



An Oracle White Paper  
June 2010

# Oracle<sup>®</sup> Solaris Studio: Using the Tuned Libraries for Peak Application Performance

Introduction .....	1
Capabilities of Perflib .....	2
Features for Shared Memory and Cluster Systems .....	2
Features Only For Shared Memory Systems .....	2
Faster Applications with Perflib .....	3
Improve Performance Without Making Code Changes .....	3
Include Perflib in the Development Environment .....	3
Use Tools to Restructure Code .....	4
Getting Peak Performance with Perflib .....	4
Uniprocessors and Shared Memory Multiprocessors .....	4
Distributed Memory Parallel Processors and Clusters .....	6
C Interfaces in Perflib .....	7
Direct Support for C .....	7
Getting Peak Performance Using Perflib with C .....	8
Getting Peak Performance Using Perflib with User Threads .....	9
Conclusion .....	9
References .....	10
Appendix A: Performance Examples .....	11
Appendix B: ScaLAPACK Example .....	13
Appendix C: BLAS List .....	16

## Introduction

Oracle Solaris Studio provides high performance C, C++, and Fortran compilers and tools for Oracle Solaris and Linux, including high-performance mathematical libraries that are critical to scientific, engineering, and financial applications.

This technical white paper discusses the Sun Performance Library (Perflib) component of Oracle Solaris Studio, and demonstrates how it can be used in applications to boost performance significantly via its highly optimized mathematical subroutines for SPARC® and x86 processor-based systems. Perflib is available for SPARC systems running Oracle Solaris and for x86 or x64 (Intel® or AMD™) systems running Oracle Solaris or Oracle Linux.

Due to its very complex nature, the creation of highly accurate, performant, and portable mathematical routines has been a joint effort within the numerical community for many years. Public versions of LINPACK, EISPACK, SPARSPAK, FFTPACK, MINPACK, BLAS, LAPACK, and more are available. With Perflib Oracle provides very highly tuned versions of important routines, taking maximum advantage of supported platform architectures in order to achieve optimal performance.

Additional Perflib information is available in the *Perflib User Guide* located at <http://docs.sun.com/app/docs/doc/821-0276>, and the *ScaLAPACK User Guide* located at <http://www.netlib.org/scalapack/slug/index.html>.

## Capabilities of Perflib

Perflib is a collection of high-speed mathematical subroutines and functions. They are callable from Fortran, C, and C++ programs, and contain Oracle's optimized implementations of the routines in

- BLAS1, BLAS2, BLAS3
- LAPACK
- ScaLAPACK
- PBLAS
- BLACS
- FFTPACK
- VFFTPACK
- SPSOLVE
- Sparse BLAS
- SuperLU

Additionally, a selection of sorting, convolution, and correlation functions is available.

Note that ScaLAPACK, PBLAS, and BLACS provide parallelism across a cluster of compute nodes and thus utilize the Message Passing Interface (MPI) for communication. The Oracle Message Passing Toolkit provides the MPI implementation needed for these libraries. See <http://www.sun.com/clustertools> for more information.

## Features for Shared Memory and Cluster Systems

- **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, LDLT, and UDUT
- **Support operations.** Condition number, in-place or out-of-place transpose, inverse, determinant, inertia

## Features Only For Shared Memory Systems

- **Sparse matrices.** Solve symmetric, structurally symmetric, and non-symmetric coefficient matrices using direct methods and a choice of fill-reducing ordering algorithms and user-specified orderings
- **Convolution and Correlation.** Convolution and correlation in one and two dimensions, Wiener deconvolution, array transpose

- **Fast Fourier Transforms.** Complex and Real FFTs in one, two and three dimensions, length of closest FFT, Fourier synthesis, cosine and 1/4-wave cosine transforms, sine and 1/4-wave sine transforms
- **Sorting.** Sorting of numerical elements of a vector

In addition to the standard subroutines with the capabilities listed above, Perflib contains C interfaces to all of the user-callable subroutines in the library to ease the development of numerical applications for C programmers. The capabilities of these interfaces are described the *C Interfaces in Perflib* section.

