_axis Length	col_axis Length
A	q	p
B	r	q
C	p	r
D	p	r

The algorithm is numerically stable.

Syntax

The C and Fortran syntax for `S3L_mat_mult` is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_mat_mult(C, A, B, row_axis, col_axis)
S3L_mat_mult_noadd(C, A, B, row_axis, col_axis)
S3L_mat_mult_addto(C, A, B, D, row_axis, col_axis)
S3L_mat_mult_t1(C, A, B, row_axis, col_axis)
S3L_mat_mult_t1_noadd(C, A, B, row_axis, col_axis)
S3L_mat_mult_t1_addto(C, A, B, D, row_axis, col_axis)
S3L_mat_mult_h1(C, A, B, row_axis, col_axis)
S3L_mat_mult_h1_noadd(C, A, B, row_axis, col_axis)
S3L_mat_mult_h1_addto(C, A, B, D, row_axis, col_axis)
S3L_mat_mult_t2(C, A, B, row_axis, col_axis)
S3L_mat_mult_t2_noadd(C, A, B, row_axis, col_axis)
S3L_mat_mult_t2_addto(C, A, B, D, row_axis, col_axis)
S3L_mat_mult_h2(C, A, B, row_axis, col_axis)
S3L_mat_mult_h2_noadd(C, A, B, row_axis, col_axis)
S3L_mat_mult_h2_addto(C, A, B, D, row_axis, col_axis)
S3L_mat_mult_t1_t2(C, A, B, row_axis, col_axis)
S3L_mat_mult_t1_t2_noadd(C, A, B, row_axis, col_axis)
S3L_mat_mult_t1_t2_addto(C, A, B, D, row_axis, col_axis)
    S3L_array_t        C
    S3L_array_t        A
    S3L_array_t        B
    S3L_array_t        D
    int                row_axis
    int                col_axis
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_mat_mult(C, A, B, row_axis, col_axis, ier)
S3L_mat_mult_noadd(C, A, B, row_axis, col_axis, ier)
S3L_mat_mult_addto(C, A, B, D, row_axis, col_axis, ier)
S3L_mat_mult_t1(C, A, B, row_axis, col_axis, ier)
S3L_mat_mult_t1_noadd(C, A, B, row_axis, col_axis, ier)
S3L_mat_mult_t1_addto(C, A, B, D, row_axis, col_axis, ier)
S3L_mat_mult_h1(C, A, B, row_axis, col_axis, ier)
S3L_mat_mult_h1_noadd(C, A, B, row_axis, col_axis, ier)
S3L_mat_mult_h1_addto(C, A, B, D, row_axis, col_axis, ier)
S3L_mat_mult_t2(C, A, B, row_axis, col_axis, ier)
S3L_mat_mult_t2_noadd(C, A, B, row_axis, col_axis, ier)
S3L_mat_mult_t2_addto(C, A, B, D, row_axis, col_axis, ier)
S3L_mat_mult_h2(C, A, B, row_axis, col_axis, ier)
S3L_mat_mult_h2_noadd(C, A, B, row_axis, col_axis, ier)
S3L_mat_mult_h2_addto(C, A, B, D, row_axis, col_axis, ier)
S3L_mat_mult_t1_t2(C, A, B, row_axis, col_axis, ier)
S3L_mat_mult_t1_t2_noadd(C, A, B, row_axis, col_axis, ier)
S3L_mat_mult_t1_t2_addto(C, A, B, D, row_axis, col_axis, ier)
    integer*8          C
    integer*8          A
    integer*8          B
    integer*8          D
    integer*4          row_axis
    integer*4          col_axis
    integer*4          ier
```

Input

The S3L_mat_mult_ functions accept the following arguments as input:

- C – Array handle for an S3L parallel array of rank >= 2. C is the destination array for all matrix multiplication operations (as discussed in the Output section). Some of these operations also use C as an input argument, adding the contents of C to their respective matrix multiplication products. The operations shown in TABLE 2-9 that include some variation of C + AB belong to this class.
- A – Array handle for an S3L parallel array of the same rank as C and B. This array contains one or more instances of the left-hand factor array A, defined by axes row_axis (which counts the rows) and col_axis (which counts the columns). Axis col_axis of A must have the same length as axis row_axis of B. The contents of A are not changed during execution.

- B – Array handle for an S3L parallel array of the same rank as C and A. This array contains one or more instances of the right-hand factor array B, defined by axes row_axis (which counts the rows) and col_axis (which counts the columns). The contents of B are not changed during execution.
- D – Parallel array of the same shape as C. This argument is used only in the calls whose names end in _addto. It contains one or more instances of the array D that is to be added to the array product, defined by axes row_axis (which counts the rows) and col_axis (which counts the columns). The contents of D are not changed during execution, unless D and C are the same variable.

  The argument D can be identical with the argument C in all matrix multiply _addto routines except _t1_t2_addto.
- row_axis – The axis of C, A, and B that counts the rows of the embedded array or arrays. Must be nonnegative and less than the rank of C.
- col_axis – The axis of C, A, and B that counts the columns of the embedded array or arrays. Must be nonnegative and less than the rank of C.

Output

The S3L_mat_mult_ functions use the following arguments for output:

- C – Array handle for an S3L parallel array, which is a destination array for all matrix multiplication operations. Upon successful completion, each array instance within C is overwritten by the result of the array multiplication call.
- ier (Fortran only) – When called from a Fortran program, these functions return error status in ier.

Error Handling

On success, the S3L_mat_mult_ functions return S3L_SUCCESS.

The S3L_mat_mult routines perform generic checking of the validity of the arrays they accept as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause these functions to terminate and return the associated error code:

- S3L_ERR_MATCH_RANK – The parallel arrays do not have the same rank.
- S3L_ERR_MATCH_EXTENTS – The extents of corresponding axes do not match.
- S3L_ERR_MATCH_DTYPE – The arguments are not the same data type and precision.

- S3L_ERR_ARG_AXISNUM – row_axis and/or col_axis contains a bad axis number. For C program calls, each of these parameters must be >= 0 and less than the rank of C. For Fortran calls, they must be >= 1 and <= the rank of C.
- S3L_ERR_CONJ_INVAL – Conjugation was requested, but data supplied was not of type S3L_complex or S3L_double_complex.

Examples

/opt/SUNWhpc/examples/s3l/dense_matrix_ops/matmult.c

/opt/SUNWhpc/examples/s3l/dense_matrix_ops-f/matmult.f

Related Functions

S3L_inner_prod(3)

S3L_2_norm(3)

S3L_outer_prod(3)

S3L_mat_vec_mult(3)

S3L_mat_vec_mult

Description

Sun S3L provides six matrix vector multiplication routines, which compute one or more instances of a matrix vector product. For each instance, these routines perform the operations listed in TABLE 2-14.

Note – In these descriptions, conj[A] denotes the conjugate of A.

TABLE 2-14 S3L Matrix Vector Multiplication Operations

Routine	Operation	Data Type
`S3L_mat_vec_mult`	y = y + Ax	real or complex
`S3L_mat_vec_mult_noadd`	y = Ax	real or complex
`S3L_mat_vec_mult_addto`	y = v + Ax	real or complex
`S3L_mat_vec_mult_c1`	y = y + conj[A]x	complex only
`S3L_mat_vec_mult_c1_noadd`	y = conj[A]x	complex only
`S3L_mat_vec_mult_c1_addto`	y = v + conj[A]x	complex only

Syntax

The C and Fortran syntax for `S3L_mat_vec_mult` is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_mat_vec_mult(y, A, x, y_vector_axis, row_axis, col_axis,
x_vector_axis)
S3L_mat_vec_mult_noadd(y, A, x, y_vector_axis, row_axis,
col_axis, x_vector_axis)
S3L_mat_vec_mult_addto(y, A, x, v, y_vector_axis, row_axis,
col_axis, x_vector_axis)
S3L_mat_vec_mult_c1(y, A, x, y_vector_axis, row_axis, col_axis,
x_vector_axis)
S3L_mat_vec_mult_c1_noadd(y, A, x, y_vector_axis, row_axis,
col_axis, x_vector_axis)
S3L_mat_vec_mult_c1_addto(y, A, x, v, y_vector_axis, row_axis,
col_axis, x_vector_axis)
    S3L_array_t       y
    S3L_array_t       A
    S3L_array_t       x
    S3L_array_t       v
    int               y_vector_axis
    int               row_axis
    int               col_axis
    int               x_vector_axis
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_mat_vec_mult(y, A, x, y_vector_axis, row_axis, col_axis,
x_vector_axis, ier)
S3L_mat_vec_mult_noadd(y, A, x, y_vector_axis, row_axis, col_axis,
x_vector_axis, ier)
S3L_mat_vec_mult_addto(y, A, x, v, y_vector_axis, row_axis,
col_axis, x_vector_axis, ier)
S3L_mat_vec_mult_c1(y, A, x, y_vector_axis, row_axis, col_axis,
x_vector_axis, ier)
S3L_mat_vec_mult_c1_noadd(y, A, x, y_vector_axis, row_axis,
col_axis, x_vector_axis, ier)
S3L_mat_vec_mult_c1_addto(y, A, x, v, y_vector_axis, row_axis,
col_axis, x_vector_axis, ier)
    integer*8         y
    integer*8         A
    integer*8         x
    integer*8         v
    integer*4         y_vector_axis
    integer*4         row_axis
    integer*4         col_axis
    integer*4         x_vector_axis
    integer*4         ier
```

Input

The `S3L_mat_vec_mult_` functions accept the following arguments as input:

- `y` – Array handle for an S3L parallel array of rank >= 1. Two matrix vector multiplication routines, `S3L_mat_vec_mult` and `S3L_mat_vec_mult_c1` add the contents of this array to the product of `Ax`. All matrix vector multiplication routines use `y` as the destination array, as described in the Output section.
- `A` – Array handle for an S3L parallel array of rank one greater than that of `y`. It contains one or more instances of the matrix `A`, defined by axes `row_axis` (which counts the rows) and `col_axis` (which counts the columns).

  The remaining axes must match the instance axes of `y` in length and order of declaration. Thus, each matrix in `A` corresponds to a vector in `y`. The contents of `A` are not changed during execution.
- `x` – Array handle for an S3L parallel array of the same rank as `y`. It contains one or more instances of `x`, the vector that will be multiplied by the matrix `A`, embedded along axis `x_vector_axis`.

Axis x_vector_axis of x must have the same length as axis col_axis of A. The remaining axes of x must match the instance axes of y in length and order of declaration. Thus, each vector in x corresponds to a vector in y. The contents of x are not changed during execution.

- v – Array handle for an S3L parallel array of the same rank and shape as y. This argument is used only in the S3L_mat_vec_mult_addto and S3L_mat_vec_mult_c1_addto calls. It contains one or more instances of the vector v, which will be added to the matrix vector product, embedded along axis y_vector_axis. The contents of v are not changed during execution, unless v is the same variable as y.

  For S3L_mat_vec_mult_addto and S3L_mat_vec_mult_c1_addto, the argument v can be identical to the argument y.

- y_vector_axis – Scalar integer variable that specifies the axis of y and v along which the elements of the embedded vectors lie. For C/C++ programs, this argument must be nonnegative and less than the rank of y. For F77/F90 programs, it must be greater than zero and less than or equal to the rank of y.
- row_axis – Scalar integer variable that counts the rows of the embedded matrix or matrices. For C/C++ programs, this argument must be nonnegative and less than the rank of A. For F77/F90 programs, it must be greater than zero and less than or equal to the rank of A.
- col_axis – Scalar integer variable that counts the columns of the embedded matrix or matrices. For C/C++ programs, this argument must be nonnegative and less than the rank of A. For F77/F90 programs, it must be greater than zero and less than or equal to the rank of A.
- x_vector_axis – Scalar integer variable that specifies the axis of x along which the elements of the embedded vectors lie. For C/C++ programs, this argument must be nonnegative and less than the rank of x. For F77/F90 programs, it must be greater than zero and less than or equal to the rank of x.

Output

The S3L_mat_vec_mult_ functions use the following arguments for output:

- y – Array handle for an S3L array of rank >= 1. This array contains one or more instances of the destination vector y embedded along the axis y_vector_axis. This axis must have the same length as axis row_axis of A. Upon completion, each vector instance is overwritten by the result of the matrix vector multiplication call.

- ier (Fortran only) – When called from a Fortran program, these functions return error status in ier.

Error Handling

On success, the S3L_mat_vec_mult routines return S3L_SUCCESS.

The S3L_mat_vec_mult routines perform generic checking of the validity of the arrays they accept as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause these functions to terminate and return the associated error code:

- S3L_ERR_MATCH_RANK – The parallel arrays do not have the same rank.
- S3L_ERR_MATCH_EXTENTS – The lengths of corresponding axes do not match.
- S3L_ERR_MATCH_DTYPE – The arguments are not all of the same data type and precision.
- S3L_ERR_ARG_AXISNUM – Either row_axis or col_axis or both contain a bad axis number. For C/C++ program calls, each of these parameters must be nonnegative and less than the rank of A. For F77/F90 calls, they must be greater than zero and less than or equal to the rank of A.
- S3L_ERR_CONJ_INVAL – Conjugation was requested, but the data supplied was not of type S3L_complex or S3L_double_complex.

Examples

/opt/SUNWhpc/examples/s3l/dense_matrix_ops/mat_vec_mult.c

/opt/SUNWhpc/examples/s3l/dense_matrix_ops-f/matvec_mult.f

Related Functions

S3L_inner_prod(3)

S3L_2_norm(3)

S3L_outer_prod(3)

S3L_mat_mult(3)

S3L_matvec_sparse

Description

S3L_matvec_sparse computes the product of a global general sparse matrix and a global dense vector. The sparse matrix is described by the S3L array handle A. The global dense vector is described by the S3L array handle x. The result is stored in the global dense vector described by the S3L array handle y.

The array handle A is produced by a prior call to one of the following routines:

- S3L_declare_sparse
- S3L_read_sparse
- S3L_rand_sparse
- S3L_convert_sparse

Syntax

The C and Fortran syntax for S3L_matvec_sparse is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_matvec_sparse(y, A, x)
    S3L_array_t           y
    S3L_array_t           A
    S3L_array_t           x
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_matvec_sparse(y, A, x, ier)
    integer*8             y
    integer*8             A
    integer*8             x
    integer*4             ier
```

Input

`S3L_matvec_sparse` uses the following arguments for output:

- `A` – S3L array handle for the global general sparse matrix.
- `x` – Global array of rank 1, with the same data type and precision as `A` and `y` and with a length equal to the number of columns in the sparse matrix.

Output

`S3L_matvec_sparse` uses the following arguments for output:

- `y` – Global array of rank 1, with the same data type and precision as `A` and `x` and with a length equal to the number of rows in the sparse matrix. Upon completion, `y` contains the product of the sparse matrix `A` and `x`.
- `ier` (Fortran only) – When called from a Fortran program, `S3L_matvec_sparse` returns error status in `ier`.

Error Handling

On success, `S3L_matvec_sparse` returns `S3L_SUCCESS`.

The `S3L_matvec_sparse` routines perform generic checking of the validity of the arrays they accept as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause `S3L_matvec_sparse` to terminate and return the associated error code:

- S3L_ERR_ARG_NULL – Invalid array x or y or sparse matrix A. x and y must be preallocated S3L arrays, and A must be a preallocated sparse matrix.
- S3L_ERR_ARG_RANK – Invalid rank for arrays x and y. They must be rank 1 arrays.
- S3L_ERR_MATCH_RANK – The ranks of x and y do not match.
- S3L_ERR_MATCH_DTYPE – Arrays x, y, and A do not have the same data type.
- S3L_ERR_MATCH_EXTENTS – The lengths of x and y are mismatched with the size of sparse matrix A. The length of x must be equal to the number of columns in A and the length of y must be equal to the number of rows in A.
- S3L_ERR_SPARSE_FORMAT – Invalid sparse format. It must be S3L_SPARSE_COO, S3L_SPARSE_CSR, S3L_SPARSE_CSC, or S3L_SPARSE_VBR.

Examples

/opt/SUNWhpc/examples/s3l/sparse/ex_sparse.c

/opt/SUNWhpc/examples/s3l/sparse-f/ex_sparse.f

/opt/SUNWhpc/examples/s3l/iter/ex_iter.c

/opt/SUNWhpc/examples/s3l/iter-f/ex_iter.f

Related Functions

S3L_declare_sparse(3)

S3L_read_sparse(3)

S3L_rand_sparse(3)

S3L_outer_prod

Description

Sun S3L provides six outer product routines that compute one or more instances of an outer product of two vectors. For each instance, the outer product routines perform the operations listed in TABLE 2-15.

Note – In these descriptions, y^T and y^H denote y transpose and y Hermitian, respectively

TABLE 2-15 S3L Outer Product Operations

Routine	Operation	Data Type
S3L_outer_prod	$A = A + xy^T$	Real or complex
S3L_outer_prod_noadd	$A = xy^T$	Real or complex
S3L_outer_prod_addto	$A = B + xy^T$	Real or complex
S3L_outer_prod_c2	$A = A + xy^H$	Complex only
S3L_outer_prod_c2_noadd	$A = xy^T$	Complex only
S3L_outer_prod_c2_addto	$A = B + xy^T$	Complex only

In elementwise notation, for each instance S3L_outer_prod computes

```
A(i,j) = A(i,j) + x(i) * y(j)
```

and S3L_outer_prod_c2 computes

```
A(i,j) = A(i,j) + x(i) * conj[y(j)]
```

where conj[y(j)] denotes the conjugate of y(j).

Syntax

The C and Fortran syntax for S3L_outer_prod is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_outer_prod(A, x, y, row_axis, col_axis, x_vector_axis,
y_vector_axis)
S3L_outer_prod_noadd(A, x, y, row_axis, col_axis, x_vector_axis,
y_vector_axis)
S3L_outer_prod_addto(A, x, y, B, row_axis, col_axis,
x_vector_axis, y_vector_axis)
S3L_outer_prod_c2(A, x, y, row_axis, col_axis, x_vector_axis,
y_vector_axis)
S3L_outer_prod_c2_noadd(A, x, y, row_axis, col_axis,
x_vector_axis, y_vector_axis)
S3L_outer_prod_c2_addto(A, x, y, B, row_axis, col_axis,
x_vector_axis, y_vector_axis)
    S3L_array_t         A
    S3L_array_t         x
    S3L_array_t         y
    S3L_array_t         B
    int                 row_axis
    int                 col_axis
    int                 x_vector_axis
    int                 y_vector_axis
```

F77/F90 Syntax

```
include ‘s3l/s3l-f.h’
include ‘s3l/s3l_errno-f.h’
subroutine
S3L_outer_prod(A, x, y, row_axis, col_axis, x_vector_axis,
y_vector_axis, ier)
S3L_outer_prod_noadd(A, x, y, row_axis, col_axis, x_vector_axis,
y_vector_axis, ier)
S3L_outer_prod_addto(A, x, y, B, row_axis, col_axis,
x_vector_axis, y_vector_axis, ier)
S3L_outer_prod_c2(A, x, y, row_axis, col_axis, x_vector_axis,
y_vector_axis, ier)
S3L_outer_prod_c2_noadd(A, x, y, row_axis, col_axis,
x_vector_axis, y_vector_axis, ier)
S3L_outer_prod_c2_addto(A, x, y, B, row_axis, col_axis,
x_vector_axis, y_vector_axis, ier)
    integer*8           A
    integer*8           x
```

```
integer*8          y
integer*8          B
integer*4          row_axis
integer*4          col_axis
integer*4          x_vector_axis
integer*4          y_vector_axis
integer*4          ier
```

Input

The S3L_outer_prod_ functions accept the following arguments as input:

- A – Array handle for an S3L parallel array of rank greater than or equal to 2. Two S3L outer product routines, S3L_outer_prod and S3L_outer_prod_c2, add the contents of this array to the product of xy. All outer product routines use A as the destination array, as described in the Output section.
- x – Array handle for an S3L parallel array of rank one less than that of A. It contains one or more instances of the first source vector x embedded along axis x_vector_axis.
  Axis x_vector_axis of x must have the same length as axis row_axis of A. The remaining axes of x must match the instance axes of A in length and order of declaration. Thus, each vector in x corresponds to a vector in A.
- y – Array handle for an S3L parallel array of rank one less than that of A. It contains one or more instances of the second source vector x embedded along axis y_vector_axis.
  y_vector_axis must have the same length as axis col_axis of A. The remaining axes of y must match the instance axes of A in length and order of declaration. Thus, each vector in y corresponds to a vector in A.

