Sun S3L 3.1 Programming and Reference Guide

THE NETWORK IS THE COMPUTER™

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900 USA
650 960-1300   Fax 650 969-9131

Part No. 806-3735-10
March 2000, 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.

Using UNIX Commands

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

See one or more of the following for such information:

- AnswerBook™ online documentation for the Solaris™ 2.x software environment
- Other software documentation that you received with your system

Typographic Conventions

TABLE P-1 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 `root` to do this.
	Command-line variable; replace with a real name or value.	To delete a file, type `rm` *filename*.

Shell Prompts

TABLE P-2 Shell Prompts

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

Related Documentation

TABLE P-3 Related Documentation

Application	Title	Part Number
All	*Sun HPC ClusterTools 3.1 Product Notes*	806-4182-10
All	*Sun HPC ClusterTools 3.1 Performance Guide*	806-3732-10
Sun MPI Programming	*Sun MPI 4.1 Programming and Reference Guide*	806-3734-10
Sun MPI Programming	*Sun HPC ClusterTools 3.1 User's Guide*	806-3733-10
Prism	*Prism 6.1 User's Guide*	806-3736-10
Prism	*Prism 6.1 Reference Manual*	806-3737-10

CHAPTER 1

Introduction to Sun S3L

This chapter contains general information about the Sun Scalable Scientific Subroutine Library (Sun S3L).

Sun S3L Overview

Sun S3L provides a set of parallel and scalable functions and tools widely used in scientific and engineering computing. It can be used on all Sun HPC Systems, from a single processor on an SMP, through multiple processors on a stand-alone SMP, to a cluster of SMPs.

The chief advantages offered by Sun S3L are summarized below.

- Sun S3L is optimized for Sun HPC Systems.
- Sun S3L functions have a simple array syntax interface that is callable from message-passing programs written in C, C++, F77, or F90.
- Sun S3L supports multiple instances.
- Sun S3L is thread safe.
- Sun S3L uses the Sun Performance Library™ for nodal computation.
- Extensive and detailed programming examples are provided online.
- Sun S3L is supported by Sun.
- Sun S3L includes built-in diagnostics.

Sun S3L uses *array handles* to provide array syntax support to message-passing programs. Array handles, which are closely analogous to the array descriptors found in the public domain packages ScaLAPACK and PETSc, facilitate argument passing by encapsulating information about distributed arrays.

Sun S3L operates on multidimensional arrays of up to 32 dimensions. This means it implements the multiple-instance paradigm, where the same function is applied to multiple, disjoint data sets concurrently.

The Sun S3L user interface includes a communicator setup routine that allows Sun S3L functions to be used in multithreaded applications. This routine causes Sun S3L to establish an independent Sun MPI communicator and thread-safe data for each thread from which the routine is called.

Sun S3L routines implement the Sun Performance Library for nodal operations. This is a collection of libraries for dense linear algebra and Fourier transforms based on the standard libraries BLAS, LINPACK, LAPACK, FFTPACK, and VFFTPACK. Besides providing appropriate nodal support to Sun S3L, routines from the Sun Performance Library can be called independently from any user codes running locally on a Sun Ultra HPC Server node.

Note – The Sun Performance Library is available to Sun S3L users as part of WorkShop Compilers Fortran or Performance WorkShop Fortran, v4.2 and v5.0.

Sun S3L routines operate on objects of various data types. However, this information is encoded in the array handle and is decoded at run time, allowing appropriate branching to occur during execution. Consequently, there is no need for separate routines with different names to implement the different data types; a single routine suffices for all types.

An extensive set of online examples illustrate correct use of all Sun S3L functions. These examples can be used as templates in developing actual code. Separate examples are provided to demonstrate C and Fortran interfaces.

Contents of Sun S3L

Sun S3L consists of a set of *core* library functions—that is, subroutines that perform the linear algebra, Fourier transform, and other scientific computations—plus a set of auxiliary utilities, referred to as the *toolkit* functions.

The toolkit functions are introduced in “Sun S3L Toolkit Functions” on page 3, with detailed descriptions provided in Chapter 6. The core library functions are introduced in “Core Scientific Library Routines” on page 4, with detailed descriptions in Chapter 7. They are also described in their online man pages.

Many of the Sun S3L core routines support the corresponding ScaLAPACK application programming interfaces (APIs). TABLE 1-1 lists the ScaLAPACK APIs that are supported.

TABLE 1-1 Supported ScaLAPACK APIs

Category	Routine
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{c,d,c,z}getrs,p{c,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

Sun S3L Toolkit Functions

Sun S3L includes an extensive set of functions that enable Sun MPI programmers to perform a variety of auxiliary tasks, such as:

- Initializing and exiting from the S3L environment.
- Creating and destroying S3L array handles for defining parallel arrays.
- Creating and destroying S3L process grid handles for defining process grids.
- Performing operations on array elements.
- Extracting information about parallel arrays and array subgrids.
- Reading a file into all or part of an S3L parallel array.
- Writing all or part of an S3L parallel array into a file.
- Printing all or part of an S3L parallel array to standard output.
- Converting ScaLAPACK descriptors into S3L array handles and S3L array handles into ScaLAPACK descriptors.
- Creating Sun MPI communicators to allow thread-safe operation of S3L functions.
- Controlling the S3L safety mechanism.

Core Scientific Library Routines

The Sun S3L core routines consist of:

- Dense matrix operations
  - 2–Norm – Compute the global 2–norm of a parallel array.
  - Inner product – Compute the global inner product over all axes of two source parallel arrays. The inner product is added to the destination. A routine that takes the conjugate of the second operand is provided for complex data.
  - Outer product – Compute one or more instances of an outer product of two vectors. The result is added to the destination. For complex data, a routine that takes the conjugate of the second operand is provided.
  - Matrix-vector multiplication – Compute one or more instances of a matrix-vector product. The result is added to the destination, or is added to a second parallel array. For complex data, a routine that takes the conjugate of the matrix is provided.
  - Matrix multiplication – Compute one or more matrix products. Each routine add the result to the destination. Routines that take the transpose of either or both operand matrices (or, for complex data, the Hermitian of either matrix) are provided.
- LU-factorization and LU-solve routines
  - LU-factorization routine – For each m x n coefficient matrix `A` of `a`, computes LU factorization using partial pivoting with row interchanges.
  - LU-solve routine – Uses the `L` and `U` factors produced by the LU-factorization routine to produce solutions to the system AX=B. B may represent one or more right-hand sides for each instance of the systems of equations.
  - LU-invert routine – For each m x m (square) instance of matrix `A`, computes the inverse of `A` using the LU-factorization results of the `S3L_lu_factor` routine.
- Parallel 1D, 2D, and 3D FFTs
  - Setup and deallocation of FFT handles – Initialize and deallocate FFT handles for both complex and real data types. Separate routines are used for the two data types.
  - Simple complex-to-complex, mixed-radix, forward and inverse FFT routines – Performs forward or inverse Fast Fourier Transform of a parallel array of type complex or double complex. Supports both power-of-two and arbitrary radix parameters.
  - Detailed complex-to-complex FFT routine – Allows independent specification along each data axis of the transform direction in a complex-to-complex FFT. Can improve performance over the simple FFT in some cases.
  - Simple real-to-complex and complex-to-real FFT routines – Perform the forward (real-to-complex) and inverse (complex-to-real) FFT operations on 1-, 2-, or 3-dimensional arrays.

- Structured solver
  - Tridiagonal solver – Solves collections of tridiagonal linear systems of equations using Gaussian elimination with pivoting.
  - Banded solver – Solves collections of banded linear systems of equations using Gaussian elimination with pivoting.
- Dense symmetric eigenvalue solver – Computes selected eigenvalues and, optionally, eigenvectors of hermitian matrices.
- Dense Singular Value Decomposition (SVD) – Computes the singular value decomposition of an M x N matrix and, optionally, the left and right singular vectors.
- Sparse routines
  - Declare array handle for a sparse matrix.
  - Read data from a file into a distributed matrix, with support for both COO and CSR sparse storage formats.
  - Compute the product of a sparse matrix with a dense vector.
- Iterative solver – Solves a general sparse linear system of equations using iterative methods, with or without preconditioning.
- Convolution / Deconvolution
  - Convolve – Computes 1D or 2D convolution of one array with another.
  - Deconvolve – Deconvolves an array into a vector.
- Iterative eigensolver – Computes selected eigenpairs of dense or sparse matrices, with optional specification of eigenpair properties.
- Autocorrelation – Computes 1D or 2D autocorrelation of a signal.
- Sort and grade – Sort and grade arrays.
- Parallel random number generators
  - Fibonacci RNG setup and deallocation – Initializes and deallocates the state table of a lagged Fibonacci random number generator (LFG).
  - Fibonacci RNG – Uses an LFG to initialize a parallel array.
  - LCG RNG setup – Defines the parameters used in the Sun S3L linear congruential random number generator (LCG).
  - LCG RNG – Uses a parallel LCG to produce random numbers that are independent of the array distribution.
- Parallel sort – Sorts a 1D parallel array.
- Parallel transpose – Performs a generalized transposition of a parallel array.
- Copy array routine – Copies the elements of one array onto another.
- Zero array elements – Replaces all elements in an array with zero.

CHAPTER 2

Sun S3L Arrays

Overview

Sun S3L distributes arrays, axis-by-axis, in blocks across multiple processes, allowing operations to be performed in parallel on different sections of the array. These arrays are referred to in this manual as S3L arrays and, more generically, as parallel arrays.

Arrays passed to Sun S3L routines by C, C++, F77, or F90 message-passing programs can have block, cyclic, or block-cyclic distributions. Regardless of the type of distribution specified by the calling program, Sun S3L will automatically select the distribution scheme that is most efficient for the routine being called. If that means Sun S3L changes the distribution method internally, it will restore the original distribution scheme on the resultant array before passing it back to the calling program.

S3L arrays can also be undistributed. That is, all the elements of the array can be located on the same process—a serial array in the conventional sense.

The balance of this chapter describes S3L arrays in more detail.

S3L Array Attributes

A principal attribute of S3L arrays is rank—the number of dimensions an array has. For example, an S3L array with three dimensions is called a rank-three array. S3L arrays can have up to 32 dimensions.

An S3L array is also defined by its extents, its length along each dimension of the array and its type, which reflects the data type of its elements. S3L arrays can be of the following types:

- `S3L_integer` (4-byte integer)
- `S3L_long_integer` (8-byte integer)
- `S3L_float` (4-byte floating point number)
- `S3L_double` (8-byte double precision floating point number)
- `S3L_integer` (4-byte integer)
- `S3L_complex` (8-byte complex number)
- `S3L_double_complex` (16-byte complex number)

The C and Fortran equivalents of these array data types are described in Chapter 3.

S3L Array Handles

When an S3L array is declared, it is associated with a unique array handle. This is an S3L internal structure that fully describes the array. An S3L array handle contains all the information needed to define both the global and local characteristics of an S3L array. For example, an array handle includes

- global features, such as the array's rank and information about how the array is distributed
- local features, such as its extents and its location in memory on the process

By describing both local and global features of an array, an array handle makes it possible for any process to easily access data in array sections that are on other processes, not just data in its local section. That is, no matter how an array has been distributed, the associated S3L array handle ensures that its layout is understood by all participating processes.

In C programs, S3L array handles are declared as type `S3L_array_t` and in Fortran programs as type `integer*8`.

Processes and Process Grids

In a Sun MPI application, each process is identified by a unique rank. This is an integer in the range 0 to np-1, where np is the total number of processes associated with the application.

Note – This use of rank is totally unrelated to references to S3L array ranks. Process ranks correspond to MPI ranks as used in interprocess communication. Array ranks indicate the number of dimensions an array has.

Sun S3L maps each S3L array onto a logical arrangement of processes, referred to as a process grid. A process grid will have the same number of dimensions as the S3L array with which it is associated. Each S3L array section that is distributed to a particular process is called a subgrid.

Sun S3L controls the ordering of the np processes within the n-dimensional process grid. FIGURE 2-1through FIGURE 2-3 illustrate this with examples of how Sun S3L might arrange eight processes in one- and two-dimensional process grids.

In FIGURE 2-1, the eight processes form a one-dimensional grid.

Process Rank	0	1	2	3	4	5	6	7
Coordinates	(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)

FIGURE 2-1 Eight Processes Arranged as a 1x8 Process Grid

FIGURE 2-2 and FIGURE 2-3 show the eight processes organized into rectangular 2x4 process grids. Although both have 2x4 extents, the array process grids differ in their majorness attribute. This attribute determines the order in which the processes are distributed onto a process grid’s axes or local subgrid axes. The two possible modes are:

- Column major – Processes are distributed along column axes first; that is, the process grid’s row indices increase fastest.
- Row major – Processes are distributed along row axes first; the process grid’s column indices increase fastest.

In FIGURE 2-2, subgrid distribution follows a column-major order. In FIGURE 2-3, process grid distribution is in row-major order.

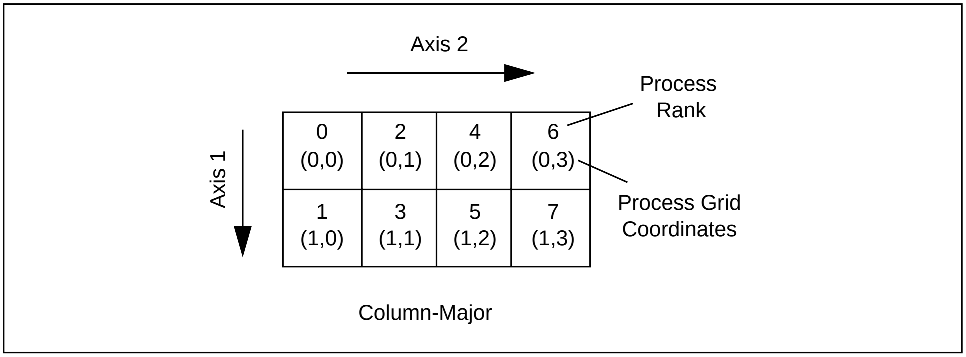

FIGURE 2-2 Eight Processes Arranged as a 2x4 Process Grid: Column-Major Order

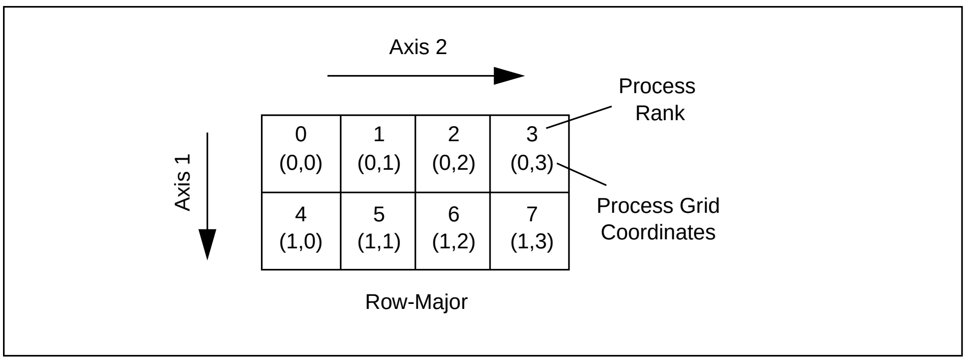

FIGURE 2-3 Eight Processes Arranged as a 2x4 Process Grid: Row-Major Order

---

Note – In these examples, axis numbers are one-based (Fortran-style). For the C-language interface, reduce each value by 1. Process ranks and process grid coordinates are always zero-based.

