Sun Performance Library User's Guide

Sun WorkShop 6 FORTRAN 77, Fortran 95, and C

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

Part No. 806-3566-10
May 2000, Revision A

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

Important Note on New Product Names

As part of Sun's new developer product strategy, we have changed the names of our development tools from Sun WorkShop™ to Forte™ Developer products. The products, as you can see, are the same high-quality products you have come to expect from Sun; the only thing that has changed is the name.

We believe that the Forte™ name blends the traditional quality and focus of Sun's core programming tools with the multi-platform, business application deployment focus of the Forte tools, such as Forte Fusion™ and Forte™ for Java™. The new Forte organization delivers a complete array of tools for end-to-end application development and deployment.

For users of the Sun WorkShop tools, the following is a simple mapping of the old product names in WorkShop 5.0 to the new names in Forte Developer 6.

Old Product Name	New Product Name
Sun Visual WorkShop™ C++	Forte™ C++ Enterprise Edition 6
Sun Visual WorkShop™ C++ Personal Edition	Forte™ C++ Personal Edition 6
Sun Performance WorkShop™ Fortran	Forte™ for High Performance Computing 6
Sun Performance WorkShop™ Fortran Personal Edition	Forte™ Fortran Desktop Edition 6
Sun WorkShop Professional™ C	Forte™ C 6
Sun WorkShop™ University Edition	Forte™ Developer University Edition 6

In addition to the name changes, there have been major changes to two of the products.

- Forte for High Performance Computing contains all the tools formerly found in Sun Performance WorkShop Fortran and now includes the C++ compiler, so High Performance Computing users need to purchase only one product for all their development needs.
- Forte Fortran Desktop Edition is identical to the former Sun Performance WorkShop Personal Edition, except that the Fortran compilers in that product no longer support the creation of automatically parallelized or explicit, directive-based parallel code. This capability is still supported in the Fortran compilers in Forte for High Performance Computing.

We appreciate your continued use of our development products and hope that we can continue to fulfill your needs into the future.

Contents

Tables

Preface

This book describes how to use the Sun™ specific extensions and features included with the Sun Performance Library subroutines that are supported by the Sun WorkShop™ 6 FORTRAN 77, Fortran 95, and C compilers.

Who Should Use This Book

This is a reference manual intended for programmers who have a working knowledge of the Fortran or C language and some understanding of the base LAPACK, BLAS, FFTPACK, VFFTPACK, and LINPACK libraries available from Netlib (`http://www.netlib.org`).

What Is in This Book

This book is organized into the following chapters and appendixes:

Chapter 1, "Introduction," describes the benefits of using the Sun Performance Library and the features of the Sun Performance Library.

Chapter 2, "Using Sun Performance Library," describes how to use the `f77`, `f95`, and C interfaces provided with the Sun Performance Library.

Chapter 3, "SPARC Optimization and Parallel Processing," shows how to use compiler and linking options to maximize library performance for specific SPARC™ instruction set architectures and different parallel processing modes.

Chapter 4, “Working With Matrices,” includes information on matrix storage schemes, matrix types, and sparse matrices.

Appendix A, “Sun Performance Library Routines,” lists the Sun Performance Library routines organized according to name, routine, and library.

What Is Not in This Book

This book does not repeat information included in existing LAPACK and LINPACK books or sources on Netlib. Refer to the section “Related Documents and Web Sites” on page 4 for a list of sources that contain reference material for the base routines upon which Sun Performance Library is based.

Typographic Conventions

TABLE P-1 shows the typographic conventions that are used in Sun WorkShop documentation.

TABLE P-1 Typographic Conventions

Typeface	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	`%` **`su`**
		`Password:`

AaBbCc123	Book titles, new words or terms, words to be emphasized	Read Chapter 6 in the *User’s Guide*. These are called *class* options. You *must* be superuser to do this.
AaBbCc123	Command-line placeholder text; replace with a real name or value	To delete a file, type `rm` *filename*.

Shell Prompts

TABLE P-2 shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P-2 Shell Prompts

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

Access to Sun WorkShop Development Tools

Because Sun WorkShop product components and man pages do not install into the standard `/usr/bin/` and `/usr/share/man` directories, you must change your `PATH` and `MANPATH` environment variables to enable access to Sun WorkShop compilers and tools.

To determine if you need to set your `PATH` environment variable:

1. **Display the current value of the `PATH` variable by typing:**

```
% echo $PATH
```

2. **Review the output for a string of paths containing `/opt/SUNWspro/bin/`.**

   If you find the paths, your `PATH` variable is already set to access Sun WorkShop development tools. If you do not find the paths, set your `PATH` environment variable by following the instructions in this section.

To determine if you need to set your `MANPATH` environment variable:

1. **Request the `workshop` man page by typing:**

```
% man workshop
```

2. **Review the output, if any.**

   If the `workshop`(1) man page cannot be found or if the man page displayed is not for the current version of the software installed, follow the instructions in this section for setting your `MANPATH` environment variable.

Note – The information in this section assumes that your Sun WorkShop 6 products were installed in the `/opt` directory. Contact your system administrator if your Sun WorkShop software is not installed in `/opt`.

The `PATH` and `MANPATH` variables should be set in your home `.cshrc` file if you are using the C shell or in your home `.profile` file if you are using the Bourne or Korn shells:

- To use Sun WorkShop commands, add the following to your `PATH` variable:

  `/opt/SUNWspro/bin`

- To access Sun WorkShop man pages with the `man` command, add the following to your `MANPATH` variable:

  `/opt/SUNWspro/man`

For more information about the `PATH` variable, see the `csh`(1), `sh`(1), and `ksh`(1) man pages. For more information about the `MANPATH` variable, see the `man`(1) man page. For more information about setting your `PATH` and `MANPATH` variables to access this release, see the *Sun WorkShop 6 Installation Guide* or your system administrator.

Related Documents and Web Sites

A number of books and web sites provide reference information on the routines in the base libraries (LAPACK, LINPACK, BLAS, and so on) upon which the Sun Performance Workshop is based. Sun Performance Library includes extensions to the base libraries that are not described in the books from the Society for Industrial and Applied Mathematics (SIAM) or the online Netlib documents.

LAPACK and LINPACK Books

The following books augment this manual and provide essential information:

- *LAPACK Users' Guide*. 3rd ed., Anderson E. and others. SIAM, 1999.
- *LINPACK User's Guide*. Dongarra J. J. and others. SIAM, 1979.

The *LAPACK Users' Guide*, 3rd ed. is the official reference for the base LAPACK version 3.0 routines. An online version of the *LAPACK 3.0 Users' Guide* is available at `http://www.netlib.org/lapack/lug/`, and the printed version is available from SIAM.

Sun Performance Library routines contain performance enhancements, extensions, and features not described in the *LAPACK Users' Guide*. However, because Sun Performance Library maintains compatibility with the base LAPACK routines, the *LAPACK Users' Guide* can be used as a reference for the types of LAPACK routines and the FORTRAN 77 interfaces.

Sparse BLAS and Sparse Solver Books and Papers

The following books and papers provide additional information for the sparse BLAS and sparse solver routines.

- Dodson, D.S, R.G. Grimes, and J.G. Lewis. "Sparse Extensions to the Fortran Basic Linear Algebra Subprograms." ACM Transactions on Mathematical Software, June 1991, Vol 17, No. 2.
- A. George and J. W-H. Liu. "Computer Solution of Large Sparse Positive Definite Systems." Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1981.
- E. Ng and B. W. Peyton. "Block Sparse Cholesky Algorithms on Advanced Uniprocessor Computers." SIAM M. Sci Comput., 14:1034-1056, 1993.
- Ian S. Duff, Roger G. Grimes and John G. Lewis, "User's Guide for the Harwell-Boeing Sparse Matrix Collection (Release I)," Technical Report TR/PA/92/86, CERFACS, Lyon, France, October 1992.

Online Resources

Online information describing the performance library routines that form the basis of the Sun Performance Library can be found at:

LAPACK version 3.0	`http://www.netlib.org/lapack/`
BLAS, levels 1 through 3	`http://www.netlib.org/blas/`
FFTPACK version 4	`http://www.netlib.org/fftpack/`
VFFTPACK version 2.1	`http://www.netlib.org/vfftpack/`
Sparse BLAS	`http://www.netlib.org/sparse-blas/index.html`
NIST (National Institute of Standards and Technology) Fortran Sparse BLAS	`http://math.nist.gov/spblas/`
LINPACK	`http://www.netlib.org/linpack/`

Related Sun WorkShop 6 Documentation

You can access documentation related to the subject matter of this book in the following ways:

- **Through the Internet at the `docs.sun.com`^{sm} Web site.** You can search for a specific book title or you can browse by subject, document collection, or product at the following Web site:

  `http://docs.sun.com`

- **Through the installed Sun WorkShop products on your local system or network.** Sun WorkShop 6 HTML documents (manuals, online help, man pages, component readme files, and release notes) are available with your installed Sun WorkShop 6 products. To access the HTML documentation, do one of the following:
  - In any Sun WorkShop or Sun WorkShop™ TeamWare window, choose Help ➤ About Documentation.
  - In your Netscape™ Communicator 4.0 or compatible version browser, open the following file:

    `/opt/SUNWspro/docs/index.html`

(Contact your system administrator if your Sun WorkShop software is not installed in the `/opt` directory.) Your browser displays an index of Sun WorkShop 6 HTML documents. To open a document in the index, click the document's title.

TABLE P-3 lists related Sun WorkShop 6 manuals by document collection.

TABLE P-3 Related Sun WorkShop 6 Documentation by Document Collection

Document Collection	Document Title	Description
Forte™ Developer 6 / Sun WorkShop 6 Release Documents	*About Sun WorkShop 6 Documentation*	Describes the documentation available with this Sun WorkShop release and how to access it.
	What's New in Sun WorkShop 6	Provides information about the new features in the current and previous release of Sun WorkShop.
	Sun WorkShop 6 Release Notes	Contains installation details and other information that was not available until immediately before the final release of Sun WorkShop 6. This document complements the information that is available in the component readme files.
Forte Developer 6 / Sun WorkShop 6	*Analyzing Program Performance With Sun WorkShop 6*	Explains how to use the new Sampling Collector and Sampling Analyzer (with examples and a discussion of advanced profiling topics) and includes information about the command-line analysis tool `er_print`, the LoopTool and LoopReport utilities, and UNIX profiling tools `prof`, `gprof`, and `tcov`.
	Debugging a Program With dbx	Provides information on using dbx commands to debug a program with references to how the same debugging operations can be performed using the Sun WorkShop Debugging window.

TABLE P-3 Related Sun WorkShop 6 Documentation by Document Collection *(Continued)*

Document Collection	Document Title	Description
	Introduction to Sun WorkShop	Acquaints you with the basic program development features of the Sun WorkShop integrated programming environment.
Forte™ C 6 / Sun WorkShop 6 Compilers C	*C User's Guide*	Describes the C compiler options, Sun-specific capabilities such as pragmas, the `lint` tool, parallelization, migration to a 64-bit operating system, and ANSI/ISO-compliant C.
Forte™ C++ 6 / Sun WorkShop 6 Compilers C++	*C++ Library Reference*	Describes the C++ libraries, including C++ Standard Library, Tools.h++ class library, Sun WorkShop Memory Monitor, Iostream, and Complex.
	C++ Migration Guide	Provides guidance on migrating code to this version of the Sun WorkShop C++ compiler.
	C++ Programming Guide	Explains how to use the new features to write more efficient programs and covers templates, exception handling, runtime type identification, cast operations, performance, and multithreaded programs.
	C++ User's Guide	Provides information on command-line options and how to use the compiler.
	Sun WorkShop Memory Monitor User's Manual	Describes how the Sun WorkShop Memory Monitor solves the problems of memory management in C and C++. This manual is only available through your installed product (see `/opt/SUNWspro/docs/index.html`) and not at the `docs.sun.com` Web site.

TABLE P-3 Related Sun WorkShop 6 Documentation by Document Collection *(Continued)*

Document Collection	Document Title	Description
Forte™ for High Performance Computing 6 / Sun WorkShop 6 Compilers Fortran 77/95	*Fortran Library Reference*	Provides details about the library routines supplied with the Fortran compiler.
	Fortran Programming Guide	Discusses issues relating to input/output, libraries, program analysis, debugging, and performance.
	Fortran User's Guide	Provides information on command-line options and how to use the compilers.
	FORTRAN 77 Language Reference	Provides a complete language reference.
	Interval Arithmetic Programming Reference	Describes the intrinsic `INTERVAL` data type supported by the Fortran 95 compiler.
Forte™ TeamWare 6 / Sun WorkShop TeamWare 6	*Sun WorkShop TeamWare 6 User's Guide*	Describes how to use the Sun WorkShop TeamWare code management tools.
Forte Developer 6 / Sun WorkShop Visual 6	*Sun WorkShop Visual User's Guide*	Describes how to use Visual to create C++ and Java™ graphical user interfaces.
Forte™ / Sun Performance Library 6	*Sun Performance Library Reference*	Discusses the optimized library of subroutines and functions used to perform computational linear algebra and fast Fourier transforms.
	Sun Performance Library User's Guide	Describes how to use the Sun-specific features of the Sun Performance Library, which is a collection of subroutines and functions used to solve linear algebra problems.
Numerical Computation Guide	*Numerical Computation Guide*	Describes issues regarding the numerical accuracy of floating-point computations.
Standard Library 2	*Standard C++ Class Library Reference*	Provides details on the Standard C++ Library.

TABLE P-3 Related Sun WorkShop 6 Documentation by Document Collection *(Continued)*

Document Collection	Document Title	Description
	Standard C++ Library User's Guide	Describes how to use the Standard C++ Library.
Tools.h++ 7	*Tools.h++ Class Library Reference*	Provides details on the `Tools.h++` class library.
	Tools.h++ User's Guide	Discusses use of the C++ classes for enhancing the efficiency of your programs.

TABLE P-4 describes related Solaris documentation available through the `docs.sun.com` Web site.

TABLE P-4 Related Solaris Documentation

Document Collection	Document Title	Description
Solaris Software Developer	*Linker and Libraries Guide*	Describes the operations of the Solaris link-editor and runtime linker and the objects on which they operate.
	Programming Utilities Guide	Provides information for developers about the special built-in programming tools that are available in the Solaris operating environment.

CHAPTER 1

Introduction

Sun Performance Library is a set of optimized, high-speed mathematical subroutines for solving linear algebra and other numerically intensive problems. Sun Performance Library is based on a collection of public domain applications available from Netlib at `http://www.netlib.org`. Sun has enhanced these public domain applications and bundled them as the Sun Performance Library.

The *Sun Performance Library User's Guide* explains the Sun-specific enhancements to the base applications available from Netlib. Reference material describing the base routines is available from Netlib and the Society for Industrial and Applied Mathematics (SIAM).

Libraries Included With Sun Performance Library

Sun Performance Library contains enhanced versions of the following standard libraries:

- LAPACK version 3.0 – For solving linear algebra problems.
- BLAS1 (Basic Linear Algebra Subprograms) – For performing vector-vector operations.
- BLAS2 – For performing matrix-vector operations.
- BLAS3 – For performing matrix-matrix operations.
- FFTPACK version 4 – For performing the fast Fourier transform.
- VFFTPACK version 2.1 – A vectorized version of FFTPACK for performing the fast Fourier transform.
- LINPACK – For solving linear algebra problems in legacy applications containing routines that have not been upgraded to LAPACK 3.0.

Note – LAPACK version 3.0 supersedes LINPACK, EISPACK, and all previous versions of LAPACK. Use LAPACK for new development and LINPACK to support legacy applications.

Sun Performance Library is available in both static and dynamic library versions optimized for the V8, V8+, and V9 architectures. Sun Performance Library supports static and shared libraries on Solaris 2.6, Solaris 7, and Solaris 8 and adds support for multiple processors.

Sun Performance Library LAPACK routines have been compiled with a Fortran 95 compiler and remain compatible with the Netlib LAPACK version 3.0 library. The Sun Performance Library versions of these routines perform the same operations as the Fortran callable routines and have the same interface as the standard Netlib versions.

Netlib

Netlib is an online repository of mathematical software, papers, and databases maintained by AT&T Bell Laboratories, the University of Tennessee, Oak Ridge National Laboratory, and professionals from around the world.

Netlib provides many libraries, in addition to the seven libraries used in Sun Performance Library. While some of these libraries can appear similar to libraries used with Sun Performance Library, they can be different from, and incompatible with Sun Performance Library.