  The argument y can be identical to the argument x.

- B – Parallel array of the same shape as A. It contains one or more embedded matrices B defined by axes row_axis (which counts the rows) and col_axis (which counts the columns). The remaining axes must match the instance axes of A in length and order of declaration. Thus, each matrix in B corresponds to a matrix in A.
  This argument is used only in the S3L_outer_prod_addto and S3L_outer_prod_c2_addto calls, which add each outer product to the corresponding matrix within B and place the result in the corresponding matrix within A. The contents of B are not changed by the operation (unless B and A are the same variable).

  For S3L_outer_prod_addto and S3L_outer_prod_c2_addto, the argument B can be identical to the argument A.

- row_axis – Scalar integer variable. The axis of A and B that counts the rows of the embedded matrix or matrices. For C/C++ programs, this argument must be nonnegative and less than the rank of A. For F77/F90 programs, it must be greater than zero and less than or equal to the rank of A.
- col_axis – Scalar integer variable. The axis of A and B that counts the columns of the embedded matrix or matrices. For C/C++ programs, this argument must be nonnegative and less than the rank of A. For F77/F90 programs, it must be greater than zero and less than or equal to the rank of A.
- x_vector_axis – Scalar integer variable that specifies the axis of x along which the elements of the embedded vectors lie. For C/C++ programs, this argument must be nonnegative and less than the rank of y. For F77/F90 programs, it must be greater than zero and less than or equal to the rank of x.
- y_vector_axis – Scalar integer variable that specifies the axis of y along which the elements of the embedded vectors lie. For C/C++ programs, this argument must be nonnegative and less than the rank of y. For F77/F90 programs, it must be greater than zero and less than or equal to the rank of y.

Output

The S3L_outer_prod_ functions use the following arguments for output:

- A – Array handle for an S3L parallel array of rank greater than or equal to 2, which contains one or more instances of the destination matrix A, defined by axes row_axis (which counts the rows) and col_axis (which counts the columns). Upon successful completion, each matrix instance is overwritten by the result of the outer product call.
- ier (Fortran only) – When called from a Fortran program, these functions return error status in ier.

Error Handling

On success, the S3L_outer_prod routines return S3L_SUCCESS.

The S3L_outer_prod routines perform generic checking of the validity of the arrays they accept as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause these functions to terminate and return the associated error code:

- S3L_ERR_MATCH_RANK – The parallel arrays do not have the same rank.

- S3L_ERR_MATCH_EXTENTS – The lengths of corresponding axes do not match.
- S3L_ERR_MATCH_DTYPE – The arguments are not all of the same data type and precision.
- S3L_ERR_ARG_AXISNUM – row_axis and/or col_axis contains a bad axis number. For C/C++ program calls, each of these parameters must be nonnegative and less than the rank of A. For F77/F90 calls, they must be greater than zero and less than or equal to the rank of A.
- S3L_ERR_CONJ_INVAL – Conjugation was requested, but the data supplied was not of type S3L_complex or S3L_double_complex
- S3L_ERR_ARG_RANK – Rank of A is less than 2.

Examples

/opt/SUNWhpc/examples/s3l/dense_matrix_ops/outer_prod.c

/opt/SUNWhpc/examples/s3l/dense_matrix_ops-f/outer_prod.f

Related Functions

S3L_inner_prod(3)

S3L_2_norm(3)

S3L_mat_vec_mult(3)

S3L_mat_mult(3)

S3L_print_array and S3L_print_sub_array

Description

S3L_print_array causes the process with MPI rank 0 to print the parallel array represented by the array handle a to standard output.

S3L_print_sub_array prints a specific section of the parallel array. This array section is defined by the lbounds, ubounds, and strides arguments. lbounds and ubounds specify the array section's lower and upper index bounds. strides specifies the stride to be used along each axis; it must be greater than zero.

Note – The values of lbounds and ubounds should refer to zero-based indexed arrays for the C interface and to one-based indexed arrays for the Fortran interface.

Syntax

The C and Fortran syntax for S3L_print_array and S3L_print_sub_array is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_print_array(a)
S3L_print_sub_array(a, lbounds, ubounds, strides)
    S3L_array_t         a
    int                 *lbounds
    int                 *ubounds
    int                 *strides
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_print_array(a, ier)
S3L_print_sub_array(a, lbounds, ubounds, strides, ier)
    integer*8           a
    integer*4           lbounds(*)
    integer*4           ubounds(*)
    integer*4           strides(*)
    integer*4           ier
```

Input

S3L_print_array and S3L_print_sub_array accept the following arguments as input:

- a – S3L array handle for the parallel array to be printed. This array handle was returned when the array was previously declared.
- lbounds – Integer vector specifying the lower bounds of the indices of a along each of its axes.
- ubounds – Integer vector specifying the upper bounds of the indices of a along each of its axes.
- strides – Integer vector specifying the strides on the indices of a along each of its axes.

Output

S3L_print_array and S3L_print_sub_array use the following argument for output:

- ier (Fortran only) – When called from a Fortran program, S3L_print_array and S3L_print_sub_array return error status in ier.

Error Handling

On success, S3L_print_array and S3L_print_sub_array return S3L_SUCCESS.

S3L_print_array and S3L_print_sub_array perform generic checking of the validity of the arrays they accept as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following condition will cause the function to terminate and return the associated error code:

- S3L_ERR_ARG_RANGE_INV – The given range of indices is invalid:
  - A lower bound is less than the smallest index of the array.
  - An upper bound is greater than the largest index of the array along the given axis.
  - A lower bound is larger than the corresponding upper bound.
  - A stride is negative or zero.

Examples

/opt/SUNWhpc/examples/s3l/io/ex_print1.c

/opt/SUNWhpc/examples/s3l/io/ex_io.c

/opt/SUNWhpc/examples/s3l/io-f/ex_io.f

Related Functions

S3L_read_array(3)

S3L_write_array(3)

S3L_print_sparse

Description

S3L_print_sparse prints all nonzero values of a global general sparse matrix and their corresponding row and column indices to standard output.

For example, the following 4x6 sample matrix:

```
  3.14       0          0          20.04       0          0
  0         27          0           0         -0.6        0
  0          0        -0.01         0          0          0
 -0.031      0          0           0.08       0        314.0
```

could be printed by a C program in the following manner.

```
  4 6 8
 (0,0)  3.140000
 (0,3)  200.040000
 (1,1)  27.000000
 (1,4)  -0.600000
 (2,2)  -0.010000
 (3,0)  -0.031000
 (3,3)  0.080000
 (3,5)  314.000000
```

Note that, for C-language applications, zero-based indices are used. For Fortran applications, one-based indices are used as follows:

```
 4 6 8
(1,1)  3.140000
(1,4)  200.040000
(2,2)  27.000000
(2,5)  -0.600000
(3,3)  -0.010000
(4,1)  -0.031000
(4,4)  0.080000
(4,6)  314.000000
```

The first line prints three integers, m, n, and nnz, which represent the number of rows, columns, and the total number of nonzero elements in the matrix, respectively. If the matrix is stored in Variable Block Row format, three additional integers are printed as well: bm, bn, and bnnz. These integers indicate the number of block rows and block columns and the total number of nonzero block entries.

The remaining lines list the all the nonzero elements in the matrix, one per line. The first two values in each line are the row and column indices for the corresponding nonzero element.

Syntax

The C and Fortran syntax for S3L_print_sparse is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_print_sparse(A)
    S3L_array_t        A
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_print_sparse(A, ier)
    integer*8         A
    integer*4         ier
```

Input

S3L_print_sparse accepts the following argument as input:

- A – S3L internal array handle for the global general sparse matrix that is produced by a prior call to one of the following sparse routines:
  - S3L_declare_sparse
  - S3L_read_sparse
  - S3L_rand_sparse
  - S3L_convert_sparse

Output

S3L_print_sparse uses the following argument for output:

- ier (Fortran only) – When called from a Fortran program, S3L_print_sparse returns error status in ier.

Error Handling

On success, S3L_print_sparse returns S3L_SUCCESS.

The S3L_print_sparse routine performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

On error, S3L_print_sparse returns the following code:

- S3L_ERR_ARG_NULL – The value specified for A is invalid; no such S3L sparse matrix has been defined.
- S3L_ERR_SPARSE_FORMAT – Invalid sparse format. It must be: S3L_SPARSE_COO, S3L_SPARSE_CSR, S3L_SPARSE_CSC, or S3L_SPARSE_VBR.

Examples

/opt/SUNWhpc/examples/s3l/sparse/ex_sparse.c

/opt/SUNWhpc/examples/s3l/sparse/ex_sparse2.c

/opt/SUNWhpc/examples/s3l/sparse-f/ex_sparse.f

Related Functions

S3L_declare_sparse(3)

S3L_read_sparse(3)

S3L_rand_sparse(3)

S3L_write_sparse(3)

S3L_qp

Description

S3L_qp applies an interior point method to solve the following linear/quadratic optimization problem:

```
min (1/2)*x'*Q*x+f'*x
```

subject to:

```
ub >= x >= lb
C*x > d
A*x = b
```

The arrays must be either S3L_float or S3L_double.

Q, A, and C should be either dense or sparse S3L arrays and all of the same type.

If convergence is achieved, the result of the optimization will be in xf.

Syntax

The C and Fortran syntax for S3L_qp is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_qp(A, Q, C, f, b, d, lb, ub, xf, iter, tol, attrib)
    S3L_array_t        A
    S3L_array_t        Q
    S3L_array_t        C
    S3L_array_t        f
    S3L_array_t        b
    S3L_array_t        d
    S3L_array_t        lb
    S3L_array_t        ub
    S3L_array_t        xf
    int                *iter
    <type>             *tol
    S3L_qp_attr_t      attrib
```

where <type> is either `float` or `double`.

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_qp(A, Q, C, f, b, d, lb, ub, xf, iter, tol, attrib, ier)
    integer*8          A
    integer*8          Q
    integer*8          C
    integer*8          f
    integer*8          b
    integer*8          d
    integer*8          lb
    integer*8          ub
    integer*8          xf
    integer*4          iter
    <type>             tol
    integer*8          attrib
    integer*4          ier
```

where <type> is either `real*4` or `real*8`.

Input

S3L_qp accepts the following argument as input:

- A – Dense or sparse S3L array of size ne x n.
- Q – Dense or sparse S3L array of size n x n.
- C – Dense or sparse S3L array of size nc x n.
- f – Dense S3L vector of length n.
- b – Dense S3L vector of length ne.
- d – Dense S3L vector of length nc.
- lb – Dense S3L vector of length n.
- ub – Dense S3L vector of length n.
- xf – Dense S3L vector of length n.
- iter – On entry, iter specifies the maximum number of iterations. Also used for output, as described below.
- tol – On entry, tol specifies the tolerance to be achieved in the linear complementarity gap for the problem to be considered solved. Also used for output, as described below.
- attrib – Attribute handle returned by an earlier call to S3L_qp_attr_init.

Output

S3L_qp use the following arguments for output:

- iter – On exit, iter contains the actual number of iterations performed.
- tol – On exit, tol contains the actual level of tolerance achieved.
- ier (Fortran only) – When called from a Fortran program, S3L_qp returns error status in ier.

Error Handling

On success, S3L_qp returns S3L_SUCCESS.

S3L_qp performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and returns an error code indicating which value was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause S3L_qp to terminate and return the associated error code:

- S3L_ERR_ARG_ARRAY – Both C and A arrays are equal to NULL.
- S3L_ERR_ARG_DTYPE – The data type of the supplied arrays is not S3L_double or S3L_float.
- S3L_ERR_MATCH_DTYPE – The data type of A is not the same as that of b.

Examples

/opt/SUNWhpc/examples/s3l/optim/ex_lp1.c

/opt/SUNWhpc/examples/s3l/optim/ex_qp1.c

/opt/SUNWhpc/examples/s3l/optim/ex_lp_sparse1.c

/opt/SUNWhpc/examples/s3l/optim/ex_qp_sparse1.c

/opt/SUNWhpc/examples/s3l/optim-f/ex_lp1.f

/opt/SUNWhpc/examples/s3l/optim-f/ex_qp1.f

/opt/SUNWhpc/examples/s3l/optim-f/ex_sp_lp1.f

Related Functions

S3L_lp_sparse(3)

S3L_qp_attr_init(3)

S3L_qp_attr_destroy(3)

S3L_qp_attr_set(3)

S3L_qp_attr_init, S3L_qp_attr_destroy, S3L_qp_attr_set

Description

`S3L_qp_attr_init` initializes a set of attributes with the handle `attrib` and loads a set of default values.

`S3L_qp_attr_destroy` destroys the set of attributes with the handle `attrib`. Once destroyed, `attrib` cannot be reused until it is reinitialized.

`S3L_qp_attr_set` specifies the type of solver to be used and the amount of error information that will be generated.

Syntax

The C and Fortran syntax for S3L_qp_attr_init, S3L_qp_attr_destroy, and S3L_qp_attr_set is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_qp_attr_init(&attrib)
S3L_qp_attr_destroy(&attrib)
S3L_qp_attr_set(&attrib, request, value)
    S3L_qp_attr_t          attrib
    S3L_qp_attr_req_t      request
    void                   *value
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_qp_attr_init(attrib, ier)
S3L_qp_attr_destroy(attrib, ier)
S3L_qp_attr_set(attrib, request, value, ier)
    integer*8              attrib
    integer*4              request
    integer*4              value
    integer*4              ier
```

Input

The S3L_qp_attr_ functions accept the following arguments as input:

- attrib – Handle for a set of attributes. This parameter is supplied by S3L_qp_attr_init and is used as input by S3L_qp_attr_destroy, S3L_qp_attr_set, and S3L_qp.
- request – For S3L_qp_attr_set, specifies the property of interest, which can be one of:

S3L_QP_SOLVER_TYPE	Set the direct solver type.
S3L_QP_VERBOSITY	Set the verbosity level.

- `value` – Specifies the value of the property named by the `request` argument. There are two kinds of values that can be set: solver type and verbosity level. The allowed values of both kinds are described below:

  For sparse constraint arrays, `value` can be used to specify the solver type, as follows:

`S3L_QP_SPLUS`	(default) Use the S+ full-pivoting asymmetric direct solver.
`S3L_QP_LIBSUNPERF`	Use the Sun Performance Library direct solver.

  `value` can also be used to set the verbosity level of reporting, as follows:

`S3L_QP_VERB_NONE`	(default) No output.
`S3L_QP_VERB_FULL`	Print the value of the error in every iteration.

Output

The `S3L_qp_attr_` functions use the following arguments for output:

- `attrib` – `S3L_qp_attr_init` returns this handle for a set of attributes.
- `ier` (Fortran only) – When called from a Fortran program, these functions return error status in `ier`.

Error Handling

On success, the `S3L_qp_attr_` functions all return `S3L_SUCCESS`.

The following conditions will cause the indicated function to terminate and return the associated error code:

- `S3L_ERR_ATTR_INVALID` – `attrib` is not a properly initialized variable.
- `S3L_ERR_NONSUPPORT` – An invalid value has been supplied.

Examples

/opt/SUNWhpc/examples/s3l/optim/ex_lp1.c

/opt/SUNWhpc/examples/s3l/optim/ex_qp1.c

/opt/SUNWhpc/examples/s3l/optim/ex_lp_sparse1.c

/opt/SUNWhpc/examples/s3l/optim/ex_qp_sparse1.c

Related Functions

S3L_qp(3)

S3L_lp_sparse(3)

S3L_qr_factor

Description

S3L_qr_factor computes the QR decomposition of real or complex S3L arrays. On exit, the Q and R factors are packed in array a.

S3L_qr_factor generates internal information related to the decomposition, such as the vector of elementary reflectors. It also returns a setup parameter, which can be used by subsequent calls to S3L_qr_solve to compute the least-squares solution to a system A*x = b, where A is an m x n array, with m > n, and b is an m x nrhs array.

S3L_qr_factor can be used for arrays with more than two dimensions. In such cases, the axis_r and axis_c arguments specify the row and column axes of 2D array slices, whose QR factorization is to be computed.

When a is a 2D array, axis_r and axis_c should be set as shown in TABLE 2-16.

TABLE 2-16 Summary of axis_r and axis_c Settings for S3L_qr_factor

QR factorization of	C/C++		F77/F90	
	axis_r	axis_c	axis_r	axis_c
a	0	1	1	2
Transpose of a	1	0	2	1

Notes

S3L_qr_factor is more efficient when both dimensions of the input array are block-cyclically distributed with equal block sizes.

If least-squares solutions are to be found for multiple A*x = b systems, where all systems have the same matrix, the same QR factorization setup can be used by all the S3L_qr_solve instances.

Syntax

The C and Fortran syntax for `S3L_qr_factor` is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_qr_factor(a, axis_r, axis_c, setup)
    S3L_array_t         a
    int                 axis_r
    int                 axis_c
    int                 *setup
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_qr_factor(a, axis_r, axis_c, setup, ier)
    integer*8           a
    integer*4           axis_r
    integer*4           axis_c
    integer*4           setup
    integer*4           ier
```

Input

`S3L_qr_factor` accepts the following arguments as input:

- `a` – Input array whose QR decomposition is to be computed. On exit, the contents of `a` are destroyed.
- `axis_r` – Integer denoting the row axis. For C program calls, `axis_r` must be >= 0 and less than the rank of `a`; for Fortran program calls, it must be >= 1 and not exceed the rank of `a`.
- `axis_c` – Integer denoting the column axis. For C program calls, `axis_c` must be >= 0 and less than the rank of `a`; for Fortran program calls, it must be >= 1 and not exceed the rank of `a`.