Defining Process Grids

When an S3L array is defined, the programmer has the choice of either defining a process grid explicitly, using the `S3L_set_process_grid` function, or letting S3L define one using an internal algorithm. The following F77 example shows how to specify a two-dimensional process grid that is defined over a set of eight processes having MPI ranks 0 through 7. The process grid has extents of 2x4 and is assigned column-major ordering.

```
include ‘s3l/s3l-f.h’
integer*8 pg
integer*4 rank
integer*4 pext(2),process_list(8)
integer*4 i,ier

rank = 2
pext(1) = 2
pext(2) = 4
do i=1,8
    process_list(i)=i-1
end do
call s3l_set_process_grid(pg,rank,S3L_MAJOR_COLUMN,
    pext,8,process_list,ier)
```

A process grid can be defined over the full set of processes being used by an application or over any subset of those processes. This flexibility can be useful when circumstances call for setting up a process grid that does not include all available processes.

For example, if an application will be running in a two-node cluster where one node has 14 CPUs and the other has 10, better load balancing may be achieved by defining the process grid to have 10 processes in each node.

For more information about explicitly defining process grids, see “S3L_set_process_grid” on page 60 or the S3L_set_process_grid(3) man page.

Declaring S3L Arrays

Sun S3L provides two subroutines for declaring S3L arrays: S3L_declare and S3L_declare_detailed. The library also includes the S3L_DefineArray interface, which maintains compatibility with the Sun HPC 2.0 release of Sun S3L.

S3L_declare and S3L_declare_detailed perform the same function, except that S3L_declare_detailed provides additional arguments that allow more detailed control over the array features. Both require the programmer to specify

- The array’s rank
- The array’s extents
- The array’s type

- Which axes will be distributed and which will be local (kept in a single block on one process).
- The method by which the array is to be allocated.

In addition, S3L_declare_detailed allows the programmer to specify the following array features:

- The starting address of the local subgrid. This value is used only if the programmer elects to allocate array subgrids explicitly by disabling automatic array allocation.
- The block size to be used in distributing the array along each axis. The programmer has the option of letting Sun S3L choose a default blocksize.
- Which processes contain the start of each array axis. Again, the programmer can let Sun S3L specify default processes. To use this option, the programmer must specify a particular process grid, rather than using one provided by Sun S3L.

The following F77 example allocates a 100 x 100 x 100 double-precision array.

```
include ‘s3l/s3l-f.h’
integer*8 A,pg_a
integer*4 ext_a(3), block_a(3), local_a(3)
ext_a(1) = 100
ext_a(2) = 100
ext_a(3) = 100
local_a(1) = 1
local_a(2) = 0
local_a(3) = 0
call s3l_declare_detailed(A,0,3,ext_a,S3L_double,block_a,
    -1,local_a,pg_a,S3L_USE_MALLOC,ier)
```

The S3L array A is distributed along each axis of the process grid. The block sizes for the three axes are specified in block_a. Because local_a is set to 1, the first axis of A will be local to the first process in the process grid’s first axis. The second and third axes of A are distributed along the corresponding axes of the process grid.

If local_a had been set to 0 instead, all three array axes would be distributed along their respective process grid axes.

For more information about this function see “S3L_declare_detailed” on page 53 or the S3L_declare_detailed(3) man page.

The simpler and more compact S3L_declare involves fewer parameters and always block-distributes the arrays. The following C program example allocates a one-dimensional, double-precision array of length 1000.

```
#include <s3l/s3l-c.h>
int local,ext,ier;
S3L_array_t A;
local = 0;
ext = 1000;
ier = S3L_declare(&A,1,&ext,S3L_double,&local,S3L_USE_MALLOC);
```

This example illustrates use of the array_is_local parameter. This parameter consists of an array containing one element per axis. Each element of the array is either 1 or 0, depending on whether the corresponding array axis should be local to a process or distributed. If array_is_local(i) is 0, the array axis i will be distributed along the corresponding axis of the process grid. If it is 1, array axis i will not be distributed. Instead, the extent of that process grid axis will be regarded as 1 and the array axis will be local to the process.

In this S3L_declare example, the array has only one axis, so array_is_local has a single value, in this case 0. If the program containing this code is run on six processes, Sun S3L will associate a one-dimensional process grid of length 6 with the S3L array A. It will set the block size of the array distribution to ceiling(1000/6)=167. As a result, processes 0 though 4 will have 167 local array elements and process 5 will have 165.

If array_is_local had been set to 1, the entire array would have been allocated to process 0.

Deallocating S3L Arrays

When S3L arrays are not needed anymore, use S3L_free to deallocate them. This makes the memory resources available for other uses.

Distributing S3L Arrays

S3L arrays are distributed, axis-by-axis, either locally or in a block-cyclic pattern. When an axis is distributed locally, all indices along that axis are made local to a particular process. A locally distributed axis is sometimes referred to as *collapsed*.

An axis that is distributed block-cyclically is partitioned into blocks of some *useful* size and the blocks are distributed onto the processes in a round-robin fashion. That is,

- The first block goes to the first process, the second block to the second process, and for on. This continues until all processes have received an initial block.
- After the last process in the sequence has received its first block, the next block is sent to the first process, the block after that to the second process, and so on. This cycle is repeated until all blocks in the axis have been distributed.

The definition of a *useful* block size will vary, depending in large part on the kind of operation to be performed. See the discussion of S3L array distribution in the Sun HPC ClusterTools 3.1 Performance Guide for additional information about block-cyclic distribution and choosing block sizes.

A special case of block-cyclic distribution is block distribution. This involves choosing a larger block size, large enough to ensure that all blocks in the axis will be distributed on the first distribution cycle—that is, no process will receive more than one block. FIGURE 2-4 through FIGURE 2-6 illustrate block and cyclic distributions with a sample 8x8 array distributed onto a 2x2 process grid.

In FIGURE 2-4 and FIGURE 2-5, block size is set to 4 along both axes and the resulting blocks are distributed in pure block fashion. As a result, all the subgrid indices on any given process are contiguous along both axes.

The only difference between these two examples is that process grid ordering is column-major in FIGURE 2-4 and row-major in FIGURE 2-5.

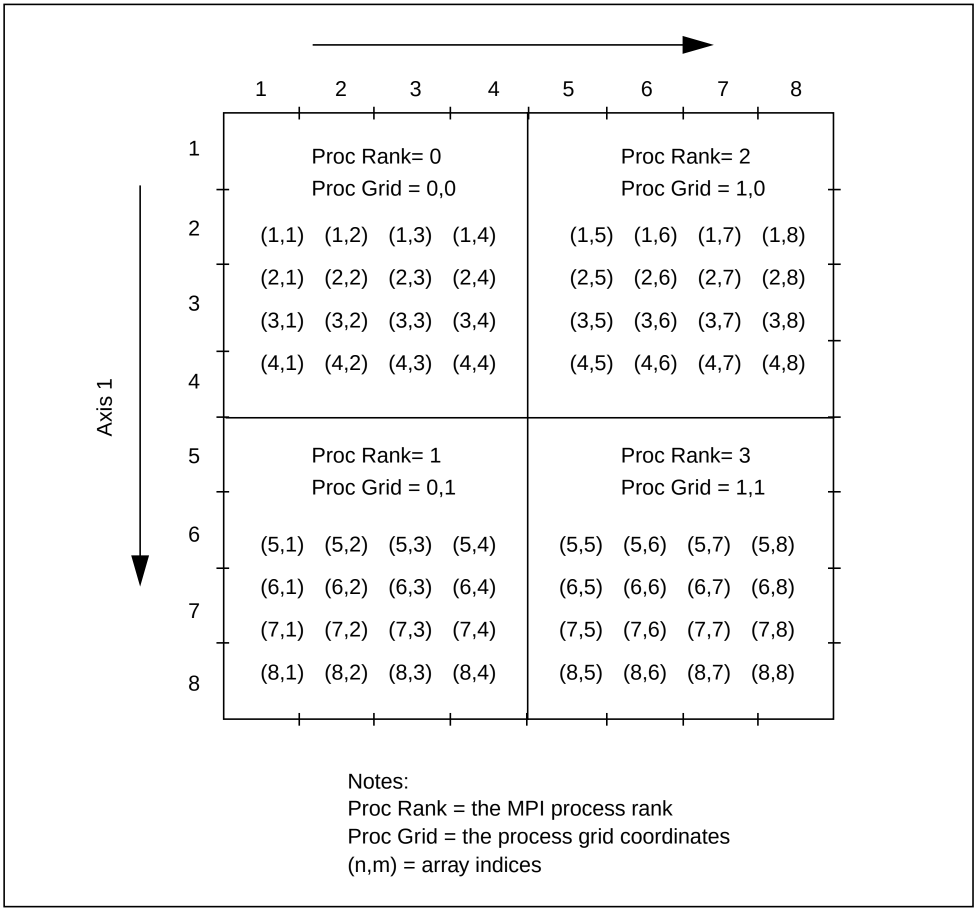

FIGURE 2-4 An 8x8 S3L Array Distributed on a 2x2 Process Grid Using Pure Block Distribution: Column-Major Order

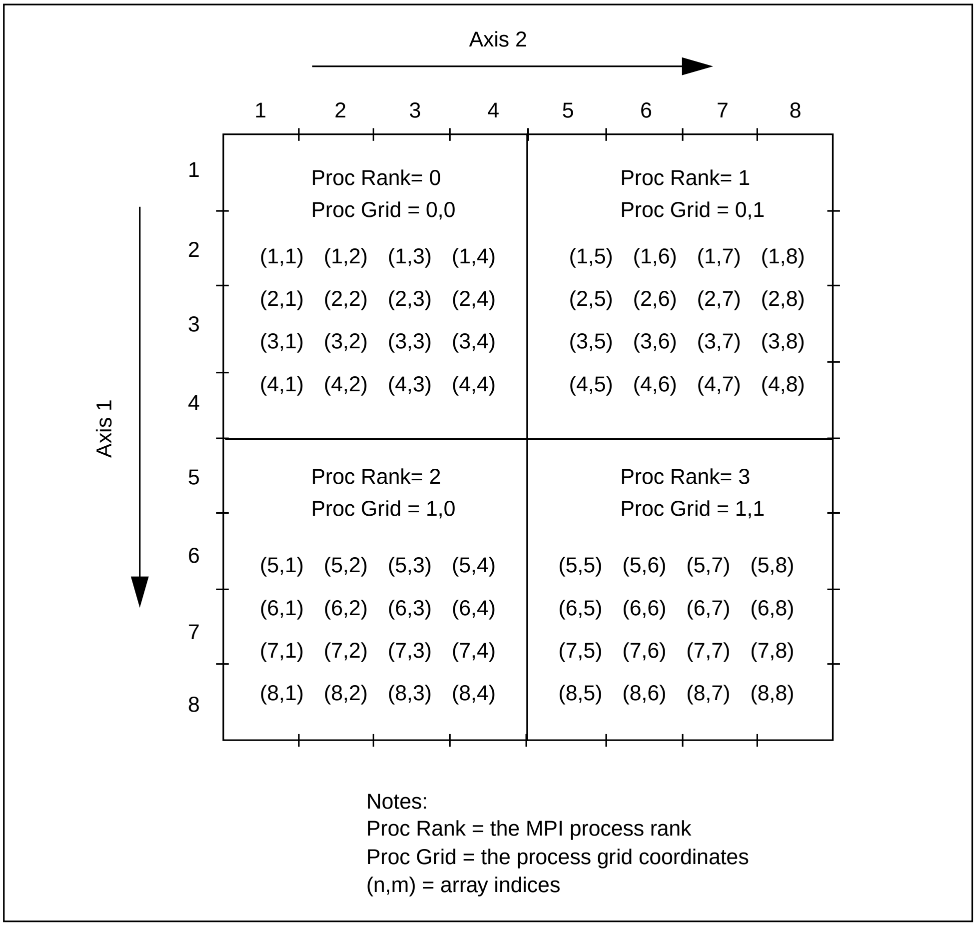

FIGURE 2-5 An 8x8 S3L Array Distribution on a 2x2 Process Grid Using Pure Block Distribution: Row-Major Ordering of Processes

FIGURE 2-6 shows block cyclic distribution of the same array. In this example, the block size for the first axis is set to 4 and the block size for the second axis is set to 2.

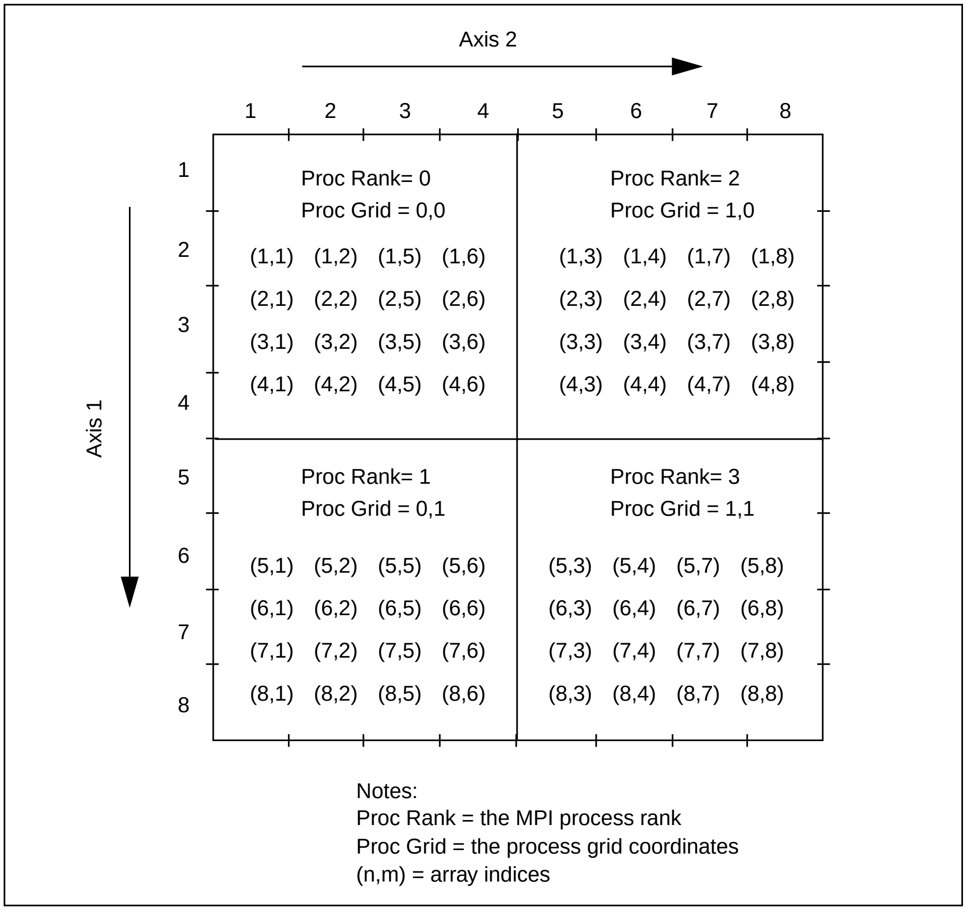

FIGURE 2-6 An 8x8 S3L Array Distributed on a 2x2 Process Grid Using Block-Cyclic Distribution: Column-Major Order

When no part of an S3L array is distributed—that is, when all axes are local—all elements of the array are on a single process. By default, this is the process with MPI rank 0. The programmer can request that an undistributed array be allocated to a particular process via the `S3L_declare_detailed` routine.