Perflib is available in both static and dynamic library versions, optimized for systems that use the SPARC and x86/x64 families of processor architectures. Perflib takes advantage of multicore or multiprocessor systems via parallel coding of select functions by using OpenMP or MPI. It can be used with programs that utilize Oracle Solaris threads, POSIX threads, OpenMP or MPI.

Note that when a function in Perflib is called from an OpenMP parallel region it executes in serial mode, although the library routine might have been parallelized. This avoids issues with nested parallelism, such as over commitment of resources or other performance overhead.

## Faster Applications with Perflib

This section discusses techniques that can help an application to achieve peak performance. For example, applications can benefit significantly by simply specifying at link time that Perflib should be used. This requires no change to the source code, or recompiling of the application. Additionally, it might be possible for an application to gain even more performance benefits with Perflib by making modifications to the application.

### Improve Performance Without Making Code Changes

Because Perflib maintains the same interfaces and function of the original Netlib libraries, user codes can automatically gain significant speed just by switching from the Netlib distribution to Perflib. In this case, use the `-library=sunperf` option when linking. For example:

```
{f95,cc,C} ... my_file -library=sunperf    (for dynamic linking)
or
{f95,cc,C} ... my_file -library=sunperf  -staticlib=sunperf (for static linking)
```

### Include Perflib in the Development Environment

For many users, subroutines in Perflib can be much faster than subroutines that perform similar functions in other libraries. For example, at the time of its introduction using Perflib on an Oracle Sun SPARC Enterprise M9000 server, the LINPACK benchmark ran in excess of 2 Teraflop/second. That was the best performance achievable on any shared memory computer at that time.

Perflib includes many key kernel routines tuned to achieve significant performance improvements on a single core. These serial enhancements automatically translate into a performance boost in multicore,

shared memory, and distributed memory parallel environments. For many routines, Perflib contains parallel algorithms to further increase parallel performance.

Perflib also provides a Fortran module for additional ease-of-use features with Fortran 95 programs. To use this module, include the following `USE` line in Fortran 95 programs: `USE sunperf`. This Perflib module contains interfaces that simplify the calling sequences and provides type independence, compile-time checking of arguments, optional arguments, and 64-bit integer support. See the Fortran User's Guide located at <http://docs.sun.com/app/docs/doc/820-7600> for more information.

Another benefit for Fortran users is the Oracle Solaris Studio Fortran option `-xknownlib=liblist`. If `liblist` is `blas`, `blas1`, `blas2`, or `blas3`, the compiler recognizes calls to the BLAS1, BLAS2, and BLAS3 library routines listed in Appendix C, and is free to optimize appropriately. The compiler ignores user-supplied versions of these library routines and links to the BLAS routines in Perflib.

If `-xknownlib=intrinsics` is specified, the compiler ignores any explicit `EXTERNAL` declarations for Fortran intrinsics, thereby ignoring any user-supplied intrinsic routines. See the Fortran Library Reference for lists of intrinsic function names.

Note that keywords can be combined, such as `-xknownlib=blas1,blas2,blas3,intrinsics`.

## Use Tools to Restructure Code

In some cases, other libraries might not directly use the subroutines in Perflib, however conversion aids might be available. For example, since Perflib implements all of the LAPACK calls, users of EISPACK can use the conversion chart in the LAPACK User's Manual that shows how to convert from EISPACK calls to the LAPACK calls in Perflib. Users of IBM's ESSL can find information at <http://www.netlib.org/lapack/essl> that can help convert programs from ESSL to Perflib. Users of other libraries may be able to find similar resources to aid in the conversion process.

## Getting Peak Performance with Perflib

Getting peak performance from Perflib for single core applications is simply a matter of identifying code constructs in an application that can be replaced by calls to subroutines in Perflib.