Output

S3L_qr_factor uses the following arguments for output:

- setup – Integer used by subsequent calls to S3L_qr_solve to access internal QR factorization information.
- ier (Fortran only) – When called from a Fortran program, S3L_qr_factor returns error status in ier.

Error Handling

On success, S3L_qr_factor returns S3L_SUCCESS.

S3L_qr_factor performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and returns an error code indicating which value was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause S3L_qr_factor to terminate and return the associated error code:

- S3L_ERR_ARG_RANK – Rank of a is less than 2, which is invalid.
- S3L_ERR_ARG_DTYPE – Invalid data type. It must be S3L_float, S3L_double, S3L_complex, or S3L_double_complex.
- S3L_ERR_ARG_AXISNUM – axis_r or axis_c or both contain invalid entries.

Examples

/opt/SUNWhpc/examples/s3l/qr/ex_qr1.c

/opt/SUNWhpc/examples/s3l/qr-f/ex_qr1.f

Related Functions

S3L_get_qr(3)

S3L_qr_solve(3)

S3L_qr_free(3)

S3L_qr_free

Description

S3L_qr_free frees all internal resources associated with a particular QR decomposition.

Syntax

The C and Fortran syntax for S3L_qr_free is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_qr_free(setup)
    int                *setup
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_qr_free(setup, ier)
    integer*4          setup
    integer*4          ier
```

Input

S3L_qr_free accepts the following argument as input:

- setup – Integer returned by a previous call to S3L_qr_factor.

Output

S3L_qr_free uses the following argument for output:

- ier (Fortran only) – When called from a Fortran program, S3L_qr_free returns error status in ier.

Error Handling

On success, S3L_qr_free returns S3L_SUCCESS.

In addition, the following condition will cause S3L_qr_free to terminate and return the associated error code:

- S3L_ERR_ARG_SETUP – Invalid setup value.

Examples

/opt/SUNWhpc/examples/s3l/qr/ex_qr1.c

/opt/SUNWhpc/examples/s3l/qr-f/ex_qr1.f

Related Functions

S3L_qr_factor(3)

S3L_qr_solve(3)

S3L_get_qr(3)

S3L_qr_solve

Description

S3L_qr_solve computes the least-squares solution to an overdetermined linear system of the form a*x = b. a is an m x n S3L array, where m > n (overdetermined). b is an m x nrhs S3L array of the same type as a. S3L_qr_solve uses the QR factorization results from a previous call to S3L_qr_factor for the computation. On exit, the first n x nrhs rows of b are overwritten with the least-squares solution of the system.

a and b can have more than two dimensions, in which case, the operation is performed over all 2D slices, which were specified by the row and column axis arguments, axis_r and axis_c, of the corresponding S3L_qr_factor call.

Notes

For m > n, the single routine S3L_gen_lsq performs the same set of operations as the sequence: S3L_qr_factor, S3L_qr_solve, S3L_qr_free. However, when multiple least-squares solutions are to be found for a set of matrices that are all the same, the explicit sequence can be more efficient. This is because S3L_gen_lsq performs the full sequence every time it is called, even though the QR factorization step is needed only the first time. In such cases, therefore, the following sequence can be used to eliminate redundant factorization operations:

- S3L_qr_factor, S3L_qr_solve, S3L_get_qr for the first solution
- S3L_qr_solve, S3L_get_qr for the second and all subsequent solutions

Syntax

The C and Fortran syntax for S3L_qr_solve is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_qr_solve(a, b, setup)
    S3L_array_t        a
    S3L_array_t        b
    int                setup
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_qr_solve(a, b, setup, ier)
    integer*8           a
    integer*8           b
    integer*4           setup
    integer*4           ier
```

Input

S3L_qr_solve accepts the following arguments as input:

- a – Input array of size m x n containing a QR decomposition computed by means of S3L_qr_factor.
- b – rhs array of size m x nrhs.
- setup – Integer returned by a previous call to S3L_qr_factor.

Output

S3L_qr_solve uses the following arguments for output:

- b – On exit, the first n rows of b contain the solution to the least-squares problem.
- ier (Fortran only) – When called from a Fortran program, S3L_qr_solve returns error status in ier.

Error Handling

On success, S3L_qr_solve returns S3L_SUCCESS.

S3L_qr_solve performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and returns an error code indicating which value was invalid. See Appendix A of this manual for a detailed list of these error codes.

The following conditions will cause S3L_qr_solve to terminate and return the associated error code:

- S3L_ERR_ARG_RANK – a and/or b are 1D arrays.
- S3L_ERR_ARG_DTYPE – The data type of a is not S3L_float, S3L_double, S3L_complex, or S3L_double_complex.

- S3L_ERR_ARG_EXTENTS – The extents of a and b are incompatible.
- S3L_ERR_ARG_SETUP – Invalid setup value.

Examples

/opt/SUNWhpc/examples/s3l/qr/ex_qr1.c

/opt/SUNWhpc/examples/s3l/qr-f/ex_qr1.f

Related Functions

S3L_qr_factor(3)

S3L_get_qr(3)

S3L_qr_free(3)

S3L_rand_fib

Description

S3L_rand_fib initializes a parallel array with a Lagged-Fibonacci random number generator (LFG). The LFG's parameters are fixed to l = 17, k = 5, and m = 32.

Random numbers are produced by the following iterative equation:

```
x[n] = (x[n-e] + x[n-k]) mod 2^m
```

The result of S3L_rand_fib depends on how the parallel array a is distributed.

When the parallel array is of type integer, its elements are filled with nonnegative integers in the range 0 . . . 2^{31} -1. When the parallel array is single- or double-precision real, its elements are filled with random nonnegative numbers in the range 0 . . . 1. For complex arrays, the real and imaginary parts are initialized to random real numbers.

Syntax

The C and Fortran syntax for S3L_rand_fib is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_rand_fib(a, setup_id)
    S3L_array_t        a
    int                setup_id
```

F77/F90 Syntax

```
include ‘s3l/s3l-f.h’
include ‘s3l/s3l_errno-f.h’
subroutine
S3L_rand_fib(a, setup_id, ier)
    integer*8          a
    integer*4          setup_id
    integer*4          ier
```

Input

`S3L_rand_fib` accepts the following arguments as input:

- `a` – S3L array handle that describes the parallel array to be initialized by the LFG.
- `setup_id` – Integer index used to access the state table associated with the array referenced by `a`.

Output

`S3L_rand_fib` uses the following arguments for output:

- `a` – On output, `a` is a randomly initialized array.
- `ier` (Fortran only) – When called from a Fortran program, `S3L_rand_fib` returns error status in `ier`.

Error Handling

On success, `S3L_rand_fib` returns `S3L_SUCCESS`.

S3L_rand_fib checks the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following condition will cause S3L_rand_fib to terminate and return the associated error code.

- S3L_ERR_ARG_SETUP – Invalid setup_id value.

Examples

/opt/SUNWhpc/examples/s3l/rand_fib/rand_fib.c

/opt/SUNWhpc/examples/s3l/rand_fib-f/rand_fib.f

Related Functions

S3L_free_rand_fib(3)

S3L_setup_rand_fib(3)

S3L_rand_lcg

Description

S3L_rand_lcg initializes a parallel array a, using a linear congruential random number generator (LCG). It produces random numbers that are independent of the distribution of the parallel array.

Arrays of type S3L_integer (integer*4) are initialized to random integers in the range 0 . . . 2^{31}-1. Arrays of type S3L_long_integer are initialized with integers in the range 0 . . . 2^{63}-1. Arrays of type S3L_float or S3L_double are initialized in the range 0 . . . 1. The real and imaginary parts of type S3L_complex and S3L_double_complex are also initialized in the range 0 . . . 1.

The random numbers are initialized by an internal iterative equation of the type:

```
x[n] = a*x[n-1] + c
```

Syntax

The C and Fortran syntax for `S3L_rand_lcg` is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_rand_lcg(a, iseed)
    S3L_array_t       a
    int               iseed
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_rand_lcg(a, iseed, ier)
    integer*8         a
    integer*4         iseed
    integer*4         ier
```

Input

`S3L_rand_lcg` accepts the following arguments as input:

- `a` – S3L array handle that describes the parallel array to be initialized by the LCG.
- `iseed` – An integer. If positive, this value is used as the initial seed for the LCG. If zero or negative, the call to `S3L_rand_lcg` produces a sequence of random numbers, which is a continuation of a sequence generated in a previous call to `S3L_rand_lcg`.

Output

`S3L_rand_lcg` uses the following arguments for output:

- `a` – On output, `a` is a randomly initialized array.
- `ier` (Fortran only) – When called from a Fortran program, `S3L_rand_lcg` returns error status in `ier`.

Error Handling

On success, S3L_rand_lcg returns S3L_SUCCESS.

S3L_rand_lcg checks the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following condition will cause the function to terminate and return the associated error code:

- S3L_ERR_ARG_RANK – Rank of a is invalid.

Examples

/opt/SUNWhpc/examples/s3l/rand_lcg/rand_lcg.c

/opt/SUNWhpc/examples/s3l/rand_lcg-f/rand_lcg.f

Related Functions

S3L_free_rand_fib(3)

S3L_setup_rand_fib(3)

S3L_rand_sparse

Description

S3L_rand_sparse creates a random sparse matrix with a random sparsity pattern in one of the four sparse formats:

- S3L_SPARSE_COO – Coordinate format
- S3L_SPARSE_CSR – Coordinate Sparse Row format
- S3L_SPARSE_CSC – Coordinate Sparse Column format
- S3L_SPARSE_VBR – Variable Block Row format

Upon successful completion, S3L_rand_sparse returns an S3L array handle in A, which represents this random sparse matrix.

The number of nonzero elements that are generated will depend primarily on the combination of the `density` argument value and the array extents given by `m` and `n`. Usually, the number of nonzero elements will approximately equal `m*n*density`. The behavior of the algorithm may cause the actual number of nonzero elements to be somewhat smaller than `m*n*density`. Regardless of the value supplied for the `density` argument, the number of nonzero elements will always be >= `m`.

Syntax

The C and Fortran syntax for `S3L_rand_sparse` is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_rand_sparse(A, spfmt, stype, m, n, density, type, seed, ...)
    S3L_array_t              *A
    S3L_sparse_storage_t     spfmt
    S3L_sparse_rand_t        stype
    int                      m
    int                      n
    real*4                   density
    S3L_data_type            type
    int                      seed
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_rand_sparse(A, spfmt, stype, m, n, density, type, seed, ...,
ier)
    integer*8                 A
    integer*4                 spfmt
    integer*4                 stype
    integer*4                 m
    integer*4                 n
    real*4                    density
    integer*4                 type
    integer*4                 seed
    integer*4                 ier
```

Input

S3L_rand_sparse accepts the following arguments as input:

- spfmt – Indicates the sparse storage format used for representing the sparse matrix. Use S3L_SPARSE_COO, S3L_SPARSE_CSR, or S3L_SPARSE_CSC to create a random point sparse matrix. Use S3L_SPARSE_VBR to create a sparse matrix with random block structure.

  If the value of spfmt is S3L_SPARSE_VBR, the following two arguments should also be supplied:

  - rptr – An integer array of length m+1, such that rptr[i] is the row index of the first point row in the i-th block row.
  - cptr – An integer array of length n+1, such that cptr[j] is the column index of the first column in the j-th block column.

  If used, rptr and cptr follow the seed argument, as indicated by the "..." in the Syntax section.

- stype – A variable of the type S3L_sparse_rand_t (C/C++) or integer*4 (F77/F90) that specifies the type of random pattern to be used, as follows:
  - S3L_SPARSE_RAND – A random pattern.
  - S3L_SPARSE_DRND – A random pattern with guaranteed nonzero diagonal.
  - S3L_SPARSE_SRND – A random symmetric sparse array.
  - S3L_SPARSE_DSRN – A random symmetric sparse array with guaranteed nonzero diagonal.
  - S3L_SPARSE_DSPD – A random symmetric positive definite sparse array.

- m – When the sparse format is S3L_SPARSE_COO, S3L_SPARSE_CSR, or S3L_SPARSE_CSC, m indicates the total number of point rows in the sparse matrix. Under S3L_SPARSE_VBR, m denotes the total number of block rows in the sparse matrix.
- n – When the sparse format is S3L_SPARSE_COO, S3L_SPARSE_CSR, or S3L_SPARSE_CSC, n indicates the total number of point columns in the sparse matrix. Under S3L_SPARSE_VBR, n denotes the total number of block columns in the sparse matrix.
- density – Positive parameter less than or equal to 1.0, which suggests the approximate density of the array. For example, if density = 0.1, approximately 10% of the array elements will have nonzero values.
- type – The type of the sparse array, which must be S3L_integer, S3L_float, S3L_double, S3L_complex, or S3L_double_complex.
- seed – An integer that is used internally to initialize the random number generators. It affects both the pattern and the values of the array elements. The results are independent of the number of processes on which the function is invoked.

Output

S3L_rand_sparse uses the following arguments for output:

- A – On return, A contains an S3L internal array handle for the distributed random sparse matrix. The handle can be used in subsequent calls to some other S3L sparse array functions.
- ier (Fortran only) – When called from a Fortran program, S3L_rand_sparse returns error status in ier.

Error Handling

On success, S3L_rand_sparse returns S3L_SUCCESS.

The S3L_rand_sparse routine performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause S3L_rand_sparse to terminate and return the associated error code:

- S3L_ERR_SPARSE_FORMAT – Invalid storage format. It must be S3L_SPARSE_COO, S3L_SPARSE_CSR, S3L_SPARSE_CSC, or S3L_SPARSE_VBR.

- S3L_ERR_SPARSE_PATTERN – Invalid random pattern. When spfmt is S3L_SPARSE_COO, S3L_SPARSE_CSR, or S3L_SPARSE_CSC, stype can be S3L_SPARSE_RAND, S3L_SPARSE_DRND, S3L_SPARSE_SRND, S3L_SPARSE_DSRN, or S3L_SPARSE_DSPD. When spfmt is S3L_SPARSE_VBR, stype must be either S3L_SPARSE_RAND or S3L_SPARSE_DRND.
- S3L_ERR_ARG_EXTENTS – Invalid m or n. Each extent value must be > 0.
- S3L_ERR_ARRNOTSQ – Invalid matrix size. When stype does not equal S3L_SPARSE_RAND, m must equal n.
- S3L_ERR_DENSITY – Invalid density value. It must be 0.0 < density <= 1.0.
- S3L_ERR_ARG_DTYPE – Invalid data type. When stype is S3L_SPARSE_DSPD, the data type of the sparse matrix must be S3L_float or S3L_double.
- S3L_ERR_ARG_NULL – Invalid arguments for rptr and cptr. When spfmt is S3L_SPARSE_VBR, both rptr and cptr must be preallocated and initialized.

Examples

/opt/SUNWhpc/examples/s3l/iter/ex_iter.c

/opt/SUNWhpc/examples/s3l/iter-f/ex_iter.f

Related Functions

S3L_declare_sparse(3)

S3L_read_sparse(3)

S3L_rc_fft and S3L_cr_fft

Description

S3L_rc_fft and S3L_cr_fft are used for computing the Fast Fourier Transform of real 1D, 2D, or 3D arrays. S3L_rc_fft performs a forward FFT of a real array and S3l_cr_fft performs the inverse FFT of a complex array with certain symmetry properties. The result of S3l_cr_fft is real.

S3L_rc_fft accepts as input a real (single- or double-precision) parallel array and, upon successful completion, overwrites the contents of the real array with the complex Discrete Fourier Transform (DFT) of the data in a packed format.

`S3L_cr_fft` accepts as input a real array, which contains the packed representation of a complex array.

`S3L_rc_fft` and `S3L_cr_fft` have been optimized for cases where the arrays are distributed only along their last dimension. They also work, however, for any `CYCLIC(`*n*`)` array layout.

For the 2D FFT, a more efficient transposition algorithm is used when the blocksizes along each dimension are equal to the extents divided by the number of processors. This arrangement can result in significantly higher performance.

The algorithms used are nonstandard extensions of the Cooley-Tuckey factorization and the Chinese Remainder Theorem. Both power-of-two and arbitrary radix FFTs are supported.

The nodal FFTs upon which the parallel FFT is based are mixed radix with prime factors 2, 3, 5, 7, 11, and 13. The parallel FFT will be more efficient when the size of the array is a product of powers of these factors. When the size of an array cannot be factored into these prime factors, a slower DFT is used for the remainder.

Supported Array Sizes

One Dimension: The array size must be divisible by 4 x p^2, where p is the number of processors.

Two Dimensions: Each of the array extents must be divisible by 2 x p, where p is the number of processors.

Three Dimensions: The first dimension must be even and must have a length of at least 4. The second and third dimensions must be divisible by 2 x p, where p is the number of processors.

Scaling

The real-to-complex and complex-to-real S3L parallel FFTs do not include scaling of the data. Consequently, for a forward 1D real-to-complex FFT of a vector of length *n*, followed by an inverse 1D complex-to-real FFT of the result, the original vector is multiplied by $n/2$.

If the data fits in a single process, a 1D real-to-complex FFT of a vector of length *n*, followed by a 1D complex-to-real FFT results in the original vector being scaled by *n*.

For a real-to-complex FFT of a 2D real array of size n x m, followed by a complex-to-real FFT, the original array is scaled by n x m.

Similarly, a real-to-complex FFT applied to a 3D real array of size n x m x k, followed by a complex-to-real FFT, results in the original array being scaled by n x m x k.

Complex Data Packed Representation

1D Real-to-Complex Periodic Fourier Transform: The periodic Fourier Transform of a real sequence X[i], i=0,...,N-1 is Hermitian (exhibits conjugate symmetry around its middle point).

If X[i],i=0,...,N-1 are the complex values of the Fourier Transform, then