Although the elements of an undistributed array are defined only on a single process, the S3L array handle enables all other processes to access the undistributed array.

Examining the Contents of S3L Arrays

Printing S3L Arrays

The Sun S3L utilities S3L_print_array and S3L_print_sub_array can be used to print the values of a distributed S3L array to standard output.

S3L_print_array prints the whole array, while S3L_print_sub_array prints a section of the array that is defined by programmer-specified lower and upper bounds.

The values of array elements will be printed out in column-major order; this is referred to as Fortran ordering, where the leftmost axis index varies fastest.

Each element value is accompanied by the array indices for that value. This is illustrated by the following example.

a is a 4 x 5 x 2 S3L array, which has been initialized to random double-precision values via a call S3L_rand_lcg. A call to S3L_print_array will produce the following output:

```
     call s3l_print_array(a)
(1,1,1)    0.000525
(2,1,1)    0.795124
(3,1,1)    0.225717
(4,1,1)    0.371280
(1,2,1)    0.225035
(2,2,1)    0.878745
(3,2,1)    0.047473
(4,2,1)    0.180571
(1,3,1)    0.432766
...
```

For large S3L arrays, it is often a good idea to print only a section of the array, rather than the entire array. This not only reduces the time it takes to retrieve the data, but it can be difficult to locate useful information in displays of large amounts of data. Printing selected sections of a large array can make the task of finding data of interest much easier. This can be done using the function `S3L_print_sub_array`. The following example shows how to print only the first column of the array shown in the previous example:

```
      integer*4 lb(3),ub(3),st(3)

c          specify the lower and upper bounds
c          along each axis. Elements whose coordinates
c          are greater or equal to lb(i) and less or
c          equal to ub(i) (and with stride st(i)) are
c          printed to the output
      lb(1) = 1
      ub(1) = 4
      st(1) = 1
      lb(2) = 1
      ub(2) = 1
      st(2) = 1
      lb(3) = 1
      ub(3) = 1
      st(3) = 1
      call s3l_print_sub_array(a,lb,ub,st,ier)
```

The following output would be produced by this call

```
(1,1,1)    0.000525
(2,1,1)    0.795124
(3,1,1)    0.225717
(4,1,1)    0.371280
```

If a stride argument other than 1 is specified, only elements at the specified stride locations will be printed. For example, the following sets the stride for axis 1 to 2

```
st(1) = 2
```

which results in the following output:

```
(1,1,1)    0.000525
(3,1,1)    0.225717
```

Visualizing Distributed S3L Arrays With Prism

S3L arrays can be visualized with Prism, the debugger that is part of the Sun HPC ClusterTools suite. Before S3L arrays can be visualized, however, the programmer must instruct Prism that a variable of interest in an MPI code describes an S3L array.

For example, if variable `a` has been declared in a Fortran program to be of type `integer*8` and a corresponding S3L array of type `S3L_float` has been allocated by a call to an S3L array allocation function, the programmer should enter the following at the prism command prompt:

```
type float a
```

Once this is done, Prism can print values of the distributed array:

```
print a(1:2,4:6)
```

Or it can assign values to it:

```
assign a(2,10)=2.0
```

or visualize it

```
print a on dedicated
```

CHAPTER **3**

Sun S3L Data Types

Data type information is encoded in the S3L array handle for both C and Fortran interfaces and is decoded at run time. This allows appropriate branching to occur during execution, which makes it unnecessary to maintain separate routines with different names for each language interface.

TABLE 3-1 shows the data types supported for the various Sun S3L routines. TABLE 3-2 lists the C and Fortran language-specific data type equivalents.

Within each subroutine call, elements of all array arguments must match in data type, unless the argument descriptions indicate otherwise.

Place one of the following include lines at the top of any C or Fortran program unit that makes an S3L call:

C and C++ Programs

```
#include <s3l/s3l-c.h>
```

F77 and F90 Programs

```
include 's3l/s3l-f.h'
```

Note – For Sun S3L 2.0, the S3L array handles for the F77 interfaces are of type `integer*4` and for Sun S3L 3.0 and 3.1, they are of type `integer*8`. Therefore, when porting F77 programs from Sun S3L 2.0 to Sun S3L 3.0 or 3.1, be sure to change the array handle data type definitions accordingly. If you want your F77 program to be compatible across Sun S3L 2.0, Sun S3L 3.0 , and Sun S3L 3.1, you should insert `#ifdef` statements in appropriate places in the code.

TABLE 3-1 Array Data Types Supported for C/C++ and F77/F90

Operation	int	long integer	float	double	complex	dcomplex
2-norm			x	x	x	x
Autocorrelation			x	x	x	x
Convolve			x	x	x	x
Copy array	x	x	x	x	x	x
Circular shift	x	x	x	x	x	x
Declare array	x	x	x	x	x	x
Deconvolve			x	x	x	x
Define array	x	x	x	x	x	x
Describe array	x	x	x	x	x	x
Exit				– N/A –		
FFT, simple and detailed complex-to-complex					x	x
FFT, inverse					x	x
FFT, simple real-to-complex			x	x		
FFT, simple complex-to-real			x	x		
Forall	x	x	x	x	x	x
Free array handle	x	x	x	x	x	x
General band solver			x	x	x	x
General iterative solver			x	x	x	x
General least squares			x	x	x	x
General singular value decomposition (SVD)			x	x	x	x
General tridiagonal			x	x	x	x
Get array elements	x	x	x	x	x	x
Get array attributes	x	x	x	x	x	x
Grade up/down	x	x	x	x	x	x

TABLE 3-1 Array Data Types Supported for C/C++ and F77/F90 *(Continued)*

Operation	int	long integer	float	double	complex	dcomplex
Initialize S3L environment				– N/A –		
Inner product			x	x	x	x
LU factor			x	x	x	x
LU solve			x	x	x	x
LU invert			x	x	x	x
Matrix multiplication			x	x	x	x
Matrix vector multiplication			x	x	x	x
Matrix vector sparse			x	x	x	x
Outer product			x	x	x	x
Print array	x	x	x	x	x	x
Print sparse array			x	x	x	x
Read array	x	x	x	x	x	x
Read sparse array			x	x	x	x
Reduce	x	x	x	x	x	x
Reduce axis	x	x	x	x	x	x
RNG, lagged Fibonacci	x	x	x	x	x	x
RNG, linear congruential	x	x	x	x	x	x
RNG, sparse matrix			x	x	x	x
Set array elements	x	x	x	x	x	x
Set process grid				– N/A –		
Set safety				– N/A –		
Sort	x	x	x	x		
Thread communicator setup				– N/A –		
Symmetric eigenvalues, eigenvectors			x	x	x	x
Transpose	x	x	x	x	x	x
Write array	x	x	x	x	x	x
Zero elements	x	x	x	x	x	x

TABLE 3-2 Equivalent S3L, Fortran, and C Array Data Types

S3L Data Types	F77/F90 Data Types	C/C++ Data Types
`S3L_integer`	`INTEGER*4`	`int`
`S3L_long_integer`	`INTEGER*8`	`long long`
`S3L_float`	`REAL*4`	`float`
`S3L_double`	`REAL*8`	`double`

`S3L_complex`	`COMPLEX*8`	`typedef struct`
		`{ float real;`
		`float imag;`
		`} S3L_cmpx8`

`S3L_double_complex`	`COMPLEX*16`	`typedef struct cmpx16_s`
		`{ float double real;`
		`float double imag;`
		`} S3L_cmpx16`

CHAPTER 4

Multiple Instance

Most Sun S3L routines support *multiple instances;* that is, they allow you to perform multiple independent operations on different data sets concurrently. The routines that support multiple instance operations are listed below:

- S3L_2_norm
- S3L_fft_detailed
- S3L_gen_band_solve
- S3L_gen_iter_solve
- S3L_gen_lsq
- S3L_gen_svd
- S3L_gen_trid_solve
- S3L_inner_prod
- S3L_mat_mult
- S3L_mat_vec_mult
- S3L_outer_prod
- S3L_lu_invert
- S3L_lu_solve
- S3L_sym_eigen

Defining Multiple Independent Data Sets

To perform a Sun S3L operation on multiple independent data sets in parallel, you must embed the multiple independent instances of each operand or result argument in a parallel array.

The shape of the parallel array is defined by two kinds of axes:

- *Data axes* define the geometry of the individual instances of the operand or result.
- *Instance axes* label the multiple instances.

FIGURE 4-1 illustrates this with an example of a matrix-vector-multiplication operation in which four independent products are computed simultaneously. It shows how the destination and source vectors and the source matrix are organized with respect to the data and instance axes.

- The four destination vectors are embedded in a 2D parallel array with one data axis and one instance axis.
- The four source vectors are similarly embedded in another parallel array.
- The source matrices are embedded in a 3D parallel array.

The instances within each variable are labeled 0 through 3.

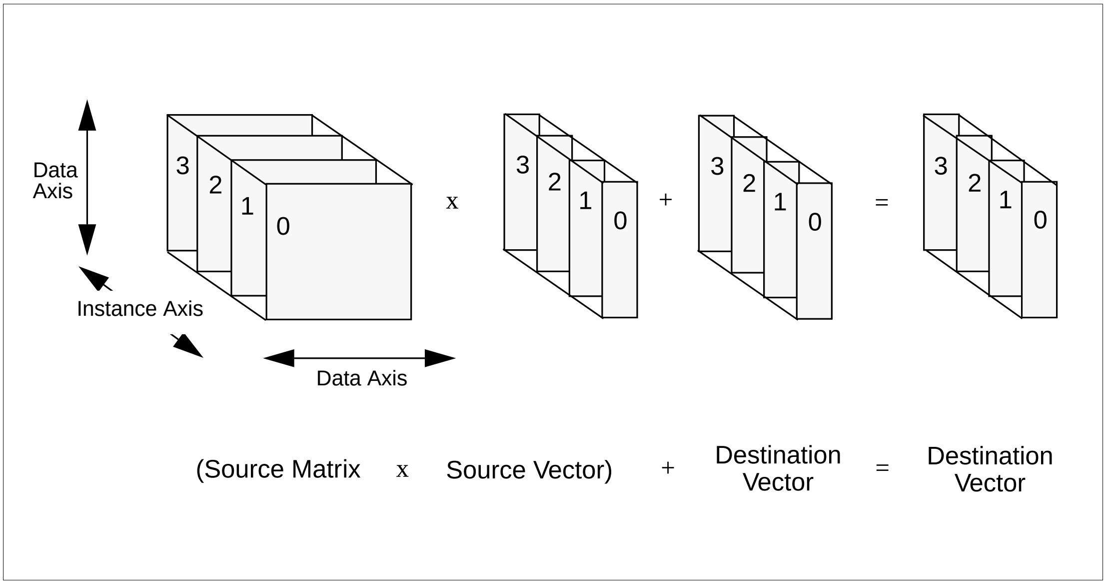

FIGURE 4-1 A Multiple-Instance Matrix-Vector Multiplication Problem

The logical unit on which the routine operates—sometimes called a *cell*—is defined by the data axes. The instance axes define the geometry of the *frame* in which the cells are embedded. The 3D parallel array shown in FIGURE 4-1 is a frame containing four 2-dimensional cells.

The product of the lengths of the instance axes is the total number of instances. The product of the lengths of the data axes is the size of the cell.

Rules for Data Axes and Instance Axes

When you organize your data to form cells and frames for a multiple-instance operation, apply the following rules:

- All parallel arrays involved in the operation must have the same number of instance axes.
- Counting up through the instance axes of the parallel arrays (excluding the data axes), corresponding instance axes must occur in the same order in each operand or result.
- The corresponding instance axes of the operands or results must have identical lengths. Certain routines also require that corresponding instance axes must also have identical layouts. The situations where identical layouts are required are identified in the applicable man pages.
- The lengths of the data axes must be defined so that the operation makes sense. For example, in matrix multiplication, the data axis lengths of the operand and result matrices must obey the standard rules for axis lengths in matrix multiplication. Specific requirements for data axis lengths are provided in the applicable man pages.
- Except where explicitly noted, Sun S3L supports all combinations of layouts for data axes and instance axes. Which layout will provide the best performance for any given operation depends largely on the nature of the operation.

In most cases, however, performance is best when the cells (that is, all of the data axes) are local to a processing element. Instance axes are typically defined as nonlocal axes. Some man pages for Sun S3L routines contain specific information about optimizing layouts.

"Specifying Single-Instance vs. Multiple-Instance Operations" on page 28 illustrates these rules being applied in a matrix-vector multiplication example.

Note – Most Sun S3L routines impose few or no restrictions on where the instance axes can occur in a parallel array.

Specifying Single-Instance vs. Multiple-Instance Operations

Sun S3L routines that support multiple instances have the same calling sequence for single-instance and multiple-instance operations. The methods for specifying single-instance and multiple-instance operations depend on which routine you are calling. The man pages for routines that are capable of multiple-instance operation contain specific information for their respective routines.

"Example 1: Matrix-Vector Multiplication" on page 28 explains the differences between single- and multiple-instance operation for the matrix-vector-multiplication routine. "Example 2: Fast Fourier Transforms" on page 33 discusses use of multiple instances in FFTs.

Example 1: Matrix-Vector Multiplication

When you call the matrix-vector-multiplication routine, `S3L_mat_vec_mult`, the dimensionality of the arguments you supply determines whether the routine performs a single-instance or multiple-instance operation. The F77 form of this Sun S3L function is

```
S3L_mat_vec_mult(y, a, x, y_vector_axis, row_axis, col_axis,
x_vector_axis, ier)
```

Note – The `S3L_mat_vec_mult` routine requires you to specify which axes you are using as data axes for each matrix or vector argument.

Single-Instance Operation

To perform a single-instance operation, specify each vector argument as a 1D parallel array and each matrix argument as a 2D parallel array. (Alternatively, you can declare these arguments to have more dimensions, but all instance axes must have length 1.)