### Uniprocessors and Shared Memory Multiprocessors

Applications can gain additional speed by identifying opportunities for parallelization. Examples of these opportunities are shown in this section. The easiest situation occurs when a block of user code exactly duplicates a capability of Perflib. Consider the code below.

```

DO 20, I = 1, N
  DO 10, J = 1, N
    Y(I) = Y(I) + A(I,J) * X(J)
20  CONTINUE
10  CONTINUE

```

This is simply the matrix-vector product  $y = Ax + y$ , which can be performed with the BLAS2 DGEMV subroutine.

```
CALL DGEMV('N',N,N,1.,A,N,X,1,1.,Y,1)
```

In other cases, a block of code may be equivalent to several Perflib calls, or may contain a mixture of code that can be replaced together with code that has no natural replacement in Perflib. Consider the code below, adapted from DYNA3D by the Methods Development Group at Lawrence Livermore National Laboratory.

```
DO 20, I = 1, N
  IF (V2(I,K) .LT. 0.0) THEN
    V2(I,K) = 0.0
  ELSE
    DO 10, J = 1, M
      X(J,I) = X(J,I) + V1(J,K) * V2(I,K)
10    CONTINUE
    END IF
20 CONTINUE
```

One way to rewrite this code with Perflib is shown below.

```
DO 10, I = 1, N
  IF (V2(I,K) .LT. 0.0) THEN
    V2(I,K) = 0.0
  END IF
10 CONTINUE
  CALL DGER (M, N, 1.0D0, X, LDX, V1(1,K), 1, V2(1,K), 1)
```

The code to replace negative numbers with zero in V2 has no natural analog in Perflib. That code is simply pulled out of the outer loop. With that code moved to its own loop, the rest of the loop could be recognized as being a rank-1 update of the general matrix X, which can be accomplished using the DGER subroutine from BLAS.

Note that the benefit of this optimization depends on the number of negative or zero values in array V2. If there are many such values in V2, it may be that the majority of the time is not spent in the rank-1 update. In that case, replacing the code with the call to DGER may not result in a large payoff.

Although there is no direct analog to DGER in the SPARSE components of Perflib, when substantial computation with sparse matrices exists it can be worthwhile to rewrite the code to use the SPARSE routines.

Note that it may be worthwhile to track down 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. Loops over DGEMV or DGER often can be converted to some form of matrix multiplication. If so, a single call to a matrix multiplication subroutine will probably bring a much larger payoff than a loop over calls to DGER.

All Perflib subroutines are multithread (MT) safe. They may be called safely from simultaneously running threads. Because the subroutines are MT-safe, additional performance is possible on multicore machines by using OpenMP to parallelize code blocks that contain calls to Perflib. As always, users are responsible for making sure that there is no data dependency in parallelized code.

An example of combining Perflib subroutines together with OpenMP parallelization is shown below. Perflib contains a subroutine named DGBMV to multiply a banded matrix by a vector. By putting this subroutine into a properly constructed loop, it is possible to use the subroutines in Perflib 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 Perflib subroutines are MT-safe, a user can use OpenMP directives to instruct the compiler to parallelize this loop, as shown below.

```
!$OMP PARALLEL DO PRIVATE(...) SHARED(...)
  DO 10, I = 1, N
    CALL DGBMV ('No transpose', N, N, ALPHA, A, LDA,
      $      B(1,I), 1, BETA, C(1,I), 1)
  10 CONTINUE
!$OMP END PARALLEL DO
```

Note compiler directives can be used to parallelize a loop with a subroutine call that ordinarily would not be parallelizable. For example, often it is not possible to parallelize a loop containing a call to some of the linear system solvers—many vendors have implemented those subroutines using code that is not MT-safe. However the versions in Perflib are MT-safe. As a result, users of multicore machines can get significant additional performance by parallelizing those loops.

## Distributed Memory Parallel Processors and Clusters

Perflib contains a tuned version of Scalable LAPACK (ScaLAPACK) to provide high performance on clusters and distributed memory multiprocessor systems. ScaLAPACK is dependent on the following libraries that are also included in Perflib: LAPACK, parallel BLAS (PBLAS), and Basic Linear Algebra Communication Subroutines (BLACS). BLACS is, in turn, dependent on the availability of an MPI implementation. For example, the Oracle Message Passing Toolkit may be used. The Oracle Solaris Studio 12 update 1 ScaLAPACK libraries are built with Oracle Solaris Studio compilers and Oracle Message Passing Toolkit 8.2.1(CT8.2.1), which is based on OpenMPI 1.3.