```
X[i] = conj(X[N-i]), i=1,...,N-1       (eq. 1)
```

Consider, for example, the real sequence:

```
X =

0
1
2
3
4
5
6
7
```

Its Fourier Transform is:

```
X =

28.0000
-4.0000 + 9.6569i
-4.0000 + 4.0000i
-4.0000 + 1.6569i
-4.0000
-4.0000 - 1.6569i
-4.0000 - 4.0000i
-4.0000 - 9.6569i
```

As you can see:

```
X[1] = conj(X[7])
X[2] = conj(X[6])
X[3] = conj(X[5])
X[4] = conj(X[4]) (i.e., X[4] is real)
X[5] = conj(X[3])
X[6] = conj(X[2])
X[7] = conj(X[1])
```

Because of the Hermitian symmetry, only N/2+1 = 5 values of the complex sequence X need to be calculated and stored. The rest can be computed from (eq. 1).

Note that X[0] and X[N/2] are real-valued so they can be grouped together as one complex number. In fact, S3L stores the sequence X as:

```
X[0]    X[N/2]
X[1]
X[2]

or

X =
28.0000 - 4.0000i
-4.0000 + 9.6569i
-4.0000 - 4.0000i
-4.0000 + 1.6569i
```

The first line in this example represents the real and imaginary parts of a complex number.

To summarize, in S3L, the Fourier transform of a real-valued sequence of length N (where N is even) is stored as a real sequence of length N. This is equivalent to a complex sequence of length N/2.

2D Fourier Transform: The method used for 2D FFTs is similar to that used for 1D FFTs. When transforming each of the array columns, only half of the data is stored.

3D Real to Hermitian FFT: As with the 1D and 2D FFTs, no extra storage is required for the 3D FFT of real data, since advantage is taken of all possible symmetries. For an array `a`(M,N,K), the result is packed in the complex `b`(M/2,N,K) array. Hermitian symmetries exist along the planes `a`(0,:,:) and `a`(M/2,:,:) and along dimension 1.

See the `rc_fft.c` and `rc_fft.f` program examples for illustrations of these concepts. The paths for these online examples are provided at the end of this section.

Syntax

The C and Fortran syntax for `S3L_rc_fft` and `S3L_cr_fft` is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_rc_fft(a, setup_id)
S3L_cr_fft(a, setup_id)
    S3L_array_t         a
    int                 setup_id
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_rc_fft(a, setup_id, ier)
S3L_cr_fft(a, setup_id, ier)
    integer*8           a
    integer*4           setup_id
    integer*4           ier
```

Input

The `S3L_rc_fft` and `S3L_cr_fft` functions accept the following arguments as input:

- `a` – S3L array handle for a parallel real array. For `S3L_rc_fft`, the contents of `a` are real values. For `S3L_cr_fft`, they are the packed representation of a complex array. Upon successful completion, both routines overwrite `a` with the results of the forward or inverse FFT. See the Output section for a discussion of the use of `a` for output.
- `setup_id` – Scalar integer variable. Use the value returned by the `S3L_rc_fft_setup` call for this argument.

Output

The S3L_rc_fft and S3L_cr_fft functions use the following arguments for output:

- a – S3L array handle for a parallel real array. Upon successful completion, S3L_rc_fft overwrites a with the packed representation of the complex result of the forward FFT. S3L_cr_fft overwrites a with the real result of the inverse FFT.
- ier (Fortran only) – When called from a Fortran program, these functions return error status in ier.

Error Handling

On success, S3L_rc_fft and S3L_cr_fft return S3L_SUCCESS.

The following condition will cause these functions to terminate and return the associated error code:

- S3L_ERR_ARG_SETUP – Invalid setup_id value.

Examples

/opt/SUNWhpc/examples/s3l/rc_fft/rc_fft.c

/opt/SUNWhpc/examples/s3l/rc_fft-f/rc_fft.f

Related Functions

S3L_rc_fft_setup(3)

S3L_rc_fft_free_setup(3)

S3L_rc_fft_free_setup

Description

S3L_rc_fft_free_setup deallocates internal memory associated with setup_id by a previous call to S3L_rc_fft_setup.

Syntax

The C and Fortran syntax for S3L_rc_fft_free_setup is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_rc_fft_free_setup(setup_id)
    int                 setup_id
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_rc_fft_free_setup(setup_id, ier)
    integer*4           setup_id
    integer*4           ier
```

Input

S3L_rand_sparse accepts the following argument as input:

- setup_id – Scalar integer variable. Use the value returned by the S3L_rc_fft_setup call for this argument.

Output

S3L_rc_fft_free_setup uses the following argument for output:

- ier (Fortran only) – When called from a Fortran program, S3L_rc_fft_free_setup returns error status in ier.

Error Handling

On success, S3L_rc_fft_free_setup returns S3L_SUCCESS.

The following condition will cause `S3L_rc_fft_free_setup` to terminate and return the associated error code:

- `S3L_ERR_ARG_SETUP` – Invalid `setup_id` value.

Examples

`/opt/SUNWhpc/examples/s3l/rc_fft/rc_fft.c`

`/opt/SUNWhpc/examples/s3l/rc_fft-f/rc_fft.f`

Related Functions

`S3L_rc_fft_setup(3)`

`S3L_rc_fft(3)`

S3L_rc_fft_setup

Description

`S3L_rc_fft_setup` allocates a real-to-complex FFT setup that includes the twiddle factors necessary for the computation and other internal structures. This setup depends only on the dimensions of the array whose FFT needs to be computed, and can be used both for the forward (real-to-complex) and inverse (complex-to-real) FFTs. Therefore, to compute multiple real-to-complex or complex-to-real Fourier transforms of different arrays whose extents are the same, the `S3L_rc_fft_setup` function has to be called only once.

Syntax

The C and Fortran syntax for `S3L_rc_fft_setup` is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_rc_fft_setup(a, setup_id)
    S3L_array_t        a
    int                *setup_id
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_rc_fft_setup(a, setup_id, ier)
    integer*8          a
    integer*4          setup_id
    integer*4          ier
```

Input

`S3L_rc_fft_setup` accepts the following argument as input:

- `a` – S3L array handle for a parallel array that will be the subject of subsequent transform operations.

Output

`S3L_rc_fft_setup` uses the following argument for output:

- `ier` (Fortran only) – When called from a Fortran program, `S3L_rc_fft_setup` returns error status in `ier`.
- `setup_id` – On output, it contains an integer value that can be used in subsequent calls to `S3L_rc_fft`, `S3L_cr_fft`, and `S3L_rc_fft_free_setup`.

Error Handling

On success, `S3L_rc_fft_setup` returns `S3L_SUCCESS`.

The following conditions will cause `S3L_rc_fft_setup` to terminate and return the associated error code:

- `S3L_ERR_ARG_RANK` – The rank of array `a` is not 1, 2, or 3.
- `S3L_ERR_ARG_NREAL` – The data type of `a` is not `real`.
- `S3L_ERR_ARG_NEVEN` – Some of the extents of `a` are not even.
- `S3L_ERR_ARG_EXTENTS` – The extents of `a` are not correct for the rank of `a` and the number of processors over which `a` is distributed. This relationship is summarized below:
  - If `a` is 1D, its length must be divisible by 4*sqr(*np*) where *np* is the number of processes over which the `a` is distributed.
  - If `a` is 2D, its extents must both be divisible by 2**np*.
  - If `a` is 3D, its first extent must be even and its last two extents must both be divisible by 2**np*.

Examples

`/opt/SUNWhpc/examples/s3l/rc_fft/rc_fft.c`

`/opt/SUNWhpc/examples/s3l/rc_fft-f/rc_fft.f`

Related Functions

`S3L_rc_fft(3)`

`S3L_cr_fft(3)`

`S3L_rc_fft_free_setup(3)`

`S3L_read_array` and `S3L_read_sub_array`

Description

`S3L_read_array` causes the process with MPI rank 0 to read the contents of a distributed array from a local file and distribute them to the processes that own the parts (subgrids) of the array. The local file is specified by the `filename` argument.

S3L_read_sub_array reads a specific section of the array, within the limits specified by the lbounds and ubounds arguments. The strides argument specifies the stride along each axis; it must be greater than zero. The format argument is a string that specifies the format of the file to be read. It can be either "ascii" or "binary".

The values of lbounds and ubounds should refer to zero-based indexed arrays for the C interface and to one-based indexed arrays for the Fortran interface.

Syntax

The C and Fortran syntax for S3L_read_array and S3L_read_sub_array is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_read_array(a, filename, format)
S3L_read_sub_array(a, lbounds, ubounds, strides, filename,
format)
    S3L_array_t        a
    int                *lbounds
    int                *ubounds
    int                *strides
    char               *filename
    char               *format
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_read_array(a, filename, format, ier)
S3L_read_sub_array(a, lbounds, ubounds, strides, filename,
format,ier)
    integer*8          a
    integer*4          lbounds(*)
    integer*4          ubounds(*)
    integer*4          strides(*)
    character*1        filename(*)
    character*1        format(*)
    integer*4          ier
```

Input

S3L_read_array and S3L_read_sub_array accept the following arguments as input:

- a – S3L array handle for the parallel array to be read. This array handle was returned when the array was declared.
- lbounds – Integer vector specifying the lower bounds of the indices of a along each of its axes.
- ubounds – Integer vector specifying the upper bounds of the indices of a along each of its axes.
- strides – Integer vector specifying the strides on the indices of a along each of its axes.
- filename – Scalar character variable specifying the name of the file from which the parallel array will be read.
- format – Scalar character variable specifying the format of the data to be read. The value can be either "ascii" or "binary".

Output

S3L_read_array and S3L_read_sub_array use the following argument for output:

- ier (Fortran only) – When called from a Fortran program, S3L_read_array and S3L_read_sub_array return error status in ier.

Error Handling

On success, S3L_read_array and S3L_read_sub_array return S3L_SUCCESS.

S3L_read_array and S3L_read_sub_array perform generic checking of the validity of the arrays they accept as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause the function to terminate and return the associated error code:

- S3L_ERR_ARG_RANGE_INV – The given range of indices is invalid:
  - A lower bound is less than the smallest index of the array.
  - An upper bound is greater than the largest index of an array along the given axis.
  - A lower bound is greater than the corresponding upper bound.
  - A stride is negative or zero.
- S3L_ERR_FILE_OPEN – Failed to open the file with the file name provided.
- S3L_ERR_EOF – Encountered EOF while reading an array from a file.
- S3L_ERR_IO_FORMAT – Format is not one of "ascii" or "binary".
- S3L_ERR_IO_FILENAME – The file name is equal to the NULL string (C/C++) or to an empty string (F77/F90).

Examples

/opt/SUNWhpc/examples/s3l/io/ex_io.c

/opt/SUNWhpc/examples/s3l/io-f/ex_io.f

Related Functions

S3L_print_array(3)

S3L_write_array(3)

S3L_read_sparse

Description

S3L_read_sparse reads sparse matrix data from an ASCII file and distributes the data to all participating processes. Upon successful completion, S3L_read_sparse returns an S3L array handle in A that represents the distributed sparse matrix.

S3L_read_sparse supports the following sparse matrix storage formats:

- S3L_SPARSE_COO – Coordinate format.
- S3L_SPARSE_CSR – Compressed Sparse Row format.
- S3L_SPARSE_CSC – Compressed Sparse Column format.
- S3L_SPARSE_VBR – Variable Block Row format.

Each of these four format files contains three sections. They begin with a header section, followed by two data sections.

The header section can be used for comments. It consists of one or more lines, each of which begins with the percent character (%).

The first data section consists of a single line. It contains a list of integers denoting the total number of matrix rows, columns, nonzero elements and, in the case of the S3L_SPARSE_VBR format for blocked matrices, the total number of block rows, block columns, and nonzero blocks.

The second data section contains the numerical data of the matrix. For its data layout, the following specifies the general rules to apply:

- Blank lines may be present anywhere in the file.
- Numerical data is separated by one or more blanks.
- Real data entries must be in floating-point decimal format or, optionally, in the e,E-format exponential notation common to C and Fortran.
- All indices must be stored using zero-based indexing when called by C or C++ applications and one-based indexing when called by F77 or F90 applications.

The details of the layout are given below for each of the sparse formats.

S3L_SPARSE_COO

Under the S3L_SPARSE_COO format, the first data section lists three integers, m, n, and nnz. m and n indicate the number of rows and columns in the matrix, respectively. nnz indicates the total number of nonzero values in the matrix.

The second data section stores all nonzero values in the matrix, one value per line. The first two entries on the line are the row and column indices for that value and the third entry is the value itself.

For example, the following 4x6 matrix:

```
 3.14      0         0        20.04      0         0
 0        27         0         0        -0.6       0
 0         0        -0.01      0         0         0
-0.031     0         0         0.08      0       314.0
```

could have the following layout in an S3L_SPARSE_COO file, using zero-based indexing:

```
% Example: 4x6 sparse matrix in an S3L_SPARSE_COO file,
% row-major order, zero-based indexing:
%
4   6   8
0   0   3.140e+00
0   3   2.004e+01
1   1   2.700e+01
1   4  -6.000e-01
2   2  -1.000e-02
3   0  -3.100e-02
3   3   8.000e-02
3   5   3.140e+02
```

The layout used for this example is row-major, but any order is supported, including random. The next two examples show this same 4x6 matrix stored in two S3L_SPARSE_COO files, both in random order. The first example illustrates zero-based indexing and the second, one-based indexing.

```
% Example: 4x6 sparse matrix in an S3L_SPARSE_COO file,
% random-major order, zero-based indexing:
%
4   6   8
3   5   3.140e+02
1   1   2.700e+01
0   3   2.004e+01
3   3   8.000e-02
2   2  -1.000e-02
0   0   3.140e+00
1   4  -6.000e-01
3   0  -3.100e-02
```

```
% Example: 4x6 sparse matrix in an S3L_SPARSE_COO file,
% random-major order, one-based indexing:
%
4   6   8
4   4   8.000e-02
2   2   2.700e+01
1   1   3.140e+00
4   1  -3.100e-02
3   3  -1.000e-02
4   6   3.140e+02
1   4   2.004e+01
2   5  -6.000e-01
```

MatrixMarket Notes

Under S3L_SPARSE_COO format, S3L_read_sparse can also read data supplied in either of two Coordinate formats distributed by MatrixMarket (http://gams.nist.gov/MatrixMarket/). The two supported MatrixMarket formats are real general and complex general.

MatrixMarket files always use one-based indexing. Consequently, they can only be used directly by Fortran programs, which also implement one-based indexing. For a C or C++ program to use a MatrixMarket file, it must call the F77 application program interface. The program example ex_sparse.c illustrates an F77 call from a C program. See the Examples section for the path to this sample program.

S3L_SPARSE_CSR

Under S3L_SPARSE_CSR format, the first data section is the same as the S3L_SPARSE_COO format. The second data section stores the S3L_SPARSE_CSR data structure in two integer arrays, ptr and indx, and one floating-point array, val. It contains, in order, the row start pointers, the column indices, and the nonzero elements.

For example, the same 4x6 sparse matrix used in the previous example could be stored under S3L_SPARSE_CSR in the manner (using zero-based indexing):

```
% Example: 4x6 sparse matrix in an S3L_SPARSE_CSR format
%
 4  6  8
 0  2  4  5  8
  0      3      4     1     2      0      3       5
 3.14  20.04  27.0  –0.6  –0.01  –0.031  0.08  314.0
```

S3L_SPARSE_CSC

The S3L_SPARSE_CSC format is almost identical to the S3L_SPARSE_CSR format except with a column orientation. Specifically, the first data section is the same as the S3L_SPARSE_CSR, while the second data section stores, in order, the column start pointers, the row indices, and the nonzero elements.

Using the same 4x6 sparse matrix example as before, a possible data layout under S3L_SPARSE_CSC follows:

```
% Example: 4x6 sparse matrix in an S3L_SPARSE_CSC format
%
 4  6  8
 0  2  3  4  6  7  8
  0      3       1     2      0     3      1      3
 3.14  -0.031  27.0  -0.01  20.04  0.08  -0.6  314.0
```

S3L_SPARSE_VBR

Unlike the first three sparse formats, which provide natural layouts for point sparse matrices, S3L_SPARSE_VBR format is well-suited to represent matrices with a block structure.

Under S3L_SPARSE_VBR format, the first data section contains six integers. They are, in order, m, n, nnz, bm, bn, and bnnz. The first three indicate the number of point rows, point columns, and point nonzero elements of the matrix. The other three represent the block partitionings of the matrix—that is, the number of block rows, block columns, and nonzero block entries of the matrix.

The second data section stores the S3L_SPARSE_VBR data structure in five integer arrays and one floating-point array. They are:

rptr	Integer array containing the row-partitioning pointers.
cptr	Integer array containing the column-partitioning pointers.
bptr	Integer array containing the block row start pointers.
bindx	Integer array containing the block column indices.
indx	Integer array containing the block start pointers.
val	Floating-point array containing the nonzero block entries, where each block entry is stored as a dense matrix, column by column.

To illustrate the data layout, consider the following 5x8 sparse matrix with variable block partitioning.