For example, a single-instance operation in F77 can be performed by first defining the block-distributed arrays:

```
 integer*8 a, x, y
 integer*4 ext(2), axis_is_local(2)
 integer*4 ier

 axis_is_local(1) = 0
 axis_is_local(2) = 0

 ext(1) = p
 ext(2) = q

 call s3l_declare(a, 2, ext, S3L_float, axis_is_local,
$    S3L_USE_MALLOC, ier)

 call s3l_declare(x, 2, ext, S3L_float, axis_is_local,
$    S3L_USE_MALLOC, ier)

 call s3l_declare(y, 2, ext, S3L_float, axis_is_local,
$    S3L_USE_MALLOC, ier)
```

and then using

```
call S3L_mat_vec_mult(y, a, x, 1, 1, 2, 1, ier)
```

Arrays `x` and `y` are 1D; the definitions of `x_vector_axis` = 1 and `col_axis` = 2 indicate that the product `a`(`i`, `j`) * `x`(`j`) will be evaluated for all values of `j`. These products will be summed over the first index of `a` (`row_axis` = 1), and the result added to the corresponding element in `y`. The equivalent code is

```
do i = 1, p
    sum = 0.0
    do j = i, q
        sum = sum + a(i, j) * x(j)
    enddo
enddo
```

Multiple-Instance Operation

To perform a multiple-instance operation, embed the multiple instances of each vector argument in a parallel array of rank greater than 1, and embed the multiple instances of each matrix argument in a parallel array of rank greater than 2.

For example, the simplest multiple-instance matrix-vector multiplication involves the definition of one instance axis.

```
  integer*8 a, x, y
  integer*4 ext(3), axis_is_local(3)
  integer*4 ier

  axis_is_local(1) = 0
  axis_is_local(2) = 0
  axis_is_local(3) = 0

  ext(1) = p
  ext(2) = q
  ext(2) = r

  call s3l_declare(a, 3, ext, S3L_float, axis_is_local,
 $    S3L_USE_MALLOC, ier)

  ext(1) = q
  ext(2) = r

  call s3l_declare(x, 2, ext, S3L_float, axis_is_local,
 $    S3L_USE_MALLOC, ier)

  ext(1) = p
  ext(2) = r

  call s3l_declare(y, 2, ext, S3L_float, axis_is_local,
 $    S3L_USE_MALLOC, ier)
```

In this code, all three arrays contain an instance axis of length `r`. In addition, each instance axis is the rightmost axis in the array declaration. In other words, the order of data axes and instance axes is the same in all three arrays. These axes definitions produce arrays whose geometries are outlined in FIGURE 4-1. In the illustration, `r` = 4. Multiplication using these arrays is then performed by

```
call S3L_mat_vec_mult(y, a, x, 1, 1, 2, 1, ier)
```

In analyzing the operations performed in this call, it is useful to define `s0`, the index along the instance axis. For a given value of `s0`, the following will be evaluated:

- The values of `x_vector_axis` = 1 and `col_axis` = 2 indicate that the product `a(i, j, s0)` * `x(j, s0)` will be calculated for all `j`.
- The above product will be summed over `i`, the first index of `a` (`row_axis` = 1), and the result added to `y(i, s0)`.

These two operations will be performed for all 1 <= `s0` <= `r`. In other words, the matrix-vector multiplication will be evaluated for all instances

```
y(:, s0) * a(:, :, s0) * x(:, s0)
```

The order in which these instances are evaluated depends on the layouts of the arrays. Since all arrays are block-distributed along all axes, it is possible for one set of processes to work on the first instance

```
y(:, 1) = a(:, :, 1) * x(:, 1)
```

while another set of processors evaluates the *N*th instance at the same time—that is, in parallel .

```
y(:, N) = a(:, :, N) * x(:, N)
```

The extent of parallelism depends on the details of the data layouts, particularly on the mapping of the data and instance axes to the underlying process grid axes. The highest degree of parallelism is achieved when all data axes are local, and all instance axes are distributed.

The use of local data axes forces each cell (all data axes) to reside entirely in just one process. The use of distributed instance axes spreads the collection of cells over the process grid, resulting in better load-balancing among processes. Use of this data layout is discussed below.

Multiple-instance operations are usually most efficient when each cell (all of the data axes) resides on one process. Use of such a layout scheme is discussed in this section. In addition, the use of several instance axes are illustrated. Declarations of arrays containing these axes can be done as

```
integer*8 a, x, y
integer*4 mat_ext(5), mat_axis_is_local(5)
integer*4 vec_ext(4), vec_axis_is_local(4)
integer*4 ier

mat_axis_is_local(1) = 1
mat_axis_is_local(2) = 1
mat_axis_is_local(3) = 0
mat_axis_is_local(4) = 0
mat_axis_is_local(5) = 0

mat_ext(1) = p
mat_ext(2) = q
mat_ext(2) = k
mat_ext(4) = m
```

```
 mat_ext(5) = n

 call s3l_declare(a, 5, mat_ext, S3L_float, mat_axis_is_local,
$    S3L_USE_MALLOC, ier)

 vec_axis_is_local(1) = 1
 vec_axis_is_local(2) = 1
 vec_axis_is_local(3) = 0
 vec_axis_is_local(4) = 0
 vec_axis_is_local(5) = 0

 vec_ext(1) = q
 vec_ext(2) = k
 vec_ext(2) = m
 vec_ext(4) = n

 call s3l_declare(x, 4, vec_ext, S3L_float, vec_axis_is_local,
$    S3L_USE_MALLOC, ier)

 vec_ext(1) = p
 vec_ext(2) = k
 vec_ext(2) = m
 vec_ext(4) = n

 call s3l_declare(y, 4, vec_ext, S3L_float, vec_axis_is_local,
$    S3L_USE_MALLOC, ier)
```

The data axes are defined to be local to a process. Each array has three instance axes, each of which is block distributed. Note that the order of instance axes is the same in all three arrays.

To analyze the results of the call

```
call S3L_mat_vec_mult(y, a, x, 1, 1, 2, 1, ier)
```

`s0`, `s1`, and `s2` are used to denote the index along each of the three instance axes. For a given set of `s0`, `s1`, and `s2`, the following will be evaluated:

- The values of `x_vector_axis` = 1 and `col_axis` = 2 indicate that the product `a(i, j, s0, s1, s2)` * `x(j, s0, s1, s2)` will be calculated for all `j`.
- This product will be summed over `i`, the first index of `a` (`row_axis` = 1), and the result added to `y(i, s0, s1, s2)`.

These two operations will be performed for all 1 <= s0 <= k, 1 <= s1 <= m, and 1 <= s2 <= n. In other words, the matrix-vector multiplication will be evaluated for all instances

```
y(:, s0, s1, s2) = A(:, :, s0, s1, s2) * x(:, s0, s1, s2)
```

However, unlike the previous example, the data axes in this case are local. This means that the evaluation of each instance does not involve any interprocess communication. Each process independently works on its own set of instances, using a purely local matrix-vector-multiplication algorithm. These local algorithms are usually faster than their global counterparts, since no communication between processes is involved.

Source code for these operations is in the file mat_vec_mult.f. This can be found in the S3L examples directory examples/s3l/dense_matrix_ops-f/, the location of which is site-specific.

Example 2: Fast Fourier Transforms

When calling the detailed complex-to-complex FFT routine, S3L_fft_detailed, you can supply a multidimensional parallel array and specify whether you want to perform a forward transform, an inverse transform, or no transform along each axis. You can also specify axes along which no transform is performed, but address bits are reversed. The axes that are transformed or bit-reversed are the data axes, and define the cell; the axes along which you perform no transformation are the instance axes.

Note – The simple FFT routine, S3L_fft, performs a transform along each axis of the supplied parallel array. Consequently, it does not support multiple instances.

CHAPTER 5

Using Sun S3L

This chapter explains how to implement calls to S3L routines into your F77, F90, C or C++ program. The following topics are included:

- Creating a program that calls Sun S3L routines
- The Sun S3L safety mechanism
- Online sample code and man pages

Sun S3L documentation includes sample online programs that demonstrate how to call each Sun S3L routine. You are encouraged to experiment with these sample programs. Online man pages are also included for all Sun S3L routines. "Online Sample Code and Man Pages" on page 41 explains how to find the program examples.

Creating a Program that Calls Sun S3L Routines

▼ To use Sun S3L routines in a program:

1. **Place calls to Sun S3L routines into your code.**
2. **Include the appropriate header file in each program unit that calls Sun S3L routines.**

   See "Include the Sun S3L Header File" on page 36 for details.

3. Use the appropriate compiler command to compile your code; include the Sun S3L link switch on the command line.

See “Compiling and Linking” on page 37 for details.

The remainder of this section describes the steps listed above more fully.

Sun S3L requires the presence of the Sun Performance Library routines and its associated license file. This library is not installed with Sun S3L and other Sun HPC ClusterTools components. Instead, it is included as part of the following compiler suites:

- Sun WorkShop Compilers Fortran 4.2
  (also included in Sun Performance WorkShop Fortran 3.0).
- Sun Performance WorkShop Fortran 5.0.

Note – If possible, use `libsunperf` versions later than 1.1 for better performance.

Include the Sun S3L Header File

Place the appropriate include line at the top of any program unit that makes an S3L call. The correct include files are shown below for both C and Fortran language interfaces:

- C or C++

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
```

- F77 or F90

```
include ‘s3l/s3l-f.h’
include ‘s3l/s3l_errno-f.h’
```

The first line allows the program to access the header file containing prototypes of the routines and defines the symbols and data types required by the interface. The second line includes the header file containing error codes the routines might return.

If the compiler cannot find the Sun S3L include file, verify that a path to the directory does exist. The standard path is

```
/opt/SUNWhpc/include/
```

If the file appears to be missing, consult your system administrator.

Compiling and Linking

Compile your program and link in Sun S3L (along with any other libraries it needs).

The link-line switch `–ls3l` does more than just link in Sun S3L subroutines. Depending on which compiler has been invoked, it also automatically links any other libraries needed to augment Sun S3L, greatly simplifying the link line.

- F77

```
% tmf77 –dalign –o program program.f –ls3l
```

- F90

```
% tmf90 –dalign –o program program.f90 –ls3l
```

- C

```
% tmcc –dalign –o program program.c –ls3l
```

- C++

```
% tmCC –dalign –o program program.cc –ls3l
```

Note – The `–dalign` option is needed because `libs3l` and `libsunperf` libraries are compiled with it.

Executing Sun S3L Programs

Execute a program that has been linked with Sun S3L just as you would any other program compiled for running on a Sun HPC System.

To submit such an application to the LSF Batch system, use the LSF `bsub` command. For example,

```
% bsub –q hpc –fln 4 hpc.job
```

submits the executable `hpc.job` to the batch queue `hpc` and requests that it run on four processors. The LSF Batch system will launch `hpc.job` as soon as it reaches the top of the queue and all required resources become available.

Refer to the *LSF Batch User's Guide*, provided by Platform Computing Corporation, for instructions on submitting Sun MPI jobs to the LSF batch queues.

To submit `hpc.job` to the Sun HPC Cluster Runtime Environment (CRE), use the `mprun` command. For example,

```
% mprun –np 4 hpc.job
```

submits `hpc.job` to the CRE and requests that it run on four processes.

Refer to the *Sun HPC ClusterTools 3.1 User's Guide* for additional information.

The Sun S3L Safety Mechanism

The Sun S3L safety mechanism offers two basic features: It synchronizes the parallel processes so that you can pinpoint the area of code that generated an error. It also performs error checking and reports errors at a user-selectable level of detail.

Synchronization

When a Sun S3L application executes on multiple processes, the processes are generally running asynchronously with respect to one another. The Sun S3L safety mechanism provides an interface for explicitly synchronizing the processes to each Sun S3L call made by your code. It traps and reports errors, indicating when the errors occurred relative to the synchronization points.

Error Checking and Reporting

The safety mechanism can perform error checking and generate run-time error information at multiple levels of detail. You can turn safety checking on at any level during all or part of a program. One level checks for errors in the usage and arguments of the Sun S3L calls in your program; a more detailed level also checks for errors generated by internal Sun S3L routines. Examples of errors found and reported by the safety mechanism include the following:

- A supplied or returned data element that should be numerical is not. For example, it is identified as a Not a Number (NaN), or as infinity. NaNs are defined in the IEEE Standard for Binary Floating-Point Arithmetic.
- The code generates a division by 0 (for example, because of bad data, a user error, or an internal software problem).

Note – For performance reasons, Sun S3L conducts most of its argument checking and error handling independently on each process. Consequently, when the safety mechanism is enabled and an error is detected, different processes may return different error values.

Levels of Error Checking

The Sun S3L safety mechanism has four selectable levels: 0, 2, 5, and 9. These levels are defined in TABLE 5-1.

At levels 2, 5, and 9, some safety mechanism error messages are displayed at the terminal when you run the program; other information appears in the backtrace when you use a debugger such as Prism.

TABLE 5-1 S3L Safety Mechanism Levels

0	Turns off the safety mechanism. Explicit synchronization and error checking are not performed. This level is appropriate for production runs of code that has 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	Detects and reports all level-2 errors. In addition, 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	Checks for and reports all level 2 and level 5 errors, as well as errors generated by lower levels of code that were 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 performs all implemented error checking and exacts a very high performance price. It is appropriate for detailed debugging when a problem occurs.

Selecting a Safety Mechanism Level

You can select the desired S3L safety mechanism level in either of two ways:

- By setting the environment variable S3L_SAFETY
- By using the subroutine calls S3L_get_safety and S3L_set_safety in a program

These methods are described in “Setting the Sun S3L Safety Environment Variable” on page 40 and “Setting the Safety Level from Within a Program” on page 40.

Setting the Sun S3L Safety Environment Variable

The S3L_SAFETY environment variable takes a single argument, which can be the integer 0, 2, 5, or 9. For example, to select the highest level, enter:

```
% setenv S3L_SAFETY 9
```

One advantage of using the S3L_SAFETY environment variable is that you can set or change the safety level without recompiling your code.

Setting the Safety Level from Within a Program

To set the Sun S3L safety level from within your program, include the following subroutine call. Specify the desired level in the integer argument *n*:

- For C Programs

  S3L_set_safety(*n*)

- For Fortran Programs

  S3L_set_safety(*n*)

To see what Sun S3L safety level is currently in effect, include the following call. Again, specify the level of interest in the integer argument *n*:

- For C Programs

  n = S3L_get_safety

- For Fortran Programs

  call S3L_set_safety()

The advantage of using these calls from within a program is that you can set or obtain the safety level at any point within your code. However, you must recompile the code each time you change these calls.

Online Sample Code and Man Pages

Sample Code Directories

The online sample programs are located in subdirectories of the S3L examples directory. Separate C and F77 versions are provided. The generic relative path for these examples is

`examples/s3l/`*operation_class*`[–`*language_suffix*`]/`*example_name.language*

where `examples/s3l` is installed in a site-specific location.

operation_class is the name of the general class of Sun S3L routines that are illustrated by the example.

The *–language_suffix* is used to denote F77 implementations. Examples implemented in C do not include the *–language_suffix*.

example_name.language is the example's file name. The *language* extension is `.c`, or `.f`. For example,