ScaLAPACK provides support for distributed memory versions of the functions in LAPACK and relies on its availability for each node used. The highly tuned version of LAPACK in Perflib provides superior performance for even the freely available version of ScaLAPACK. More information on ScaLAPACK can be found in the user guide located at <http://www.netlib.org/scalapack/slug/index.html>.

ScaLAPACK applications can be built and linked using compatible OpenMPI compiler wrappers. The compiler wrapper calls the Oracle Solaris Studio compilers with all the options necessary for compiling and linking MPI code as shown in the following example. ScaLAPACK and BLACS libraries need to be specified on the link line. In this example, Perflib supplies the highly optimized version of LAPACK via the `-library=sunperf` option.

```
$ mpif90 scalapack_prog.f90 -o scalapack_prog -lscalapack lblacs_openmpi -library=sunperf
```

Likewise, OpenMPI compiler wrappers are provided for compiling C and C++ applications. For example:

```
$ mpicc -c scalapack_prog_c.c
$ mpiCC -c scalapack_prog_c++.c
```

ScaLAPACK applications are run like any MPI application. The `mpirun` command runs a number of copies of an MPI program as specified by the `-np` option. The command below runs four MPI processes of the `scalapack_prb1` application.

```
$ mpirun -np 4 -- scalapack_prb1
```

See the *Sun HPC ClusterTools 8.2 User's Guide* located at <http://docs.sun.com/app/docs/doc/820-6793-10?l=en> for more options. See Appendix B for an example using the ScaLAPACK interfaces in Perflib to solve the linear matrix equation  $Ax = B$ . The source code for this example is available at <http://www.netlib.org/scalapack/examples/example1.f>

Note that it is longer and more complex than previous examples due to the inherent increased complexity of programming distributed memory computers with MPI. The code for SUBROUTINE MATINIT, which initializes and distributes the elements of matrices *A* and *B*, has been omitted for brevity. The example shows the basic requirements for calling ScaLAPACK functions: initializing the process grid, assigning the matrix to the processes, and calling the ScaLAPACK function.

## C Interfaces in Perflib

Perflib contains C interfaces for each of the routines contained in LAPACK, BLAS 1-3, FFTPACK, VFFTPACK, and SPARSE BLAS. These significantly improve on the CLAPACK library available on Netlib, which is just a straightforward translation from Fortran to C. The C interfaces in Perflib have C names, use the function interface conventions to which C programmers are accustomed, and do not have arguments that are redundant or unnecessary for a C function. C++ programmers also can use these interfaces by modifying their prototype declarations with the extern “C” declaration.

### Direct Support for C

The first thing to notice about the C interfaces is that function names are not followed by an underscore. In the CLAPACK library in Netlib, all of the subroutines and functions are followed by a trailing underscore to maintain compatibility with Fortran compilers, which often postfix subroutine names in the object (.o) file with an underscore. The C interfaces in Perflib are intended for C programmers, so the underscore is not required.

Another important improvement in Perflib is that arguments are passed in the way that C programmers prefer to use. Input-only scalars are passed by value rather than by reference, which gives

added safety and allows constants to be passed without the need for creating a separate variable to hold their value. Array references are based at zero to be compatible with the C convention rather than based at one to be compatible with Fortran. Complex scalars can be passed as either structures or arrays of length 2.

## Getting Peak Performance Using Perflib with C

As with the Fortran examples in the previous section, the key to using Perflib to get peak performance from applications is to recognize opportunities to transform user written code sequences into calls to Perflib functions. The code sequence below is adapted from LAPACK.

```
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 subprogram in Perflib that exactly replicates the functionality of the code above. However, the code can be accelerated by replacing it with the several calls to Perflib as shown below.

```
int    i, large_index;
float  a[n], b[n], largest;
large_index = isamax (n, a, 1);
largest = a[large_index];
large_index = isamax (n, b, 1);
if (b[large_index] > largest)
    largest = b[large_index];
```

Note the differences between the call to the C `isamax` function in Perflib above, and the more elaborate call shown below to a comparable function in CLAPACK.