```
     0  1   2  3  4    5    6   7   8
   +------+---------+----+--------+
 0 | 1  3  5 | 0  0 |  9 |  0   0 |
 1 | 2  4  6 | 0  0 | 10 |  0   0 |
   +---------+------+----+--------+
 2 | 0  0  0 | 7  8 | 11 |  0   0 |
   +------+---------+----+--------+
 3 | 0  0  0 | 0  0 | 12 | 14  16 |
 4 | 0  0  0 | 0  0 | 13 | 15  17 |
   +------+---------+----+--------+
 5
```

It could be stored in `S3L_SPARSE_VBR` format as follows:

```
% Example: 5x8 sparse matrix in an S3L_SPARSE_VBR format
%
 5  8  17  3  4  6

 0  2  3  5
 0  3  5  6  8

 0  2  4  6
 0  2  1  2  2  3
 0  6  8  10  11  13  17

 1.0  2.0  3.0  4.0  5.0  6.0  9.0  10.0
 7.0  8.0  11.0
 12.0  13.0  14.0  15.0  16.0  17.0
```

Syntax

The C and Fortran syntax for `S3L_read_sparse` is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_read_sparse(A, spfmt, m, n, nnz, type, fname, dfmt)
    S3L_array_t                *A
    S3L_sparse_storage_t       spfmt
    int                        m
    int                        n
    int                        nnz
    S3L_data_type              type
    char                       *fname
    char                       *dfmt
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_read_sparse(A, spfmt, m, n, nnz, type, fname, dfmt, ier)
    integer*8                  A
    integer*4                  spfmt
    integer*4                  m
    integer*4                  n
    integer*4                  nnz
    integer*4                  type
    character*1                fname
    character*1                dfmt
    integer*4                  ier
```

Input

`S3L_read_sparse` accepts the following arguments as input:

- `spfmt` – Specifies the sparse storage format used for representing the sparse matrix. The supported formats are `S3L_SPARSE_COO`, `S3L_SPARSE_CSR`, `S3L_SPARSE_CSC`, and `S3L_SPARSE_VBR`.
- `m` – Indicates the total number of rows in the sparse matrix.
- `n` – Indicates the total number of columns in the sparse matrix.
- `nnz` – Indicates the total number of nonzero elements in the sparse matrix.
- `type` – Indicates the type of the sparse array, which must be `S3L_float`, `S3L_double`, `S3L_complex`, or `S3L_double_complex`.

- fname – Scalar character variable that names the ASCII file containing the sparse matrix data.
- dfmt – Specifies the format of the data to be read from the data file. Allowed strings are 'ascii' and 'ASCII'.

Output

S3L_read_sparse uses the following arguments for output:

- A – S3L array handle for the global general sparse matrix output.
- ier (Fortran only) – When called from a Fortran program, S3L_read_sparse returns error status in ier.

Error Handling

On success, S3L_read_sparse returns S3L_SUCCESS.

The S3L_read_sparse routine performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause S3L_read_sparse to terminate and return the associated error code:

- S3L_ERR_ARG_EXTENTS – Invalid m, n, or nnz. These arguments must all be > 0.
- S3L_ERR_SPARSE_FORMAT – Invalid storage format. It must be S3L_SPARSE_COO, S3L_SPARSE_CSR, S3L_SPARSE_CSC, or S3L_SPARSE_VBR.
- S3L_ERR_ARG_DTYPE – Invalid data type. It must be S3L_float, S3L_double, S3L_complex, or S3L_double_complex.
- S3L_ERR_IO_FILENAME – Invalid file name.
- S3L_ERR_IO_FORMAT – Invalid data file format. The error could be either of the following:
  - The dfmt value supplied was not 'ascii' or 'ASCII'.
  - An unsupported MatrixMarket format was supplied. When a MatrixMarket file is used, the first line of its comment section must contain either the words 'real general' or 'complex general'.
- S3L_ERR_FILE_OPEN – Failed to open the data file; the file either does not exist or the name is specified incorrectly.
- S3L_ERR_EOF – The input data ended before expected.

Examples

/opt/SUNWhpc/examples/s3l/sparse/ex_sparse.c

/opt/SUNWhpc/examples/s3l/sparse-f/ex_sparse.f

Related Functions

S3L_convert_sparse(3)

S3L_declare_sparse(3)

S3L_matvec_sparse(3)

S3L_rand_sparse(3)

S3L_reduce

Description

S3L_reduce performs a predefined reduction function over all elements of a parallel array. The array is described by the S3L array handle argument A. The argument op specifies the type of reduction operations, which can be one of the following:

- S3L_OP_SUM – Finds the sum of all the elements.
- S3L_OP_MIN – Finds the smallest value among all the elements.
- S3L_OP_MAX – Finds the largest value among all the elements.

Syntax

The C and Fortran syntax for S3L_reduce is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_reduce(A, op, res)
    S3L_array_t       A
    S3L_op_type       op
    void              *res
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_reduce(A, op, res, ier)
    integer*8         A
    integer*4         op
    <type>            res
    integer*4         ier
```

where `<type>` is one of: `real*4`, `real*8`, `complex*8`, or `complex*16`.

Input

`S3L_reduce` accepts the following arguments as input:

- `A` – Array handle for the parallel array to be reduced.
- `op` – Specifies the type of operation to be performed.

Output

`S3L_reduce` uses the following arguments for output:

- `res` – Contains the result of the reduction function.
- `ier` (Fortran only) – When called from a Fortran program, `S3L_reduce` returns error status in `ier`.

Error Handling

On success, S3L_reduce returns S3L_SUCCESS.

S3L_reduce performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause the function to terminate and return the associated error code:

- S3L_ERR_ARG_OP – Requested operation is not supported.
- S3L_ERR_ARG_DTYPE – Invalid data type.

Examples

/opt/SUNWhpc/examples/s3l/utils/cshift_reduce.c

/opt/SUNWhpc/examples/s3l/utils-f/cshift_reduce.f

Related Function

S3L_reduce_axis(3)

S3L_reduce_axis

Description

S3L_reduce_axis applies a predefined reduction operation along a given axis of a parallel S3L array. If *n* is the rank (number of dimensions) of a, the result b is a parallel array of rank *n*-1. The argument op specifies the operation to be performed. The value of op must be one of:

- S3L_OP_SUM – The sum reduction operation is applied.
- S3L_OP_MIN – The minimum reduction operation is applied.
- S3L_OP_MAX – The maximum reduction operation is applied.

Syntax

The C and Fortran syntax for `S3L_reduce_axis` is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_reduce_axis(a, op, axis, b)
    S3L_array_t         a
    S3L_op_type         op
    int                 axis
    S3L_array_t         b
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_reduce_axis(a, op, axis, b, ier)
    integer*8           a
    integer*4           op
    integer*4           axis
    integer*8           b
    integer*4           ier
```

Input

`S3L_reduce_axis` accepts the following arguments as input:

- `a` – S3L array handle for the parallel array on which the operation will be applied.
- `op` – Predefined constant specifying the operation to be applied.
- `axis` – Specifies the axis along which the reduction will be performed. When `S3L_reduce_axis` is called from a C program, this value must reflect zero-based indexing of the array axes. If called from a Fortran program, it must reflect one-based indexing.

Output

S3L_reduce_axis uses the following arguments for output:

- b – S3L array handle for the parallel array that will contain the result of the reduction.
- ier (Fortran only) – When called from a Fortran program, S3L_reduce_axis returns error status in ier.

Error Handling

On success, S3L_reduce_axis returns S3L_SUCCESS.

S3L_reduce_axis performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause the function to terminate and return the associated error code:

- S3L_ERR_ARG_OP – Requested operation is not supported.
- S3L_ERR_MATCH_EXTENTS – The extents of a and b do not match. For example, if a is a 4D array with extents n1 x n2 x n3 x n4, and axis is equal to 2 (Fortran interface), b must be a 3D array with extents n1 x n3 x n4.
- S3L_ERR_MATCH_RANK – The rank of b is not equal to the rank of a minus 1.
- S3L_ERR_ARG_AXISNUM – The axis specified is not valid; that is, it is either larger than the rank of the array or smaller than 1 (Fortran interface). For the C interface, the axis value is equal to or larger than the rank of the array or smaller than 0.

Examples

/opt/SUNWhpc/examples/s3l/utils/cshift_reduce.c

/opt/SUNWhpc/examples/s3l/utils-f/cshift_reduce.f

Related Function

S3L_reduce(3)

S3L_set_array_element, S3L_get_array_element, S3L_set_array_element_on_proc, and S3L_get_array_element_on_proc

Description

The four subroutines described in this section enable the user to alter (set) and retrieve (get) individual elements of an array. Two of these subroutines also enable the user to know which process will participate in the set or get activity.

`S3L_set_array_element` assigns the value stored in `val` to a specific element of a distributed S3L array whose global coordinates are specified by `coor`. The `val` variable is colocated with the array subgrid containing the target element.

Note – Because an S3L array is distributed across a set of processes, each process has a subsection of the global array local to it. These array subsections are also referred to as *array subgrids*.

For example, if a parallel array is distributed across four processes, P0–P3, and `coor` specifies an element in the subgrid that is local to P2, the `val` that is located on P2 will be the source of the value used to set the target element.

`S3L_get_array_element` is similar to `S3L_set_array_element`, but operates in the opposite direction. It assigns the value stored in the element specified by `coor` to `val` on every process. Since `S3L_get_array_element` broadcasts the element value to every process, upon completion, every process contains the same value in `val`.

`S3L_set_array_element_on_proc` specifies which process will be the source of the value to be assigned to the target element. That is, the argument `pnum` specifies the MPI rank of a particular process. The value of the variable `val` on that process will be assigned to the target element—that is, the element whose coordinates are specified by `coor`.

Note – The MPI rank of a process is defined in the global communicator `MPI_COMM_WORLD`.

`S3L_get_array_element_on_proc` updates the variable `val` on the process whose MPI rank is supplied in `pnum`, and uses the element whose indices are given in `coor` as the source for the update.

Syntax

The C and Fortran syntax for `S3L_set_array_element` and its related routines is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_set_array_element(a, coor, val)
S3L_get_array_element(a, coor, val)
S3L_set_array_element_on_proc(a, coor, val, pnum)
S3L_get_array_element_on_proc(a, coor, val, pnum)
    S3L_array_t       a
    int               coor
    void              val
    int               pnum
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_set_array_element(a, coor, val, ier)
S3L_get_array_element(a, coor, val, ier)
S3L_set_array_element_on_proc(a, coor, val, pnum, ier)
S3L_get_array_element_on_proc(a, coor, val, pnum, ier)
    integer*8          a
    integer*4          coor
    <type>             val
    integer*4          pnum
    integer*4          ier
```

where `<type>` is `integer*4`, `real*4`, `real*8`, `complex*8`, or `complex*16`.

Input

`S3L_set_array_element`, `S3L_set_array_element_on_proc`, `S3L_get_array_element`,and `S3L_get_array_element_on_proc` accept the following arguments as input:

- `a` – Array handle describing the parallel array containing the element of interest.
- `coor` – Integer vector specifying the coordinates of an element of the distributed array `a`. This value follows zero-based notation for C/C++ programs and one-based notation for F77/F90 programs.
- `val` – Variable that holds the value to be assigned to an element of an array or that accepts the value of that element.
- `pnum` – Integer variable specifying the MPI rank of a process to supply or accept the value `val`. `pnum` is used only with `S3L_set_array_element_on_proc` and `S3L_get_array_element_on_proc.`

Output

`S3L_set_array_element`, `S3L_set_array_element_on_proc`, `S3L_get_array_element`,and `S3L_get_array_element_on_proc` use the following argument for output:

- `ier` (Fortran only) – When called from a Fortran program, these functions return error status in `ier`.

Error Handling

On success, these functions return S3L_SUCCESS.

In addition, the following conditions will cause these functions to terminate and return the associated error code and terminate:

- S3L_ERR_ARG_DTYPE – The data type of array a is not:
  - S3L_integer
  - S3L_float
  - S3L_double
  - S3L_complex
  - S3L_double_complex
- S3L_ERR_ARG_COOR – The supplied coordinates are not valid; that is, they do not specify an element of a.

Examples

/opt/SUNWhpc/examples/s3l/utils/cshift_reduce.c

/opt/SUNWhpc/examples/s3l/utils-f/cshift_reduce.f

S3L_set_process_grid

Description

S3L_set_process_grid allows the user to define various aspects of an internal process grid. It returns a process grid handle, which subsequent calls to other Sun S3L functions can use to refer to that process grid.

Syntax

The C and Fortran syntax for S3L_set_process_grid is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
S3L_set_process_grid(pgrid, rank, majorness, grid_extents,
plist_length, process_list)
    S3L_pgrid_t        *pgrid
    int                rank
    int                majorness
    int                *grid_extents
    int                plist_length
    int                *process_list
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_set_process_grid(pgrid, rank, majorness, grid_extents,
plist_length, process_list, ier)
    integer*8          pgrid
    integer*4          rank
    integer*4          majorness
    integer*4          grid_extents(*)
    integer*4          plist_length
    integer*4          process_list(*)
    integer*4          ier
```

Input

`S3L_set_process_grid` accepts the following arguments as input:

- `rank` – Specifies the rank of the process grid to be created. The range of legal values for `rank` is the same as for S3L arrays, which is 1 <= `rank` <= 31.
- `majorness` – Uses one of the following predefined values to specify the order of loop execution:
  - `S3L_MAJOR_ROW` – Rightmost axis varies fastest.
  - `S3L_MAJOR_COLUMN` – Leftmost axis varies fastest.
- `grid_extents` – Integer array whose length equals the rank of the process grid. It contains a list of process grid extents. Each element in the array specifies the extent of the corresponding process grid axis. Note that axis indexing is zero-based for the C/C++ interface and one-based for the F77/F90 interface, as follows:

- When called from a C or C++ application, the first element of grid_extents corresponds to axis 0, the second element to axis 1, and so forth.
- When called from an F77 or F90 application, the first element corresponds to axis 1, the second to axis 2, and so forth.

- plist_length – Length of process list. Note that, if the product of all grid extents is N and if a value greater than N is specified for plist_length, only the first N elements of process_list will be used.
- process_list – Array of integers of length plist_length, which contains a list of the processes that will constitute the process grid. For example, if you are running your program on four MPI processes but you wish to create a process grid consisting of only processes 1 and 3, you should set plist_length to 2 and have

```
process_list[0] = 1
process_list[1] = 3
```

  If plist_length is 0, process_list is ignored. The process grid is then created using all available processes in MPI_COMM_WORLD.

Output

S3L_set_process_grid uses the following arguments for output:

- pgrid – The process grid handle returned by the function.
- ier (Fortran only) – When called from a Fortran program, S3L_set_process_grid returns error status in ier.

Error Handling

On success, S3L_set_process_grid returns S3L_SUCCESS.

S3L_set_process_grid performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes:

In addition, the following conditions will cause the function to terminate and return the associated error code:

- S3L_ERR_ARG_RANK – Invalid rank argument value.
- S3L_ERR_ARG_MAJOR – Invalid majorness value.
- S3L_ERR_PGRID_EXTENTS – Grid size (calculated as product of process grid extents) is less than 1.

- S3L_ERR_ARRTOOSMALL – plist_length is greater than 0 but less than the size of the grid (calculated from the product of process grid extents).
- S3L_ERR_ARG_NULL – In a C/C++ program, plist_length is greater than 0, but process_list is a NULL pointer.

Examples

/opt/SUNWhpc/examples/s3l/utils/scalapack_conv.c

/opt/SUNWhpc/examples/s3l/utils-f/scalapack_conv.f

Related Functions

S3L_declare_detailed(3)

S3L_free_process_grid(3)

S3L_set_safety

Description

The S3L safety mechanism offers two types of services:

- It performs error checking and reporting during execution of S3L routines.
- It synchronizes S3L processes so that, when an error is detected, the section of code associated with the error can be more readily identified.

The S3L safety mechanism can be set to operate at any one of four levels, which are described in TABLE 2-17.

TABLE 2-17 Setting S3L Safety Levels

Safety Level	Description
0	Turns the safety mechanism off. Explicit synchronization and error checking are not performed. This level is appropriate for production runs of code that have already been thoroughly tested.
2	Detects potential race conditions in multithreaded S3L operations on parallel arrays. To avoid race conditions, an S3L function locks all parallel array handles in its argument list before proceeding. This safety level causes warning messages to be generated if more than one S3L function attempts to use the same parallel array at the same time.
5	In addition to checking for and reporting level 2 errors, performs explicit synchronization before and after each call and locates each error with respect to the synchronization points. This safety level is appropriate during program development or during runs for which a small performance penalty can be tolerated.
9	Checks for and reports all level 2 and level 5 errors, as well as errors generated by any lower levels of code called from within S3L. Performs explicit synchronization in these lower levels of code and locates each error with respect to the synchronization points. This level is appropriate for detailed debugging following the occurrence of a problem.

The S3L safety mechanism can be controlled in either of two ways:

- By setting the environment variable `S3L_SAFETY`.
- By using the call `S3L_set_safety` in a program.

To set the S3L safety level using the `S3L_SAFETY` environment variable, issue the command:

```
setenv S3L_SAFETY number
```

where number is one of: 0, 2, 5, or 9.

The value of `S3L_SAFETY` is read in when `S3L_init()` is called. This value can be overridden by a call to `S3L_set_safety()` at any point in the user’s program. When `S3L_set_safety()` is called, its value overrides `S3L_SAFETY` until the program completes.

If `S3L_set_safety()` is called again, the new safety level value will override the the previous call. In other words, `S3L_set_safety()` can be called multiple times within a single program. The next time the program is run, the safety level specified by `S3L_SAFETY` will be reasserted.

Syntax

The C and Fortran syntax for `S3L_set_safety` is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_set_safety(n)
    int                 n
```

F77/F90 Syntax

```
include ‘s3l/s3l-f.h’
include ‘s3l/s3l_errno-f.h’
subroutine
S3L_set_safety(n, ier)
    integer*4           n
    integer*4           ier
```

Input

`S3L_set_safety` accepts the following argument as input:

- `n` – An integer specifying one of four safety levels: `0`, `2`, `5`, and `9`. See the Description section for details.

Output

`S3L_set_safety` uses the following argument for output:

- `ier` (Fortran only) – When called from a Fortran program, `S3L_set_safety` returns error status in `ier`.

Error Handling

On success, `S3L_set_safety` returns `S3L_SUCCESS`.

On error, the following condition will cause the function to terminate and return the associated error code:

- `S3L_ERR_SAFELEV_INVAL` – Value specified for `n` is invalid.

Examples

/opt/SUNWhpc/examples/s3l/utils/copy_array.c

/opt/SUNWhpc/examples/s3l/utils-f/copy_array.f

Related Function

S3L_get_safety(3)

S3L_setup_rand_fib

Description

`S3L_setup_rand_fib` initializes the Lagged-Fibonacci random number generator's (LFG's) state table with the fixed parameters:

l = 17, k = 5, m = 32.

The state table is initialized in a manner that ensures that the random numbers generated for each node are from a different period of the LFG. A Linear Multiplicative Generator (LMG) is used to initialize the noncritical elements of the state table.

Use `S3L_free_rand_fib` to deallocate an LFG setup.

Syntax

The C and Fortran syntax for `S3L_setup_rand_fib` is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_setup_rand_fib(setup_id, seed)
    int                 *setup_id
    int                 seed
```

F77/F90 Syntax

```
include ‘s3l/s3l-f.h’
include ‘s3l/s3l_errno-f.h’
subroutine
S3L_setup_rand_fib(setup_id, seed, ier)
    integer*4           setup_id
    integer*4           seed
    integer*4           ier
```

Input

`S3L_setup_rand_fib` accepts the following argument as input:

- `seed` – An integer value used to initialize the LMG that initializes the noncritical elements of the LFG’s state table.