```
examples/s3l/dense_matrix_ops-f/outer_prod.f
```

is the F77 version of a program example that illustrates use of `s3l_outer_prod` routines. The equivalent examples for C applications is

```
examples/s3l/dense_matrix_ops/outer_prod.c
```

Compiling and Running the Examples

Each example subdirectory has a makefile. Each makefile references the file `../Make.simple`. If you are copying the example sources and makefiles to one of your own subdirectories, you should also copy `Make.simple` to your subdirectory's parent directory. `Make.simple` contains definitions of compilers, compiler flags and other variables that are needed to compile and run the examples. Note that the compiler flags in this file will *not* provide you with highly optimized executables. Information on optimization flags is best obtained from the documentation for the compiler of interest.

Each makefile has several targets that are meant to simplify the compilation and execution of examples. If you want to compile the source codes and create all executables in a particular example directory, use the command `make`.

If you wish to run the executables, enter `make run`. This command will also perform any necessary compilation and linking steps, so you need not issue `make` before entering `make run`.

By default, your executables will be run on two processes. You can change this by specifying the `NPROCS` variable on the command line. For example,

```
% make run NPROCS=4
```

will start your runs on four processes.

Executables and object files can be deleted by `make clean`.

Man Pages

To read the online man page for a Sun S3L routine, enter

```
% man routine_name
```

Chapter 6 and Chapter 7 also describe the Sun S3L routines. Chapter 6 covers the set of toolkit routines and Chapter 7 describes the core (computational) routines.

CHAPTER 6

Sun S3L Toolkit Routines

Sun S3L provides an extensive subset of auxiliary routines, referred to as the *toolkit* functions, which enable you to do the following.

- Set up the proper environment to support calls to Sun S3L subroutines:
  - `S3L_init` – See “`S3L_init`” on page 45
- Exit the Sun S3L environment once use of the library is over:
  - `S3l_exit` – See “`S3L_exit`” on page 47
- Set up parallel arrays and allocate memory for them:
  - `S3L_declare` – See “`S3L_declare`” on page 49
  - `S3L_declare_detailed` – See “`S3L_declare_detailed`” on page 53
  - `S3L_DefineArray` – See “`S3L_DefineArray`” on page 57
- Defining and freeing parallel process grids:
  - `S3L_set_process_grid` – See “`S3L_set_process_grid`” on page 60
  - `S3L_free_process_grid` – See “`S3L_free_process_grid`” on page 63
- Undefine a parallel array:
  - `S3L_free` – See “`S3L_free`” on page 64
  - `S3L_UnDefineArray` – See “`S3L_UnDefineArray`” on page 66
- Perform operations on elements of parallel arrays:
  - `S3L_array_op1` – See “`S3L_array_op1`” on page 68
  - `S3L_array_op2` – See “`S3L_array_op2`” on page 70
  - `S3L_array_scalar_op2` – See “`S3L_array_scalar_op2`” on page 73
  - `S3L_cshift` – See “`S3L_cshift`” on page 75
  - `S3L_forall` – See “`S3L_forall`” on page 78
  - `S3L_reduce` – See “`S3L_reduce`” on page 81
  - `S3L_reduce_axis` – See “`S3L_reduce_axis`” on page 83

- S3L_set_array_element – See “S3L_set_array_element, S3L_get_array_element, S3L_set_array_element_on_proc, and S3L_get_array_element_on_proc” on page 86
- S3L_set_array_element_on_proc – “S3L_set_array_element, S3L_get_array_element, S3L_set_array_element_on_proc, and S3L_get_array_element_on_proc” on page 86
- S3L_get_array_element – See “S3L_set_array_element, S3L_get_array_element, S3L_set_array_element_on_proc, and S3L_get_array_element_on_proc” on page 86
- S3L_get_array_element_on_proc – See “S3L_set_array_element, S3L_get_array_element, S3L_set_array_element_on_proc, and S3L_get_array_element_on_proc” on page 86
- S3L_zero_elements – See “S3L_zero_elements” on page 89

- Extract various kinds of information about parallel arrays and subgrids:
  - S3L_describe – See “S3L_describe” on page 90
  - S3L_get_attribute – See “S3L_get_attribute” on page 93
- Read a file into a parallel array, write all or part of a parallel array to a file, and print all or part of a parallel array to standard out:
  - S3L_read_array – See “S3L_read_array and S3L_read_sub_array” on page 97
  - S3L_read_sub_array – See “S3L_read_array and S3L_read_sub_array” on page 97
  - S3L_print_array – See “S3L_print_array and S3L_print_sub_array” on page 100
  - S3L_print_sub_array – See “S3L_print_array and S3L_print_sub_array” on page 100
  - S3L_write_array – See “S3L_write_array and S3L_write_sub_array” on page 103
  - S3L_write_sub_array – See “S3L_write_array and S3L_write_sub_array” on page 103
- Copy the contents of one parallel array into another:
  - S3L_copy_array – See “S3L_copy_array” on page 106
- Convert ScaLAPACK descriptors to S3L arrays and vice versa:
  - S3L_from_ScaLAPACK_desc – See “S3L_from_ScaLAPACK_desc” on page 108
  - S3L_to_ScaLAPACK_desc – See “S3L_to_ScaLAPACK_desc” on page 110
- Miscellaneous general control functions:
  - S3L_thread_comm_setup – See “S3L_thread_comm_setup” on page 113
  - S3L_set_safety – See “S3L_set_safety” on page 115

- S3L_get_safety – See "S3L_get_safety" on page 118

Setting Up a Sun S3L Environment

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 are illustrated below.

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

`../examples/s3l/utils/copy_array.c`

`../examples/s3l/utils/copy_array.f`

Related Functions

S3L_exit(3)

Leaving a Sun S3L Environment

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 are illustrated below.

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 value:

- `S3L_ERR_NOT_INIT` – S3L has not been initialized.

Examples

`../examples/s3l/dense_matrix_ops/inner_prod.c`

`../examples/s3l/dense_matrix_ops-f/inner_prod.f`

`../examples/s3l/utils/copy_array.f`

Related Functions

S3L_init(3)

Declaring Parallel Arrays

The Sun S3L toolkit functions described in this section share a common purpose—they all enable you to define parallel arrays that can then be operated on by other Sun S3L routines. Each of these routines returns an S3L array handle, which the application uses to reference the parallel array in subsequent S3L calls.

Each array declaring routine is described separately below.

- Declare a parallel array – See "S3L_declare" on page 49.
- Declare a parallel array with detailed control over attributes – See "S3L_declare_detailed" on page 53
- Define a parallel array – "S3L_DefineArray" on page 57.

Note – Use either S3L_declare() or S3L_declare_detailed() rather than S3L_DefineArray(). S3L_declare() and S3L_declare_detailed() are much simpler to use.

Sun S3L also provides a routine for declaring a sparse matrix. This routine, called, S3L_declare_sparse is described in "S3L_declare_sparse" on page 148.

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` are shown below.

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

- `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 Chapter 3 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-bit 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 value:

- S3L_ERR_ARG_RANK – The rank specified is invalid.
- 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 can not 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

```
../examples/s3l/transpose/ex_trans1.c
../examples/s3l/grade-f/ex_grade.f
```

Related Functions

```
S3L_declare_detailed(3)
S3L_free(3)
```

S3L_declare_detailed

Description

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

Syntax

The C and Fortran syntax for `S3L_declare_detailed` are shown below.

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*8          type
    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 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 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. See Chapter 3 for a complete list of supported data types.
- blocksizes – Specifies the blocksize to be used in distributing the array axes. If blocksizes is NULL (C/C++) or if its first element is equal to –1 (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 less than zero (F77/F90), default values will be used. Along each axis, the process whose process grid coordinate along that axis is equal to 0 contains the first array element. If present, the pgrid argument (process grid) should also be present. Otherwise an error code will be returned.
- axis_is_local – An integer vector whose length equals the number of dimensions in the array. 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.

  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.

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

- 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 – 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-bit 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.

Output

- A – S3L_declare_detailed returns the array handle in A.
- 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 value:

- S3L_ERR_ARG_RANK – The rank specified is invalid in one of the following ways:
  - If called from a C/C++ program, it is negative or greater than 31
  - If called from an F77/F90 program, it is zero or greater than 32.
- S3L_ERR_ARG_EXTENTS – One or more of the array extents is less than 1.
- S3L_ERR_ARG_BLKSIZE – One or more blocksizes 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

../examples/s3l/utils/copy_array.c

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

../examples/s3l/utils/get_attribute.c

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

../examples/s3l/utils/scalapack_conv.c

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

Related Functions

S3L_declare(3)

S3L_free(3)

S3L_get_process_grid(3)

S3L_set_process_grid(3)

S3L_DefineArray

Description

S3L_DefineArray associates an internal S3L array handle to a user-distributed parallel array. The array must be distributed in such a manner that it can be expressed as a block cyclic distribution. The array handle returned by S3L_DefineArray can then be used in subsequent calls by Sun MPI programs to S3L functions.

S3L_DefineArray does not allocate the memory required to store the local (process specific) part on the array. The user must allocate sufficient memory on each process to hold the local part of the parallel array before calling this function.

Syntax

The C and Fortran syntax for S3L_DefineArray are shown below.

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
S3L_array_t
S3L_DefineArray(address, rank, type, extents, blocks, sprocs,
p_ext)
    void              address
    int               rank
    int               type
    int               *extents
    int               *blocks
    int               *sprocs
    int               *p_ext
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
integer*8 function
S3L_DefineArray(address, rank, type, extents, blocks, sprocs,
p_ext)
    <type>            array(1)
    pointer           (addr_a,array)
    integer*4         rank
    integer*8         type
    integer*4         extents(*)
    integer*4         blocks(*)
    integer*4         sprocs(*)
    integer*4         p_ext(*)
```

Input

S3L_DefineArray accepts the following arguments as input:

- address – The starting address of the local (within the process) portion of a parallel array. In C, the user must allocate this local memory (for example, via the malloc function). In F77, the address is defined as a pointer to a local storage area.
- rank – Specifies the number of dimensions the array will have. The range of legal values for rank is 1 <= rank <= 31.
- type – Denotes the parallel array's data type. In C, it is a variable of type S3L_data_type. In F77, it is a variable of type integer*4.
- extents – Specifies the extents of the parallel array.
- blocks – Specifies the block sizes to be used in distributing each axis of the parallel array.
- sprocs – The starting processes of the block cyclic array distributions. If sprocs[i] = j, block 0 of the block cyclic distribution of the array along axis [i] is located in process j.
- p_ext – The extents of the process grid upon which the array is distributed. If, for example, p_ext[0] = 2, p_ext[1] = 3, p_ext[2] = 4, the three-dimensional parallel array will be distributed on a 2 x 3 x 4 process grid.

Note that process ordering within the process grid is always column major (F77 major) and that the product of the extents of the process grid must equal the total number of processes participating in the computation (that is, must equal the size of MPI_COMM_WORLD).

Error Handling

On success, S3L_DefineArray returns an S3L array handle that can be used for subsequent calls to other Sun S3L functions.

On error, it returns 0.

Examples

../examples/s3l/api

../examples/s3l/api-f

Related Functions

S3L_UnDefineArray(3)

Parallel Process Grids

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_DefineArray are shown below.

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 number of dimensions the array will have. The range of legal values for `rank` is 1 <= `rank` <= 31.
- `majorness` – Use 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` – Specifies the length of the process. In C/C++ programs, if the `process_list` argument is a `NULL` pointer, `plist_length` must be 0.
- `process_list` – Integer array whose length equals the number of processes in the grid. It contains a list of processes. For C/C++ programs, if `process_list` is a `NULL` pointer, `plist_length` must be 0.

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 indicating which value of the array handle was invalid 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_RANK – Invalid rank.
- S3L_ERR_ARG_MAJOR – Invalid majorness.
- 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

```
../examples/s3l/utils/scalapack_conv.c
../examples/s3l/utils-f/scalapack_conv.f
```

Related Functions

```
S3L_declare_detailed(3)
S3L_free_process_grid(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_DefineArray are shown below.

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 arguments 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 arguments 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, the following error code may be returned:

- S3L_ERR_PGRID_NULL – An invalid process grid argument was supplied.

Examples

../examples/s3l/utils/scalapack_conv.c

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

Related Functions

S3L_set_process_grid(3)

Deallocating Parallel Arrays

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` option set to `S3L_DONOT_ALLOCATE`.

Syntax

The C and Fortran syntax for `S3L_free` are shown below.

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, the following error code may be returned:

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

Examples

```
../examples/s3l/io/ex_print1.c
../examples/s3l/io-f/ex_print1.f
```

Related Functions

```
S3L_declare(3)
S3L_declare_detailed(3)
```

S3L_UnDefineArray

Description

S3L_UnDefineArray frees the array handle and the associated memory that were set up by a previous call to S3L_DefineArray.

Note – S3L_UnDefineArray does not free the local (process-resident) memory, where the local part of a parallel array is stored. The user is responsible for deallocating local memory assigned to the parallel array before the parallel program exits.

Syntax

The C and Fortran syntax for S3L_UnDefineArray are shown below.

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_UnDefineArray(a)
    S3L_array_t        a
```

F77/F90 Syntax

```
include 's3l/s3l-f.h'
include 's3l/s3l_errno-f.h'
subroutine
S3L_UnDefineArray(a)
    integer*8          a
```

Input

S3L_UnDefineArray 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_DefineArray.

Error Handling

S3L_UnDefineArray does not return any value.

Examples

../examples/s3l/api

../examples/s3l/api.f

../examples/s3l/array_utils

Related Functions

S3L_DefineArray(3)

S3L_declare(3)

S3L_declare_detailed(3)

S3L_free(3)

Performing Operations on S3L Parallel Arrays

The toolkit functions described in this section enable the user to apply various kinds of operations on a parallel array's elements.

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 are shown below.

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 indicating which value of the array handle was invalid is returned. 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

../examples/s3l/fft/ex_fft1.c

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

Related Functions

S3L_array_op2(3)

S3L_array_scalar_op2(3)

S3L_reduce_scalar(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 differences.

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 are shown below.

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 indicating which value of the array handle was invalid 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_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

```
../examples/s3l/fft/ex_fft1.c
../examples/s3l/fft-f/ex_fft1.f
```

Related Functions

S3L_array_op1(3)

S3L_array_scalar_op2(3)

S3L_reduce_scalar(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 are shown below.

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
    <void>             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 indicating which value of the array handle was invalid is returned. 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

../examples/s3l/fft/ex_fft1.c

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

Related Functions

S3L_array_op1(3)

S3L_array_op2(3)

S3L_reduce_scalar(3)

S3L_cshift

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 upwards 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 upwards, as shown below:

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 are shown below.

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 indicating which value of the array handle was invalid is returned. 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_AXISNUM – Invalid axis value.
- S3L_ERR_INDX_INVALID – index value is out of range.