Using routines from other libraries can produce compatibility problems, not only with Sun Performance Library routines, but also with the base Netlib LAPACK routines. When using non-Sun Performance Library routines, refer to the documentation provided with that library.

For example, Netlib provides a CLAPACK library, but the CLAPACK interfaces differ from the C interfaces included with Sun Performance Library. A LAPACK 90 library package is also available on Netlib. The LAPACK 90 library contains interfaces that differ from the Sun Performance Library Fortran 95 interfaces and the Netlib LAPACK version 3.0 interfaces. If using LAPACK 90, refer to the documentation provided with that library.

For the base libraries supported by Sun Performance Library, Netlib provides detailed information that can supplement this user's guide. The *LAPACK 3.0 Users' Guide* describes LAPACK algorithms and how to use the routines. However, these documents do not describe the Sun-specific extensions made to the base routines.

Sun Performance Library Features

Sun Performance Library provides the following optimizations and extensions to the base Netlib libraries:

- Extensions that support Fortran 95 and C language interfaces
- Fortran 95 language features, including type independence, compile time checking, and optional arguments.
- Consistent API across the different libraries in Sun Performance Library.
- Compatibility with LAPACK 1.x, LAPACK 2.0, and LAPACK 3.0 libraries
- Increased performance, and in some cases, greater accuracy
- Optimizations for specific SPARC instruction set architectures
- Support for 64-bit code on UltraSPARC™
- Support for parallel processing compiler options
- Support for multiple processor hardware options

Mathematical Routines

The Sun Performance Library routines are used to solve the following types of linear algebra and numerical problems:

- *Elementary vector and matrix operations* – Vector and matrix products; plane rotations; 1, 2-, and infinity-norms; rank-1, 2, k, and 2k updates
- *Linear systems* – Solve full-rank systems, compute error bounds, solve Sylvester equations, refine a computed solution, equilibrate a coefficient matrix
- *Least squares* – Full-rank, generalized linear regression, rank-deficient, linear equality constrained
- *Eigenproblems* – Eigenvalues, generalized eigenvalues, eigenvectors, generalized eigenvectors, Schur vectors, generalized Schur vectors
- *Matrix factorizations or decompositions* – SVD, generalized SVD, QL and LQ, QR and RQ, Cholesky, LU, Schur, LDL^T and UDU^T
- *Support operations* – Condition number, in-place or out-of-place transpose, inverse, determinant, inertia
- *Sparse matrices* – Solve symmetric, structurally symmetric, and unsymmetric coefficient matrices using direct methods and a choice of fill-reducing ordering algorithms, including user specified orderings

- Convolution and correlation in one and two dimensions
- Fast Fourier transforms, Fourier synthesis, cosine and quarter-wave cosine transforms, cosine and quarter-wave sine transforms
- Complex vector FFTs and FFTs in two and three dimensions

Compatibility With Previous LAPACK Versions

The Sun Performance Library routines that are based on LAPACK support the expanded capabilities and improved algorithms in LAPACK 3.0, but are completely compatible with both LAPACK l.x and LAPACK 2.0. Maintaining compatibility with previous LAPACK versions:

- Reduces linking errors due to changes in subroutine names or argument lists.
- Ensures results are consistent with results generated with previous LAPACK versions.
- Minimizes programs terminating due to differences between argument lists.

With Sun Performance Library, users can safely use programs intended for the original LAPACK 1.x or 2.0. At the same time, developers can gradually upgrade the portions of their applications that use LAPACK 3.0.

Getting Started With Sun Performance Library

This section shows the most basic compiler options used to compile an application using the Sun Performance Library.

To use the Sun Performance Library, type one of the following commands:

```
my_system% f95 -dalign my_file.f -xlic_lib=sunperf
```

or

```
my_system% cc -dalign my_file.c -xlic_lib=sunperf
```

The routines in the Sun Performance Library are compiled with `-dalign`. For best performance, compiling applications with `-dalign` is suggested. If there are cases when `-dalign` cannot be used, enable trap 6, which allows misaligned data, as described in the following section.

Additional compiler options exist that optimize application performance for:

- Specific SPARC instruction set architectures, as described in "Compiling for SPARC Platforms" on page 30.
- Shared or dedicated parallel processing models, as described in "Optimizing for Parallel Processing" on page 33.

Enabling Trap 6

If an application cannot be compiled using `-dalign`, enable trap 6 to provide a handler for misaligned data. To enable trap 6 on SPARC, do the following:

1. **Place this assembly code in a file called `trap6_handler.s`:**

```
    .global trap6_handler_
    .text
    .align 4
trap6_handler_:
    retl
    ta    6
```

2. **Assemble `trap6_handler.s`:**

```
my_system% fbe trap6_handler.s
```

The first parallelizable subroutine invoked from Sun Performance Library will call a routine named `trap6_handler_`. If a `trap6_handler_` is not specified, Sun Performance Library will call a default handler that does nothing. Not supplying a handler for any misaligned data will cause a trap that will be fatal. (`fbe` (1) is the Sun WorkShop assembler for SPARC platforms.)

3. **Include `trap6_handler.o` on the command line:**

```
my_system% f95 any.f trap6_handler.o -xlic_lib=sunperf
```

CHAPTER **2**

Using Sun Performance Library

This chapter describes using the Sun Performance Library to improve the execution speed of applications written in either FORTRAN 77, Fortran 95, or C. Although some modifications to applications might be required to gain peak performance, many applications can benefit significantly from using Sun Performance Library without making source code changes or recompiling.

Improving Application Performance

Use Sun Performance Library in the following ways to improve the speed of user code without making any code changes:

- Use Sun Performance Library routines instead of the base Netlib routines. See the next section “Replacing Routines With Sun Performance Library Routines.”
- Use Sun Performance Library to speed up the other libraries, if an application already uses libraries in addition to those in the Sun Performance Library. See “Improving Performance of Other Libraries” on page 18.
- Use tools that automatically modify an application to use Sun Performance Library. See “Using Tools to Restructure Code” on page 18.

Replacing Routines With Sun Performance Library Routines

Many applications are built using one or more of the base Netlib libraries supported by the Sun Performance Library. Third-party vendors can also use BLAS and LAPACK as building blocks in their applications. Because Sun Performance Library maintains the same interfaces and functionality of these libraries, base Netlib routines can be replaced with Sun Performance Library routines.

Sun Performance Library can be included in a user's development environment to improve application performance on single processor and multiprocessor (MP) platforms. Sun Performance Library routines can be faster than the corresponding Netlib routines or routines provided by other vendors that perform similar functions. The serial speed of many Sun Performance Library routines has been increased, and many routines have been parallelized that might be serial in other products.

Improving Performance of Other Libraries

Users of other mathematical libraries can replace the BLAS in their library with the BLAS in Sun Performance Library, while leaving other routines unchanged. This is helpful when an application has a dependency on proprietary interfaces in another library that prevent the other library from being completely replaced. Many commercial math libraries are built around a core of generic BLAS and LAPACK routines, so replacing those generic routines with the highly optimized BLAS and LAPACK routines in Sun Performance Library can give speed improvements on both serial and MP platforms. Because replacing the core routines does not require any code changes, the proprietary library features can still be used.

Even libraries that already have fast core routines may get additional speedups by using Sun Performance Library. For example, if another vendor's core routines are based on BLAS, these routines can be replaced with Sun Performance Library routines, which have SPARC specific optimizations. Many Sun Performance Library routines have also been parallelized.

Using Tools to Restructure Code

In some cases, other libraries may not directly use the routines in the Sun Performance Library; however, there might be conversion aids available. For example, EISPACK users can refer to a conversion chart in the *LAPACK Users' Manual* that shows how to convert EISPACK calls to LAPACK calls.

Several vendors market automatic code restructuring tools that replace existing code with Sun Performance Library code. For example, a source- to- source conversion tool can replace existing BLAS code structures with calls to the BLAS in Sun Performance Library. These tools can also recognize many user written matrix multiplications and replace them with calls to the matrix multiplication subroutine in Sun Performance Library.

Fortran `f77/f95` Interfaces

The Sun Performance Library routines can be called from within a FORTRAN 77, Fortran 95, or a C program. However, C programs must still use the FORTRAN 77 calling sequence.

Sun Performance Library `f77/f95` interfaces use the following conventions:

- All arguments are passed by reference.
- The number of arguments to a routine is fixed.
- Types of arguments must match.
- Arrays are stored columnwise.
- Indices are based at one, in keeping with standard Fortran practice.

When calling Sun Performance Library routines:

- Do not prototype the subroutines with the Fortran 95 `INTERFACE` statement. Use the `USE SUNPERF` statement instead.
- Do not use `-ext_names=plain` to compile routines that call routines from Sun Performance Library.

Using Fortran 95 Features

This release supports Fortran 95 language features. To use the Sun Performance Library Fortran 95 modules and definitions, including the `USE SUNPERF` statement in the program. The `USE SUNPERF` statement enables the following features:

- *Type Independence* – In the FORTRAN 77 routines, the type must be specified as part of the name. `DGEMM` is a double precision matrix multiply and `SGEMM` is single precision. With the Fortran 95 interfaces, when calling `GEMM`, Fortran will infer the type from the arguments that are passed. Passing single-precision arguments to `GEMM` gets results that are equivalent to specifying `SGEMM`, passing double-precision arguments gets results that are equivalent to `DGEMM`, and so on. For example, `CALL DSCAL(20,5.26D0,X,1)` could be changed to `CALL SCAL(20, 5.26D0, X, 1)`.
- *Compile-Time Checking* – In FORTRAN 77, it is generally impossible for the compiler to determine what arguments should be passed to a particular routine. In Fortran 95, the `USE SUNPERF` statement allows the compiler to determine the

number, type, size, and shape of each argument to each Sun Performance Library routine. It can check the calls against the expected value and display errors during compilation.

- *Optional `f95` Interfaces* – In FORTRAN 77, all arguments must be specified in the order determined by the interface for all routines. All interfaces will support `f95` style `OPTIONAL` attributes on arguments that are not required. To determine the optional arguments for a routine, refer to the man pages. Optional arguments are enclosed in square brackets [ ].

For example, the `SAXPY` routine is defined as follows in the man page:

```
SUBROUTINE SAXPY([N], ALPHA, X, [INCX], Y, [INCY])
REAL ALPHA
INTEGER INCX, INCY, N
REAL X(*), Y(*)
```

Note that the arguments `N`, `INCX`, and `INCY` are optional.

Suppose the user tries to call the `SAXPY` routine with the following arguments:

```
USE SUNPERF
COMPLEX ALPHA
REAL    X(100), Y(100), XA(100,100), RALPHA
INTEGER INCX, INCY
```

If mismatches in the type, shape, or number of arguments occur, the compiler would issue the following error message:

```
ERROR: No specific match can be found for the generic subprogram
call "AXPY".
```

Using the arguments defined above, the following examples show incorrect calls to the `SAXPY` routine due type, shape, or number mismatches.

- *Incorrect type of the arguments*–If `SAXPY` is called as follows:

  ```
      CALL AXPY(100, ALPHA, X, INCX, Y, INCY)
  ```

  A compiler error occurs because the variable `ALPHA` is type `COMPLEX`, but the interface describes it as being type `REAL`.

- *Incorrect shape of the arguments*– If SAXPY is called as follows:

```
    CALL AXPY(N, RALPHA, XA, INCX, Y, INCY)
```

  A compiler error occurs because the XA argument is two dimensional, but the interface is expecting a one-dimensional argument.

- *Incorrect number of arguments*– If SAXPY is called as follows:

```
    CALL AXPY(RALPHA, X, INCX, Y)
```

- A compiler error occurs because the compiler cannot find a routine in the AXPY interface group that takes four parameters of the form

```
    AXPY(REAL, REAL 1-D ARRAY, INTEGER, REAL 1-D ARRAY)
```

  In the last example, the f95 keyword parameter passing capability can allow a user to make essentially the same call using that capability.

```
    CALL AXPY(ALPHA=RALPHA,X=X,INCX=INCX,Y=Y)
```

  This is a valid call to the AXPY interface. It is necessary to use keyword parameter passing on any parameter that appears in the list *after* the first OPTIONAL parameter is omitted.

The following calls to the AXPY interface are valid.

```
    CALL AXPY(N,RALPHA,X,Y=Y,INCY=INCY)
    CALL AXPY(N,RALPHA,X,INCX,Y)
    CALL AXPY(N,RALPHA,X,Y=Y)
    CALL AXPY(ALPHA=RALPHA,X=X,Y=Y)
```

Fortran Examples

Getting peak performance from Sun Performance Library for single processor applications is a matter of identifying code constructs in an application that can be replaced by calls to subroutines in Sun Performance Library. Multiprocessor applications can get additional speed by identifying opportunities for parallelization.

The easiest situation occurs when a block of user code exactly duplicates a capability of Sun Performance Library. Consider the code below:

```
      DO I = 1, N
          DO J = 1, N
              Y(I) = Y(I) + A(I,J) * X(J)
          END DO
      END DO
```

This is the matrix-vector product $y \leftarrow Ax + y$, which can be performed with the `DGEMV` subroutine.

As another example, consider the following code fragment:

```
      DO I = 1, N
          IF (V2(I,K) .LT. 0.0) THEN
              V2(I,K) = 0.0
          ELSE
              DO J = 1, M
                  X(J,I) = X(J,I) + Vl(J,K) * V2(I,K)
              END DO
          END IF
      END DO
```

In other cases, a block of code can be equivalent to several Sun Performance Library calls or contain a mixture of code that can be replaced together with code that has no natural replacement in Sun Performance Library. One way to rewrite the code with Sun Performance Library is shown below:

```
      DO I = 1, N
          IF (V2(I,K) .LT. 0.0) THEN
             V2(I,K) = 0.0
          END IF
      END DO
      CALL DGER (M, N, 1.0D0, X, LDX, Vl(l,K), 1, V2(1,K), 1)
```

An `f95` specific example is also shown.

```
WHERE (V(1:N,K) .LT. 0.0) THEN
       V(1:N,K) = 0.0
END WHERE
CALL DGER (M, N, 1.0D0, X, LDX, Vl(l,K), 1, V2(1,K), 1)
```

The code to replace negative numbers with zero in `V2` has no natural analog in Sun Performance Library, so that code is pulled out of the outer loop. With that code removed to its own loop, the rest of the loop can be recognized as being a rank- 1 update of the general matrix x, which can be accomplished using the `DGER` routine from BLAS.

Note that if there are many negative or zero values in `V2`, it may be that the majority of the time is not spent in the rank- 1 update and so replacing that code with the call to `DGER` might not bring a large payoff. It might be worthwhile to evaluate the reference to K. If it is a loop index, it may be that the loops shown here are part of a larger code structure, and loops over `DGEMV` or `DGER` can often be converted to some form of matrix multiplication. If so, a single call to a matrix multiplication routine will probably bring a much larger payoff than a loop over calls to `DGER`.

All Sun Performance Library routines are MT-safe (multithread safe). Because the routines are MT-safe, additional performance is possible on MP platforms by using the auto-parallelizing compiler to parallelize loops that contain calls to Sun Performance Library.

An example of an effective combination of a Sun Performance Library routine together with an auto-parallelizing compiler parallelization directive is shown in the following example.

```
      C$PAR DOALL
      DO I = 1, N
             CALL DGBMV ('No transpose', N, N, ALPHA, A, LDA,
     $     B(l,I), 1, BETA, C(l,I), 1)
      END DO
```

Sun Performance Library contains a routine named DGBMV to multiply a banded matrix by a vector. By putting this routine into a properly constructed loop, it is possible to use the routines in Sun Performance Library to multiply a banded matrix by a matrix. The compiler will not parallelize this loop by default because the presence of subroutine calls in a loop inhibits parallelization. However, because Sun Performance Library routines are MT-safe, a user may use parallelization directives as shown below to instruct the compiler to parallelize this loop.

Note that a user can also use compiler directives to parallelize a loop with a subroutine call that ordinarily would not be parallelizable. For example, it is ordinarily not possible to parallelize a loop containing a call to some of the linear system solvers, because some vendors have implemented those routines using code that is not MT-safe. Loops containing calls to the expert drivers of the linear system solvers (routines whose names end in SVX) are usually not parallelizable with other implementations of LAPACK. The implementation of LAPACK in Sun Performance Library allows parallelization of loops containing such calls. Because the versions in Sun Performance Library are MT-safe, users of MP platforms can get additional performance by parallelizing these loops.