```
/* 1.Declare scratch variable to allow 1 to be passed by value */
int  one = 1;
/* 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) - 1;
largest = a[large_index];
large_index = isamax_ (&n, b, &one) - 1;
if (b[large_index] > largest)
    largest = b[large_index];
```

## Getting Peak Performance Using Perflib with User Threads

As mentioned in the previous section all Perflib routines are MT-safe so may be called from parallel regions. This also applies to code with user-managed POSIX threads.

As an example of a program that uses Perflib subroutines from user-managed threads, consider a real-time signal processing application running on a 4 CPU server with one CPU dedicated to acquiring the data, two CPUs dedicated to performing FFTs on the data, and one CPU dedicated to post-processing 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] = 1;
    (void)thr_create ((void *)0, (size_t)N, fft_func,
                    (void *)&who[i], (long)0, (thread_t *)0);
}
```

In the above example, N is the stack size in bytes needed for the thread. Perflib requires that threads for 32-bit programs have a minimum stack size of 4 MB, and 8 MB for 64-bit programs.

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

```
cpu_id = *who_am_i;
while (1) {
    while (!do_fft[cpu_id]) {}
    fftf (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;
}
```

## Conclusion

This paper briefly described the capabilities of Perflib, as well as how to realize faster applications with Perflib for uniprocessor, shared memory parallel (including multicore) computers, and distributed memory parallel systems. In addition, it described the C interfaces in Perflib. By using Perflib, users of Oracle computing systems can obtain excellent—and often close to optimal—performance on current and future systems.

## References

**TABLE 1. REFERENCES**

ScaLAPACK User's Guide	<a href="http://www.netlib.org/scalapack/slug/index.html">http://www.netlib.org/scalapack/slug/index.html</a>
Sun Performance Library User's Guide	<a href="http://docs.sun.com/app/docs/doc/821-0276">http://docs.sun.com/app/docs/doc/821-0276</a>
Sun Studio 12u1 Fortran User's Guide	<a href="http://docs.sun.com/app/docs/doc/820-7600">http://docs.sun.com/app/docs/doc/820-7600</a>
Sun HPC Cluster Tools 8.1 User's Guide	<a href="http://docs.sun.com/app/docs/doc/820-6793-10?l=en">http://docs.sun.com/app/docs/doc/820-6793-10?l=en</a>
The ScaLAPACK Project	<a href="http://www.netlib.org/scalapack">http://www.netlib.org/scalapack</a>
The ESSL Library	<a href="http://www.netlib.org/lapack/essl">http://www.netlib.org/lapack/essl</a>

## Appendix A: Performance Examples

TABLE 2. NETLIB VERSUS PERFLIB FOR DGEMM RESULTS.

M=N=K	NETLIB MFLOPS	PERFLIB MFLOPS	NETLIB POP	PERFLIB POP	PERFLIB MFLOPS VERSUS NETLIB MFLOPS RATIO
100	3,879	8,629	37%	81%	2.22X
200	4,023	9,288	38%	87%	2.31X
300	4,280	9,653	40%	91%	2.26X
400	4,381	9,960	41%	94%	2.27X
500	4,347	9,999	41%	94%	2.30X
1,000	3,739	9,997	35%	94%	2.67X
2,000	2,414	10,091	23%	95%	4.18X
3,000	2,317	10,116	22%	95%	4.37X

- TRANSA, TRANSB = 'N'
- ALPHA, BETA = 1.0D0
- PoP = "Percent of Peak"

TABLE 3. NETLIB VERSUS PERLIB FOR CFFTF RESULTS

N	NETLIB MFLOPS	PERFLIB MFLOPS	PERFLIB / NETLIB MFLOPS
8	370	400	1.08X
16	322	930	2.89X
32	1,305	1,968	1.51X
64	1,983	3,948	1.99X
128	1,968	6,075	3.09X
256	2,739	8,825	3.22X
512	2,630	10,308	3.92X
1,024	2,858	12,009	4.20X
2,048	2,774	11,730	4.23X
4,096	2,803	11,860	4.23X
8,192	2,715	11,563	4.26X
16,384	2,769	11,834	4.27X
32,768	2,670	9,251	3.46X
65,536	2,632	9,379	3.56X
131,072	2,625	8,185	3.12X
262,144	2,080	4,606	2.21X
524,288	2,073	4,422	2.13X
1,048,576	1,400	4,405	3.15X