Examples

```
../examples/s3l/utils/cshift_reduce.c
../examples/s3l/utils-f/cshift_reduce.f
```

Related Functions

S3L_reduce

S3L_reduce_axis

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 6-1.

TABLE 6-1 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_ELEM_FN	void user_fn(void *elem_addr, int *coord);	`subroutine user_fn(a, 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_cshift` are shown below.

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. A triplet takes the form:

inclusive lower bound
inclusive upper bound
stride

for each axis of the array. 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 indicating which value of the array handle was invalid is returned. 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_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

../examples/s3l/forall/ex_forall.c

../examples/s3l/forall/ex_forall2.cc

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

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 are shown below.

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; it can be one of:
  - `S3L_OP_SUM`
  - `S3L_OP_MIN`
  - `S3L_OP_MAX`

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 indicating which value of the array handle was invalid 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_OP – Requested operation is not supported.
- S3L_ERR_ARG_DTYPE – Invalid data type.

Examples

../examples/s3l/utils/cshift_reduce.c

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

Related Functions

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 are shown below.

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 indicating which value of the array handle was invalid 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_OP – An illegal operation was requested.
- 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 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 thearray or smaller than 1 (Fortran interface). For the C interface, the axis value would be equal to or larger than the rank of the array or smaller than 0.

Examples

../examples/s3l/utils/cshift_reduce.c

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

Related Functions

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 allow the user to know which process will participate in the set/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, using 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 are shown below.

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_set_array_element(a, coor, val)
S3L_set_array_element(a, coor, val)
S3L_set_array_element_on_proc(a, coor, val, pnum)
S3L_set_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_set_array_element(a, coor, val, ier)
S3L_set_array_element_on_proc(a, coor, val, pnum, ier)
S3L_set_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 one of: `integer*4`, `real*4`, `real*8`, `complex*8`, or `complex*16`.

Input

`S3L_set_array_element` and `S3L_get_array_element` 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 only used with S3L_set_array_element_on_proc and S3L_get_array_element_on_proc.

Output

These functions 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 return the associated error code and terminate.

- S3L_ERR_ARG_DTYPE – The data type of array a is not one of:
  - 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

../examples/s3l/utils/cshift_reduce.c

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

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 are illustrated below.

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

- A – S3L internal array handle for the parallel array that is to be initialized to zero.

Output

This function 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_zero_elements returns S3L_SUCCESS.

S3L_zero_elements checks the arrays it accepts as argument. If the array argument contains an invalid or corrupted value, the function terminates and an error code indicating which value of the array handle was invalid is returned. 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

../examples/s3l/utils/zero_elements.c

../examples/s3l/utils-f/zero_elements.f

Extracting Information About S3L Parallel Arrays

The functions described in this section allow users to retrieve information about parallel arrays for which an array handle exists.

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 is provided:

- Information on the rank extents and the data type of the array, as well as the starting address in memory of its subgrid.

Note: 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.

- A description of the underlying grid of processes to which data is mapped.

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 are shown below.

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 indicating which value of the array handle was invalid is returned. 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

```
../examples/s3l/utils/scalapack_conv.c
../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_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_COLUMN (F77 major) or S3L_MAJOR_ROW (C 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-bit 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_NONZEROS – For an S3L sparse matrix, S3L_NONZEROS returns in attr the number of nonzero elements of that matrix.
- 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, 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: User must not change the data returned in attr. It is created for internal use only.

- S3L_NRZS_SGRID_ADDR – For an S3L sparse matrix, 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, 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_NRZS_SGRID_SIZE – For an S3L sparse matrix, 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 are shown below.

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 type pointer. In all other cases, it should be of type integer*4.

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 indicating which value of the array handle was invalid is returned. 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

```
../examples/s3l/utils/get_attribute.c
../examples/s3l/utils-f/get_attribute.f
```

Related Functions

S3L_set_array_element(3)

Reading Data Into and Printing From S3L Parallel Arrays

The toolkit functions described in this section allow the user to read data from a file into a parallel array, to write data from a parallel array into a local file, and to print data from a parallel array to standard output.

S3L_read_array and S3L_read_sub_array

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 are shown below.

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                d *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 indicating which value of the array handle was invalid 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_RANGE_INV – The given range of indices is invalid:
  - A lower bound is less that 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_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

`../examples/s3l/io/ex_io.c`

`../examples/s3l/io-f/ex_io.f`

Related Functions

`S3L_print_array(3)`

`S3L_write_array(3)`

`S3L_print_array` and `S3L_print_sub_array`

`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` are shown below.

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 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 indicating which value of the array handle was invalid is returned. 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 that 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.

Examples

```
../examples/s3l/io/ex_print1.c
../examples/s3l/io/ex_io.c
../examples/s3l/io-f/ex_io.f
```

Related Functions

```
S3L_read_array(3)
S3L_write_array(3)
```

S3L_write_array and S3L_write_sub_array

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 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_write_array and S3L_write_sub_array are shown below.

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 indicating which value of the array handle was invalid 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_RANGE_INV` – The given range of indices is invalid:
  - A lower bound is less that 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

```
../examples/s3l/io/ex_io.c
../examples/s3l/io-f/ex_io.f
```

Related Functions

```
S3L_print_array(3)
S3L_read_array(3)
```

Copy Array

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 are illustrated below.

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

- A – S3L_array handle for the parallel array to be copied.

Output

This function 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 indicating which value of the array handle was invalid 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 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

```
../examples/s3l/utils/copy_array.c
../examples/s3l/utils-f/copy_array.f
```

Converting Between ScaLAPACK Descriptors and S3L Array Handles

The functions described in this section make it possible to convert ScaLAPACK descriptors to S3L array handles and vice versa.

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 are shown below.

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 either be 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 indicating which value of the array handle was invalid 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_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

../examples/s3l/utils/scalapack_conv.c

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

Related Functions

S3L_to_ScaLAPACK_desc(3)

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 are shown below.

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 arguments 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 argument 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_from_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 indicating which value of the array handle was invalid 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_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

```
../examples/s3l/utils/scalapack_conv.c
../examples/s3l/utils-f/scalapack_conv.f
```

Related Functions

S3L_from_ScaLAPACK_desc(3)

Performing Miscellaneous S3L Control Functions

This section describes three toolkit functions that support the following tasks:

- Enabling thread-safe operation of Sun S3L routines.
- Checking the current safety level.
- Setting the safety level.

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 via 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 are shown below.

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 indicating which value of the array handle was invalid 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_NULL – The comm argument is a NULL pointer (C/C++) or 0 (F77/F90)
- S3L_ERR_COMM_INVAL – The comm argument is not congruent to MPI_COMM_WORLD.
- S3L_ERR_THREAD_UNSAFE – The application program is using libraries that are not thread-safe.

Examples

See the following C and Fortran 77 program examples for illustrations of S3L_thread_comm_setup in use:

```
../examples/s3l/dense_matrix_ops/inner_prod_mt.c
../examples/s3l/dense_matrix_ops/matmult_mt.c
```

Related Functions

The following may be of related interest.

MPI_Comm_dup(3)

S3L_set_safety(3)

threads(3T)

Also, "Multithreaded Programming" is a relevant section in the *Sun HPC ClusterTools 3.1 User's Guide*.

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 6-2.

TABLE 6-2 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 calls `S3L_set_safety` and `S3L_get_safety` in a program.

To set the S3L safety level using the `S3L_SAFETY` environment variable, issue the command:

```
setenv S3L_SAFETY {0|2|5|9}
```

Syntax

The C and Fortran syntax for `S3L_set_safety` are shown below.

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 return the associated error code and terminate:

- `S3L_ERR_SAFELEV_INVAL` – Safety level has an invalid value.

Examples

```
../examples/s3l/utils/copy_array.c
../examples/s3l/utils-f/copy_array.f
```

Related Functions

S3L_get_safety(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 6-2.

Syntax

The C and Fortran syntax for S3L_set_safety are shown below.

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`.

Examples

`../examples/s3l/utils/copy_array.c`

`../examples/s3l/utils-f/copy_array.f`

Related Functions

`S3L_set_safety(3)`

CHAPTER 7

Sun S3L Core Library Functions

This chapter describes the set of computational functions, which form the core of the scientific subroutine library. These descriptions are organized as follows:

- S3L_rc_fft – See "S3L_rc_fft and S3L_cr_fft" on page 188
- S3L_cr_fft – See "S3L_rc_fft and S3L_cr_fft" on page 188
- S3L_fft_setup – See Section "S3L_fft_setup" on page 193
- S3L_rc_fft_setup – See "S3L_rc_fft_setup" on page 196
- S3L_fft_free_setup – See "S3L_fft_free_setup" on page 198
- S3L_rc_fft_free_setup – See "S3L_rc_fft_free_setup" on page 200

- Structured Solvers
  - S3L_gen_band_factor – See "S3L_gen_band_factor" on page 202
  - S3L_gen_band_free_factors – See "S3L_gen_band_free_factors" on page 205
  - S3L_gen_band_solve – See "S3L_gen_band_solve" on page 207
  - S3L_gen_trid_factor – See "S3L_gen_trid_factor" on page 211
  - S3L_gen_trid_free_factors – See "S3L_gen_trid_free_factors" on page 214
  - S3L_gen_trid_solve – See "S3L_gen_trid_solve" on page 215
- Dense Symmetric Eigenvalue Solve
  - S3L_sym_eigen – See "S3L_sym_eigen" on page 218
- Parallel Random Number Generators
  - S3L_setup_rand_fib – See "S3L_setup_rand_fib" on page 222
  - S3L_free_rand_fib – See "S3L_free_rand_fib" on page 224
  - S3L_rand_fib – See "S3L_rand_fib" on page 226
  - S3L_rand_lcg – See "S3L_rand_lcg" on page 228
- Least Squares Solver
  - S3L_gen_lsq – See "S3L_gen_lsq" on page 230
- Dense Singular Value Decomposition
  - S3L_gen_svd – See "S3L_gen_svd" on page 233
- Iterative Solver
  - S3L_gen_iter_solve – See "S3L_gen_iter_solve" on page 236
- Auto-correlation
  - S3L_acorr_setup – See "S3L_acorr_setup" on page 244
  - S3L_acorr_free_setup – See "S3L_acorr_free_setup" on page 246
  - S3L_acorr – See "S3L_acorr" on page 248
- Convolution / Deconvolution
  - S3L_conv_setup – See "S3L_conv_setup" on page 251
  - S3L_conv_free_setup – see "S3L_deconv_free_setup" on page 260

- S3L_conv – See “S3L_conv” on page 255
- S3L_deconv_setup – See “S3L_deconv_setup” on page 258
- S3L_deconv_free_setup – See “S3L_deconv_free_setup” on page 260
- S3L_deconv – See “S3L_deconv” on page 261

- Multidimensional Sort and Grade
  - S3L_grade_up – See “S3L_grade_down, S3L_grade_up, S3L_grade_down_detailed, S3L_grade_up_detailed” on page 265
  - S3L_grade_down – See “S3L_grade_down, S3L_grade_up, S3L_grade_down_detailed, S3L_grade_up_detailed” on page 265
  - S3L_grade_detailed_up – See “S3L_grade_down, S3L_grade_up, S3L_grade_down_detailed, S3L_grade_up_detailed” on page 265
  - S3L_grade_detailed_down – See “S3L_grade_down, S3L_grade_up, S3L_grade_down_detailed, S3L_grade_up_detailed” on page 265
  - S3L_sort – See “S3L_sort, S3L_sort_up, S3L_sort_down, S3L_sort_detailed_up, S3L_sort_detailed_down” on page 270
  - S3L_sort_up – See “S3L_sort, S3L_sort_up, S3L_sort_down, S3L_sort_detailed_up, S3L_sort_detailed_down” on page 270
  - S3L_sort_down – See “S3L_sort, S3L_sort_up, S3L_sort_down, S3L_sort_detailed_up, S3L_sort_detailed_down” on page 270
  - S3L_sort_detailed_up – See “S3L_sort, S3L_sort_up, S3L_sort_down, S3L_sort_detailed_up, S3L_sort_detailed_down” on page 270
  - S3L_sort_detailed_down – See “S3L_sort, S3L_sort_up, S3L_sort_down, S3L_sort_detailed_up, S3L_sort_detailed_down” on page 270
- Parallel Transpose
  - S3L_trans – See “S3L_trans” on page 275

Dense Matrix Routines

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 7-1.

TABLE 7-1 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 7-2.

TABLE 7-2 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, S3L_gbl_2_norm overwrites a with the global 2-norm of x.

Syntax

The C and Fortran syntax for S3L_2_norm and S3L_gbl_2_norm are shown below.

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

- 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

These functions use the following argument for 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 indicating which value of the array handle was invalid is returned. 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

../examples/s3l/dense_matrix_ops/norm2.c

../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_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 7-3.

TABLE 7-3 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
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 it 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 via 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 the instance axes of x and y—that is, the axes along which the inner product will be taken—each 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 7-4.

TABLE 7-4 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

TABLE 7-4 S3L Single-Instance Inner Product Operations *(Continued)*

Routine	Operation	Data Type
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

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 are shown below.

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)
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)
```

```
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

- `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 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 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

These 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 indicating which value of the array handle was invalid 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_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_t or S3L_dcomplex_t.

Examples

../examples/s3l/dense_matrix_ops/inner_prod.c

../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_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 7-5.

Note – In these descriptions, A^T and A^H denote A transpose and A Hermitian, respectively.

TABLE 7-5 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	$C = C + A^TB$	real or complex
S3L_mat_mult_t1_noadd	$C = A^TB$	real or complex
S3L_mat_mult_t1_addto	$C = D + A^TB$	real or complex
S3L_mat_mult_h1	$C = C + A^HB$	complex only
S3L_mat_mult_h1_noadd	$C = A^HB$	complex only
S3L_mat_mult_h1_addto	$C = D + A^HB$	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	$C = C + A^TB^T$	real or complex
S3L_mat_mult_t1_t2_noadd	$C = A^TB^T$	real or complex
S3L_mat_mult_t1_t2_addto	$C = D + A^TB^T$	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 7-6.

TABLE 7-6 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` (A^T), the variables conform correctly with the axis lengths for `row_axis` and `col_axis` shown in TABLE 7-7.

TABLE 7-7 Recommended `row_axis` and `col_axis` Values When Matrices Are 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 (B^T), the variables conform correctly with the axis lengths for `row_axis` and `col_axis` shown in TABLE 7-8.

TABLE 7-8 Recommended `row_axis` and `col_axis` Values When Matrix `B` Is Transposed

Variable	`row_axis` **Length**	`col_axis` **Length**
A	q	q
B	r	q
C	p	r
D	p	r

For calls that transpose both A and B ($A^T B^T$), the variables conform correctly with the axis lengths for `row_axis` and `col_axis` shown in TABLE 7-9.

TABLE 7-9 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` are shown below.

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_axisb)
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_nodto(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

- 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 7-5 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.

  Note: 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

These 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 routines 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 indicating which value of the array handle was invalid is 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_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 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_t or S3L_dcomplex_t.

Examples

../examples/s3l/dense_matrix_ops/matmult.c

../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 7-10.

Note – In these descriptions, conj[A] denotes the conjugate of A.

TABLE 7-10 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_noadd	y = v + conj[A]x	complex only

Syntax

The C and Fortran syntax for S3L_mat_vec_mult are shown below.

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

- 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.

  Note: 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. It 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 y. For F77/F90 programs, it must be greater than zero and less than or equal to the rank of x.

Output

These 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 indicating which value of the array handle was invalid is 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_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_t or S3L_dcomplex_t.

Examples

```
../examples/s3l/dense_matrix_ops/mat_vec_mult.c
../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_outer_prod

Description

Sun S3L provides six outer product routines which 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 7-11.