C Interfaces

Sun Performance Library contains native C interfaces for each of the routines contained in LAPACK, BLAS, FFTPACK, VFFTPACK, and LINPACK. The Sun Performance Library C interfaces have the following features:

- Function names have C names
- Function interfaces follow C conventions
- C functions do not contain redundant or unnecessary arguments for a C function

The following example compares the standard LAPACK Fortran interface and the Sun Performance Library C interfaces for the `DGBCON` routine.

```
CALL DGBCON (NORM, N, NSUB, NSUPER, DA, LDA, IPIVOT, DANORM,
             DRCOND, DWORK, IWORK2, INFO)
void dgbcon(char norm, int n, int nsub, int nsuper, double *da,
            int lda, int *ipivot, double danorm, double drcond,
            int *info)
```

Note that the names of the arguments are the same and that arguments with the same name have the same base type. Scalar arguments that are used only as input values, such as `NORM` and `N`, are passed by value in the C version. Arrays and scalars that will be used to return values are passed by reference.

The Sun Performance Library C interfaces improve on CLAPACK, available on Netlib, which is an `f2c` translation of the standard libraries. For example, all of the CLAPACK routines are followed by a trailing underscore to maintain compatibility with Fortran compilers, which often postfix routine names in the object (`.o`) file with an underscore. The Sun Performance Library C interfaces do not require a trailing underscore.

Sun Performance Library C interfaces use the following conventions:

- Input-only scalars are passed by value rather than by reference, which gives added safety and allows constants to be passed without creating a separate variable to hold their value. Complex and double complex arguments are not considered scalars because they are not implemented as a scalar type by C.
- Complex scalars can be passed as either structures or arrays of length 2
- Arguments relating to workspace are not used in Sun Performance Library.
- Types of arguments must match even after C does type conversion. For example, be careful when passing a single precision real value because a C compiler can automatically promote the argument to double precision.
- Arrays are stored columnwise.
- Array indices are based at zero in conformance with C conventions rather than being based at one to conform to Fortran conventions.

  For example, the Fortran interface to `IDAMAX`, which C programs access as `idamax_`, would return a 1 to indicate the first element in a vector. The C interface to `idamax`, which C programs access as `idamax`, would return a 0 to indicate the first element of a vector. This convention is observed in function return values, permutation vectors, and anywhere else that vector or array indices are used.

Note – Some of the routines in Sun Performance Library use `malloc` internally, so user codes that make calls to Sun Performance Library and to `sbrk` may not work correctly.

Sun Performance Library uses global integer registers `%g2`, `%g3`, and `%g4` in 32-bit mode and `%g2` through `%g5` in 64-bit mode as scratch registers. User code should not use these registers for temporary storage, and then call a Sun Performance Library routine. The data will be overwritten when the Sun Performance Library routine uses these registers.

C Examples

The key to using Sun Performance Library to get peak performance from applications is to recognize opportunities to transform user-written code sequences into calls to Sun Performance Library functions. The following code sequence adapted from LAPACK shows one example:

```
int    i;
float a[n], b[n], largest;

largest = a[0];
for (i = 0; i < n; i++)
{
if (a[i] > largest)
    largest = a[i];
    if (b[i] > largest
    largest = b[i];
}
```

There is no subroutine in Sun Performance Library that exactly replicates the functionality of the code above. However, the code can be accelerated by replacing it with the several calls to Sun Performance Library as shown below:

```
int    i, large_index;
float a[n], b[n], largest;

large_index = isamax (n, a, l);
largest = a[large_index];
large_index = isamax (n, b, l);
if (b[large_index] > largest)
     largest = b[large_index];
```

Note the differences between the call to the native C `isamax` in Sun Performance Library above and the call shown below to a comparable function in CLAPACK:

```
/* 1. Declare scratch variable to allow 1 to be passed by value */
int one = l;
/* 2. Append underscore to conform to FORTRAN naming system     */
/* 3. Pass all arguments, even scalar input-only, by reference  */
/* 4. Subtract one to convert from FORTRAN indexing conventions */
large_index = isamax_ (&n, a, &one) - l;
largest = a[large_index]; large_index = isamax_ (&n, b, &one) - l;
if (b[large_index] > largest)
     largest = b[large_index];
```

As an example of a program that uses Sun Performance Library routines from user-managed threads, consider a real-time signal processing application running on a 4-processor server with one processor dedicated to acquiring the data, two processors dedicated to performing FFTs on the data, and one processor dedicated to postprocessing the data after the FFTs. It begins by creating multiple running instances of the function that performs the FFT:

```
for (i = 0; i < NCPUS_FOR_FFT; i++) {
   who[i] = i;
   do_fft[i] = 0;
   fft_done_buff_available[i] = l;
   (void)thr_create ((void *)0, (size_t)0, fft_func,
                     (void *)&who[i], (long)0, (thread_t *)0);
```

The code below is a simplified implementation of part of `fft_func` started by `thr_create` in the loop above. Note that production code should check the return value from `thr_create` above and should use semaphores rather than busy waits at the synchronization points in the code below.

```
cpu_id = *who_am_i;
while (1) {
  while (!do_fft[cpu_id]) {}
  rfftf (n, &dataset[0][cpu_id], &scratch[0][cpu_id]);
  while (!fft_done_buff_available[cpu_id]) {}
  fft_done_buff_available[cpu_id] = 0;
  scopy (n, &dataset[0][cpu_id], 1, &fft_done_buff[0][cpu_id], 1);
  do_fft[cpu_id] = 0;
}
```

CHAPTER 3

SPARC Optimization and Parallel Processing

This chapter describes how to use compiler and linking options to optimize applications for:

- Specific SPARC instruction set architectures
- 64-bit code
- Parallel processing

Using Sun Performance Library on SPARC Platforms

The Sun Performance Library was compiled using the `f95` compiler provided with this release. The Sun Performance Library routines were compiled using `-dalign` and `-xarch` set to `v8`, `v8plusa`, or `v9a`.

For each `-xarch` option used to compile the libraries, there is a library compiled with `-xparallel` and a library compiled without `-xparallel`. When linking the program, use `-dalign`, `-xlic_lib=sunperf`, and the same `-xarch` option that was used when compiling. If `-dalign` cannot be used in the program, supply a trap 6 handler as described in "Getting Started With Sun Performance Library" on page 14. If compiling with a value of `-xarch` that is not one of `[v8|v8plusa|v9a]`, the compiler driver will select the closest match.

Sun Performance Library is linked into an application with the `-xlic_lib` switch rather than the `-l` switch that is used to link in other libraries, as shown below:

```
my_system% f95 -dalign my_file.f -xlic_lib=sunperf
```

The `-xlic_lib` switch gives the same effect as if `-l` was used to specify the Sun Performance Library and added `-l` switches for all of the supporting libraries that Sun Performance Library requires.

Compiling for SPARC Platforms

Applications using Sun Performance Library can be optimized for specific SPARC instruction set architectures and for 64-bit code. The optimization for each architecture is targeted at one implementation of that architecture and includes optimizations for other architectures when it does not degrade the performance of the primary target.

Compile with the most appropriate `-xarch=` option for best performance. At link time, use the same `-xarch=` option that was used at compile time to select the version of the Sun Performance Library optimized for a specific SPARC instruction set architecture.

Note – Using SPARC-specific optimization options increases application performance on the selected instruction set architecture, but limits code portability. When using these optimization options, the resulting code can be run only on systems using the specific SPARC chip from Sun Microsystems and, in some cases, a specific Solaris operating environment (32- or 64-bit Solaris 7 or Solaris 8).

The SunOS™ command `isalist`(1) can be used to display a list of the native instruction sets executable on a particular platform. The names output by `isalist` are space-separated and are ordered in the sense of best performance.

For a detailed description of the different `-xarch` options, refer to the *Fortran User's Guide* or *C User's Guide*.

To compile for 32-bit addressing in a 32-bit enabled Solaris operating environment:

- UltraSPARC I™ or UltraSPARC II™ systems – use `-xarch=v8plus` or `-xarch=v8plusa`.
- UltraSPARC III™ systems – use `-xarch=v8plus` or `-xarch=v8plusb`.

To compile for 64-bit addressing in a 64-bit enabled Solaris operating environment:

- UltraSPARC I or UltraSPARC - II systems–, use `-xarch=v9` or `-xarch=v9a`.
- UltraSPARC III systems – use `-xarch=v9` or `-xarch=v9b`.

Compiling Code for 64-Bit UltraSPARC

To compile 64-bit code on UltraSPARC, use `-xarch=v9[a|b]` and convert all integer arguments to 64-bit arguments. 64-bit routines require the use of 64-bit integers.

Sun Performance Library provides 32-bit and 64-bit interfaces. To use the 64-bit interfaces:

- Modify the Sun Performance Library routine name: For C, FORTRAN 77, and Fortran 95 code (without the `USE SUNPERF` statement), `_64` must be appended to the names of Sun Performance Library routines (for example, `dgbcon_64` or `CAXPY_64`). For `f95` code with the `USE SUNPERF` statement, do not append `_64` to the Sun Performance Library routine names. The compiler will infer the correct interface from the presence or absence of `INTEGER*8` arguments.
- Promote integers to 64 bits. Double precision variables and the real and imaginary parts of double complex variables are already 64 bits. Only the size of the integers is affected.

To control promotion of integer arguments, do one of the following:

- To promote all integers from 32 bits to 64 bits, compile with `-xtypemap=integer:64`.
- When using Fortran, to avoid promoting all integers, change `INTEGER` or `INTEGER*4` declarations to `INTEGER*8`.

When passing constants in Fortran 95 code that have not been compiled with `-xtypemap`, append `_8` to literal constants to effect the promotion. For example, when using Fortran 95, change `CALL DSCAL(20,5.26D0,X,1)` to `CALL DSCAL(20_8,5.26D0,X,1_8)`. This example assumes `USE SUNPERF` is included in the code.

The following example shows calling `CAXPY` from FORTRAN 77 or Fortran 95 using 32-bit arguments:

```
      SUBROUTINE CAXPY (N, ALPHA, X, INCY, Y, INCY)
      COMPLEX ALPHA
      INTEGER INCX, INCY, N
      COMPLEX X( * ), Y( * )
```

The following example shows calling CAXPY from FORTRAN 77 or Fortran 95 (without the USE SUNPERF statement) using 64-bit arguments:

```
      SUBROUTINE CAXPY_64 (N, ALPHA, X, INCY, Y, INCY)
      COMPLEX   ALPHA
      INTEGER*8 INCX, INCY, N
      COMPLEX   X( * ), Y( * )
```

The following example shows calling CAXPY from Fortran 95 (with the USE SUNPERF statement) using 64-bit arguments:

```
      SUBROUTINE CAXPY (N, ALPHA, X, INCY, Y, INCY)
      COMPLEX   ALPHA
      INTEGER*8 INCX, INCY, N
      COMPLEX   X( * ), Y( * )
```

In C routines, the size of long is 32 bits when compiling for V8 or V8plus and 64 bits when compiling for V9. The following example shows calling the dgbcon routine using 32-bit arguments.

```
void dgbcon(char norm, int n, int nsub, int nsuper, double *da,
            int lda, int *ipivot, double danorm, double drcond,
            int *info)
```

The following example shows calling the dgbcon routine using 64-bit arguments.

```
void dgbcon_64 (char norm, long n, long nsub, long nsuper,
                double *da, long lda, long *ipivot, double danorm,
                double *drcond, long *info)
```

Optimizing for Parallel Processing

Note – The Fortran compiler parallelization features require a Sun WorkShop HPC license.

Sun Performance Library can be used with the shared or dedicated modes of parallelization, that are user selectable at link time. Specifying the parallelization mode improves application performance by using the parallelization enhancements made to Sun Performance Library routines.

The shared multiprocessor model of parallelism has the following features:

- Delivers peak performance to applications that do not use compiler parallelization and that run on a platform shared with other applications.
- Parallelization is implemented with threads library synchronization primitives.

The dedicated multiprocessor model of parallelism has the following features:

- Delivers peak performance to applications using automatic compiler parallelization and running on an MP platform dedicated to a single processor-intensive application
- Parallelization is implemented with spin locks.

On a dedicated system, the dedicated model can be faster than the shared model due to lower synchronization overhead. On a system running many different tasks, the shared model can make better use of available resources.

Specifying the Parallelization Mode

To specify the parallelization mode:

- *Shared model* – Use `-mt` on the link line without one of the compiler parallelization options.
- *Dedicated model* – Use one of the compiler parallelization options `[-xparallel|-xexplicitpar|-xautopar]` on the compile and link lines.
- *Single processor* – Do not specify any of the compiler parallelization options or `-mt` on the link line.

Note – Using the shared model with one of the compiler parallelization options, `-xparallel`, `-xexplicitpar`, or `-xautopar`, produces unpredictable behavior.

If compiling with one of the compiler parallelization options:

- Use the same parallelization option on the linking command.
- To use multiple processors, add `-mt` to the link line, and then specify the number of processors at runtime with the `PARALLEL` environment variable.

For example, to use 24 processors, type the commands shown below:

```
my_system% f95 -dalign -mt my_app.f -xlic_lib=sunperf
my_system% setenv PARALLEL 24
my_system% ./a.out
```

Note – Parallel processing options require using either the `-dalign` command-line option or establishing a trap 6 handler, as described in "Enabling Trap 6" on page 15. When using C, do not use `-misalign`.

Starting Threads

When Sun Performance Library starts threads in shared mode, it uses a stack size that it determines as follows:

1. Checks the value of the `STACKSIZE` environment variable and interpret the units as kbytes (1024 bytes).
2. Computes the maximum stack size required by Sun Performance Library.
3. Uses the largest of the values determined in steps 1 and 2 for the size of the stack in the created thread.

When Sun Performance Library starts threads in dedicated mode, use the `STACKSIZE` environment variable to specify a stack size of at least 4 MB:

```
setenv STACKSIZE 4000
```

Parallel Processing Examples

The following sections demonstrate using the `PARALLEL` environment variable and the compile and linking options for creating code that supports using:

- A single processor
- Multiple processors in shared mode
- Multiple processors in dedicated mode

Using a Single Processor

To use a single processor:

1. **Call one or more of the routines.**
2. **Set `PARALLEL` equal to 1.**
3. **Link with `-xlic_lib=sunperf` specified at the end of the command line.**

   Do not compile or link with `-parallel`, `-explicitpar`, or `-autopar.`

   For example, compile and link with `libsunperf.so` (default):

```
cc -dalign -xarch=... any.c -xlic_lib=sunperf
or
f77 -dalign -xarch=... any.f -xlic_lib=sunperf
or
f95 -dalign -xarch=... any.f95 -xlic_lib=sunperf
```

For example: Compile and link with `libsunperf.a` statically:

```
cc -dalign -xarch=... any.c -Bstatic -xlic_lib=sunperf -Bdynamic
or
f77 -dalign -xarch=... any.f -Bstatic -xlic_lib=sunperf -Bdynamic
or
f95 -dalign -xarch=... any.f95 -Bstatic -xlic_lib=sunperf -Bdynamic
```

Using Multiple Processors in Shared Mode

To use multiple processors in shared mode:

1. **Call one or more of the routines.**
2. **Set `PARALLEL` to a number greater than 1.**
3. **Compile and link with `-mt`.**
4. **Link with `-xlic_lib=sunperf` specified at the end of the command line.**

   Do not compile or link with `-parallel`, `-explicitpar`, or `-autopar.`

   For example, compile and link with `libsunperf.so` (default):

```
cc -dalign -xarch=... any.c -xlic_lib=sunperf -mt
or
f77 -dalign -xarch=... any.f -xlic_lib=sunperf -mt
or
f95 -dalign -xarch=... any.f95 -xlic_lib=sunperf -mt
```

For example: Compile and link with `libsunperf.a` statically:

```
cc -dalign -xarch=... any.c -Bstatic -xlic_lib=sunperf -Bdynamic -mt
or
f77 -dalign -xarch=... any.f -Bstatic -xlic_lib=sunperf -Bdynamic -mt
or
f95 -dalign -xarch=... any.f95 -Bstatic -xlic_lib=sunperf -Bdynamic -mt
```

Using Multiple Processors in Dedicated Mode (With Parallelization Options)

To use multiple processors in dedicated mode:

1. **Call one or more of the routines.**
2. **Set `PARALLEL` to the number of available processors.**
3. **Link with `-xlic_lib=sunperf` specified at the end of the command line.**

   Compile and link with `-parallel`, `-explicitpar`, or `-autopar.`

For example, compile and link with `libsunperf_mt.so` (default):

```
cc -dalign -xarch=... -xparallel any.c -xlic_lib=sunperf
or
f77 -dalign -xarch=... -parallel any.f -xlic_lib=sunperf
or
f95 -dalign -xarch=... -parallel any.f95 -xlic_lib=sunperf
```

For example, compile and link with `libsunperf_mt.a` statically:

```
cc -dalign -xarch=... -xparallel any.c  -Bstatic -xlic_lib=sunperf -Bdynamic
or
f77 -dalign -xarch=... -parallel any.f -Bstatic -xlic_lib=sunperf -Bdynamic
or
f95 -dalign -xarch=... -parallel any.f95 -Bstatic -xlic_lib=sunperf -Bdynamic
```

CHAPTER 4

Working With Matrices

Most matrices can be stored in ways that save both storage space and computation time. Sun Performance Library uses the following storage schemes:

- Banded storage
- Packed storage

The Sun Performance Library processes matrices that are in one of four forms:

- General
- Triangular
- Symmetric
- Tridiagonal

Storage schemes and matrix types are described in the following sections.