Output

`S3L_setup_rand_fib` uses the following arguments for output:

- `setup_id` – On output, `setup_id` contains an index that can be used as input to `S3L_rand_fib`.
- `ier` (Fortran only) – When called from a Fortran program, `S3L_setup_rand_fib` returns error status in `ier`.

Error Handling

On success, `S3L_setup_rand_fib` returns `S3L_SUCCESS`.

Examples

/opt/SUNWhpc/examples/s3l/rand_fib/rand_fib.c

/opt/SUNWhpc/examples/s3l/rand_fib-f/rand_fib.f

Related Functions

S3L_free_rand_fib(3)

S3L_rand_fib(3)

S3L_sort, S3L_sort_up, S3L_sort_down, S3L_sort_detailed_up, and S3L_sort_detailed_down

Description

The S3L_sort function sorts the elements of a one-dimensional array in ascending order.

S3L_sort_up and S3L_sort_down sort the elements of one-dimensional or multidimensional array in ascending and descending order, respectively.

Note – S3L_sort is a special case of S3L_sort_up.

When A is one-dimensional, the result is a vector that contains the same elements as A, but arranged in ascending order (S3L_sort or S3L_sort_up) or descending order. For example, if A contains:

```
| 7   2   4   3   1   8   6   9   5 |
```

calling S3L_sort or S3L_sort_up would produce the result:

```
| 1   2   3   4   5   6   7   8   9 |
```

If A is multidimensional, the elements are sorted into an index-based sequence, starting with the first row-column index and progressing through the row indices first before advancing to the next column index position.

For example, if A contains:

```
| 6   2   7 |
|           |
| 1   4   3 |
|           |
| 9   5   8 |
```

S3L_sort_up would produce the result:

```
| 1   4   7 |
|           |
| 2   5   8 |
|           |
| 3   6   9 |
```

and S3L_sort_down would produce the result:

```
| 9   6   3 |
|           |
| 8   5   2 |
|           |
| 7   4   1 |
```

S3L_sort_detailed_up and S3L_sort_detailed_down sort the elements of one-dimensional or multidimensional arrays in ascending and descending order along the axis specified by the axis argument.

Note – The value of the axis argument is language dependent. For C/C++ applications, it must be zero-based and for F77/F90 applications, it must be one-based.

If the array referenced by A contains:

```
| 6   2   7 |
|           |
| 1   4   3 |
|           |
| 9   5   8 |
```

and a C program calls S3L_sort_detailed_up with axis = 0, upon completion, A will contain:

```
| 1   2   3 |
|           |
| 6   4   7 |
|           |
| 9   5   8 |
```

Or, if a C program calls S3L_sort_detailed_up with axis = 1, upon completion, A will contain:

```
| 2   6   7 |
|           |
| 1   3   4 |
|           |
| 5   8   9 |
```

If these calls were made from an F77 or F90 program, the axis values would need to be one greater (that is, 1 and 2, respectively) to achieve the same results.

Syntax

The C and Fortran syntax for these functions is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_sort(A)
S3L_sort_up(A)
S3L_sort_down(A)
S3L_sort_detailed_up(A, axis)
S3L_sort_detailed_down(A, axis)
    S3L_array_t       A
    int               axis
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_sort(A, ier)
S3L_sort_up(A, ier)
S3L_sort_down(A, ier)
S3L_sort_detailed_up(A, axis, ier)
S3L_sort_detailed_down(A, axis, ier)
    integer*8         A
    integer*4         axis
    integer*4         ier
```

Input

The family of sort functions accept one or both of the following arguments as input:

- A – For S3L_sort, A must be a one-dimensional array. For S3L_sort_up, S3L_sort_down, S3L_sort_detailed_up, and S3L_sort_detailed_down, A can be one-dimensional or multidimensional.
- axis – Used with S3L_sort_detailed_up and S3L_sort_detailed_down to specify which axis of A is to be sorted. If A is one-dimensional, axis must be 0 (for C/C++) or 1 (for F77/F90). It may not be used in S3L_sort, S3L_sort_up, or S3L_sort_down calls.

Output

These sort functions use the following arguments for output:

- A – On output, A contains the sorted array.
- ier (Fortran only) – When called from a Fortran program, these functions return error status in ier.

Error Handling

On success, the sort functions return S3L_SUCCESS.

These functions all check the arrays they accept as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause the functions to terminate and return the associated code:

- S3L_ERR_ARG_DTYPE – The type of the array is invalid. It must be S3L_integer, S3L_long_integer, S3L_float, or S3L_double.
- S3L_ERR_ARG_AXISNUM – The axis argument has an invalid value. The correct values for axis are:
  - 0 <= axis < rank of a (C/C++)
  - 0 < axis <= rank of a (F77/F90)

Examples

/opt/SUNWhpc/examples/s3l/sort/sort1.c

/opt/SUNWhpc/examples/s3l/sort/ex_sort2.c

/opt/SUNWhpc/examples/s3l/sort-f/sort1.f

Related Functions

S3L_grade_up(3)

S3L_grade_detailed_down(3)

S3L_grade_detailed_up(3)

S3L_sort_detailed

Description

S3L_sort_detailed enables the user to sort S3L arrays on one or more dimensions with detailed control. The axis and options arguments support:

- specifying the axis along which the sort will be done (for multidimensional arrays).
- specifying the sort direction—descending or ascending order
- specifying either radix or quick-sort as the sort algorithm

Syntax

The C and Fortran syntax for S3L_sort_detailed is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_sort_detailed(A, axis, options)
    S3L_array_t        A
    int                axis
    int                *options
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_sort_detailed(A, axis, options, ier)
    integer*8          A
    integer*4          axis
    integer*4          options
    integer*4          ier
```

Input

S3L_sort_detailed accepts the following arguments as input:

- A – On entry, A contains the S3L array to be sorted.
- axis – Integer value that specifies which axis of A is to be sorted. If A is one-dimensional, axis must be 0 (for C/C++) or 1 (for F77/F90).
- options – Integer vector with two elements. These elements are used as follows:

  The first element specifies the algorithm to be used. It can be either S3L_QUICKSORT or S3L_RADIXSORT.

  The second element specifies the sort direction. It can be either S3L_DOWN or S3L_UP.

Output

S3L_sort_detailed uses the following arguments for output:

- A – On exit, A contains the sorted array.
- ier (Fortran only) – When called from a Fortran program, S3L_sort_detailed returns error status in ier.

Error Handling

On success, S3L_sort_detailed returns S3L_SUCCESS.

S3L_sort_detailed performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause S3L_sort_detailed to terminate and return the associated error code:

- S3L_ERR_ARG_DTYPE – The data type of the array is invalid. It must be S3L_integer, S3L_long_integer, S3L_float, or S3L_double.
- S3L_ERR_ARG_AXISNUM – The axis argument has an invalid value. The correct values for axis are:

C/C++7	0 <= axis < rank of A
F77/F90	1 <= axis <= rank of A

- S3L_ERR_ARG_OP – The first element of options is not either S3L_QUICKSORT or S3L_RADIXSORT, or the second element of options is not either S3L_UP or S3L_DOWN.

Examples

/opt/SUNWhpc/examples/s3l/sort/ex_sort3.c

/opt/SUNWhpc/examples/s3l/sort-f/sort2.f

Related Functions

S3L_grade_down(3)

S3L_grade_up(3)

S3L_grade_detailed_down(3)

S3L_grade_detailed_up(3)

S3L_sparse_solve

Description

S3L_sparse_solve solves a linear system of equations A*x = y, where A is a sparse S3L array and A and y are both single- or double-precision real parallel arrays.

Notes

When calling S3L_sparse_solve to solve a new (unfactored) sparse linear system, specify S3L_FULL_FACTOR_SOLVE as the first element of the option argument vector. This will cause S3L_sparse_solve to reduce fill by reordering the array and to perform symbolic and numeric factoring before solving the system. It will also return a setup value that identifies the internal setup created by the factoring.

If the same linear system is to be solved again, but with a different right-hand-side, specify S3L_SOLVE_ONLY as the first element of the options argument. Also specify the setup value returned by the S3L_sparse_solve call that factored the sparse array. The new solution will make use of the internal setup created by the earlier S3L_sparse_solve call.

If a previously factored sparse array contains new values, but the sparsity pattern has not changed, it can be solved without specifying S3L_FULL_FACTOR_SOLVE. Instead, specify S3L_SAME_SPARSITY_SOLVE and the previously returned setup value. This causes S3L_sparse_solve to perform numeric factorization on the sparse array and then solve the linear system.

When the internal setup for a linear system is no longer needed, the resources associated with it can be freed by calling S3L_sparse_solve_free and specifying the applicable setup value.

The S+ message-passing direct sparse solver was developed by Kai Shen and Tao Yang of the University of California at Santa Barbara. S+ can be used for general (asymmetric) sparse matrices.

The Sun Performance Library direct solver solves a sparse linear system on a single process.

Syntax

The C and Fortran syntax for S3L_sparse_solve is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_sparse_solve(A, y, options, roptions, setup)
    S3L_array_t         A
    S3L_array_t         y
    int                 *options
    double              *roptions
    int                 *setup
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_sparse_solve(A, y, options, roptions, setup, ier)
    integer*8           A
    integer*8           y
    integer*4           options(*)
    <type>              roptions(*)
    integer*4           setup
    integer*4           ier
```

where <type> is either real*4 or real*8.

Input

S3L_sparse_solve accepts the following arguments as input:

- A – A parallel single- or double-precision, real sparse S3L array.
- y – A parallel real array of rank 1 (vector) or rank 2 (matrix), containing the RHS of the linear system A*x = y. See Output section for use at exit.
- options – An array of rank 1 whose elements control S3L_sparse_solve behavior as follows:

options[0] = S3L_FULL_FACTOR_SOLVE	Perform fill-reducing reordering, symbolic factorization, numeric factorization, and solve the linear system.
options[0] = S3L_SAME_SPARSITY_SOLVE	Do only numeric factorization and solve the linear system. Use this option when the sparsity pattern of a previously factored array stays the same but it has a new set of values.
options[0] = S3L_SOLVE_ONLY	Only solve the linear system, using a previously computed factorization.
options[1] = S3L_SPLUS_SOLVER	Use S+ sparse solver. See "Notes" on page 296 for attribution information.
options[1] = S3L_PERFLIB_SOLVER	Use the Sun Performance Library 6.0 sparse solver.

If options[1] = S3L_PERFLIB_SOLVER, specify the following options as well:

options[2] = S3L_NON_SYMMETRIC	The sparse array has asymmetric structure and asymmetric values.
options[2] = S3L_SYMMETRIC	The sparse array has symmetric structure and symmetric values.
options[2] = S3L_SYM_STRUCT	The sparse array has symmetric structure but asymmetric values.
options[3] = S3L_NO_PIVOT	Do not use pivoting.
options[3] = S3L_DO_PIVOT	Use pivoting.

- roptions – Not currently used. It may be used in the future for specifying such parameters as a drop tolerance for pivoting, a threshold value for determining when a block is considered dense, and the amalgamation constant.

Output

S3L_sparse_solve uses the following arguments for output:

- y – On exit, y is overwritten with the solution of the system.
- setup – Integer associated with the sparse linear solution that results from this call to S3L_sparse_solve. If the internal setup will be used for additional solutions of the linear system, this setup value will be used by the subsequent S3L_sparse_solve calls. It will also be used in a subsequent call to S3L_sparse_solve_free to free the internal data associated with this solution of the sparse system.
- ier (Fortran only) – When called from a Fortran program, S3L_sparse_solve returns error status in ier.

Error Handling

On success, S3L_sparse_solve returns S3L_SUCCESS.

S3L_sparse_solve performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and returns an error code indicating which value was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause S3L_sparse_solve to terminate and return the associated error code:

- S3L_ERR_ARG_DTYPE – The data type of the input arrays A and/or y is not S3L_float or S3L_double, or the data type of A is not the same as that of y.
- S3L_ERR_ARRNOTSQ – Input array A is not square.
- S3L_ERR_ARG_RANK – y is not a 1D or 2D array.
- S3L_ERR_ARG_EXTENTS – The length of y is not compatible with the extents of A.
- S3L_ERR_ARG_OP – One or more of the following conditions exist:
  - The first element of the integer vector options is not S3L_FULL_FACTOR_SOLVE, S3L_SOLVE_ONLY, or S3L_SAME_SPARSITY_SOLVE.
  - The second element of options is not S3L_SPLUS_SOLVER or S3L_PERFLIB_SOLVER.
  - The second element of options is S3L_PERFLIB_SOLVER, but one or more of the following conditions exist:

    The third element of options is not S3L_NON_SYMMETRIC, S3L_SYMMETRIC, or S3L_SYM_STRUCT.

    The fourth element of options is not S3L_NO_PIVOT or S3L_DO_PIVOT.
- S3L_ERR_ARG_SETUP – Invalid setup value.

Examples

/opt/SUNWhpc/examples/s3l/spsolve/ex_sparse_solve1.c

/opt/SUNWhpc/examples/s3l/spsolve-f/ex_sp_solve1.f

Related Function

S3L_sparse_solve_free(3)

S3L_sparse_solve_free

Description

S3L_sparse_solve_free frees all internal data associated with the solution of a sparse linear system.

Syntax

The C and Fortran syntax for S3L_sparse_solve_free is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_sparse_solve_free(setup)
    int                *setup
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_sparse_solve_free(setup, ier)
    integer*4          setup
    integer*4          ier
```

Input

S3L_sparse_solve_free accepts the following argument as input:

- setup – An integer associated with a particular sparse linear solution. It was previously returned by the S3L_sparse_solve call that produced the solution.

Output

S3L_sparse_solve_free uses the following argument as output:

- ier (Fortran only) – When called from a Fortran program, S3L_sparse_solve_free returns error status in ier.

Error Handling

On success, S3L_sparse_solve_free returns S3L_SUCCESS.

On error, S3L_sparse_solve_free returns one of the following error codes:

- S3L_ERR_ARG_SETUP – Invalid setup value.
- S3L_ERR_NOTSUPPORT – The software layer upon which the S3L sparse solver is built could not be found. See the S3L_sparse_solve Error Handling section for details. Input array A is not square.

Examples

/opt/SUNWhpc/examples/s3l/spsolve/ex_sparse_solve1.c

/opt/SUNWhpc/examples/s3l/spsolve-f/ex_sp_solvef1.f

Related Function

S3L_sparse_solve(3)

S3L_sym_eigen

Description

S3L_sym_eigen finds selected eigenvalues and, optionally, eigenvectors of Hermitian matrices. The eigenvalues and eigenvectors can be selected by specifying a range of values or a range of indices for the desired eigenvalues/vectors.

