Using Sun Performance Library

Introduction

Sun Performance Library™ is a library of high-speed mathematical subroutines for Solaris on SPARC and x86 based on seven of the most widely-used numerical subroutine libraries in the world. Sun Performance Library has native interfaces to FORTRAN 77, Fortran 90, and C. It can even be used within Java™ applets and applications to animate technical documents or as an integral building block for high-speed spiders and intelligent agents whose work involves computationally intensive tasks.

This writeup describes the capabilities of Sun Performance Library and shows how it can be used in real applications to boost speed by a factor of four or more. The sections that follow will introduce Sun Performance Library as shown below:

⇒ Capabilities of Sun Performance Library

⇒ Faster Applications with Sun Performance Library

⇒ Native C Interfaces in Sun Performance Library

Capabilities of Sun Performance Library

Sun Performance Library is a library of high-speed mathematical subroutines that contains Sun's optimized implementation of the subroutines in LAPACK 2.0, levels 1-3 of BLAS, FFTPACK, VFFTPACK, and LINPACK. These libraries include the following capabilities:

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

Fourier transforms: Fourier transforms, Fourier synthesis, cosine and quarter-wave cosine transforms, cosine and quarter-wave sine transforms

Matrix factorizations or decompositions: SVD, generalized SVD, QL and LQ, QR and RQ, Cholesky, LU, Schur, LDL and UDU^T

Support operations: Condition number, in-place or out-of-place transpose, inverse, determinant, inertia

The operations in Sun Performance Library that are modeled on LAPACK support the expanded capabilities and improved algorithms in LAPACK 2.0, but are completely compatible with both LAPACK 1.x and LAPACK 2.0. This is important because simply replacing LAPACK 1.x with LAPACK 2.0 can cause programs based on the older version to have linking errors, get wrong answers without warning, or simply abort. An example is shown below:

```
DOUBLE PRECISION WORK(N)
CALL DPTEQR ('NO EIGENVECTORS', N, DIAG, SUBD, Z, LDZ, WORK, INFO)
IF (INFO .EQ. 0) THEN
   CALL DLAZRO (M, N, ALPHA, BETA, A, LDA)
```

The code above conforms to the interface used by LAPACK 1.0 and CraySoft's™ libSci version 1.0, but will not link when moved to an environment that uses LAPACK 2.0 because DLAZRO has been replaced with a subroutine that has a different name and a different argument list. The arguments to DPTEQR have also changed, but in this case the subroutine name is the same. That means that it will link with LAPACK 2.0, but return wrong answers or abort. With Sun Performance Library, users can safely use programs intended for the original LAPACK 1.0. At the same time, they can gradually upgrade those portions of their applications that can take advantage of the power of LAPACK 2.0.

In addition to the standard subroutines with the capabilities listed above, Sun Performance Library contains native C interfaces to all of the user-callable subroutines in the library to ease development of numerical applications for C and programmers. The capabilities of these interfaces are described below in the section entitled *Native C Interfaces in Sun Performance* Library.

Sun Performance Library is available in both static and dynamic library versions optimized for the V7, V8, and V8+ architectures, and a V9-optimized version will be available with the first Sun shipment of V9. The optimization for each architecture is targeted at one implementation of that architecture, and then includes optimizations for other architectures when it does not harm the primary target. For example, the V8-optimized version is targeted at SuperSPARC-I™ and takes advantage of the superscalar capabilities of that chip, including special features of the SuperSPARC-II floating point pipeline, but also takes advantage of the special pipeline structure of the microSPARC™ memory interface and the Fast Constant and Fast Branch features of the hyperSPARC™.

Sun Performance Library supports static and shared libraries on Solaris™ 2.5.1, 2.6, and 2.7 and adds support for multiple processors.

Sun Performance Library uses two modes of parallelization that are selectable by the user at link time, referred to as Dedicated and Shared. The Dedicated mode of parallelization delivers peak performance to applications using automatic compiler parallelization and running on an MP machine that is dedicated to a single CPU-intensive application. The Shared mode of parallelization delivers peak performance to applications that do not use compiler parallelizatoin and that run on a machine that is shared with other applications.