Matrix Storage Schemes

Some Sun Performance Library routines that work with arrays stored normally have corresponding routines that take advantage of these special storage forms. For example, `DGBMV` will form the product of a general matrix in banded storage and a vector, and `DTPMV` will form the product of a triangular matrix in packed storage and a vector.

Banded Storage

A banded matrix is stored so the *j*th column of the matrix corresponds to the *j*th column of the Fortran array.

The following code copies a banded general matrix in a general array into banded storage mode.

```
C     Copy the matrix A from the array AG to the array AB. The
C     matrix is stored in general storage mode in AG and it will
C     be stored in banded storage mode in AB. The code to copy
C     from general to banded storage mode is taken from the
C     comment block in the original DGBFA by Cleve Moler.
C
      NSUB = 1
      NSUPER = 2
      NDIAG = NSUB + 1 + NSUPER
      DO ICOL = 1, N
        I1 = MAX0 (1, ICOL - NSUPER)
        I2 = MIN0 (N, ICOL + NSUB)
        DO IROW = I1, I2
          IROWB = IROW - ICOL + NDIAG
          AB(IROWB,ICOL) = AG(IROW,ICOL)
        END DO
      END DO
```

Note that this method of storing banded matrices is compatible with the storage method used by LAPACK, BLAS, and LINPACK, but is inconsistent with the method used by EISPACK.

Packed Storage

A packed vector is an alternate representation for a triangular, symmetric, or Hermitian matrix. An array is packed into a vector by storing the elements sequentially column by column into the vector. Space for the diagonal elements is always reserved, even if the values of the diagonal elements are known, such as in a unit diagonal matrix.

An upper triangular matrix or a symmetric matrix whose upper triangle is stored in general storage in the array A, can be transferred to packed storage in the array AP as shown below. This code comes from the comment block of the LAPACK routine DTPTRI.

```
    JC = 1
    DO J = 1, N
       DO I = 1, J
          AP(JC+I-1) = A(I,J)
       END DO
       JC = JC + J
     END DO
```

Similarly, a lower triangular matrix or a symmetric matrix whose lower triangle is stored in general storage in the array A, can be transferred to packed storage in the array AP as shown below:

```
    JC = 1
    DO J = 1, N
       DO I = J, N
          AP(JC+I-1) = A(I,J)
       END DO
       JC = JC + N - J + 1
    END DO
```

Matrix Types

The general matrix form is the most common matrix, and most operations performed by the Sun Performance Library can be done on general arrays. In many cases, there are routines that will work with the other forms of the arrays. For example, DGEMM will form the product of two general matrices and DTRMM will form the product of a triangular and a general matrix.

General Matrices

A general matrix is stored so that there is a one-to-one correspondence between the elements of the matrix and the elements of the array. Element A_{ij} of a matrix A is stored in element `A(I,J)` of the corresponding array A. The general form is the most common form. A general matrix, because it is dense, has no special storage scheme. In a general banded matrix, however, the diagonal of the matrix is stored in the row below the upper diagonals.

For example, as shown below, the general banded matrix can be represented with banded storage. Elements shown with the symbol × are never accessed by routines that process banded arrays.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 \\ a_{21} & a_{22} & a_{23} & a_{24} & 0 \\ 0 & a_{32} & a_{33} & a_{34} & a_{35} \\ 0 & 0 & a_{43} & a_{44} & a_{45} \\ 0 & 0 & 0 & a_{54} & a_{55} \end{bmatrix} \qquad \begin{bmatrix} \times & \times & a_{13} & a_{24} & a_{35} \\ \times & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & \times \end{bmatrix}$$

General Banded Matrix General Banded Array in Banded Storage

Triangular Matrices

A triangular matrix is stored so that there is a one-to-one correspondence between the nonzero elements of the matrix and the elements of the array, but the elements of the array corresponding to the zero elements of the matrix are never accessed by routines that process triangular arrays.

A triangular matrix can be stored using packed storage.

$$\begin{bmatrix} a_{11} & 0 & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \qquad \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ a_{22} \\ a_{32} \\ a_{33} \end{bmatrix}$$

Triangular Matrix Triangular Array in Packed Storage

A triangular banded matrix can be stored using banded storage as shown below. Elements shown with the symbol × are never accessed by routines that process banded arrays.

$$\begin{bmatrix} a_{11} & 0 & 0 \\ a_{21} & a_{22} & 0 \\ 0 & a_{32} & a_{33} \end{bmatrix} \qquad \begin{bmatrix} a_{11} & a_{22} & a_{33} \\ a_{21} & a_{32} & \times \end{bmatrix}$$

Triangular Banded Matrix

Triangular Banded Array in Banded Storage

Symmetric Matrices

A symmetric matrix is similar to a triangular matrix in that the data in either the upper or lower triangle corresponds to the elements of the array. The contents of the other elements in the array are assumed and those array elements are never accessed by routines that process symmetric or Hermitian arrays.

A symmetric matrix can be stored using packed storage.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \qquad \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ a_{22} \\ a_{32} \\ a_{33} \end{bmatrix}$$

Symmetric Matrix

Symmetric Array in Packed Storage

A symmetric banded matrix can be stored using banded storage as shown below. Elements shown with the symbol × are never accessed by routines that process banded arrays.

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ 0 & a_{32} & a_{33} & a_{34} \\ 0 & 0 & a_{43} & a_{44} \end{bmatrix}$$

Symmetric Banded Matrix

$$\begin{bmatrix} \times & a_{12} & a_{23} & a_{34} \\ a_{11} & a_{22} & a_{33} & a_{44} \\ a_{21} & a_{32} & a_{43} & \times \end{bmatrix}$$

Symmetric Banded Array
in Banded Storage

Tridiagonal Matrices

A tridiagonal matrix has elements only on the main diagonal, the first superdiagonal, and the first subdiagonal. It is stored using three 1-dimensional arrays.

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ 0 & a_{32} & a_{33} & a_{34} \\ 0 & 0 & a_{43} & a_{44} \end{bmatrix}$$

Tridiagonal Matrix

$$\begin{bmatrix} a_{21} \\ a_{32} \\ a_{43} \end{bmatrix} \quad \begin{bmatrix} a_{11} \\ a_{22} \\ a_{33} \\ a_{44} \end{bmatrix} \quad \begin{bmatrix} a_{12} \\ a_{23} \\ a_{34} \end{bmatrix}$$

Tridiagonal Array in Tridiagonal Storage

Sparse Matrices

The Sun Performance Library sparse solver package is a collection of routines that efficiently factor and solve sparse linear systems of equations. Use the sparse solver package to:

- Solve symmetric, structurally symmetric, and unsymmetric coefficient matrices
- Specify a choice of ordering methods, including user-specified orderings

The sparse solver package contains interfaces for FORTRAN 77. Fortran 95 and C interfaces are not currently provided. To use the sparse solver routines from Fortran 95, use the FORTRAN 77 interfaces. To use the sparse solver routines with C, append an underscore to the routine name (`dgssin_()`, `dgssor_()`, and so on), pass arguments by reference, and use 1-based array indexing.

Sparse Solver Matrix Data Formats

Sparse matrices are usually represented in formats that minimize storage requirements. By taking advantage of the sparsity and not storing zeros, considerable storage space can be saved. The storage format used by the general sparse solver is the compressed sparse column (CSC) format (also called the Harwell-Boeing format).

The CSC format represents a sparse matrix with two integer arrays and one floating point array. The integer arrays (colptr and rowind) specify the location of the nonzeros of the sparse matrix, and the floating point array (values) is used for the nonzero values.

The column pointer (colptr) array consists of n+1 elements where colptr(i) points to the beginning of the ith column, and colptr(i + 1) – 1 points to the end of the ith column. The row indices (rowind) array contains the row indices of the nonzero values. The values arrays contains the corresponding nonzero numerical values.

The following matrix data formats exist for a sparse matrix of *neqns* equations and *nnz* nonzeros:

- Symmetric
- Structurally symmetric
- Unsymmetric

The most efficient data representation often depends on the specific problem. The following sections show examples of sparse matrix data formats.

Symmetric Sparse Matrices

A symmetric sparse matrix is a matrix where a(i, j) = a(j, i) for all i and j. Because of this symmetry, only the lower triangular values need to be passed to the solver routines. The upper triangle can be determined from the lower triangle.

An example of a symmetric matrix is shown below. This example is derived from A. George and J. W-H. Liu. "Computer Solution of Large Sparse Positive Definite Systems."

$$A = \begin{bmatrix} 4.0 & 1.0 & 2.0 & 0.5 & 2.0 \\ 1.0 & 0.5 & 0.0 & 0.0 & 0.0 \\ 2.0 & 0.0 & 3.0 & 0.0 & 0.0 \\ 0.5 & 0.0 & 0.0 & 0.625 & 0.0 \\ 2.0 & 0.0 & 0.0 & 0.0 & 16.0 \end{bmatrix}$$

To represent A in CSC format:

- colptr: 1, 6, 7, 8, 9, 10
- rowind: 1, 2, 3, 4, 5, 2, 3, 4, 5
- values: 4.0, 1.0, 2.0, 0.5, 2.0, 0.5, 3.0, 0.625, 16.0

Structurally Symmetric Sparse Matrices

A structurally symmetric sparse matrix has nonzero values with the property that if a(i, j) ≠ 0, then a(j, i) ≠ 0 for all i and j. When solving a structurally symmetric system, the entire matrix must be passed to the solver routines.

An example of a structurally symmetric matrix is shown below.

$$A = \begin{bmatrix} 1.0 & 3.0 & 0.0 & 0.0 \\ 2.0 & 4.0 & 0.0 & 7.0 \\ 0.0 & 0.0 & 6.0 & 0.0 \\ 0.0 & 5.0 & 0.0 & 8.0 \end{bmatrix}$$

To represent A in CSC format:

- colptr: 1, 3, 6, 7, 9
- rowind: 1, 2, 1, 2, 4, 3, 2, 4
- values: 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0

Unsymmetric Sparse Matrices

An unsymmetric sparse matrix does not have a(*i*, *j*) = a(*j*, *i*) for all *i* and *j*. The structure of the matrix does not have an apparent pattern. When solving an unsymmetric system, the entire matrix must be passed to the solver routines. An example of an unsymmetric matrix is shown below.

$$A = \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 2.0 & 6.0 & 0.0 & 0.0 & 9.0 \\ 3.0 & 0.0 & 7.0 & 0.0 & 0.0 \\ 4.0 & 0.0 & 0.0 & 8.0 & 0.0 \\ 5.0 & 0.0 & 0.0 & 0.0 & 10.0 \end{bmatrix}$$

To represent *A* in CSC format:

- colptr: 1, 6, 7, 8, 9, 11
- rowind: 1, 2, 3, 4, 5, 2, 3, 4, 2, 5
- values: 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0

Sun Performance Library Sparse BLAS

The Sun Performance Library sparse BLAS package is based on the following two packages:

- Netlib Sparse BLAS package, by Dodson, Grimes, and Lewis consists of sparse extensions to the Basic Linear Algebra Subroutines that operate on sparse vectors.
- NIST (National Institute of Standards and Technology) Fortran Sparse BLAS Library consists of routines that perform matrix products and solution of triangular systems for sparse matrices in a variety of storage formats.

Refer to the following sources for additional sparse BLAS information.

- For information on the Sun Performance Library Sparse BLAS routines, refer to the section 3P man pages for the individual routines.
- For more information on the Netlib Sparse BLAS package refer to `http://www.netlib.org/sparse-blas/index.html`.
- For more information on the NIST Fortran Sparse BLAS routines, refer to `http://math.nist.gov/spblas/`

Naming Conventions

The Netlib Sparse BLAS and NIST Fortran Sparse BLAS Library routines each use their own naming conventions, as described in the following two sections.

Netlib Sparse BLAS

Each Netlib Sparse BLAS routine has a name of the form Prefix-Root-Suffix where the:

- Prefix represents the data type.
- Root represents the operation.
- Suffix represents whether or not the routine is a direct extension of an existing dense BLAS routine.

TABLE 4-1 lists the naming conventions for the Netlib Sparse BLAS vector routines.

TABLE 4-1 Netlib Sparse BLAS Naming Conventions

Operation	Root of Name	Prefix and Suffix
Dot product	`-DOT-`	`S-I  D-I  C-UI Z-UI C-CI Z-CI`
Scalar times a vector added to a vector	`-AXPY-`	`S-I  D-I  C-I  Z-I`
Apply Givens rotation	`-ROT-`	`S-I  D-I`
Gather x into y	`-GTHR-`	`S-  D-  C-  Z-  S-Z  D-Z  C-Z Z-Z`
Scatter x into y	`-SCTR-`	`S-  D-  C-  Z-`

The prefix can be one of the following data types:

- S: `SINGLE`
- D: `DOUBLE`
- C: `COMPLEX`
- Z: `COMPLEX*16` or `DOUBLE COMPLEX`

The `I`, `CI`, and `UI` suffixes denote sparse BLAS routines that are direct extensions to dense BLAS routines.

NIST Fortran Sparse BLAS

Each NIST Fortran Sparse BLAS routine has a six-character name of the form *XYYYZZ* where:

- *X* represents the data type.
- *YYY* represents the sparse storage format.
- *ZZ* represents the operation.

TABLE 4-2 shows the values for *X*, *Y*, and *Z*.

TABLE 4-2 NIST Fortran Sparse BLAS Routine Naming Conventions

X: Data Type		
X	S: single precision D: double precision	
YYY: Sparse Storage Format		
YYY	Single entry formats:	COO: coordinate CSC: compressed sparse column CSR: compressed sparse row DIA: diagonal ELL: ellpack JAD: jagged diagonal SKY: skyline
	Block entry formats:	BCO: block coordinate BSC: block compressed sparse column BSR: block compressed sparse row BDI: block diagonal BEL: block ellpack VBR: block compressed sparse row
ZZ: Operation		
ZZ	MM: matrix-matrix product SM: solution of triangular system (supported for all formats except COO) RP: right permutation (for JAD format only)	

Sparse Solver Routines

The Sun Performance Library sparse solver package contains the routines listed in TABLE 4-3.

TABLE 4-3 Sparse Solver Routines

Routine	Function
`DGSSFS()`	One call interface to sparse solver
`DGSSIN()`	Sparse solver initialization
`DGSSOR()`	Fill reducing ordering and symbolic factorization
`DGSSFA()`	Matrix value input and numeric factorization
`DGSSSL()`	Triangular solve
Utility Routine	**Function**
`DGSSUO()`	Sets user-specified ordering permutation.
`DGSSRP()`	Returns permutation used by solver.
`DGSSCO()`	Returns condition number estimate of coefficient matrix.
`DGSSDA()`	De-allocates sparse solver.
`DGSSPS()`	Prints solver statistics.

Use the regular interface to solve multiple matrices with the same structure, but different numerical values, as shown below:

```
call dgssin() ! {initialization, input coefficient matrix
              !  structure}
call dgssor() ! {fill-reducing ordering, symbolic factorization}
do m = 1, number_of_structurally_identical_matrices
    call dgssfa() ! {input coefficient matrix values, numeric
                  ! factorization}
    do r = 1, number_of_right_hand_sides
        call dgsssl() ! {triangular solve}
    enddo
enddo
```

The one-call interface is not as flexible as the regular interface, but it covers the most common case of factoring a single matrix and solving some number right-hand sides. Additional calls to `dgsssl()` are allowed to solve for additional right-hand sides, as shown below.

```
call dgssfs() ! {initialization, input coefficient matrix
              ! structure}
              ! {fill-reducing ordering, symbolic factorization}
              ! {input coefficient matrix values, numeric
              ! factorization}
              ! {triangular solve}
do r = 1, number_of_right_hand_sides
    call dgsssl() ! {triangular solve}
enddo
```

Routine Calling Order

To solve problems with the sparse solver package, use the sparse solver routines in the order shown in TABLE 4-4.