## Appendix B: ScaLAPACK Example

```

PROGRAM EXAMPLE1
*
*   Example Program solving Ax=b via ScaLAPACK routine PDGESV
*
*   .. Parameters ..
INTEGER          DLEN_, IA, JA, IB, JB, M, N, MB, NB, RSRC,
$               CSRC, MXLLDA, MXLLDB, NRHS, NBRHS, NOUT,
$               MXLOCR, MXLOCC, MXRHSC
PARAMETER       ( DLEN_ = 9, IA = 1, JA = 1, IB = 1, JB = 1,
$               M = 9, N = 9, MB = 2, NB = 2, RSRC = 0,
$               CSRC = 0, MXLLDA = 5, MXLLDB = 5, NRHS = 1,
$               NBRHS = 1, NOUT = 6, MXLOCR = 5, MXLOCC = 4,
$               MXRHSC = 1 )
DOUBLE PRECISION ONE
PARAMETER       ( ONE = 1.0D+0 )
*
*   ..
*   .. Local Scalars ..
INTEGER          ICTXT, INFO, MYCOL, MYROW, NPCOL, NPROW
DOUBLE PRECISION ANORM, BNORM, EPS, RESID, XNORM
*
*   ..
*   .. Local Arrays ..
INTEGER          DESCA( DLEN_ ), DESCB( DLEN_ ),
$               IPIV( MXLOCR+NB )
DOUBLE PRECISION A( MXLLDA, MXLOCC ), A0( MXLLDA, MXLOCC ),
$               B( MXLLDB, MXRHSC ), B0( MXLLDB, MXRHSC ),
$               WORK( MXLOCR )
*
*   ..
*   .. External Functions ..
DOUBLE PRECISION PDLAMCH, PDLANGE
EXTERNAL         PDLAMCH, PDLANGE
*
*   ..
*   .. External Subroutines ..
EXTERNAL        BLACS_EXIT, BLACS_GRIDEXIT, BLACS_GRIDINFO,
$               DESCINIT, MATINIT, PDGEMM, PDGESV, PDLACPY,
$               SL_INIT
*
*   ..
*   .. Intrinsic Functions ..
INTRINSIC       DBLE
*
*   ..
*   .. Data statements ..
DATA           NPROW / 2 / , NPCOL / 3 /
*
*   ..
*   .. Executable Statements ..
*

```

```

*   INITIALIZE THE PROCESS GRID
*
CALL SL_INIT( ICTXT, NPROW, NPCOL )
CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )
*
*   If I'm not in the process grid, go to the end of the program
*
IF( MYROW.EQ.-1 )
$   GO TO 10
*
*   DISTRIBUTE THE MATRIX ON THE PROCESS GRID
*   Initialize the array descriptors for the matrices A and B
*
CALL DESCINIT( DESCA, M, N, MB, NB, RSRC, CSRC, ICTXT, MXLLDA,
$             INFO )
CALL DESCINIT( DESCRB, N, NRHS, NB, NBRHS, RSRC, CSRC, ICTXT,
$             MXLLDB, INFO )
*
*   Generate matrices A and B and distribute to the process grid
*
CALL MATINIT( A, DESCA, B, DESCRB )
*
*   Make a copy of A and B for checking purposes
*
CALL PDLACPY( 'All', N, N, A, 1, 1, DESCA, A0, 1, 1, DESCA )
CALL PDLACPY( 'All', N, NRHS, B, 1, 1, DESCRB, B0, 1, 1, DESCRB )
*
*   CALL THE SCALAPACK ROUTINE
*   Solve the linear system A * X = B
*
CALL PDGESV( N, NRHS, A, IA, JA, DESCA, IPIV, B, IB, JB, DESCRB,
$           INFO )
*
IF( MYROW.EQ.0 .AND. MYCOL.EQ.0 ) THEN
WRITE( NOUT, FMT = 9999 )
WRITE( NOUT, FMT = 9998 )M, N, NB
WRITE( NOUT, FMT = 9997 )NPROW*NPCOL, NPROW, NPCOL
WRITE( NOUT, FMT = 9996 )INFO
END IF
*
*   Compute residual ||A * X - B|| / ( ||X|| * ||A|| * eps * N )
*
EPS = PDLAMCH( ICTXT, 'Epsilon' )
ANORM = PDLANGE( 'I', N, N, A, 1, 1, DESCA, WORK )
BNORM = PDLANGE( 'I', N, NRHS, B, 1, 1, DESCRB, WORK )
CALL PDGEMM( 'N', 'N', N, NRHS, N, ONE, A0, 1, 1, DESCA, B, 1, 1,
$           DESCRB, -ONE, B0, 1, 1, DESCRB )
XNORM = PDLANGE( 'I', N, NRHS, B0, 1, 1, DESCRB, WORK )
RESID = XNORM / ( ANORM*BNORM*EPS*DBLE( N ) )