Syntax

The C and Fortran syntax for S3L_sym_eigen is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_sym_eigen(A, axis1, axis2, E, V, J, job, range, limits,
tolerances)
    S3L_array_t       A
    int               axis1
    int               axis2
    S3L_array_t       E
    S3L_array_t       V
    S3L_array_t       J
    int               job
    int               range
    void              *limits
    void              *tolerances
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_sym_eigen(A, axis1, axis2, E, V, J, job, range, limits,
tolerances, ier)
    integer*8        A
    integer*4        axis1
    integer*4        axis2
    integer*8        E
    integer*8        V
    integer*8        J
    integer*4        job
    integer*4        range
    <type_lim>       limits(2)
    <type_tol>       tolerances(2)
    integer*4        ier
```

where <type_lim> is either `integer*4` or `real*4` and <type_tol> is either `real*4` or `real*8`.

Input

`S3L_sym_eigen` accepts the following arguments as input:

- `A` – S3L array handle describing a real or complex parallel array. On entry, `A` contains one or more two-dimensional Hermitian matrices, `b`, each of which is assumed to be dense and square. The axes of `b` are identified by the arguments `axis1` and `axis2`. Upon exit, the contents of `A` are destroyed.
- `axis1` – Integer variable denoting the axis of `A` that contains the rows of each Hermitian matrix, `b`.
- `axis2` – Integer variable denoting the axis of `A` that contains the columns of each Hermitian matrix, `b`. `axis2` must be greater than `axis1`.
- `job` – Integer variable indicating whether or not eigenvectors are to be computed. A value of 0 indicates that only eigenvalues are desired. Otherwise, both eigenvalues and eigenvectors are calculated.
- `range` – Integer variable indicating the range of eigenvalues to be computed, as follows:
  - `0` – Return all eigenvalues.
  - `1` – Compute all eigenvalues within the specified interval.

- 2 – Return a range of eigenvalue indices (when eigenvalues are sorted in ascending order).
- limits – Defines the eigenvalue interval when the value of range is 1 or 2. Specifically, when range equals:
  - 0 – limits is not used.
  - 1 – limits must be a scalar real vector of length 2. Its values bracket the interval in which eigenvalues are requested—that is, all eigenvalues in the interval [limits(1), limits(2)] will be found.
  - 2 – limits must be a scalar integer vector of length 2. For eigenvalues sorted in ascending order, eigenvalues corresponding to limits(1) through limits(2) will be found.
- tolerances – Real vector of length 2. Its precision must match that of A. That is, if A is of type S3L_float or S3L_complex, tolerances must be single-precision. If A is of type S3L_double or S3L_double_complex, tolerances must be double-precision.

  tolerances(1) gives the absolute error tolerance for the eigenvalues. If tolerances(1) is less than or equal to zero, the value eps * norm(b) will be used in its place, where eps is the machine tolerance and norm(b) is the 1-norm of the tridiagonal matrix obtained by reducing b to tridiagonal form.

  tolerances(2) controls the reorthogonalization of eigenvectors. Eigenvectors corresponding to eigenvalues that are within tolerances(2) * norm(b) of each other will be reorthogonalized. If tolerances(1) is less than or equal to zero, the value 1.0e-03 will be used in its place.

Output

S3L_sym_eigen uses the following arguments for output:

- A – Upon exit, the contents of A are destroyed.
- E – S3L array handle describing a real parallel array with rank(E) = rank(A) -1. axis1 of E must have the same extent as axis1 of A. The remaining axes are instance axes matching those of A in order of declaration and extents. Thus, each vector f within E corresponds to a matrix b within A.
  On return, each f contains the eigenvalues of the corresponding matrix b.
- V – S3L array handle describing a parallel array with the same rank, extents, and data type as A. For each instance of matrix b within A, there is a corresponding two-dimensional array, w, within V. axis1 denotes the axis of V that contains the rows of w; axis2 denotes the axis of V that contains the columns of w.

  On return, each column of w will contain an eigenvector of w.

- J – S3L array handle describing an integer parallel array with rank(J) = rank(A) - 1. axis1 of J should have an extent of 2. The remaining axes are instance axes matching those of A in order of declaration and extents. Thus, J will contain vectors of length 2 corresponding to the matrices b embedded within A.

  On return, the first element of each vector will contain the number of eigenvalues found. The second element of each vector will contain the number of eigenvectors found.
- ier (Fortran only) – When called from a Fortran program, S3L_sym_eigen returns error status in ier.

Error Handling

On success, S3L_sym_eigen returns S3L_SUCCESS.

S3L_sym_eigen performs generic checking of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause the function to terminate and return the associated error code:

- S3L_ERR_ARG_AXISNUM – Invalid value of axis1 or axis2.
- S3L_ERR_MATCH_RANK – Ranks of the parallel arrays do not match.
- S3L_ERR_ARRNOTSQ – The two-dimensional arrays in A are not square.
- S3L_ERR_MATCH_EXTENTS – The extents of the parallel arrays do not match.
- S3L_ERR_MATCH_DTYPE – The arguments are not all of the same data type and precision.
- S3L_ERR_ARG_RANGE_INV – Invalid value used for range or limits.
- S3L_ERR_ARG_NULL – Value of range is 1 or 2, but limits is a NULL pointer (C/C++) or 0 (F77/F90).

Examples

/opt/SUNWhpc/examples/s3l/eigen/eigen.c

/opt/SUNWhpc/examples/s3l/eigen-f/eigen.f

S3L_thread_comm_setup

Description

S3L_thread_comm_setup establishes the appropriate internal MPI communicators and data for thread-safe operation of S3L functions. It should be called from each thread in which S3L functions will be used.

Only S3L_init can be called before S3L_thread_comm_setup.

The argument comm specifies an MPI communicator, which should be congruent with, but not identical to, MPI_COMM_WORLD.

A unique communicator must be used for each thread or set of cooperating threads. The term *cooperating threads* refers to a set of threads that will be working on the same data. For example, one thread can initialize a random number generator, obtain a setup ID, and pass this to a fellow cooperating thread, which will then use the random number generator.

In such cases, the user must ensure that the threads within a cooperating set are properly synchronized.

A unique communicator is required because S3L performs internal communications. For example, when S3L_mat_mult is called from a multithreaded program, the thread on one node needs to communicate with the appropriate thread on another node. This can be done only if a communicator that is unique to these threads has been previously defined and passed to the communications routines within S3L.

S3L_thread_comm_setup need not be invoked if S3L functions are called from only one thread in the user's program.

Note – S3L_thread_comm_setup is useful when a user performs explicit multithreading by means of threads library functions. Since threads library functions are not available in F77, the F77 interface for S3L_thread_comm_setup is not provided.

Syntax

The C and Fortran syntax for S3L_thread_comm_setup is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_thread_comm_setup(comm)
    MPI_Comm            comm
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_thread_comm_setup(comm, ier)
    integer*4           comm
    integer*4           ier
```

Input

S3L_thread_comm_setup accepts the following argument as input:

- comm – An MPI communicator that is congruent with, but not identical to, MPI_COMM_WORLD.

Output

S3L_thread_comm_setup uses the following argument for output:

- ier (Fortran only) – When called from a Fortran program, S3L_thread_comm_setup returns error status in ier.

Error Handling

On success, S3L_thread_comm_setup returns S3L_SUCCESS.

S3L_thread_comm_setup performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause the function to terminate and return the associated error code:

- S3L_ERR_ARG_NULL – The comm argument is a NULL pointer (C/C++) or 0 (F77/F90).
- S3L_ERR_COMM_INVAL – The comm argument is not congruent with MPI_COMM_WORLD.
- S3L_ERR_THREAD_UNSAFE – The application program is using libraries that are not thread-safe.

Examples

/opt/SUNWhpc/examples/s3l/dense_matrix_ops/inner_prod_mt.c

/opt/SUNWhpc/examples/s3l/dense_matrix_ops/matmult_mt.c

Related Functions

MPI_Comm_dup(3)

S3L_set_safety(3)

threads(3T)

Also, "Multithreaded Programming" is a relevant section in the *Sun HPC ClusterTools User's Guide*.

S3L_to_ScaLAPACK_desc

Description

S3L_to_ScaLAPACK_desc converts the S3L array handle specified by s3ldesc into a ScaLAPACK array descriptor and subgrid address, which are returned in scdesc and address, respectively.

The array referred to by s3ldesc must be two-dimensional—that is, it must be a rank 2 array.

Syntax

The C and Fortran syntax for `S3L_to_ScaLAPACK_desc` is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_to_ScaLAPACK_desc(s3ldesc, scdesc, data_type, address)
    S3L_array_t          *s3ldesc
    int                  *scdesc
    int                  data_type
    void                 **address
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_to_ScaLAPACK_desc(s3ldesc, scdesc, data_type, address, ier)
    integer*8            s3ldesc
    integer*4            scdesc(*)
    integer*4            data_type
    pointer              address
    integer*4            ier
```

Input

`S3L_to_ScaLAPACK_desc` accepts the following argument as input:

- `s3ldesc` – Contains the S3L array handle that is provided as input to `S3L_to_ScaLAPACK_desc`.

Output

`S3L_to_ScaLAPACK_desc` uses the following arguments for output:

- `scdesc` – Contains the ScaLAPACK descriptor output generated by `S3L_to_ScaLAPACK_desc`.

- `data_type` – Contains the data type of the S3L array. It must specify a data type supported by Sun S3L.
- `address` – This argument will hold the starting address of an existing array subgrid.
- `ier` (Fortran only) – When called from a Fortran program, `S3L_to_ScaLAPACK_desc` returns error status in `ier`.

Error Handling

On success, `S3L_to_ScaLAPACK_desc` returns `S3L_SUCCESS`.

`S3L_to_ScaLAPACK_desc` performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause the function to terminate and return the associated error code:

- `S3L_ERR_ARG_NULL` – The `s3ldesc` argument is a `NULL` pointer (C/C++) or `0` (F77/F90).
- `S3L_ERR_ARG_RANK` – The S3L array handle refers to an array with a rank not equal to 2.
- `S3L_ERR_PGRID_NOPROCS` – The ScaLAPACK descriptor has an invalid BLACS context.

Examples

`/opt/SUNWhpc/examples/s3l/utils/scalapack_conv.c`

`/opt/SUNWhpc/examples/s3l/utils-f/scalapack_conv.f`

Related Function

`S3L_from_ScaLAPACK_desc(3)`

S3L_trans

Description

S3L_trans performs a generalized transposition of a parallel array. A generalized transposition is defined as a general permutation of the axes. The array axis_perm contains a description of the permutation to be performed.

The distribution characteristics of a and b must be compatible—that is, they must have the same rank and type, and corresponding axes must be of the same length.

A faster algorithm is used in the 2D case when the array meets the following conditions:

- The first axis of the array is local.
- The second axis of the array is global.
- The size of each dimension is divisible by the number of processes.
- The blocksizes are equal to the result of the division.

Syntax

The C and Fortran syntax for S3L_trans is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_trans(a, b, axis_perm)
    S3L_array_t      a
    S3L_array_t      b
    int              *axis_perm
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_trans(a, b, axis_perm, ier)
    integer*8         a
    integer*8         b
    integer*4         axis_perm
    integer*4         ier
```

Input

S3L_trans accepts the following arguments as input:

- a – S3L_array handle for the parallel array to be transposed.
- axis_perm – A vector of integers that specifies the axis permutation to be performed.

Output

S3L_trans uses the following arguments for output:

- b – S3L_array handle for a parallel array. Upon successful completion, S3L_trans stores the transposed array in b.
- ier (Fortran only) – When called from a Fortran program, S3L_trans returns error status in ier.

Error Handling

On success, S3L_trans returns S3L_SUCCESS.

S3L_trans checks the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause S3L_trans to terminate and return the associated code:

- S3L_ERR_MATCH_RANK – The ranks of a and b do not match.

- S3L_ERR_MATCH_EXTENTS – The extents of a and b are not compatible with the transpose operation requested. That is, the following relationship is not satisfied for all array axes i:

```
ext(a,axis_perm[i] = ext(b,i)
```

- S3L_ERR_TRANS_PERMAX – The supplied permutation is not valid (every axis must appear exactly once).
- S3L_ERR_ARG_AXISNUM – The axis argument has an invalid value. The correct values for axis are:
  - 0 <= axis < rank of the array (C/C++)
  - 0 < axis <= rank of the array (F77/F90)

Examples

/opt/SUNWhpc/examples/s3l/transpose/transp.c

/opt/SUNWhpc/examples/s3l/transpose/ex_trans1.c

/opt/SUNWhpc/examples/s3l/transpose-f/transp.f

S3L_walsh

Description

S3L_walsh computes the discrete Walsh/Hadamard transform of 1D and 2D S3L arrays. The arrays can have any of the supported S3L data types. For 1D transforms, the length of the array has to be a power of two. Similarly for the 2D case, the lengths along both dimensions should be a power of two.

The transform can be computed either in-place or out-of-place. If computed in-place, the result is in a and in-order. If it is computed out-of-place, the result is in b and out-of-order.

Arrays a and b must be the same rank and type and the extents of b must be compatible with the extents of a. For the 1D case, a and b should have the same extent. For the 2D case, the extents of array b should be such that array a can be transposed into b.

Notes

Efficient Distribution: The `S3L_walsh` function is more efficient when the arrays are block-distributed along their last dimension. When the calling program does not specify this distribution, S3L performs an internal redistribution of the arrays, which may result in additional overhead.

Inverse: The inverse transform is the transform itself.

Scaling: When a forward transform of an array is followed by the inverse transform, the original array is scaled by a factor that is the inverse of the product of the array extents. The following shows the scaling factors for one- and two-dimensional arrays:

1D	reconstructed array is scaled by 1/n, where n is the extent of the original array
2D	reconstructed array is scaled by 1/(m*n), where m and n are the array extents

Out-of-place: When computing the out-of-place transform, a different setup must be used for the forward and inverse transforms. For the 1D case, the length decomposition should be the reverse of the forward decomposition.

The following shows the sequence of steps required for a 1D array of length m*n.

```
decomp[0] = m;
decomp[1] = n;
S3L_walsh_setup(A,&setup,decomp);

decomp_t[0] = n;
decomp_t[1] = m;
S3L_walsh_setup(B,&setup_t,decomp_t);

dist = S3L_WALSH_OUTOFPL;
S3L_walsh(A,B,setup,&dist);
S3L_walsh(B,A,setup_t,&dist);
```

The out-of-place transform can avoid one internal global transposition. Consequently, it is generally more efficient than the in-place transform.

S3L computes the unordered Hadamard transform, whose matrix is a permutation of the ordered Hadamard and Walsh transforms. In particular, the transform matrix can be constructed recursively by the 2x2 matrix:

```
H2 = |  1   1  |
     |  1  -1  |
```

as follows:

```
    H4 = | H2  H2 |
         | H2 -H2 |

   [...]

   H2N = | HN  HN |
         | HN -HN |
```

Syntax

The C and Fortran syntax for `S3L_walsh` is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_walsh(a, b, setup, dist)
    S3L_array_t            a
    S3L_array_t            b
    int                    setup
    int                    *dist
```

F77/F90 Syntax

```
include ‘s3l/s3l-f.h’
include ‘s3l/s3l_errno-f.h’
subroutine
S3L_walsh(a, b, setup, dist, ier)
    integer*8              a
    integer*8              b
    integer*4              setup
    integer*4              dist
    integer*4              ier
```

Input

S3L_walsh accepts the following arguments as input:

- a – Input array whose Walsh transform is to be computed.
- b – Input array that is used as a storage array for the in-place case. It is used as the output array for the out-of-place case. For the 1D case, it should be of the same type and length as a. For the 2D case, its extents should be such that a can be transposed into b.
- setup – Integer corresponding to an appropriate Walsh transform setup. It is initialized with S3L_walsh_setup.
- dist – Integer that can have the following values:

S3L_WALSH_INPLACE	specifies in-place computation
S3L_WALSH_OUTOFPL	specifies out-of-place computation

Output

S3L_walsh uses the following arguments for output:

- a – Upon exit, if dist is equal to S3L_WALSH_INPLACE, its elements are replaced with the values of the Walsh transform. Otherwise, if dist is equal to S3L_WALSH_OUTOFPL, the contents of a are destroyed and the output of the transform is in array b.
- b – When dist is equal to S3L_WALSH_OUTOFPL, the results of the transform are in array b.
- ier (Fortran only) – When called from a Fortran program, S3L_walsh returns error status in ier.

Error Handling

On success, S3L_walsh returns S3L_SUCCESS.

S3L_walsh performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause S3L_walsh to terminate and return the associated error code:

- S3L_ERR_ARG_SETUP – Invalid setup value.
- S3L_ERR_PARAM_INVALID – The first element of the dist vector is not S3L_WALSH_INPLACE or S3L_WALSH_OUTOFPL.

- S3L_ERR_MATCH_RANK – The rank of a is not equal to that of b.
- S3L_ERR_MATCH_DTYPE – The data type of a is not equal to that of b.
- S3L_ERR_MATCH_EXTENTS – The extents of b are not compatible with those of a.

Examples

/opt/SUNWhpc/examples/s3l/walsh/ex_walsh1.c

/opt/SUNWhpc/examples/s3l/walsh/ex_walsh2.c

/opt/SUNWhpc/examples/s3l/walsh-f/ex_walsh1.f

/opt/SUNWhpc/examples/s3l/walsh-f/ex_walsh2.f

Related Functions

S3L_walsh_setup(3)

S3L_walsh_free_setup(3)

S3L_walsh_free_setup

Description

S3L_walsh_free_setup frees all internal data structures required for the computation of a parallel discrete Walsh/Hadamard transform.

Syntax

The C and Fortran syntax for S3L_walsh_free_setup is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_walsh_free_setup(setup)
    int                     *setup
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_walsh_free_setup(setup, ier)
    integer*4               setup
    integer*4               ier
```

Input

`S3L_walsh_free_setup` accepts the following argument as input:

- `setup` – Integer corresponding to a Walsh transform setup.

Output

`S3L_walsh_free_setup` uses the following argument for output:

- `ier` (Fortran only) – When called from a Fortran program, `S3L_walsh_free_setup` returns error status in `ier`.

Error Handling

On success, `S3L_walsh_free_setup` returns `S3L_SUCCESS`.

`S3L_walsh_free_setup` performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following condition will cause S3L_walsh_free_setup to terminate and return the associated error code:

- S3L_ERR_ARG_SETUP – Invalid setup value.

Examples

/opt/SUNWhpc/examples/s3l/walsh/ex_walsh1.c

/opt/SUNWhpc/examples/s3l/walsh/ex_walsh2.c

/opt/SUNWhpc/examples/s3l/walsh-f/ex_walsh1.f

/opt/SUNWhpc/examples/s3l/walsh-f/ex_walsh2.f

Related Functions

S3L_walsh(3)

S3L_walsh_setup(3)

S3L_walsh_setup

Description

S3L_walsh_setup initializes internal data structures required for the computation of a parallel discrete Walsh/Hadamard transform. Depending on the size, data type and distribution of the parallel array a, and the user-specified length decomposition, S3L_walsh_setup allocates an internal structure that can be used to compute the Walsh transform of array a or any other array with the same size, data type, and distribution. This internal structure is referenced by the integer variable setup and can be freed by using S3L_walsh_free_setup.

Syntax

The C and Fortran syntax for S3L_walsh_setup is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_walsh_setup(a, setup, decomp)
    S3L_array_t            a
    int                    *setup
    int                    decomp[2]
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_walsh_setup(a, setup, decomp, ier)
    integer*8              a
    integer*4              setup
    integer*4              decomp(2)
    integer*4              ier
```

Input

`S3L_walsh_setup` accepts the following arguments as input:

- `a` – Input array whose Walsh/Hadamard transform is to be computed. The data contained in the array are not modified.
- `decomp` – In the 1D case, `decomp` is an integer vector of length 2, whose elements correspond to the user-specified decomposition of the length of the array. For example, if the length of `a` is `l`, the elements of `decomp` should be specified such that `l` = `decomp[0] * decomp[1]`.

Output

`S3L_walsh_setup` uses the following arguments for output:

- `setup` – Integer corresponding to an appropriate Walsh transform setup. This parameter can be used in a subsequent call to `S3L_walsh` so long as the data type and extents of the array to be transformed are the same those of the array `a` set up through this call.

- ier (Fortran only) – When called from a Fortran program, S3L_walsh_setup returns error status in ier.

Error Handling

On success, S3L_walsh_setup returns S3L_SUCCESS.

S3L_walsh_setup performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause S3L_walsh_setup to terminate and return the associated error code:

- S3L_ERR_ARG_RANK – Array a has a rank greater than 2.
- S3L_ERR_ARG_EXTENTS – Some of the extents of array a are not powers of 2.

Examples

/opt/SUNWhpc/examples/s3l/walsh/ex_walsh1.c

/opt/SUNWhpc/examples/s3l/walsh/ex_walsh2.c

/opt/SUNWhpc/examples/s3l/walsh-f/ex_walsh1.f

/opt/SUNWhpc/examples/s3l/walsh-f/ex_walsh2.f

Related Functions

S3L_walsh(3)

S3L_walsh_free_setup(3)

S3L_write_array and S3L_write_sub_array

Description

S3L_write_array causes the process with MPI rank 0 to write the parallel array represented by the array handle a into a file specified by the filename argument. The file filename resides on the process with rank 0.

S3L_write_sub_array writes a specific section of the parallel array to *filename*. This section is defined by the lbounds, ubounds, and strides arguments. lbounds and ubounds specify the array section's lower and upper index bounds. strides specifies the stride along each axis; it must be greater than 0.

Note – The values of lbounds and ubounds should refer to zero-based indexed arrays for the C interface and to one-based indexed arrays for the Fortran interface.

Syntax

The C and Fortran syntax for S3L_write_array and S3L_write_sub_array is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_write_array(a, filename, format)
S3L_write_sub_array(a, lbounds, ubounds, strides, filename,
format)
    S3L_array_t         a
    int                 *lbounds
    int                 *ubounds
    int                 *strides
    char                *filename
    char                *format
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_write_array(a, filename, format, ier)
S3L_write_sub_array(a, lbounds, ubounds, strides, filename,
format, ier)
    integer*8          a
    integer*4          lbounds(*)
    integer*4          ubounds(*)
    integer*4          strides(*)
    character*1        filename(*)
    character*1        format(*)
    integer*4          ier
```

Input

`S3L_write_array` and `S3L_write_sub_array` accept the following arguments as input:

- `a` – S3L array handle for the parallel array to be written. This array handle was returned when the array was declared.
- `lbounds` – Integer vector specifying the lower bounds of the indices of `a` along each of its axes.
- `ubounds` – Integer vector specifying the upper bounds of the indices of `a` along each of its axes.
- `strides` – Integer vector specifying the strides on the indices of `a` along each of its axes.