TABLE 4-4 Sparse Solver Routine Calling Order

One Call Interface: For solving single matrix	
Start	
`DGSSFS()`	Initialize, order, factor, solve
`DGSSSL()`	Additional solves (optional): repeat `dgsssl()` as needed
`DGSSDA()`	Deallocate working storage
Finish	
End of One-Call Interface	

TABLE 4-4 Sparse Solver Routine Calling Order *(Continued)*

Regular Interface: For solving multiple matrices with the same structure	
Start	
`DGSSIN()`	Initialize
`DGSSOR()`	Order
`DGSSFA()`	Factor
`DGSSSL()`	Solve: repeat `dgssfa()` or `dgsssl()` as needed
`DGSSDA()`	Deallocate working storage
Finish	
End of Regular Interface	

Sparse Solver Examples

CODE EXAMPLE 4-1 shows solving a symmetric system using the one-call interface, and CODE EXAMPLE 4-2 on page 55 shows solving a symmetric system using the regular interface.

CODE EXAMPLE 4-1 Solving a Symmetric System–One-Call Interface

```
my_system% cat example_1call.f
      program example_1call
c
c  This program is an example driver that calls the sparse solver.
c    It factors and solves a symmetric system, by calling the
c    one-call interface.
c
      implicit none

      integer           neqns, ier, msglvl, outunt, ldrhs, nrhs
      character         mtxtyp*2, pivot*1, ordmthd*3
      double precision  handle(150)
      integer           colstr(6), rowind(9)
      double precision  values(9), rhs(5), xexpct(5)
      integer           i
c
c  Sparse matrix structure and value arrays.  From George and Liu,
c  page 3.
```

CODE EXAMPLE 4-1 Solving a Symmetric System–One-Call Interface *(Continued)*

```
c    Ax = b, (solve for x) where:
c
c      4.0   1.0   2.0   0.5   2.0        2.0       7.0
c      1.0   0.5   0.0   0.0   0.0        2.0       3.0
c  A = 2.0   0.0   3.0   0.0   0.0   x = 1.0   b = 7.0
c      0.5   0.0   0.0   0.625 0.0       -8.0      -4.0
c      2.0   0.0   0.0   0.0  16.0       -0.5      -4.0
c
      data colstr / 1, 6, 7, 8, 9, 10 /
      data rowind / 1, 2, 3, 4, 5, 2, 3, 4, 5 /
     data values / 4.0d0, 1.0d0, 2.0d0, 0.5d0, 2.0d0, 0.5d0, 3.0d0,
     &              0.625d0, 16.0d0 /
      data rhs    / 7.0d0, 3.0d0, 7.0d0, -4.0d0, -4.0d0 /
      data xexpct / 2.0d0, 2.0d0, 1.0d0, -8.0d0, -0.5d0 /
c
c  set calling parameters
c
      mtxtyp= 'ss'
      pivot = 'n'
      neqns  = 5
      nrhs   = 1
      ldrhs  = 5
      outunt = 6
      msglvl = 0
      ordmthd = 'mmd'
c
c  call single call interface
c
      call dgssfs ( mtxtyp, pivot,  neqns , colstr, rowind,
     &              values, nrhs  , rhs,    ldrhs , ordmthd,
     &              outunt, msglvl, handle, ier             )
      if ( ier .ne. 0 ) goto 110
c
c  deallocate sparse solver storage
c
      call dgssda ( handle, ier )
      if ( ier .ne. 0 ) goto 110
c
c  print values of sol
c
      write(6,200) 'i', 'rhs(i)', 'expected rhs(i)', 'error'
```

CODE EXAMPLE 4-1 Solving a Symmetric System–One-Call Interface *(Continued)*

```
      do i = 1, neqns
        write(6,300) i, rhs(i), xexpct(i), (rhs(i)-xexpct(i))
      enddo
      stop
  110 continue
c
c call to sparse solver returns an error
c
      write ( 6 , 400 )
     &      ' example: FAILED sparse solver error number = ', ier
      stop

  200 format(a5,3a20)

  300 format(i5,3d20.12) ! i/sol/xexpct values

  400 format(a60,i20) ! fail message, sparse solver error number

      end
my_system% f95 -dalign example_1call.f -xlic_lib=sunperf
my_sytem% a.out
    i              rhs(i)     expected rhs(i)               error
    1  0.200000000000D+01  0.200000000000D+01 -0.528466159722D-13
    2  0.200000000000D+01  0.200000000000D+01  0.105249142734D-12
    3  0.100000000000D+01  0.100000000000D+01  0.350830475782D-13
    4 -0.800000000000D+01 -0.800000000000D+01  0.426325641456D-13
    5 -0.500000000000D+00 -0.500000000000D+00  0.660582699652D-14
```

CODE EXAMPLE 4-2 Solving a Symmetric System–Regular Interface

```
my_system% cat example_ss.f
      program example_ss
c
c  This program is an example driver that calls the sparse solver.
c  It factors and solves a symmetric system.

      implicit none

      integer           neqns, ier, msglvl, outunt, ldrhs, nrhs
      character         mtxtyp*2, pivot*1, ordmthd*3
      double precision  handle(150)
      integer           colstr(6), rowind(9)
      double precision  values(9), rhs(5), xexpct(5)
      integer           i
c
c  Sparse matrix structure and value arrays.  From George and Liu,
c  page 3.
c    Ax = b, (solve for x) where:
c
c       4.0   1.0   2.0   0.5   2.0       2.0       7.0
c       1.0   0.5   0.0   0.0   0.0       2.0       3.0
c  A = 2.0   0.0   3.0   0.0   0.0   x = 1.0   b = 7.0
c       0.5   0.0   0.0   0.625 0.0      -8.0      -4.0
c       2.0   0.0   0.0   0.0  16.0      -0.5      -4.0
c
      data colstr / 1, 6, 7, 8, 9, 10 /
      data rowind / 1, 2, 3, 4, 5, 2, 3, 4, 5 /
      data values / 4.0d0, 1.0d0, 2.0d0, 0.5d0, 2.0d0, 0.5d0,
     &             3.0d0, 0.625d0, 16.0d0 /
      data rhs    / 7.0d0, 3.0d0, 7.0d0, -4.0d0, -4.0d0 /
      data xexpct / 2.0d0, 2.0d0, 1.0d0, -8.0d0, -0.5d0 /
c
c  initialize solver
c
      mtxtyp= 'ss'
      pivot = 'n'
      neqns  = 5
      outunt = 6
      msglvl = 0
```

CODE EXAMPLE 4-2 Solving a Symmetric System–Regular Interface *(Continued)*

```
c
c  call regular interface
c
      call dgssin ( mtxtyp, pivot,  neqns , colstr, rowind,
     &              outunt, msglvl, handle, ier             )
      if ( ier .ne. 0 ) goto 110
c
c  ordering and symbolic factorization
c
      ordmthd = 'mmd'
      call dgssor ( ordmthd, handle, ier )
      if ( ier .ne. 0 ) goto 110
c
c  numeric factorization
c
      call dgssfa ( neqns, colstr, rowind, values, handle, ier )
      if ( ier .ne. 0 ) goto 110
c
c  solution
c
      nrhs   = 1
      ldrhs  = 5
      call dgsssl ( nrhs, rhs, ldrhs, handle, ier )
      if ( ier .ne. 0 ) goto 110
c
c  deallocate sparse solver storage
c
      call dgssda ( handle, ier )
      if ( ier .ne. 0 ) goto 110
c
c  print values of sol
c
      write(6,200) 'i', 'rhs(i)', 'expected rhs(i)', 'error'
      do i = 1, neqns
        write(6,300) i, rhs(i), xexpct(i), (rhs(i)-xexpct(i))
      enddo
      stop
  110 continue
```

CODE EXAMPLE 4-2 Solving a Symmetric System–Regular Interface *(Continued)*

```
c
c call to sparse solver returns an error
c
      write ( 6 , 400 )
     &      ' example: FAILED sparse solver error number = ', ier
      stop

  200 format(a5,3a20)

  300 format(i5,3d20.12) ! i/sol/xexpct values

  400 format(a60,i20) ! fail message, sparse solver error number

      end
my_system% f95 -dalign example_ss.f -xlic_lib=sunperf
my_sytem% a.out
    i             rhs(i)     expected rhs(i)               error
    1  0.200000000000D+01  0.200000000000D+01 -0.528466159722D-13
    2  0.200000000000D+01  0.200000000000D+01  0.105249142734D-12
    3  0.100000000000D+01  0.100000000000D+01  0.350830475782D-13
    4 -0.800000000000D+01 -0.800000000000D+01  0.426325641456D-13
    5 -0.500000000000D+00 -0.500000000000D+00  0.660582699652D-14
```

CODE EXAMPLE 4-3 Solving a Structurally Symmetric System With Unsymmetric Values–Regular Interface

```
my_system% cat example_su.f
      program example_su
c
c  This program is an example driver that calls the sparse solver.
c    It factors and solves a structurally symmetric system
c    (w/unsymmetric values).
c
      implicit none

      integer           neqns, ier, msglvl, outunt, ldrhs, nrhs
      character         mtxtyp*2, pivot*1, ordmthd*3
      double precision  handle(150)
      integer           colstr(5), rowind(8)
      double precision  values(8), rhs(4), xexpct(4)
      integer           i
c
c  Sparse matrix structure and value arrays.  Coefficient matrix
c    has a symmetric structure and unsymmetric values.
c    Ax = b, (solve for x) where:
c
c      1.0   3.0   0.0   0.0       1.0        7.0
c      2.0   4.0   0.0   7.0       2.0       38.0
c  A = 0.0   0.0   6.0   0.0   x = 3.0   b = 18.0
c      0.0   5.0   0.0   8.0       4.0       42.0
c
      data colstr / 1, 3, 6, 7, 9 /
      data rowind / 1, 2, 1, 2, 4, 3, 2, 4 /
     data values / 1.0d0, 2.0d0, 3.0d0, 4.0d0, 5.0d0, 6.0d0, 7.0d0,
     &              8.0d0 /
      data rhs    / 7.0d0, 38.0d0, 18.0d0, 42.0d0 /
      data xexpct / 1.0d0, 2.0d0, 3.0d0, 4.0d0 /
c
c  initialize solver
c
      mtxtyp= 'su'
      pivot = 'n'
      neqns  = 4
      outunt = 6
      msglvl = 0
```

CODE EXAMPLE 4-3 Solving a Structurally Symmetric System With Unsymmetric Values–Regular Interface *(Continued)*

```
c
c  call regular interface
c
      call dgssin ( mtxtyp, pivot,  neqns , colstr, rowind,
     &              outunt, msglvl, handle, ier             )
      if ( ier .ne. 0 ) goto 110
c
c  ordering and symbolic factorization
c
      ordmthd = 'mmd'
      call dgssor ( ordmthd, handle, ier )
      if ( ier .ne. 0 ) goto 110
c
c  numeric factorization
c
      call dgssfa ( neqns, colstr, rowind, values, handle, ier )
      if ( ier .ne. 0 ) goto 110
c
c  solution
c
      nrhs   = 1
      ldrhs  = 4
      call dgsssl ( nrhs, rhs, ldrhs, handle, ier )
      if ( ier .ne. 0 ) goto 110
c
c  deallocate sparse solver storage
c
      call dgssda ( handle, ier )
      if ( ier .ne. 0 ) goto 110
c
c  print values of sol
c
      write(6,200) 'i', 'rhs(i)', 'expected rhs(i)', 'error'
      do i = 1, neqns
        write(6,300) i, rhs(i), xexpct(i), (rhs(i)-xexpct(i))
      enddo
      stop
  110 continue
```

CODE EXAMPLE 4-3 Solving a Structurally Symmetric System With Unsymmetric Values–Regular Interface *(Continued)*

```
c
c call to sparse solver returns an error
c
      write ( 6 , 400 )
     &     ' example: FAILED sparse solver error number = ', ier
      stop

  200 format(a5,3a20)

  300 format(i5,3d20.12)     ! i/sol/xexpct values

  400 format(a60,i20)   ! fail message, sparse solver error number

      end
my_system% f95 -dalign example_su.f -xlic_lib=sunperf
my_system% a.out
    i              rhs(i)     expected rhs(i)               error
    1  0.100000000000D+01  0.100000000000D+01  0.000000000000D+00
    2  0.200000000000D+01  0.200000000000D+01  0.000000000000D+00
    3  0.300000000000D+01  0.300000000000D+01  0.000000000000D+00
    4  0.400000000000D+01  0.400000000000D+01  0.000000000000D+00
```

CODE EXAMPLE 4-4 Solving an Unsymmetric System–Regular Interface

```
my_system% cat example_uu.f
      program example_uu
c
c  This program is an example driver that calls the sparse solver.
c    It factors and solves an unsymmetric system.
c
      implicit none

      integer           neqns, ier, msglvl, outunt, ldrhs, nrhs
      character         mtxtyp*2, pivot*1, ordmthd*3
      double precision  handle(150)
      integer           colstr(6), rowind(10)
      double precision  values(10), rhs(5), xexpct(5)
      integer           i
c
c  Sparse matrix structure and value arrays.  Unsummetric matrix A.
c    Ax = b, (solve for x) where:
c
c      1.0   0.0   0.0   0.0   0.0       1.0        1.0
c      2.0   6.0   0.0   0.0   9.0       2.0       59.0
c  A = 3.0   0.0   7.0   0.0   0.0   x = 3.0   b = 24.0
c      4.0   0.0   0.0   8.0   0.0       4.0       36.0
c      5.0   0.0   0.0   0.0  10.0       5.0       55.0
c
      data colstr / 1, 6, 7, 8, 9, 11 /
      data rowind / 1, 2, 3, 4, 5, 2, 3, 4, 2, 5 /
     data values / 1.0d0, 2.0d0, 3.0d0, 4.0d0, 5.0d0, 6.0d0, 7.0d0,
     &              8.0d0, 9.0d0, 10.0d0 /
      data rhs    / 1.0d0, 59.0d0, 24.0d0, 36.0d0, 55.0d0 /
      data xexpct / 1.0d0, 2.0d0, 3.0d0, 4.0d0, 5.0d0 /
c
c  initialize solver
c
      mtxtyp= 'uu'
      pivot = 'n'
      neqns  = 5
      outunt = 6
      msglvl = 3
      call dgssin ( mtxtyp, pivot,  neqns , colstr, rowind,
     &              outunt, msglvl, handle, ier             )
      if ( ier .ne. 0 ) goto 110
```

CODE EXAMPLE 4-4 Solving an Unsymmetric System–Regular Interface *(Continued)*

```
c
c  ordering and symbolic factorization
c
      ordmthd = 'mmd'
      call dgssor ( ordmthd, handle, ier )
      if ( ier .ne. 0 ) goto 110
c
c  numeric factorization
c
      call dgssfa ( neqns, colstr, rowind, values, handle, ier )
      if ( ier .ne. 0 ) goto 110
c
c  solution
c
      nrhs   = 1
      ldrhs  = 5
      call dgsssl ( nrhs, rhs, ldrhs, handle, ier )
      if ( ier .ne. 0 ) goto 110
c
c  deallocate sparse solver storage
c
      call dgssda ( handle, ier )
      if ( ier .ne. 0 ) goto 110
c
c  print values of sol
c
      write(6,200) 'i', 'rhs(i)', 'expected rhs(i)', 'error'
      do i = 1, neqns
        write(6,300) i, rhs(i), xexpct(i), (rhs(i)-xexpct(i))
      enddo
      stop
  110 continue
```

CODE EXAMPLE 4-4 Solving an Unsymmetric System–Regular Interface *(Continued)*

```
c
c call to sparse solver returns an error
c
      write ( 6 , 400 )
     &      ' example: FAILED sparse solver error number = ', ier
      stop

  200 format(a5,3a20)

  300 format(i5,3d20.12)     ! i/sol/xexpct values

  400 format(a60,i20)    ! fail message, sparse solver error number

      end
my_system% f95 -dalign example_uu.f -xlic_lib=sunperf
my_system% a.out
  i              rhs(i)     expected rhs(i)               error
    1  0.100000000000D+01  0.100000000000D+01  0.000000000000D+00
    2  0.200000000000D+01  0.200000000000D+01  0.000000000000D+00
    3  0.300000000000D+01  0.300000000000D+01  0.000000000000D+00
    4  0.400000000000D+01  0.400000000000D+01  0.000000000000D+00
    5  0.500000000000D+01  0.500000000000D+01  0.000000000000D+00
```

APPENDIX **A**

Sun Performance Library Routines

This appendix lists the Sun Performance Library routines by library, routine name, and function.