```

```
*
  IF( MYROW.EQ.0 .AND. MYCOL.EQ.0 ) THEN
    IF( RESID.LT.10.0D+0 ) THEN
      WRITE( NOUT, FMT = 9995 )
      WRITE( NOUT, FMT = 9993 )RESID
    ELSE
      WRITE( NOUT, FMT = 9994 )
      WRITE( NOUT, FMT = 9993 )RESID
    END IF
  END IF
*
*
*   RELEASE THE PROCESS GRID
*   Free the BLACS context
*
  CALL BLACS_GRIDEXIT( ICTXT )
10 CONTINUE
*
*   Exit the BLACS
*
  CALL BLACS_EXIT( 0 )
*
9999 FORMAT( / 'ScaLAPACK Example Program #1 -- May 1, 1997' )
9998 FORMAT( / 'Solving Ax=b where A is a ', I3, ' by ', I3,
  $         ' matrix with a block size of ', I3 )
9997 FORMAT('Running on ', I3, ' processes, where the process grid',
  $         ' is ', I3, ' by ', I3 )
9996 FORMAT( / 'INFO code returned by PDGESV = ', I3 )
9995 FORMAT( /
  $         'According to the normalized residual the solution is correct.'
  $         )
9994 FORMAT( /
  $         'According to the normalized residual the solution is incorrect.'
  $         )
9993 FORMAT( / ' ||A*x - b|| / ( ||x||*||A||*eps*N ) = ', 1P, E16.8 )
  STOP
  END
```

## Appendix C: BLAS List

The compiler recognizes calls to the following subroutines.

- **BLAS1 library routines.** caxpy, ccopy, cdotc, cdotu, crotg, cscal, csrot, csscal, cswap, dasum, daxpy, dcopy, ddot, drot, drotg, drotm, drotmg, dscal, dsdot, dswap, dnorm2, dzasum, dznorm2, icamax, idamax, isamax, izamax, sasum, saxpy, scasum, scnrm2, scopy, sdot, sdsdot, snrm2, srot, srotg, srotm, srotmg, sscal, sswap, zaxpy, zcopy, zdotc, zdotu, zdrot, zdscal, zrotg, zscal, zswap
- **BLAS2 library routines.** cgemv, cgerc, cgeru, ctrmv, ctrsv, dgemv, dger, dsymv, dsyr, dsyr2, dtrmv, dtrsv, sgemv, sger, ssymv, ssyr, ssyr2, strmv, strsv, zgemv, zgerc, zgeru, ztrmv, ztrsv
- **BLAS3 library routines.** cgemm, csymm, csyr2k, csyrk, ctrmm, ctrsm, dgemm, dsymm, dsyr2k, dsyrk, dtrmm, dtrsm, sgemm, ssymm, ssyr2k, ssyrk, strmm, strsm, zgemm, zsymm, zsyr2k, zsyrk, ztrmm, ztrsm



Using the Oracle® Solaris Studio  
Performance Library with Applications  
June 2010

Oracle Corporation  
World Headquarters  
500 Oracle Parkway  
Redwood Shores, CA 94065  
U.S.A.

Worldwide Inquiries:  
Phone: +1.650.506.7000  
Fax: +1.650.506.7200  
oracle.com



Oracle is committed to developing practices and products that help protect the environment

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

This document is provided for information purposes only and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. UNIX is a registered trademark licensed through X/Open Company, Ltd. 0310

**SOFTWARE. HARDWARE. COMPLETE.**