Note – In these descriptions, y^T and y^H denote y transpose and y Hermitian, respectively

---

TABLE 7-11 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_noadd	$A = B + xy^T$	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` are shown below.

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

- 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.

  Note: 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).

  Note: 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 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.

Output

These 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 indicating which value of the array handle was invalid is 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_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_t or S3L_dcomplex_t.
- S3L_ERR_ARG_RANK – Rank of A is less than 2.

Examples

../examples/s3l/dense_matrix_ops/outer_prod.c

../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)

Sparse Matrix Operations

S3L_declare_sparse

Description

S3L_declare_sparse creates an internal S3L array handle that describes a sparse matrix. The sparse matrix may be represented in either the Coordinate format or the Compressed Sparse Row (CSR) 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 three arrays: a, r, and c. Array a stores the nonzero elements of the sparse matrix in any order. r and c are integer arrays that hold the corresponding row and column indices of the sparse matrix, respectively.

The contents of r, c, and a are supplied by the arguments row, col, and val, respectively. row, col, and val are all rank 1 parallel arrays.

The CSR format stores the sparse matrix in arrays ia, ja, and a. As with the Coordinate format, array a stores the nonzero elements of the matrix. ja, an integer array, contains the column indices of the nonzeros as stored in the array a. ia, also an integer array, contains pointers to the beginning of each row in arrays a and ja.

The ia, ja, and a arrays take their contents from the row, col, and val arguments, respectively. As with the Coordinate format, row, col, and val are all rank 1 parallel arrays.

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

S3L_declare_sparse assumes that the row and column indices of the sparse matrix are stored using zero-based indexing when called by C or C++ applications and one-based indexing when called by F77 or F90 applications. See "S3L_read_sparse" on page 160 for a discussion of S3L_read_sparse.

Syntax

The C and Fortran syntax for S3L_declare_sparse are noted next.

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*4               row
    integer*8               col
    integer*8               val
    integer*8               ier
```

Input

- `spfmt` – Indicates the sparse storage format used for representing the sparse matrix. Use S3L_SPARSE_COO to specify the Coordinate format and S3L_SPARSE_CSR for the Compressed Sparse Row 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 the sparse storage format 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`.
- `col` – Integer global array of rank 1 with the same length as array `val`. It contains column indices of the corresponding elements stored in 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, they should be stored row by row, from the first row to the last. The length of val for both S3L_SPARSE_COO and S3L_SPARSE_CSR is, nnz, 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).

Output

This function 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 indicating which value of the array handle was invalid is 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_SPARSE_FORMAT – Invalid storage format. It must be either S3L_SPARSE_COO or S3L_SPARSE_CSR.
- S3L_ERR_ARG_EXTENTS – Invalid m or n. Each must be > 0.
- S3L_ERR_ARG_NULL – Invalid arrays row, col, or val. They must all 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. For S3L_SPARSE_COO, they all must be of the same size. For S3L_SPARSE_CSR, the length of array col must be equal to that of array val and array row must be of size m+1.

Examples

../examples/s3l/sparse/ex_sparse2.c

../examples/s3l/dense_matrix_ops-f/outer_prod.f

Related Functions

S3L_matvec_sparse(3)

S3L_rand_sparse(3)

S3L_read_sparse(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 are shown below.

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 via 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, the following error code may be returned:

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

Examples

`../examples/s3l/sparse/ex_sparse.c`

```
../examples/s3l/sparse/ex_sparse2.c
../examples/s3l/iter/ex_iter.c
../examples/s3l/sparse-f/ex_sparse.f
../examples/s3l/iter-f/ex_iter.f
```

Related Functions

S3L_declare_sparse(3)
S3L_read_sparse(3)
S3L_rand_sparse(3)

S3L_rand_sparse

Description

S3L_rand_sparse creates a random sparse matrix with random sparsity pattern in either the Coordinate format or the Compressed Sparse Row format. Upon successful completion, it returns an S3L array handle in A representing 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 are shown below.

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                      m
    real4                    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

- `spfmt` – Indicates the sparse storage format used for representing the sparse matrix. Use `S3L_SPARSE_COO` to specify the Coordinate format and `S3L_SPARSE_CSR` for the Compressed Sparse Row format.
- `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.

- m – Indicates the total number of rows in the sparse matrix.
- n – Indicates the total number of 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 one of: S3L_integer, S3L_float, S3L_double, S3L_complex, or S3L_dcomplex.
- 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

This function uses the following arguments for output:

- A – On return, 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, this function 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 indicating which value of the array handle was invalid is 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_SPARSE_FORMAT – Invalid storage format. It must be either S3L_SPARSE_COO or S3L_SPARSE_CSR.
- S3L_ERR_ARG_EXTENTS – Invalid m or n. Each must be > 0.
- S3L_ERR_DENSITY – Invalid density value. It must be 0.0 < density <= 1.0.
- S3L_ERR_ARG_OP – Invalid random pattern. It must be one of: S3L_SPARSE_RAND, S3L_SPARSE_DRND, S3L_SPARSE_SRND, or S3L_SPARSE_DSRN.
- S3L_ERR_ARRNOTSQ – Invalid matrix size. When stype does not equal S3L_SPARSE_RAND, m must equal n.

Examples

```
../examples/s3l/iter/ex_iter.c
../examples/s3l/iter-f/ex_iter.f
```

Related Functions

S3L_declare_sparse(3)

S3L_matvec_sparse(3)

S3L_read_sparse(3)

S3L_matvec_sparse

Description

`S3L_matvec_sparse` computes the product of a global general sparse matrix with 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`

Syntax

The C and Fortran syntax for `S3L_matvec_sparse` are shown below.

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

- 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

These functions use 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, this function 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 indicating which value of the array handle was invalid is 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 – 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.

Examples

../examples/s3l/sparse/ex_sparse.c

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

../examples/s3l/iter/ex_iter.c

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

Related Functions

S3L_declare_sparse(3)

S3L_read_sparse(3)

S3L_rand_sparse(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.

These two formats are described below.

S3L_SPARSE_COO – *Coordinate Format*

S3L_SPARSE_COO files consist of three sections, which are illustrated below and described immediately after.

```
% <comments>
%
%
m       n       nnz
i1      j1      a(i1, j1)
i1      j1      a(i1, j1)
i1      j1      a(i1, j1)
i1      j1      a(i1, j1)
    :       :       :
innz    jnnz    a(innz, jnnz)
```

The first section can be used for comments. It consists of one or more lines, each of which begins with the percent "%" character.

The second section consists of a single line containing three integers, shown above as m, n, and nnz. m and n indicate the number of rows and columns of the matrix, respectively, and nnz indicates the total number of nonzero values in the matrix.

The third section lists all nonzero values in the matrix, one value per line. The first two entries on a line are the row and column indices for that value and the third entry is the value itself.

Note – `S3L_read_sparse` assumes that row and column indices are stored using zero-based indexing when called by C or C++ applications and one-based indexing when called by F77 or F90 applications.

This is illustrated by 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
```

This sample matrix could have the `S3L_SPARSE_COO` files consist of three sections, which are below and described immediately after.

```
% 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 – *Compressed Sparse Row Format*

The S3L_SPARSE_CSR files also consist of three sections. The first two sections are the same as in S3L_SPARSE_COO files. The third section stores the sparse matrix in the arrays a, ja, and ia. As with S3L_SPARSE_COO, array a stores the nnz elements of the matrix. `ja`, an integer array, contains the column indices of the nonzeros and `ia`, also an integer array, contains pointers to the beginning of each row in arrays `a` and `ja`.

For example, the same 4x6 sparse matrix used in previous examples could be stored under `S3L_SPARSE_CSR` in the manner shown in (using zero-based indexing).

```
% Example: 4x6 sparse matrix in an S3L_SPARSE_CSR file,
% zero-based indexing:
%
%
4       6       8
0    2    4    5    8
0    3    4    1    2    0    5    3
3.140000   200.400000   -0.600000   27.000000
-0.010000   -0.031000   314.000000   0.080000
```

Syntax

The C and Fortran syntax for `S3L_read_sparse` are shown below.

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                     m
    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

- `spfmt` – Specifies the sparse storage format used for representing the sparse matrix. The supported formats are S3L_SPARSE_COO and S3L_SPARSE_CSR.
- `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` – The type of the sparse array, which must be one of: `S3L_float`, `S3L_double`, `S3L_complex`, or `S3L_dcomplex`.
- `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. The supported format is ASCII.

Output

This function uses the following argument for output:

- A – S3L internal array handle for the global general sparse matrix output.
- ier (Fortran only) – When called from a Fortran program, this function 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 indicating which value of the array handle was invalid is 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_EXTENTS – Invalid m, n, or nnz. These arguments must all be > 0.
- S3L_ERR_SPARSE_FORMAT – Invalid storage format. It must be either S3L_SPARSE_COO or S3L_SPARSE_CRS.
- S3L_ERR_ARG_DTYPE – Invalid data type. It must be S3L_float, S3L_double, S3L_complex, or S3L_dcomplex.
- 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'.
  - 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 ends before expected.

Examples

../examples/s3l/sparse/ex_sparse.c

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

Related Functions

S3L_declare_sparse(3)

S3L_matvec_sparse(3)

S3L_rand_sparse(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        3.14000
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. When S3L_print_sparse is called from a Fortran program, one-based indices are used, as shown below.

```
4       6        8
1       1        3.14000
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
```

Syntax

The C and Fortran syntax for `S3L_print_sparse` are shown below.

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

- 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

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 indicating which value of the array handle was invalid is returned. See Appendix A of this manual for a detailed list of these error codes.

On error, it returns the following code.

- S3L_ERR_ARG_NULL – The value specified for A is invalid; no such S3L sparse matrix has been defined.

Examples

../examples/s3l/sparse/ex_sparse.c

../examples/s3l/sparse/ex_sparse2.c

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

Related Functions

S3L_declare_sparse(3)

S3L_read_sparse(3)

S3L_rand_sparse(3)

Gaussian Elimination for Dense Systems

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 on 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 178 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 are shown below.

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

- 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

This function 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, this function 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

```
../examples/s3l/lu/lu.c
../examples/s3l/lu/ex_lu1.c
../examples/s3l/lu/ex_lu2.c
../examples/s3l/lu-f/lu.f
../examples/s3l/lu-f/ex_lu1.f
```

Related Functions

S3L_lu_deallocate(3)

S3L_lu_invert(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 are shown below.

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

- `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

This function 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, this function 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.
- S3L_ERR_FACTOR_SING – a contains singular factors; its inverse could not be computed.

Examples

../examples/s3l/lu/lu.c

../examples/s3l/lu/ex_lu1.c

../examples/s3l/lu/ex_lu2.c

../examples/s3l/lu-f/lu.f

../examples/s3l/lu-f/ex_lu1.f

Related Functions

S3L_lu_factor(3)

S3L_lu_invert(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-1B
```

followed by back substitution:

```
X = U-1C = U-1(L-1B)
```

To obtain this solution, the S3L_lu_solve routine performs the following steps:

1. **Applies L^{-1} to B.**
2. **Applies U^{-1} to $L^{-1}B$.**

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 are shown below.

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

- `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

This function 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, this function 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, 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; 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 – Unmatched data type between a and b.
- S3L_ERR_ARRNOTSQ – Invalid matrix size; each coefficient matrix must be square.
- S3L_ERR_ARG_SETUP – Invalid setup_id value. It does not match the value returned by S3L_lu_factor.

Examples

../examples/s3l/lu/lu.c

../examples/s3l/lu/ex_lu1.c

../examples/s3l/lu/ex_lu2.c

../examples/s3l/lu-f/lu.f

../examples/s3l/lu-f/ex_lu1.f

Related Functions

S3L_lu_deallocate(3)

S3L_lu_factor(3)

S3L_lu_invert(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 up 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 are shown below.

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

- setup_id – Scalar integer variable. Use the value returned by the corresponding S3L_lu_factor call for this argument.

Output

This function 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_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. It does not match the value returned by `S3L_lu_factor`.

Examples

```
../examples/s3l/lu/lu.c
../examples/s3l/lu/ex_lu1.c
../examples/s3l/lu/ex_lu2.c
../examples/s3l/lu-f/lu.f
../examples/s3l/lu-f/ex_lu1.f
```

Related Functions

```
S3L_lu_factor(3)
S3L_lu_solve(3)
S3L_lu_invert(3)
```

Fast Fourier Transforms

S3L_fft

Description

S3L_fft performs a simple 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 blocksizes 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 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 are shown below.

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

- `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

This function 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, this function 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 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 – The setup_id supplied is not valid.

Examples

../examples/s3l/fft/fft.c

../examples/s3l/fft/ex_fft1.c

../examples/s3l/fft/ex_fft2.c

../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` 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` are shown below.

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

- `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

This function 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, this function 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` – The `setup_id` supplied is not valid.
- `S3L_ERR_FFT_INVIFLAG` – The `iflag` argument is invalid.

Examples

```
../examples/s3l/fft/fft.c
../examples/s3l/fft/ex_fft1.c
../examples/s3l/fft/ex_fft2.c
../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_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 will perform 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 are shown below.

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

- `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

This function 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, this function 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 – The setup_id supplied is not valid.

Examples

../examples/s3l/fft/fft.c

../examples/s3l/fft-f/fft.f

Related Functions

S3L_fft_setup(3)

S3L_fft_free_setup(3)

S3L_fft_detailed(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 non-standard 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 lengths 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 Transforms: 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 (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 represent 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 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` are shown below.

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

- `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

These 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, ese 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 – The setup_id supplied is not valid.

Examples

```
../examples/s3l/rc_fft/rc_fft.c
../examples/s3l/rc_fft-f/rc_fft.f
```

Related Functions

S3L_rc_fft_setup(3)

S3L_rc_fft_free_setup(3)

S3L_fft_setup

Description

A call to S3L_fft_setup is the first step in executing Sun S3L Fast Fourier Transforms. You supply it with the parallel array (a) that is to be transformed. It returns a setup value in setup_id, which you use in subsequent calls to other S3L FFT routines.

When calling S3L_fft_setup, you may supply arbitrary values in a; 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.

You may have more than one setup ID active at a time; that is, you may call the setup routine more than once before deallocating any setup IDs. For this reason, be careful that you 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 phase and transform phases distinct.

When `a` is no longer needed, call `S3L_fft_free_setup` to deallocate the FFT `setup_id`.

Syntax

The C and Fortran syntax for `S3L_fft_setup` are shown below.

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

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

Output

This function uses the following argument for output:

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

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 indicating which value of the array handle was invalid is returned. 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` – `a` is not of type `S3L_complex` or `S3L_double_complex`.
- `S3L_ERR__FFT_EXTSQPROCS` – `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

```
../examples/s3l/fft/fft.c
../examples/s3l/fft/ex_fft1.c
../examples/s3l/fft/ex_fft2.c
../examples/s3l/fft-f/fft.f
../examples/s3l/fft-f/ex_fft1.f
```

Related Functions

```
S3L_fft(3)
S3L_fft_free_setup(3)
S3L_ifft(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 are shown below.