- filename – Scalar character variable specifying the name of the file to which the parallel array will be written.
- format – Scalar character variable specifying the format of the data to be written. The value can be either "ascii" or "binary".

Output

S3L_write_array and S3L_write_sub_array use the following argument for output:

- ier (Fortran only) – When called from a Fortran program, S3L_write_array and S3L_write_sub_array return error status in ier.

Error Handling

On success, S3L_write_array and S3L_write_sub_array return S3L_SUCCESS.

S3L_write_array and S3L_write_sub_array perform generic checking of the validity of the arrays they accept as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause the function to terminate and return the associated error code:

- S3L_ERR_ARG_RANGE_INV – The given range of indices is invalid:
  - A lower bound is less than the smallest index of the array.
  - An upper bound is greater than the largest index of an array along the given axis.
  - A lower bound is larger than the corresponding upper bound.
  - A stride is negative or zero.
- S3L_ERR_FILE_OPEN – Failed to open the file with the file name provided.
- S3L_ERR_IO_FORMAT – Format is not one of "ascii" or "binary".
- S3L_ERR_IO_FILENAME – The file name is equal to the NULL string (C/C++) or to an empty string (F77/F90).

Examples

/opt/SUNWhpc/examples/s3l/io/ex_io.c

/opt/SUNWhpc/examples/s3l/io-f/ex_io.f

Related Functions

S3L_print_array(3)

S3L_read_array(3)

S3L_write_sparse

Description

S3L_write_sparse causes the process with MPI rank 0 to write the global sparse matrix A into a file. The matrix data will be written in a user-specified format, which can be any one of:

- S3L_SPARSE_COO – Coordinate format.
- S3L_SPARSE_CSR – Compressed Sparse Row format.
- S3L_SPARSE_CSC – Compressed Sparse Column format.
- S3L_SPARSE_VBR – Variable Block Row format.

Each of these formats consists of a header and two data sections, which S3L_write_sparse fills in the following manner:

Header section	This is a one-line section that begins with two percent '%%' characters, followed by a sequence of keywords. It indicates which sparse format and what numerical data type are used to write out data in the matrix data structure.
First data section	This is also a one-line section. It contains metric information about the sparse matrix, such as the number of rows (m), columns (n), and nonzero elements (nnz). In the case of the S3L_SPARSE_VBR format, it also writes the number of block rows (bm), block columns (bn), and nonzero blocks (bnnz).
Second data section	Writes sparse matrix data in the specified format.

Examples of the four supported formats are presented below.

S3L_SPARSE_COO Format

The S3L_SPARSE_COO format is explained with this sample 4x6 sparse matrix:

```
 3.14      0          0         20.04     0          0
 0        27          0          0       -0.6        0
 0         0         -0.01       0        0          0
-0.031     0          0          0.08     0        314.0
```

The following shows how this matrix data might be written in S3L_SPARSE_COO format:

```
%% S3L_SPARSE_COO matrix single precision real

4 6 8
  0  0   3.14000000e+00
  0  3   2.00400000e+01
  1  1   2.70000000e+01
  1  4  -6.00000000e-01
  2  2  -1.00000000e-02
  3  0  -3.10000000e-02
  3  3   8.00000000e-02
  3  5   3.14000000e+02
```

In this example, the first line indicates that the matrix is written in S3L_SPARSE_COO format and that the data type is single-precision real.

The second line is the first data section. It lists m = 4, n = 6, and nnz = 8. These values represent the number of rows, columns, and the total number of nonzero elements, respectively.

The third line begins the second data section, which contains the eight nonzero values, each on a separate line and preceded by their row and column indices.

S3L_SPARSE_CSR Format

The following example shows how the same 4x6 sparse matrix used in the S3L_SPARSE_COO format example might appear when written in the S3L_SPARSE_CSR format:

```
%% S3L_SPARSE_CSR matrix single precision real
 4 6 8
 0 2 4 5 8
 0 3 1 4 2 0 3 5
 3.14000000e+00  2.00400000e+01
 2.70000000e+01 -6.00000000e-01
-1.00000000e-02
-3.10000000e-02  8.00000000e-02  3.14000000e+02
```

The S3L_SPARSE_CSR format differs from the S3L_SPARSE_COO format in the second data section. It represents the CSR structure as:

```
ptr  = (0, 2, 4, 5, 8)
indx = ( 0, 3, 1, 4, 2, 0, 3, 5)
val  = (3.14000000e+00,  2.00400000e+01,
         2.70000000e+01, -6.00000000e-01,
        -1.00000000e-02, -3.10000000e-02,
         8.00000000e-02,  3.14000000e+02)
```

For example, ptr[1] = 2 indicates that the first nonzero element in row 1 is stored in val[2] (= val[ptr[1]]), which is 2.70000000e+01.

S3L_SPARSE_CSC Format

The S3L_SPARSE_CSC format is, in effect, the CSR format for the transpose of A. In other words, for the S3L_SPARSE_CSC format, the ptr and indx arrays exchange roles: ptr contains column start pointers and indx contains row indices. The val array contains the nonzero elements.

The following shows the S3L_SPARSE_CSC layout for the same 4x6 sparse matrix example as was used before:

```
%% S3L_SPARSE_CSC matrix single precision real
 4 6 8
 0 2 3 4 6 7 8
 0 3 1 2 0 3 1 3
 3.14000000e+00 -3.10000000e-02
 2.70000000e+01 -1.00000000e-02
 2.00400000e+01  8.00000000e-02
-6.00000000e-01  3.14000000e+02
```

Again, the S3L_SPARSE_CSC data structure is written in the second data section, which begins on the third line:

```
ptr  = ( 0, 2, 3, 4, 6, 7, 8 )
indx = ( 0, 3, 1, 2, 0, 3, 1, 3 )
val  = ( 3.14000000e+00 -3.10000000e-02
         2.70000000e+01 -1.00000000e-02
         2.00400000e+01  8.00000000e-02
        -6.00000000e-01  3.14000000e+02)
```

Note that, in the S3L_SPARSE_CSC format, the nonzero elements in val are stored column-by-column, instead of row-by-row, as in the S3L_SPARSE_CSR format.

For example, ptr[5] = 7 means that the first nonzero element of column 5 is stored in val[7] (= val[ptr[5]]), which is 3.14000000e+02, and its row index is stored in indx[7] (= indx[ptr[5]]), which is 3.

S3L_SPARSE_VBR Format

The first three sparse matrix formats all provide natural layouts for point sparse matrices. However, for matrices with nonzero elements clustered in blocks, S3L_SPARSE_VBR offers a more efficient representation.

Like the other three formats, S3L_SPARSE_VBR begins with a section that identifies the format and the data type.

The second section contains three additional integers beyond the basic three used in the point-based formats. The six integers in the second section are:

m	Indicates the number of point rows.
n	Indicates the number of point columns.
nnz	Indicates the number of point nonzero elements.
bm	Indicates the number of block rows.
bn	Indicates the number of block columns.
bmnz	Indicates the number of nonzero block entries.

The third section contains the S3L_SPARSE_VBR data structure in five integer arrays and one floating-point array. For their definition, see the man page for S3L_convert_sparse().

To illustrate the S3L_SPARSE_VBR data layout, consider the following 5x8 sparse matrix with a variable block partitioning:

```
     0  1   2  3  4    5    6   7   8
   +------+---------+----+--------+
 0 | 1  3  5 | 0  0 |  9 |  0   0 |
 1 | 2  4  6 | 0  0 | 10 |  0   0 |
   +---------+------+----+--------+
 2 | 0  0  0 | 7  8 | 11 |  0   0 |
   +------+---------+----+--------+
 3 | 0  0  0 | 0  0 | 12 | 14  16 |
 4 | 0  0  0 | 0  0 | 13 | 15  17 |
   +------+---------+----+--------+
 5
```

This matrix could be written in S3L_SPARSE_VBR format as follows:

```
%% S3L_SPARSE_VBR matrix single precision real
 5 8 17 3 4 6

 0 2 3 5
 0 3 5 6 8

 0 2 4 6
 0 2 1 2 2 3
 0 6 8 10 11 13 17

 1.00000000e+00  2.00000000e+00  3.00000000e+00  4.00000000e+00
 5.00000000e+00  6.00000000e+00  9.00000000e+00  1.00000000e+01
 7.00000000e+00  8.00000000e+00  1.10000000e+01  1.20000000e+01
 1.30000000e+01  1.40000000e+01  1.50000000e+01  1.60000000e+01
 1.70000000e-01
```

In this example, the second line lists:

```
m = 5, n = 8, nnz = 17, bm = 3, bn = 4, bnnz = 6
```

The rest of lines lay out the matrix data in S3L_SPARSE_VBR format. The first two lines are filled with data from arrays rptr and cptr:

```
rptr = (0, 2, 3, 5)
cptr = (0, 3, 5, 6, 8)
```

In array rptr, 0, 2, 3, and 5 are pointers to the boundaries of the block rows. Likewise in array cptr, 0, 3, 5, 6, and 8 are pointers to the boundaries of the block columns.

Data in the remaining lines are from arrays bptr, bindx, indx, and val:

```
bptr  = (0, 2, 4, 6)
bindx = (0, 2, 1, 2, 2, 3)
indx  = (0, 6, 8, 10, 11, 13, 17)
val   = (1.0, 4.0, 2.0, 5.0, 3.0, 6.0, 7.0, 8.0, 9.0, 10.0,
         11.0, 14.0, 17.0, 12.0, 15.0, 18.0, 13.0, 16.0, 19.0 )
         1.00000000e+00, 4.00000000e+00, 3.00000000e+00,
         4.00000000e+00, 5.00000000e+00, 6.00000000e+00,
         7.00000000e+00, 8.00000000e+00, 9.00000000e+00,
         1.00000000e+01, 1.10000000e+01, 1.20000000e+01,
         1.30000000e+01, 1.40000000e+01, 1.50000000e+01,
         1.60000000e+01, 1.70000000e-01,
```

In array bptr, 0, 2, 4, and 6 are pointers to the location in bindx of the first nonzero block entry of each block row.

These block-based pointers are illustrated in the following figure, which represents the block structure of the original 5x8 sparse matrix. It shows the first block row with two nonzero blocks, one in block column 0 and the other in block column 2. The next nonzero block is at block row 1 and block column 1, and so forth. Block 6 is the outer boundary of the block rows.

```
     0    1    2    3    4
   +----+----+----+----+
 0 | b0 |    | b1 |    |
   +----+----+----+----+
 1 |    | b2 | b3 |    |
   +----+----+----+----+
 2 |    |    | b4 | b5 |
   +----+----+----+----+
 3
```

In array `bindx`, `0`, `2`, `1`, `2`, `2`, and `3` are indices for the block columns.

In array `indx`, `0`, `6`, `8`, `10`, 11, `13`, and `17` point to the locations in `val` of the first nonzero block entry from each block row.

The last array, `val`, stores nonzero blocks `b0`, `b1`, ..., `b5` block-by-block with each block stored as a dense matrix in standard column-by-column form. Morever, the starting location in `val`, where the first element of each block gets stored is indexed by array `indx`.

The `S3L_SPARSE_VBR` data structure can be understood by analyzing the representation of block row 1 for example.

First, `bptr[1]` = `2` indicates that `b2`, the first nonzero block from block row 1 is from block column 1, as indicated by `bindx[2]` = `bindx[bptr[1]]` = `1`.

Second, `bptr[1]` = `2` also indexes into `indx`. That is, `indx[bptr[1]]` = `indx[2]` = `8` points to val[8] (= `val[indx[bptr[1]]` = `val[indx[2]]`), where `8` is the location in `val` at which the first element of `b2`, `7.0`, is stored.

The next nonzero block in block row 1 is `b3`, its block column index is 2, as indicated by `bindx[bptr[1]+1]` = `bindx[3]` = `2`, and the first element of block `b3` is stored in `val[10]` (= `val[indx[bptr[1]+1]]` = `val[indx[3]]`), which is `11.0`.

Syntax

The C and Fortran syntax for `S3L_write_sparse` is as follows:

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_write_sparse(A, spfmt, fname, dfmt)
    S3L_array_t             A
    S3L_sparse_storage_t    spfmt
    char                    *fname
    char                    *dfmt
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_write_sparse(A, spfmt, fname, dfmt, ier)
    integer*8               A
    integer*4               spfmt
    character*1             fname(*)
    character*1             dfmt(*)
    integer*4               ier
```

Input

S3L_write_sparse accepts the following arguments as input:

- A – S3L array handle for the global general sparse matrix.
- spfmt – Specifies the sparse storage format to be used in writing the matrix data to a file. The supported formats are: S3L_SPARSE_COO, S3L_SPARSE_CSR, S3L_SPARSE_CSC, and S3L_SPARSE_VBR.
- fname – Scalar character variable that names the file to which the sparse matrix data will be written.
- dfmt – Scalar character variable that specifies the data file format to be used in writing the sparse matrix data. The allowed values are 'ascii' and 'ASCII'.

Output

S3L_write_sparse uses the following argument for output:

- ier (Fortran only) – When called from a Fortran program, S3L_write_sparse returns error status in ier.

Error Handling

On success, S3L_write_sparse returns S3L_SUCCESS.

The S3L_write_sparse routine performs generic checking of the validity of the arrays it accepts as arguments. If an array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following conditions will cause S3L_write_sparse to terminate and return the associated error code:

- S3L_ERR_ARG_NULL – The value specified for A is invalid. No such S3L sparse matrix has been defined.
- S3L_ERR_SPARSE_FORMAT – Invalid storage format. It must be S3L_SPARSE_COO, S3L_SPARSE_CSR, S3L_SPARSE_CSC, or S3L_SPARSE_VBR.
- S3L_ERR_IO_FILENAME – Invalid file name.
- S3L_ERR_IO_FORMAT – Invalid data file format. The dfmt value supplied was not 'ascii' or 'ASCII'.

Examples

/opt/SUNWhpc/examples/s3l/sparse/ex_sparse.c

/opt/SUNWhpc/examples/s3l/sparse-f/ex_sparse.f

Related Functions

S3L_read_sparse(3)

S3L_print_sparse(3)

S3L_zero_elements

Description

S3L_zero_elements sets to zero all elements of the S3L array whose array handle is A.

Syntax

The C and Fortran syntax for S3L_zero_elements is as follows.

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_zero_elements(A)
    S3L_array_t        A
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_zero_elements(A, ier)
    integer*8          A
    integer*4          ier
```

Input

S3L_zero_elements accepts the following argument as input:

- A – S3L internal array handle for the parallel array that is to be initialized to zero.

Output

S3L_zero_elements uses the following argument for output:

- ier (Fortran only) – When called from a Fortran program, S3L_zero_elements returns error status in ier.

Error Handling

On success, S3L_zero_elements returns S3L_SUCCESS.

S3L_zero_elements checks the array it accepts as argument. If the array argument contains an invalid or corrupted value, the function terminates and an error code is returned that indicates which value of the array handle was invalid. See Appendix A of this manual for a detailed list of these error codes.

In addition, the following condition will cause the function to terminate and return the associated code:

- S3L_ERR_ARG_DTYPE – The data type of A is invalid.

Examples

/opt/SUNWhpc/examples/s3l/utils/zero_elements.c

/opt/SUNWhpc/examples/s3l/utils-f/zero_elements.f

APPENDIX **A**

S3L Array Checking Errors

Sun S3L interfaces do generic checking of the validity of the array handles that are passed as arguments to them. If such an array handle contains an invalid or corrupted value, the function terminates and one of the error codes listed in TABLE A-1 is returned.

TABLE A-1 Return Codes Associated With Array Handle Errors

Error Code	Definition
`S3L_ERR_ARG_DTYPE`	The data type specified for an array is not supported by Sun S3L.
`S3L_ERR_ARG_ELEMSIZE`	An array argument includes an invalid element size.
`S3L_ERR_ARG_RANK`	An invalid rank is specified for an array; it is either negative or larger than 32 (the largest supported array rank).
`S3L_ERR_ARG_EXTENTS`	An array argument includes a negative extent.
`S3L_ERR_ARG_BLKSIZE`	An array argument includes a negative blocksize.
`S3L_ERR_ARG_BLKSTART`	For a block-cyclic array distribution, an invalid starting process is specified; it is either negative or is larger than the extent of the corresponding process grid axis.
`S3L_ERR_ARG_SFSIZE`	An array argument includes an invalid subgrid size; it is either negative or is larger than the extent along the corresponding array axis.
`S3L_ERR_ARG_MAJOR`	An array argument includes an invalid majorness value.
`S3L_ERR_ARG_PGRID_EXTENTS`	An array argument includes an invalid process grid extent; it is either negative or larger than the total number of processes over which the array is defined.

TABLE A-1 Return Codes Associated With Array Handle Errors *(Continued)*

Error Code	Definition
S3L_ERR_ARG_PGRID_RANK	The rank of a process grid does not equal the rank of the corresponding array.
S3L_ERR_ARG_PGRID_MAJOR	An array argument specifies an invalid majorness value for a process grid.
S3L_ERR_ARG_PGRID_COOR	An array argument specifies a process grid coordinate that is either negative or larger than the process grid extent along that axis.