For a description of the function and a listing of the Fortran and C interfaces, refer to the section 3P man pages for the individual routines. For example, to display the man page for the `SBDSQR` routine, type **`man -s 3P sbdsqr`**. The man page routine names use lowercase letters.

For many routines, separate routines exist that operate on different data types. Rather than list each routine separately, a lowercase *x* is used in a routine name to denote single, double, complex, and double complex data types. For example, the routine *x*`BDSQR` is available as four routines that operate with the following data types:

- `SBDSQR` – Single data type
- `BBDSQR` – Double data type
- `CBDSQR` – Complex data type
- `ZBDSQR` – Double complex data type

If a routine name is not available for `S`, `B`, `C`, and `Z`, the *x* prefix will not be used and each routine name will be listed.

LAPACK Routines

TABLE A-1 LAPACK (Linear Algebra Package) Routines

Routine	Function
Bidiagonal Matrix	
`SBDSDC` or `DBDSDC`	Computes the singular value decomposition (SVD) of a bidirectional matrix, using a divide and conquer method.
x`BDSQR`	Computes SVD of real upper or lower bidiagonal matrix, using the bidirectional QR algorithm.
Diagonal Matrix	
`SDISNA` or `DDISNA`	Computes the reciprocal condition numbers for eigenvectors of real symmetric or complex Hermitian matrix.
General Band Matrix	
x`GBBRD`	Reduces real or complex general band matrix to upper bidiagonal form.
x`GBCON`	Estimates the reciprocal of the condition number of general band matrix using LU factorization.
x`GBEQU`	Computes row and column scalings to equilibrate a general band matrix and reduce its condition number.
x`GBRFS`	Refines solution to general banded system of linear equations.
x`GBSV`	Solves a general banded system of linear equations (simple driver).
x`GBSVX`	Solves a general banded system of linear equations (expert driver).
x`GBTRF`	LU factorization of a general band matrix using partial pivoting with row interchanges.
x`GBTRS`	Solves a general banded system of linear equations, using the factorization computed by *x*`GBTRF`.
General Matrix (Unsymmetric or Rectangular)	
x`GEBAK`	Forms the right or left eigenvectors of a general matrix by backward transformation on the computed eigenvectors of the balanced matrix output by *x*`GEBAL.`
x`GEBAL`	Balances a general matrix.
x`GEBRD`	Reduces a general matrix to upper or lower bidiagonal form by an orthogonal transformation.
x`GECON`	Estimates the reciprocal of the condition number of a general matrix, using the factorization computed by *x*`GETRF`.
x`GEEQU`	Computes row and column scalings intended to equilibrate a general rectangular matrix and reduce its condition number.

TABLE A-1 LAPACK (Linear Algebra Package) Routines *(Continued)*

Routine	Function
*x*GEES	Computes the eigenvalues and Schur factorization of a general matrix (simple driver).
*x*GEESX	Computes the eigenvalues and Schur factorization of a general matrix (expert driver).
*x*GEEV	Computes the eigenvalues and left and right eigenvectors of a general matrix (simple driver).
*x*GEEVX	Computes the eigenvalues and left and right eigenvectors of a general matrix (expert driver).
*x*GEGS	Depreciated routine replaced by *x*GGES.
*x*GEGV	Depreciated routine replaced by *x*GGEV.
*x*GEHRD	Reduces a general matrix to upper Hessenberg form by an orthogonal similarity transformation.
*x*GELQF	Computes LQ factorization of a general rectangular matrix.
*x*GELS	Computes the least squares solution to an over-determined system of linear equations using a QR or LQ factorization of A.
*x*GELSD	Computes the least squares solution to an over-determined system of linear equations using a divide and conquer method using a QR or LQ factorization of A.
*x*GELSS	Computes the minimum-norm solution to a linear least squares problem by using the SVD of a general rectangular matrix (simple driver).
*x*GELSX	Depreciated routine replaced by *x*SELSY.
*x*GELSY	Computes the minimum-norm solution to a linear least squares problem using a complete orthogonal factorization.
*x*GEQLF	Computes QL factorization of a general rectangular matrix.
*x*GEQP3	Computes QR factorization of general rectangular matrix using Level 3 BLAS.
*x*GEQPF	Depreciated routine replaced by *x*GEQP3.
*x*GEQRF	Computes QR factorization of a general rectangular matrix.
*x*GERFS	Refines solution to a system of linear equations.
*x*GERQF	Computes RQ factorization of a general rectangular matrix.
*x*GESDD	Computes SVD of general rectangular matrix using a divide and conquer method.
*x*GESV	Solves a general system of linear equations (simple driver).
*x*GESVX	Solves a general system of linear equations (expert driver).
*x*GESVD	Computes SVD of general rectangular matrix.

TABLE A-1 LAPACK (Linear Algebra Package) Routines *(Continued)*

Routine	Function
*x*GETRF	Computes an LU factorization of a general rectangular matrix using partial pivoting with row interchanges.
*x*GETRI	Computes inverse of a general matrix using the factorization computed by *x*GETRF.
*x*GETRS	Solves a general system of linear equations using the factorization computed by *x*GETRF.
General Matrix-Generalized Problem (Pair of General Matrices)	
*x*GGBAK	Forms the right or left eigenvectors of a generalized eigenvalue problem based on the output by *x*GGBAL.
*x*GGBAL	Balances a pair of general matrices for the generalized eigenvalue problem.
*x*GGES	Computes the generalized eigenvalues, Schur form, and left and/or right Schur vectors for two nonsymmetric matrices.
*x*GGESX	Computes the generalized eigenvalues, Schur form, and left and/or right Schur vectors.
*x*GGEV	Computes the generalized eigenvalues and the left and/or right generalized eigenvalues for two nonsymmetric matrices.
*x*GGEVX	Computes the generalized eigenvalues and the left and/or right generalized eigenvectors.
*x*GGGLM	Solves the GLM (Generalized Linear Regression Model) using the GQR (Generalized QR) factorization.
*x*GGHRD	Reduces two matrices to generalized upper Hessenberg form using orthogonal transformations.
*x*GGLSE	Solves the LSE (Constrained Linear Least Squares Problem) using the GRQ (Generalized RQ) factorization.
*x*GGQRF	Computes generalized QR factorization of two matrices.
*x*GGRQF	Computes generalized RQ factorization of two matrices.
*x*GGSVD	Computes the generalized singular value decomposition.
*x*GGSVP	Computes an orthogonal or unitary matrix as a preprocessing step for calculating the generalized singular value decomposition.
General Tridiagonal Matrix	
*x*GTCON	Estimates the reciprocal of the condition number of a tridiagonal matrix, using the LU factorization as computed by *x*GTTRF.
*x*GTRFS	Refines solution to a general tridiagonal system of linear equations.
*x*GTSV	Solves a general tridiagonal system of linear equations (simple driver).
*x*GTSVX	Solves a general tridiagonal system of linear equations (expert driver).

TABLE A-1 LAPACK (Linear Algebra Package) Routines *(Continued)*

Routine	Function
*x*GTTRF	Computes an LU factorization of a general tridiagonal matrix using partial pivoting and row exchanges.
*x*GTTRS	Solves general tridiagonal system of linear equations using the factorization computed by *x*.
Hermitian Band Matrix	
CHBEV or ZHBEV	(Replacement with newer version CHBEVD or ZHBEVD suggested) Computes all eigenvalues and eigenvectors of a Hermitian band matrix.
CHBEVD or ZHBEVD	Computes all eigenvalues and eigenvectors of a Hermitian band matrix and uses a divide and conquer method to calculate eigenvectors.
CHBEVX or ZHBEVX	Computes selected eigenvalues and eigenvectors of a Hermitian band matrix.
CHBGST or ZHBGST	Reduces Hermitian-definite banded generalized eigenproblem to standard form.
CHBGV or ZHBGV	(Replacement with newer version CHBGVD or ZHBGVD suggested) Computes all eigenvalues and eigenvectors of a generalized Hermitian-definite banded eigenproblem.
CHBGVD or ZHBGVD	Computes all eigenvalues and eigenvectors of generalized Hermitian-definite banded eigenproblem and uses a divide and conquer method to calculate eigenvectors.
CHBGVX or ZHBGVX	Computes selected eigenvalues and eigenvectors of a generalized Hermitian-definite banded eigenproblem.
CHBTRD or ZHBTRD	Reduces Hermitian band matrix to real symmetric tridiagonal form by using a unitary similarity transform.
Hermitian Matrix	
CHECON or ZHECON	Estimates the reciprocal of the condition number of a Hermitian matrix using the factorization computed by CHETRF or ZHETRF.
CHEEV or ZHEEV	(Replacement with newer version CHEEVR or ZHEEVR suggested) Computes all eigenvalues and eigenvectors of a Hermitian matrix (simple driver).
CHEEVD or ZHEEVD	(Replacement with newer version CHEEVR or ZHEEVR suggested) Computes all eigenvalues and eigenvectors of a Hermitian matrix and uses a divide and conquer method to calculate eigenvectors.
CHEEVR or ZHEEVR	Computes selected eigenvalues and the eigenvectors of a complex Hermitian matrix.
CHEEVX or ZHEEVX	Computes selected eigenvalues and eigenvectors of a Hermitian matrix (expert driver).
CHEGST or ZHEGST	Reduces a Hermitian-definite generalized eigenproblem to standard form using the factorization computed by CPOTRF or ZPOTRF.

TABLE A-1 LAPACK (Linear Algebra Package) Routines *(Continued)*

Routine	Function
CHEGV or ZHEGV	(Replacement with newer version CHEGVD or ZHEGVD suggested) Computes all the eigenvalues and eigenvectors of a complex generalized Hermitian-definite eigenproblem.
CHEGVD or ZHEGVD	Computes all the eigenvalues and eigenvectors of a complex generalized Hermitian-definite eigenproblem and uses a divide and conquer method to calculate eigenvectors.
CHEGVX or ZHEGVX	Computes selected eigenvalues and eigenvectors of a complex generalized Hermitian-definite eigenproblem.
CHERFS or ZHERFS	Improves the computed solution to a system of linear equations when the coefficient matrix is Hermitian indefinite.
CHESV or ZHESV	Solves a complex Hermitian indefinite system of linear equations (simple driver).
CHESVX or ZHESVX	Solves a complex Hermitian indefinite system of linear equations (simple driver).
CHETRD or ZHETRD	Reduces a Hermitian matrix to real symmetric tridiagonal form by using a unitary similarity transformation.
CHETRF or ZHERTF	Computes the factorization of a complex Hermitian indefinite matrix, using the diagonal pivoting method.
CHETRI or ZHETRI	Computes the inverse of a complex Hermitian indefinite matrix, using the factorization computed by CHETRF or ZHETRF.
CHETRS or ZHETRS	Solves a complex Hermitian indefinite matrix, using the factorization computed by CHETRF or ZHETRF.
Hermitian Matrix in Packed Storage	
CHPCON or ZHPCON	Estimates the reciprocal of the condition number of a Hermitian indefinite matrix in packed storage using the factorization computed by CHPTRF or ZHPTRF.
CHPEV or ZHPEV	(Replacement with newer version CHPEVD or ZHPEVD suggested) Computes all the eigenvalues and eigenvectors of a Hermitian matrix in packed storage (simple driver).
CHPEVX or ZHPEVX	Computes selected eigenvalues and eigenvectors of a Hermitian matrix in packed storage (expert driver).
CHPEVD or ZHPEVD	Computes all the eigenvalues and eigenvectors of a Hermitian matrix in packed storage and uses a divide and conquer method to calculate eigenvectors.
CHPGST or ZHPGST	Reduces a Hermitian-definite generalized eigenproblem to standard form where the coefficient matrices are in packed storage and uses the factorization computed by CPPTRF or ZPPTRF.

TABLE A-1 LAPACK (Linear Algebra Package) Routines *(Continued)*

Routine	Function
CHPGV or ZHPGV	(Replacement with newer version CHPGVD or ZHPGVD suggested) Computes all the eigenvalues and eigenvectors of a generalized Hermitian-definite eigenproblem where the coefficient matrices are in packed storage (simple driver).
CHPGVX or ZHPGVX	Computes selected eigenvalues and eigenvectors of a generalized Hermitian-definite eigenproblem where the coefficient matrices are in packed storage (expert driver).
CHPGVD or ZHPGVD	Computes all the eigenvalues and eigenvectors of a generalized Hermitian-definite eigenproblem where the coefficient matrices are in packed storage, and uses a divide and conquer method to calculate eigenvectors.
CHPRFS or ZHPRFS	Improves the computed solution to a system of linear equations when the coefficient matrix is Hermitian indefinite in packed storage.
CHPSV or ZHPSV	Computes the solution to a complex system of linear equations where the coefficient matrix is Hermitian in packed storage (simple driver).
CHPSVX or ZHPSVX	Uses the diagonal pivoting factorization to compute the solution to a complex system of linear equations where the coefficient matrix is Hermitian in packed storage (expert driver).
CHPTRD or ZHPTRD	Reduces a complex Hermitian matrix stored in packed form to real symmetric tridiagonal form.
CHPTRF or ZHPTRF	Computes the factorization of a complex Hermitian indefinite matrix in packed storage, using the diagonal pivoting method.
CHPTRI or ZHPTRI	Computes the inverse of a complex Hermitian indefinite matrix in packed storage using the factorization computed by CHPTRF or ZHPTRF.
CHPTRS or ZHPTRS	Solves a complex Hermitian indefinite matrix in packed storage, using the factorization computed by CHPTRF or ZHPTRF.
Upper Hessenberg Matrix	
*x*HSEIN	Computes right and/or left eigenvectors of upper Hessenberg matrix using inverse iteration.
*x*HSEQR	Computes eigenvectors and Shur factorization of upper Hessenberg matrix using multishift QR algorithm.
Upper Hessenberg Matrix-Generalized Problem (Hessenberg and Triangular Matrix)	
*x*HGEQZ	Implements single-/double-shift version of QZ method for finding the generalized eigenvalues of the equation det(A - w(i) * B) = 0.

TABLE A-1 LAPACK (Linear Algebra Package) Routines *(Continued)*

Routine	Function
Real Orthogonal Matrix in Packed Storage	
SOPGTR or DOPGTR	Generates an orthogonal transformation matrix from a tridiagonal matrix determined by SSPTRD or DSPTRD.
SOPMTR or DOPMTR	Multiplies a general matrix by the orthogonal transformation matrix reduced to tridiagonal form by SSPTRD or DSPTRD.
Real Orthogonal Matrix	
SORGBR or DORGBR	Generates the orthogonal transformation matrices from reduction to bidiagonal form, as determined by SGEBRD or DGEBRD.
SORGHR or DORGHR	Generates the orthogonal transformation matrix reduced to Hessenberg form, as determined by SGEHRD or DGEHRD.
SORGLQ or DORGLQ	Generates an orthogonal matrix Q from an LQ factorization, as returned by SGELQF or DGELQF.
SORGQL or DORGQL	Generates an orthogonal matrix Q from a QL factorization, as returned by SGEQLF or DGEQLF.
SORGQR or DORGQR	Generates an orthogonal matrix Q from a QR factorization, as returned by SGEQRF or DGEQRF.
SORGRQ or DORGRQ	Generates orthogonal matrix Q from an RQ factorization, as returned by SGERQF or DGERQF.
SORGTR or DORGTR	Generates an orthogonal matrix reduced to tridiagonal form by SSYTRD or DSYTRD.
SORMBR or DORMBR	Multiplies a general matrix with the orthogonal matrix reduced to bidiagonal form, as determined by SGEBRD or DGEBRD.
SORMHR or DORMHR	Multiplies a general matrix by the orthogonal matrix reduced to Hessenberg form by SGEHRD or DGEHRD.
SORMLQ or DORMLQ	Multiplies a general matrix by the orthogonal matrix from an LQ factorization, as returned by SGELQF or DGELQF.
SORMQL or DORMQL	Multiplies a general matrix by the orthogonal matrix from a QL factorization, as returned by SGEQLF or DGEQLF.
SORMQR or DORMQR	Multiplies a general matrix by the orthogonal matrix from a QR factorization, as returned by SGEQRF or DGEQRF.
SORMR3 or DORMR3	Multiplies a general matrix by the orthogonal matrix returned by STZRZF or DTZRZF.
SORMRQ or DORMRQ	Multiplies a general matrix by the orthogonal matrix from an RQ factorization returned by SGERQF or DGERQF.
SORMRZ or DORMRZ	Multiplies a general matrix by the orthogonal matrix from an RZ factorization, as returned by STZRZF or DTZRZF.