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

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

Output

This function uses the following argument for output:

- ier (Fortran only) – When called from a Fortran program, this function 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 calls.

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

../examples/s3l/rc_fft/rc_fft.c

../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_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 are shown below.

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

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

Output

This function 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_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` – The `setup_id` supplied does not correspond to a valid FFT setup.

Examples

`../examples/s3l/fft/fft.c`

```
../examples/s3l/fft/ex_fft1.c
../examples/s3l/fft/ex_fft2.c
../examples/s3l/fft-f/fft.f
../examples/s3l/fft-f/ex_fft1.f
```

Related Functions

```
S3L_fft_setup(3)
S3L_fft(3)
S3L_ifft(3)
S3L_fft_detailed(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` are shown below.

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

- `setup_id` – Scalar integer variable. Use the value returned by the `S3L_rc_fft_setup` call for this argument.

Output

This function 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_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` – The `setup_id` supplied does not correspond to a valid `S3L_rc_fft_setup`.

Examples

`../examples/s3l/rc_fft/rc_fft.c`

```
../examples/s3l/rc_fft-f/rc_fft.f
```

Related Functions

`S3L_rc_fft_setup(3)`

`S3L_rc_fft(3)`

Structured Solvers

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 bl. The non-zero 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 are shown below.

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

- 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

This function 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, this function 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 indicating which value of the array handle was invalid 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_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:
  - Less than 0 (C/C++) or less than 1 (F77/F90).
  - 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; that is, it is either:
  - 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

../examples/s3l/band/ex_band.c

../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 are shown below.

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

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

Output

This function 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_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 factors value is invalid.

Examples

```
../examples/s3l/band/ex_band.c
../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 axes axis_r and axis_d of a and axes axis_row and axis_col are local (not distributed).

Syntax

The C and Fortran syntax for S3L_gen_band_solve are shown below.

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

- 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.
- b – S3L array handle containing the right-hand side of the matrix equation ax=b.

Output

This function uses the following argument 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, this function 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 indicating which value of the array handle was invalid 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_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 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
  - 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

../examples/s3l/band/ex_band.c

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

Related Functions

S3L_gen_band_factor(3)

S3L_gen_band_free_factors(3)

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 are shown below.

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

- 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

This function 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, this function 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 indicating which value of the array handle was invalid 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_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 (that is, it is 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

../examples/s3l/trid/ex_trid.c

../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 are shown below.

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

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

Output

This function 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_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

```
../examples/s3l/trid/ex_trid.c
../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 via 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` are shown below.

C/C++ Syntax

```
#include <s3l/s3l-c.h>
#include <s3l/s3l_errno-c.h>
int
S3L_gen_trid_solve(D, U, L, factors, B, 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

- `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

This function uses the following argument for output:

- B – On output, B is overwritten with the solution to the tridiagonal system.
- ier (Fortran only) – When called from a Fortran program, this function 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 indicating which value of the array handle was invalid 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_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 (that is, it is 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.
  - row_b is equal to col_b.
- S3L_ERR_ARG_SETUP – The factors value does not correspond to a valid setup.

Examples

```
../examples/s3l/trid/ex_trid.c
../examples/s3l/trid-f/ex_trid.f
```

Related Functions

```
S3L_gen_trid_factor(3)
S3L_gen_trid_free_factors(3)
```

Dense Symmetric Eigenvalue Solver

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` are shown below.

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

- 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

This function 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 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, this function 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 indicating which value of the array handle was invalid 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 – 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

```
../examples/s3l/eigen/eigen.c
../examples/s3l/eigen-f/eigen.f
```

Parallel Random Number Generators

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 are shown below.

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

- `setup_id` – Integer index used to access the state table associated with a particular LFG.
- `seed` – An integer value used to initialize the LMG that initializes the noncritical elements of the LFG's state table.

Output

This function uses the following argument 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, this function returns error status in `ier`.

Error Handling

On success, `S3L_setup_rand_fib` returns `S3L_SUCCESS`.

Examples

../examples/s3l/rand_fib/rand_fib.c

../examples/s3l/rand_fib-f/rand_fib.f

Related Functions

S3L_free_rand_fib(3)

S3L_rand_fib(3)

S3L_free_rand_fib

Description

S3L_free_rand_fib frees the state table associated with a particular Lagged-Fibonacci random number Generator (LFG).

Syntax

The C and Fortran syntax for S3L_free_rand_fib are shown below.

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

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

Output

This function 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_setup_rand_fib` returns `S3L_SUCCESS`.

On error, the following error code may be returned:

- `S3L_ERR_ARG_SETUP` – The `setup_id` value does not correspond to a valid setup.

Examples

`../examples/s3l/rand_fib/rand_fib.c`

`../examples/s3l/rand_fib-f/rand_fib.f`

Related Functions

`S3L_rand_fib(3)`

`S3L_setup_rand_fib(3)`

S3L_rand_fib

Description

S3L_rand_fib initializes a parallel array using 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:

$$x[n] = (x[n-e] + x[n-k]) \bmod 2^m$$

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 are shown below.

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

- 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

This function uses the following argument for output:

- a – On output, a is a randomly initialized array.
- ier (Fortran only) – When called from a Fortran program, this function 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 indicating which value of the array handle was invalid is returned. 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_SETUP – The setup_id value does not correspond to a valid setup.

Examples

../examples/s3l/rand_fib/rand_fib.c

../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 (integer4) 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 are shown below.

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

- `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

This function uses the following argument for output:

- `a` – On output, a is a randomly initialized array.
- `ier` (Fortran only) – When called from a Fortran program, this function 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 indicating which value of the array handle was invalid is returned. 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 – Invalid rank of a.

Examples

../examples/s3l/rand_lcg/rand_lcg.c

../examples/s3l/rand_lcg-f/rand_lcg.f

Related Functions

S3L_free_rand_fib(3)

S3L_setup_rand_fib(3)

Least Squares Solver

S3L_gen_lsq

Description

If m >= n, S3L_gen_lsq finds the least squares solution of 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 of 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 are shown below.

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

- 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 that 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

This function uses the following argument 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, this function 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; that is, it is either:
  - Less than 0 (C/C++) or less than 1 (F77/F90).
  - 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 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

```
../examples/s3l/lsq/ex_lsq.c
../examples/s3l/lsq-f/ex_lsq.f
```

Dense Singular Value Decomposition

S3L_gen_svd

Description

S3L_gen_svd computes the singular value of a parallel array A and, optionally, the right and/or left singular vectors. 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 are shown below.

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

- 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 (see the Output section).

- 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 (see the Output section).
- 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

This function 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, this function 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 indicating which value of the array handle was invalid 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; that is, it is either:
  - Less than 0 (C/C++) or less than 1 (F77/F90).
  - Greater than the rank of the referenced array.
  - axis_r is equal to axis_c.
- S3L_ERR_MATCH_DTYPE – The arrays are not the same data type.
- S3L_ERR_MATCH_RANK – The arrays are not the same rank.
- 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

../examples/s3l/svd/ex_svd.c

../examples/s3l/svd-f/ex_svd.f

Iterative Solver

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`

The global rank 1 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,” “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 them 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 7-12

TABLE 7-12 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 7-13.

TABLE 7-13 iparm[S3L_iter_pc] Options

Option	Description
S3L_none	No preconditioning will be done (default).
S3L_jacobi	Point Jacobi preconditioner will be used.
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.

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 to 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` are shown below.

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

- `A` – S3L internal array handle for the global general sparse matrix. It is produced by a prior call to one of the following sparse routines:
  - `S3L_declare_sparse`
  - `S3L_read_sparse`
  - `S3L_rand_sparse`
- `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` contains the initial guess for the solution to the linear system. Upon successful completion, `x` contains the converged solution (see the Output section).
- `iparm` – Integer local array of rank 1 and length `s3l_iter_iparm_size`, where:

- 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.

- rparm – Specifies options for computing all or part of the matrix U.

Output

This function 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, this function 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 indicating which value of the array handle was invalid is returned. 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_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.

Computational Errors

- S3L_ERR_ILU_ZRPVT – Encountered a zero pivot in the course of ILU preconditioning.
- S3L_ERR_JACOBI_ZRDIAG – Encountered a zero diagonal in the course of 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

../examples/s3l/iter/ex_iter.c

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

Related Functions

S3L_declare_sparse(3)

S3L_read_sparse(3)

S3L_rand_sparse(3)

Autocorrelation

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 are shown below.

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

- `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

This function uses the following arguments for output:

- `setup` – Integer value retuned 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 indicating which value of the array handle was invalid is returned. 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 one of:
  - S3L_float
  - S3L_double
  - S3L_complex
  - S3L_double_complex
- S3L_ERR_MATCH_DTYPE – The array arguments are not all of the same 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.
- S3L_ERR_ARG_EXTENTS – The extents of c are less than the extents of A.

Examples

../examples/s3l/acorr/ex_acorr.c

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

Related Functions

S3L_acorr(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` are shown below.

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

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

Output

This function 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_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

```
../examples/s3l/acorr/ex_acorr.c
../examples/s3l/acorr-f/ex_acorr.f
```

Related Functions

```
S3L_acorr(3)
S3L_acorr_setup(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 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 wrap-around 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 Effects

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 via S3L_fft, or S3L_rc_fft. See "S3L_fft" on page 180 and "S3L_rc_fft and S3L_cr_fft" on page 188 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 are shown below.

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

- 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

This function uses the following arguments for 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 indicating which value of the array handle was invalid is returned. 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 one of:
  - `S3L_float`
  - `S3L_double`
  - `S3L_complex`
  - `S3L_double_complex`
- `S3L_ERR_MATCH_DTYPE` – The array arguments are not of the same data type.
- `S3L_ERR_MATCH_RANK` – The array arguments are not 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 smaller than 2*ma-1 (1D case) or 2*ma-1 x 2*na-1 (2D case).

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

../examples/s3l/acorr/ex_acorr.c
../examples/s3l/acorr-f/ex_acorr.f

Related Functions

S3L_acorr_setup(3)
S3L_acorr_free_setup(3)

Convolution / Deconvolution

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 are shown below.

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
```

where `<type>` is `real*4` or `real*8` for both C/C++ and F77/F90.

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

This function uses the following arguments for output:

- `setup_id` – Integer value retuned 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 indicating which value of the array handle was invalid is returned. 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_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 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

../examples/s3l/conv/ex_conv.c

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

Related Functions

S3L_conv(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` are shown below.

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
```

where `<type>` is `real*4` or `real*8` for both C/C++ and F77/F90.

Input

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

Output

This function 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 value.

Examples

```
../examples/s3l/conv/ex_conv.c
../examples/s3l/conv-f/ex_conv.f
```

Related Functions

```
S3L_conv(3)
S3L_conv_setup(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 Effects

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 via `S3L_fft`, or `S3L_rc_fft`. See "`S3L_fft`" on page 180 and "`S3L_rc_fft` and `S3L_cr_fft`" on page 188 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` are shown below.

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
```

where `<type>` is `real*4` or `real*8` for both C/C++ and F77/F90.

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

This function 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 indicating which value of the array handle was invalid is returned. 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_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

```
../examples/s3l/conv/ex_conv.c
../examples/s3l/conv-f/ex_conv.f
```

Related Functions

S3L_conv_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 are shown below.

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
```

where <type> is real*4 or real*8 for both C/C++ and F77/F90.

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

This function uses the following arguments for output:

- setup_id – Integer value retuned by this function. Use this value for the setup_id argument in subsequent calls to S3_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 indicating which value of the array handle was invalid is returned. 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_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 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

```
../examples/s3l/deconv/ex_deconv.c
../examples/s3l/deconv-f/ex_deconv.f
```

Related Functions

S3L_deconv(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 are shown below.

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
```

where <type> is real*4 or real*8 for both C/C++ and F77/F90.

Input

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

Output

This function 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 value.

Examples

../examples/s3l/deconv/ex_deconv.c

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

Related Functions

S3L_deconv(3)

S3L_deconv_setup(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 180 and "S3L_rc_fft and S3L_cr_fft" on page 188 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 and by n. In all other cases, the result is scaled by the product of the extents of c.

Side Effects

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 are shown below.

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

- `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

This function 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 indicating which value of the array handle was invalid is returned. 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_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 180 and "S3L_rc_fft and S3L_cr_fft" on page 188 for more details.

Examples

../examples/s3l/deconv/ex_deconv.c

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

Related Functions

S3L_deconv_setup(3)

S3L_deconv_free_setup(3)

Multidimensional Sort and Grade

S3L_grade_down, S3L_grade_up, S3L_grade_down_detailed, S3L_grade_up_detailed

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 – S3L_grade_detailed_down
  - The j-th largest value along the specified axis – 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 are shown below.

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

- 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

These 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, this function returns 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 indicating which value of the array handle was invalid is returned. 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

```
../examples/s3l/grade/ex_grade.c
../examples/s3l/grade-f/ex_grade.f
```

Related Functions

```
S3L_sort(3)
S3L_sort_detailed_up(3)
S3L_sort_detailed_down(3)
```

S3L_sort, S3L_sort_up, S3L_sort_down, S3L_sort_detailed_up, 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 are shown below.

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
```

where `<type>` is `real*4` or `real*8` for both C/C++ and F77/F90.

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 zero (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 functions use the following arguments for output:

- `A` – On output, A contains the sorted array.

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

Error Handling

On success, these 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 indicating which value of the array handle was invalid is returned. 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_DTYPE` – The type of the array is invalid. It must be one of: `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

```
../examples/s3l/sort/sort1.c
../examples/s3l/sort/ex_sort2.c
../examples/s3l/sort-f/sort1.f
```

Related Functions

```
S3L_grade_up(3)
S3L_grade_detailed_down(3)
S3L_grade_detailed_up(3)
```

Parallel Transpose

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 are shown below.

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
```

where `<type>` is `real*4` or `real*8` for both C/C++ and F77/F90.

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

These functions use 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, this function 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 indicating which value of the array handle was invalid is returned. 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_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

`../examples/s3l/transpose/transp.c`

`../examples/s3l/transpose/ex_trans1.c`

`../examples/s3l/transpose-f/transp.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 following error codes 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

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.