TABLE A-1 LAPACK (Linear Algebra Package) Routines *(Continued)*

Routine	Function
SORMTR or DORMTR	Multiplies a general matrix by the orthogonal transformation matrix reduced to tridiagonal form by SSYTRD or DSYTRD.
Symmetric or Hermitian Positive Definite Band Matrix	
*x*PBCON	Estimates the reciprocal of the condition number of a symmetric or Hermitian positive definite band matrix, using the Cholesky factorization returned by *x*PBTRF.
*x*PBEQU	Computes equilibration scale factors for a symmetric or Hermitian positive definite band matrix.
*x*PBRFS	Refines solution to a symmetric or Hermitian positive definite banded system of linear equations.
*x*PBSTF	Computes a split Cholesky factorization of a real symmetric positive definite band matrix.
*x*PBSV	Solves a symmetric or Hermitian positive definite banded system of linear equations (simple driver).
*x*PBSVX	Solves a symmetric or Hermitian positive definite banded system of linear equations (expert driver).
*x*PBTRF	Computes Cholesky factorization of a symmetric or Hermitian positive definite band matrix.
*x*PBTRS	Solves symmetric positive definite banded matrix, using the Cholesky factorization computed by *x*PBTRF.
Symmetric or Hermitian Positive Definite Matrix	
*x*POCON	Estimates the reciprocal of the condition number of a symmetric or Hermitian positive definite matrix, using the Cholesky factorization returned by *x*POTRF.
*x*POEQU	Computes equilibration scale factors for a symmetric or Hermitian positive definite matrix.
*x*PORFS	Refines solution to a linear system in a Cholesky-factored symmetric or Hermitian positive definite matrix.
*x*POSV	Solves a symmetric or Hermitian positive definite system of linear equations (simple driver).
*x*POSVX	Solves a symmetric or Hermitian positive definite system of linear equations (expert driver).
*x*POTRF	Computes Cholesky factorization of a symmetric or Hermitian positive definite matrix.
*x*POTRI	Computes the inverse of a symmetric or Hermitian positive definite matrix using the Cholesky-factorization returned by *x*POTRF.

TABLE A-1 LAPACK (Linear Algebra Package) Routines *(Continued)*

Routine	Function
*x*POTRS	Solves a symmetric or Hermitian positive definite system of linear equations, using the Cholesky factorization returned by *x*POTRF.
Symmetric or Hermitian Positive Definite Matrix in Packed Storage	
*x*PPCON	Reciprocal condition number of a Cholesky-factored symmetric positive definite matrix in packed storage.
*x*PPEQU	Computes equilibration scale factors for a symmetric or Hermitian positive definite matrix in packed storage.
*x*PPRFS	Refines solution to a linear system in a Cholesky-factored symmetric or Hermitian positive definite matrix in packed storage.
*x*PPSV	Solves a linear system in a symmetric or Hermitian positive definite matrix in packed storage (simple driver).
*x*PPSVX	Solves a linear system in a symmetric or Hermitian positive definite matrix in packed storage (expert driver).
*x*PPTRF	Computes Cholesky factorization of a symmetric or Hermitian positive definite matrix in packed storage.
*x*PPTRI	Computes the inverse of a symmetric or Hermitian positive definite matrix in packed storage using the Cholesky-factorization returned by *x*PPTRF.
*x*PPTRS	Solves a symmetric or Hermitian positive definite system of linear equations where the coefficient matrix is in packed storage, using the Cholesky factorization returned by *x*PPTRF.
Symmetric or Hermitian Positive Definite Tridiagonal Matrix	
*x*PTCON	Estimates the reciprocal of the condition number of a symmetric or Hermitian positive definite tridiagonal matrix using the Cholesky factorization returned by *x*PTTRF.
*x*PTEQR	Computes all eigenvectors and eigenvalues of a real symmetric or Hermitian positive definite system of linear equations.
*x*PTRFS	Refines solution to a symmetric or Hermitian positive definite tridiagonal system of linear equations.
*x*PTSV	Solves a symmetric or Hermitian positive definite tridiagonal system of linear equations (simple driver).
*x*PTSVX	Solves a symmetric or Hermitian positive definite tridiagonal system of linear equations (expert driver).
*x*PTTRF	Computes the LDL^H factorization of a symmetric or Hermitian positive definite tridiagonal matrix.
*x*PTTRS	Solves a symmetric or Hermitian positive definite tridiagonal system of linear equations using the LDL^H factorization returned by *x*PTTRF.

TABLE A-1 LAPACK (Linear Algebra Package) Routines *(Continued)*

Routine	Function
Real Symmetric Band Matrix	
SSBEV or DSBEV	(Replacement with newer version SSBEVD or DSBEVD suggested) Computes all eigenvalues and eigenvectors of a symmetric band matrix.
SSBEVD or DSBEVD	Computes all eigenvalues and eigenvectors of a symmetric band matrix and uses a divide and conquer method to calculate eigenvectors.
SSBEVX or DSBEVX	Computes selected eigenvalues and eigenvectors of a symmetric band matrix.
SSBGST or DSBGST	Reduces symmetric-definite banded generalized eigenproblem to standard form.
SSBGV or DSBGV	(Replacement with newer version SSBGVD or DSBGVD suggested) Computes all eigenvalues and eigenvectors of a generalized symmetric-definite banded eigenproblem.
SSBGVD or DSBGVD	Computes all eigenvalues and eigenvectors of generalized symmetric-definite banded eigenproblem and uses a divide and conquer method to calculate eigenvectors.
SSBGVX or DSBGVX	Computes selected eigenvalues and eigenvectors of a generalized symmetric-definite banded eigenproblem.
SSBTRD or DSBTRD	Reduces symmetric band matrix to real symmetric tridiagonal form by using an orthogonal similarity transform.
Symmetric Matrix in Packed Storage	
*x*SPCON	Estimates the reciprocal of the condition number of a symmetric packed matrix using the factorization computed by xSPTRF.
SSPEV or DSPEV	(Replacement with newer version SSPEVD or DSPEVD suggested) Computes all the eigenvalues and eigenvectors of a symmetric matrix in packed storage (simple driver).
SSPEVX or DSPEVX	Computes selected eigenvalues and eigenvectors of a symmetric matrix in packed storage (expert driver).
SSPEVD or DSPEVD	Computes all the eigenvalues and eigenvectors of a symmetric matrix in packed storage and uses a divide and conquer method to calculate eigenvectors.
SSPGST or DSPGST	Reduces a real symmetric-definite generalized eigenproblem to standard form where the coefficient matrices are in packed storage and uses the factorization computed by SPPTRF or DPPTRF.
SSPGVD or DSPGVD	Computes all the eigenvalues and eigenvectors of a real generalized symmetric-definite eigenproblem where the coefficient matrices are in packed storage, and uses a divide and conquer method to calculate eigenvectors.

TABLE A-1 LAPACK (Linear Algebra Package) Routines *(Continued)*

Routine	Function
SSPGV or DSPGV	(Replacement with newer version SSPGVD or DSPGVD suggested) Computes all the eigenvalues and eigenvectors of a real generalized symmetric-definite eigenproblem where the coefficient matrices are in packed storage (simple driver).
SSPGVX or DSPGVX	Computes selected eigenvalues and eigenvectors of a real generalized symmetric-definite eigenproblem where the coefficient matrices are in packed storage (expert driver).
*x*SPRFS	Improves the computed solution to a system of linear equations when the coefficient matrix is symmetric indefinite in packed storage.
*x*SPSV	Computes the solution to a system of linear equations where the coefficient matrix is a symmetric matrix in packed storage (simple driver).
*x*SPSVX	Uses the diagonal pivoting factorization to compute the solution to a system of linear equations where the coefficient matrix is a symmetric matrix in packed storage (expert driver).
SSPTRD or DSPTRD	Reduces a real symmetric matrix stored in packed form to real symmetric tridiagonal form using an orthogonal similarity transform.
*x*SPTRF	Computes the factorization of a symmetric packed matrix using the Bunch-Kaufman diagonal pivoting method.
*x*SPTRI	Computes the inverse of a symmetric indefinite matrix in packed storage using the factorization computed by xSPTRF.
*x*SPTRS	Solves a system of linear equations by the symmetric matrix stored in packed format using the factorization computed by xSPTRF.
Real Symmetric Tridiagonal Matrix	
SSTEBZ or DSTEBZ	Computes the eigenvalues of a real symmetric tridiagonal matrix.
*x*STEDC	Computes all the eigenvalues and eigenvectors of a symmetric tridiagonal matrix using a divide and conquer method.
*x*STEGR	Computes selected eigenvalues and eigenvectors of a real symmetric tridiagonal matrix using Relatively Robust Representations.
*x*STEIN	Computes selected eigenvectors of a real symmetric tridiagonal matrix using inverse iteration.
*x*STEQR	Computes all the eigenvalues and eigenvectors of a real symmetric tridiagonal matrix using the implicit QL or QR algorithm.
SSTERF or DSTERF	Computes all the eigenvalues and eigenvectors of a real symmetric tridiagonal matrix using a root-free QL or QR algorithm variant.
SSTEV or DSTEV	(Replacement with newer version SSTEVR or DSTEVR suggested) Computes all eigenvalues and eigenvectors of a real symmetric tridiagonal matrix (simple driver).

TABLE A-1 LAPACK (Linear Algebra Package) Routines *(Continued)*

Routine	Function
SSTEVX or DSTEVX	Computes selected eigenvalues and eigenvectors of a real symmetric tridiagonal matrix (expert driver).
SSTEVD or DSTEVD	(Replacement with newer version SSTEVR or DSTEVR suggested) Computes all the eigenvalues and eigenvectors of a real symmetric tridiagonal matrix using a divide and conquer method.
SSTEVR or DSTEVR	Computes selected eigenvalues and eigenvectors of a real symmetric tridiagonal matrix using Relatively Robust Representations.
*x*STSV	Computes the solution to a system of linear equations where the coefficient matrix is a symmetric tridiagonal matrix.
*x*STTRF	Computes the factorization of a symmetric tridiagonal matrix.
*x*STTRS	Computes the solution to a system of linear equations where the coefficient matrix is a symmetric tridiagonal matrix.
Symmetric Matrix	
*x*SYCON	Estimates the reciprocal of the condition number of a symmetric matrix using the factorization computed by SSYTRF or DSYTRF.
SSYEV or DSYEV	(Replacement with newer version SSYEVR or DSYEVR suggested) Computes all eigenvalues and eigenvectors of a symmetric matrix.
SSYEVX or DSYEVX	Computes eigenvalues and eigenvectors of a symmetric matrix (expert driver).
SSYEVD or DSYEVD	(Replacement with newer version SSYEVR or DSYEVR suggested) Computes all eigenvalues and eigenvectors of a symmetric matrix and uses a divide and conquer method to calculate eigenvectors.
SSYEVR or DSYEVR	Computes selected eigenvalues and eigenvectors of a symmetric tridiagonal matrix.
SSYGST or DSYGST	Reduces a symmetric-definite generalized eigenproblem to standard form using the factorization computed by SPOTRF or DPOTRF.
SSYGV or DSYGV	(Replacement with newer version SSYGVD or DSYGVD suggested) Computes all the eigenvalues and eigenvectors of a generalized symmetric-definite eigenproblem.
SSYGVX or DSYGVX	Computes selected eigenvalues and eigenvectors of a generalized symmetric-definite eigenproblem.
SSYGVD or DSYGVD	Computes all the eigenvalues and eigenvectors of a generalized symmetric-definite eigenproblem and uses a divide and conquer method to calculate eigenvectors.
*x*SYRFS	Improves the computed solution to a system of linear equations when the coefficient matrix is symmetric indefinite.
*x*SYSV	Solves a real symmetric indefinite system of linear equations (simple driver).

TABLE A-1 LAPACK (Linear Algebra Package) Routines *(Continued)*

Routine	Function
*x*SYSVX	Solves a real symmetric indefinite system of linear equations (expert driver).
SSYTRD or DSYTRD	Reduces a symmetric matrix to real symmetric tridiagonal form by using a orthogonal similarity transformation.
*x*SYTRF	Computes the factorization of a real symmetric indefinite matrix using the diagonal pivoting method.
*x*SYTRI	Computes the inverse of a symmetric indefinite matrix using the factorization computed by *x*SYTRF.
*x*SYTRS	Solves a system of linear equations by the symmetric matrix using the factorization computed by *x*SYTRF.
Triangular Band Matrix	
*x*TBCON	Estimates the reciprocal condition number of a triangular band matrix.
*x*TBRFS	Determines error bounds and estimates for solving a triangular banded system of linear equations.
*x*TBTRS	Solves a triangular banded system of linear equations.
Triangular Matrix-Generalized Problem (Pair of Triangular Matrices)	
*x*TGEVC	Computes right and/or left generalized eigenvectors of two upper triangular matrices.
*x*TGEXC	Reorders the generalized Schur decomposition of a real or complex matrix pair using an orthogonal or unitary equivalence transformation.
*x*TGSEN	Reorders the generalized real-Schur or Schur decomposition of two matrixes and computes the generalized eigenvalues.
*x*TGSJA	Computes the generalized SVD from two upper triangular matrices obtained from *x*GGSVP.
*x*TGSNA	Estimates reciprocal condition numbers for specified eigenvalues and eigenvectors of two matrices in real-Schur or Schur canonical form.
*x*TGSYL	Solves the generalized Sylvester equation.
Triangular Matrix in Packed Storage	
*x*TPCON	Estimates the reciprocal or the condition number of a triangular matrix in packed storage.
*x*TPRFS	Determines error bounds and estimates for solving a triangular system of linear equations where the coefficient matrix is in packed storage.
*x*TPTRI	Computes the inverse of a triangular matrix in packed storage.
*x*TPTRS	Solves a triangular system of linear equations where the coefficient matrix is in packed storage.

TABLE A-1 LAPACK (Linear Algebra Package) Routines *(Continued)*

Routine	Function
Triangular Matrix	
*x*TRCON	Estimates the reciprocal or the condition number of a triangular matrix.
*x*TREVC	Computes right and/or left eigenvectors of an upper triangular matrix.
*x*TREXC	Reorders Schur factorization of matrix using an orthogonal or unitary similarity transformation.
*x*TRRFS	Determines error bounds and estimates for triangular system of a linear equations.
*x*TRSEN	Reorders Schur factorization of matrix to group selected cluster of eigenvalues in the leading positions on the diagonal of the upper triangular matrix T and the leading columns of Q form an orthonormal basis of the corresponding right invariant subspace.
*x*TRSNA	Estimates the reciprocal condition numbers of selected eigenvalues and eigenvectors of an upper quasi-triangular matrix.
*x*TRSYL	Solves Sylvester matrix equation.
*x*TRTRI	Computes the inverse of a triangular matrix.
*x*TRTRS	Solves a triangular system of linear equations.
Trapezoidal Matrix	
*x*TZRQF	Depreciated routine replaced by routine *x*TZRZF.
*x*TZRZF	Reduces a rectangular upper trapezoidal matrix to upper triangular form by means of orthogonal transformations.
Unitary Matrix	
CUNGBR or ZUNGBR	Generates the unitary transformation matrices from reduction to bidiagonal form, as determined by CGEBRD or ZGEBRD.
CUNGHR or ZUNGHR	Generates the orthogonal transformation matrix reduced to Hessenberg form, as determined by CGEHRD or ZGEHRD.
CUNGLQ or ZUNGLQ	Generates a unitary matrix Q from an LQ factorization, as returned by CGELQF or ZGELQF.
CUNGQL or ZUNGQL	Generates a unitary matrix Q from a QL factorization, as returned by CGEQLF or ZGEQLF.
CUNGQR or ZUNGQR	Generates a unitary matrix Q from a QR factorization, as returned by CGEQRF or ZGEQRF.
CUNGRQ or ZUNGRQ	Generates a unitary matrix Q from an RQ factorization, as returned by CGERQF or ZGERQF.
CUNGTR or ZUNGTR	Generates a unitary matrix reduced to tridiagonal form, by CHETRD or ZHETRD.

TABLE A-1 LAPACK (Linear Algebra Package) Routines *(Continued)*

Routine	Function
CUNMBR or ZUNMBR	Multiplies a general matrix with the unitary transformation matrix reduced to bidiagonal form, as determined by CGEBRD or ZGEBRD.
CUNMHR or ZUNMHR	Multiplies a general matrix by the unitary matrix reduced to Hessenberg form by CGEHRD or ZGEHRD.
CUNMLQ or ZUNMLQ	Multiplies a general matrix by the unitary matrix from an LQ factorization, as returned by CGELQF or ZGELQF.
CUNMQL or ZUNMQL	Multiplies a general matrix by the unitary matrix from a QL factorization, as returned by CGEQLF or ZGEQLF.
CUNMQR or ZUNMQR	Multiplies a general matrix by the unitary matrix from a QR factorization, as returned by CGEQRF or ZGEQRF.
CUNMRQ or ZUNMRQ	Multiplies a general matrix by the unitary matrix from an RQ factorization, as returned by CGERQF or ZGERQF.
CUNMRZ or ZUNMRZ	Multiplies a general matrix by the unitary matrix from an RZ factorization, as returned by CTZRZF or ZTZRZF.
CUNMTR or ZUNMTR	Multiplies a general matrix by the unitary transformation matrix reduced to tridiagonal form by CHETRD or ZHETRD.
Unitary Matrix in Packed Storage	
CUPGTR or ZUPGTR	Generates the unitary transformation matrix from a tridiagonal matrix determined by CHPTRD or ZHPTRD.
CUPMTR or ZUPMTR	Multiplies a general matrix by the unitary transformation matrix reduced to tridiagonal form by CHPTRD or ZHPTRD.

BLAS1 Routines

TABLE A-2 BLAS1 (Basic Linear Algebra Subprograms, Level 1) Routines

Routine	Function
SASUM, DASUM, SCASUM, DZASUM	Sum of the absolute values of a vector
xAXPY	Product of a scalar and vector plus a vector
xCOPY	Copy a vector
SDOT, DDOT, DSDOT, SDSDOT, CDOTU, ZDOTU, DQDOTA, DQDOTI	Dot product (inner product)

TABLE A-2 BLAS1 (Basic Linear Algebra Subprograms, Level 1) Routines *(Continued)*

Routine	Function
`CDOTC, ZDOTC`	Dot product conjugating first vector
`SNRM2, DNRM2, SCNRM2, DCNRM2, DZNRM2`	Euclidean norm of a vector
`xROTG`	Set up Givens plane rotation
`xROT, CSROT, ZDROT`	Apply Given's plane rotation
`SROTMG, DROTMG`	Set up modified Given's plane rotation
`SROTM, DROTM`	Apply modified Given's rotation
`ISAMAX, DAMAX, ICAMAX, IZAMAX`	Index of element with maximum absolute value
`xSCAL, CSSCAL, ZDSCAL`	Scale a vector
`xSWAP`	Swap two vectors
`CVMUL, ZVMUL`	Compute scaled product of complex vectors

BLAS2 Routines

TABLE A-3 BLAS2 (Basic Linear Algebra Subprograms, Level 2) Routines

Routine	Function
x`GBMV`	Product of a matrix in banded storage and a vector
x`GEMV`	Product of a general matrix and a vector
`SGER, DGER, CGERC, ZGERC, CGERU, ZGERU`	Rank-1 update to a general matrix
`CHBMV, ZHBMV`	Product of a Hermitian matrix in banded storage and a vector
`CHEMV, ZHEMV`	Product of a Hermitian matrix and a vector
`CHER, ZHER`	Rank-1 update to a Hermitian matrix
`CHER2, ZHER2`	Rank-2 update to a Hermitian matrix
`CHPMV, ZHPMV`	Product of a Hermitian matrix in packed storage and a vector

TABLE A-3 BLAS2 (Basic Linear Algebra Subprograms, Level 2) Routines *(Continued)*

Routine	Function
`CHPR, ZHPR`	Rank-1 update to a Hermitian matrix in packed storage
`CHPR2, ZHPR2`	Rank-2 update to a Hermitian matrix in packed storage
`SSBMV, DSBMV`	Product of a symmetric matrix in banded storage and a vector
x`SPMV`	Product of a Symmetric matrix in packed storage and a vector
`SSPR, DSPR`	Rank-1 update to a real symmetric matrix in packed storage
`SSPR2, DSPR2`	Rank-2 update to a real symmetric matrix in packed storage
`SSYMV, DSYMV`	Product of a symmetric matrix and a vector
`SSYR, DSYR`	Rank-1 update to a real symmetric matrix
`SSYR2, DSYR2`	Rank-2 update to a real symmetric matrix
x`TBMV`	Product of a triangular matrix in banded storage and a vector
x`TBSV`	Solution to a triangular system in banded storage of linear equations
x`TPMV`	Product of a triangular matrix in packed storage and a vector
x`TPSV`	Solution to a triangular system of linear equations in packed storage
x`TRMV`	Product of a triangular matrix and a vector
x`TRSV`	Solution to a triangular system of linear equations

BLAS3 Routines

TABLE A-4 BLAS3 (Basic Linear Algebra Subprograms, Level 3) Routines

Routine	Function
x`GEMM`	Product of two general matrices
`CHEMM` or `ZHEMM`	Product of a Hermitian matrix and a general matrix
`CHERK` or `ZHERK`	Rank-k update of a Hermitian matrix
`CHER2K` or `ZHER2K`	Rank-2k update of a Hermitian matrix
x`SYMM`	Product of a symmetric matrix and a general matrix

TABLE A-4 BLAS3 (Basic Linear Algebra Subprograms, Level 3) Routines *(Continued)*

Routine	Function
*x*SYRK	Rank-k update of a symmetric matrix
*x*SYR2K	Rank-2k update of a symmetric matrix
*x*TRMM	Product of a triangular matrix and a general matrix
*x*TRSM	Solution for a triangular system of equations

Sparse BLAS Routines

TABLE A-5 Sparse BLAS Routines

Routines	Function
*x*AXPYI	Adds a scalar multiple of a sparse vector *X* to a full vector *Y*.
SBCOMM or DBCOMM	Block coordinate matrix-matrix multiply.
SBDIMM or DBDIMM	Block diagonal format matrix-matrix multiply.
SBDISM or DBDISM	Block Diagonal format triangular solve.
SBELMM or DBELMM	Block Ellpack format matrix-matrix multiply.
SBELSM or DBELSM	Block Ellpack format triangular solve.
SBSCMM or DBSCMM	Block compressed sparse column format matrix-matrix multiply.
SBSCSM or DBSCSM	Block compressed sparse column format triangular solve.
SBSRMM or DBSRMM	Block compressed sparse row format matrix-matrix multiply.
SBSRSM or DBSRSM	Block compressed sparse row format triangular solve.
SCOOMM or DCOOMM	Coordinate format matrix-matrix multiply.
SCSCMM or DCSCMM	Compressed sparse column format matrix-matrix multiply
SCSCSM or DCSCSM	Compressed sparse column format triangular solve

TABLE A-5 Sparse BLAS Routines *(Continued)*

Routines	Function
`SCSRMM` or `DCSRMM`	Compressed sparse row format matrix-matrix multiply.
`SCSRSM` or `DCSRSM`	Compressed sparse row format triangular solve.
`SDIAMM` or `DDIAMM`	Diagonal format matrix-matrix multiply.
`SDIASM` or `DDIASM`	Diagonal format triangular solve.
`SDOTI`, `DDOTI`, `CDOTUI`, or `ZDOTUI`	Computes the dot product of a sparse vector and a full vector.
`CDOTCI`, or `ZDOTCI`,	Computes the conjugate dot product of a sparse vector and a full vector.
`SELLMM` or `DELLMM`	Ellpack format matrix-matrix multiply.
`SELLSM` or `DELLSM`	Ellpack format triangular solve.
x`CGTHR`	Given a full vector, creates a sparse vector and corresponding index vector.
x`CGTHRZ`	Given a full vector, creates a sparse vector and corresponding index vector and zeros the full vector.
`SJADMM` or `DJADMM`	Jagged diagonal matrix-matrix multiply.
`SJADRP` or `DJADRP`	Right permutation of a jagged diagonal matrix.
`SJADSM` or `DJADSM`	Jagged diagonal triangular solve.
`SROTI` or `DROTI`	Applies a Givens rotation to a sparse vector and a full vector.
x`CSCTR`	Given a sparse vector and corresponding index vector, puts those elements into a full vector.
`SSKYMM` or `DSKYMM`	Skyline format matrix-matrix multiply.

TABLE A-5 Sparse BLAS Routines *(Continued)*

Routines	Function
SSKYSM or DSKYSM	Skyline format triangular solve.
SVBRMM or DVBRMM	Variable block sparse row format matrix-matrix multiply.
SVBRSM or DVBRSM	Variable block sparse row format triangular solve.

Sparse Solver Routines

TABLE A-6 Sparse Solver Routines

Routines	Function
DGSSFS	One call interface to sparse solver.
DGSSIN	Sparse solver initialization.
DGSSOR	Fill reducing ordering and symbolic factorization.
DGSSFA	Matrix value input and numeric factorization.
DGSSSL	Triangular solve.
DGSSUO	Sets user-specified ordering permutation.
DGSSRP	Returns permutation used by solver.
DGSSCO	Returns condition number estimate of coefficient matrix.
DGSSDA	De-allocates sparse solver.
DGSSPS	Prints solver statistics.

FFTPACK and VFFTPACK Routines

Routines with a V prefix are vectorized routines that belong to VFFTPACK.

TABLE A-7 FFTPACK and VFFTPACK (Fast Fourier Transform and Vectorized Fast Fourier Transform) Routines

Routine	Function
`COSQB, DCOSQB, VCOSQB, VDCOSQB`	Cosine quarter-wave synthesis
`COSQF, DCOSQF, VCOSQF, VDCOSQF`	Cosine quarter-wave transform
`COSQI, DCOSQI, VCOSQI, VDCOSQI`	Initialize cosine quarter-wave transform and synthesis
`COST, DCOST, VCOST, VDCOST`	Cosine even-wave transform
`COSTI, DCOSTI, VCOSTI, VDCOSTI`	Initialize cosine even-wave transform
`EZFFTB`	EZ Fourier synthesis
`EZFFTF`	EZ Fourier transform
`EZFFTI`	Initialize EZ Fourier transform and synthesis
`RFFTB, DFFTB, CFFTB, ZFFTB, VRFFTB, VDFFTB, VCFFTB, VZFFTB`	Fourier synthesis
`RFFTF, DFFTF, CFFTF, ZFFTF, VRFFTF, VDFFTF, VCFFTF, VZFFTF`	Fourier transform
`RFFTI, DFFTI, CFFTI, ZFFTI, VRFFTI, VDFFTI, VCFFTI, VZFFTI`	Initialize Fourier transform and synthesis
`SINQB, DSINQB, VSINQB, VDSINQB`	Sine quarter-wave synthesis
`SINQF, DSINQF, VSINQF, VDSINQF`	Sine quarter-wave transform
`SINQI, DSINQI, VSINQI, VDSINQI`	Initialize sine quarter-wave transform and synthesis

TABLE A-7 FFTPACK and VFFTPACK (Fast Fourier Transform and Vectorized Fast Fourier Transform) Routines *(Continued)*

Routine	Function
`SINT, DSINT, VSINT, VDSINT`	Sine odd-wave transform
`SINTI, DSINT, VSINTI, VDSINTI`	Initialize sine odd-wave transform
`RFFT2B, DFFT2B, CFFT2B, ZFFT2B`	Two-dimensional Fourier synthesis
`RFFT2F, DFFT2F, CFFT2F, ZFFT2F`	Two-dimensional Fourier transform
`RFFT2I, DFFT2I, CFFT2I, ZFFT2I`	Initialize two-dimensional Fourier transform or synthesis
`RFFT3B, DFFT3B, CFFT3B, ZFFT3B`	Three-dimensional Fourier synthesis
`RFFT3F, DFFT3F, CFFT3F, DFFT3F`	Three-dimensional Fourier transform
`RFFT3I, DFFT3I, CFFT3I, DFFT3I`	Initialize three-dimensional Fourier transform or synthesis

Other Routines

TABLE A-8 Other Routines

Routines	Function
x`CNVCOR`	Computes convolution or correlation
x`CNVCOR2`	Computes two-dimensional convolution or correlation
x`TRANS`	Transposes array
`SWIENER` or `DWEINER`	Performs Wiener deconvolution of two signals

LINPACK Routines

TABLE A-9 LINPACK Routines

Routine	Function
*x*CHDC	Cholesky decomposition of a symmetric positive definite matrix
*x*CHDD	Downdate an augmented Cholesky decomposition
*x*CHEX	Update an augmented Cholesky decomposition with permutations
*x*CHUD	Update an augmented Cholesky decomposition
*x*GBCO	LU Factorization and condition number of a general matrix in banded storage
*x*GBDI	Determinant of an LU-factored general matrix in banded storage
*x*GBFA	LU factorization of a general matrix in banded storage
*x*GBSL	Solution to a linear system in an LU-factored matrix in banded storage
*x*GECO	LU factorization and condition number of a general matrix
*x*GEDI	Determinant and inverse of an LU-factored general matrix
*x*GEFA	LU factorization of a general matrix
*x*GESL	Solution to a linear system in an LU-factored general matrix
*x*GTSL	Solution to a linear system in a tridiagonal matrix
CHICO or ZHICO	UDU factorization and condition number of a Hermitian matrix
CHIDI or ZHIDI	Determinant, inertia, and inverse of a UDU-factored Hermitian matrix
CHIFA or ZHIFA	UDU factorization of a Hermitian matrix
CHISL or ZHISL	Solution to a linear system in a UDU-factored Hermitian matrix
CHPCO or ZHPCO	UDU factorization and condition number of a Hermitian matrix in packed storage
CHPDI or ZHPDI	Determinant, inertia, and inverse of a UDU-factored Hermitian matrix in packed storage
CHPFA or ZHPFA	UDU factorization of a Hermitian matrix in packed storage
CHPSL or ZHPSL	Solution to a linear system in a UDU-factored Hermitian matrix in packed storage
*x*PBCO	Cholesky factorization and condition number of a symmetric positive definite matrix in banded storage

TABLE A-9 LINPACK Routines *(Continued)*

Routine	Function
*x*PBDI	Determinant of a Cholesky-factored symmetric positive definite matrix in banded storage
*x*PBFA	Cholesky factorization of a symmetric positive definite matrix in banded storage
*x*PBSL	Solution to a linear system in a Cholesky-factored symmetric positive definite matrix in banded storage
*x*POCO	Cholesky factorization and condition number of a symmetric positive definite matrix
*x*PODI	Determinant and inverse of a Cholesky-factored symmetric positive definite matrix
*x*POFA	Cholesky factorization of a symmetric positive definite matrix
*x*POSL	Solution to a linear system in a Cholesky-factored symmetric positive definite matrix
*x*PPCO	Cholesky factorization and condition number of a symmetric positive definite matrix in packed storage
*x*PPDI	Determinant and inverse of a Cholesky-factored symmetric positive definite matrix in packed storage
*x*PPFA	Cholesky factorization of a symmetric positive definite matrix in packed storage
*x*PPSL	Solution to a linear system in a Cholesky-factored symmetric positive definite matrix in packed storage
*x*PTSL	Solution to a linear system in a symmetric positive definite tridiagonal matrix
*x*QRDC	QR factorization of a general matrix
*x*QRSL	Solution to a linear system in a QR-factored general matrix
*x*SICO	UDU factorization and condition number of a symmetric matrix
*x*SIDI	Determinant, inertia, and inverse of a UDU-factored symmetric matrix
*x*SIFA	UDU factorization of a symmetric matrix
*x*SISL	Solution to a linear system in a UDU-factored symmetric matrix
*x*SPCO	UDU factorization and condition number of a symmetric matrix in packed storage
*x*SPDI	Determinant, inertia, and inverse of a UDU-factored symmetric matrix in packed storage
*x*SPFA	UDU factorization of a symmetric matrix in packed storage

TABLE A-9 LINPACK Routines *(Continued)*

Routine	Function
x`SPSL`	Solution to a linear system in a UDU-factored symmetric matrix in packed storage
x`SVDC`	Singular value decomposition of a general matrix
x`TRCO`	Condition number of a triangular matrix
x`TRDI`	Determinant and inverse of a triangular matrix
x`TRSL`	Solution to a linear system in a triangular matrix

Index