Faster Applications with Sun Performance Library

This section describes some of the ways in which Sun Performance Library can significantly boost the speeds of applications written in either FORTRAN 77, Fortran 90, or C. Although it may be necessary to make some modifications to an application in order to get peak performance, many codes can benefit significantly from Sun Performance Library without any source code changes or recompilations.

Improve Performance Without Making Code Changes

There are three common techniques for using Sun Performance Library to improve the speed of user code without making any changes. First, many applications are built using one or more of the seven libraries that comprise Sun Performance Library. It is particularly common for third-party vendors to use BLAS and LAPACK as building blocks. Because Sun Performance Library maintains the same interfaces and functionality of these libraries, those codes can automatically gain significant speed just by switching from the Netlib distribution. Second, if an application already uses some libraries besides those in Sun Performance Library, it is often possible to use Sun Performance Library to speed up the other libraries. A third approach is to use some of the tools that automatically transform an application to use Sun Performance Library. These tools can make significant improvements without requiring the user to make any changes. Each of these techniques is described below.

Include Sun Performance Library in the Development Environment. For many users, subroutines in Sun Performance Library can be much faster than subroutines that perform similar functions in CraySoft's libSci™, Visual Numerics' IMSL® libraries for both FORTRAN and C, NAG's® Mark 16, Rogue Wave's™ math libraries, Fujitsu's SSL-II, and others. For example, Sun Performance Library runs the LINPACK 1000x1000 benchmark about 1.5 times faster than libSci on a SPARCstation 5. In addition to significantly improved performance on single-CPU machines, Sun Performance Library can also give better performance on MP machines because of the higher serial speed of many of the subroutines and because Sun Performance Library has parallelized subroutines that may be serial in other products. Performance-conscious users can find it beneficial to include Sun Performance Library in their development environment.

Use Sun Performance Library to Speed Up Other Libraries. Another approach for users of other mathematical libraries is to replace the BLAS that is included in their library with the BLAS in Sun Performance Library, but leave the rest of the subroutines. This is a helpful approach 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 subroutines, so replacing those generic subroutines with the highly optimized BLAS and LAPACK subroutines in Sun Performance Library can give large speed improvements on both serial and MP machines. However, replacing the core routines does not require the user to make any code changes and so proprietary features of the library may still be used.

Even libraries that already have fast core routines may get additional speedups by using Sun Performance Library. For example, although the core subroutines in CraySoft's libSci do have excellent performance on SuperSPARC machines, the subroutines in Sun Performance Library can be faster on other platforms such as the SPARCstation 5 and machines based on the new UltraSPARC™ processor.

Users of other libraries can get the benefits of the optimized subroutines in Sun Performance Library while keeping any proprietary capabilities of their current library in one of two ways. If the other library is statically linked (`.a`), users can use `remove_old_subroutines`, available from Sun sales offices, to remove redundant subroutines from the other library. The example below shows how a user of a new Ultra-1 workstation could use the BLAS and LAPACK from Sun Performance Library while continuing to get the random number generator from libSci.

```
remove_old_subroutines libSci.a
```

Users who do not have access to `remove_old_subroutines` or users of dynamically linked (`.so`) libraries can still often cause most subroutines from Sun Performance Library to take precedence over subroutines in the other library by specifying `-lsunperf` in the link step immediately before the reference to the other library.

```
f77 -dalign app.f -lsunperf -lSci
```

Note that neither source code changes nor recompilation are required.

Use Tools to Restructure Code. In some cases, other libraries may not directly use the subroutines in Sun Performance Library, however there may be conversion aids available. For example, users of EISPACK can find a conversion chart in the LAPACK User's Manual that will show how to convert from EISPACK calls to the LAPACK calls in Sun Performance Library. Users of IBM's® ESSL can find information at `http://www.netlib.org/lapack/essl` that will help convert programs from ESSL to Sun Performance Library. Users of other libraries may be able to find similar resources to aid in the conversion process.

Even users who do not use other libraries in their applications can get significant speed improvements without necessarily making code modifications by using Sun Performance Library. One way to do that is to use a code restructurer such as VAST® from Pacific-Sierra Research[1] to replace BLAS code structures in existing user code with calls to the BLAS in Sun Performance Library. For example, VAST is able to recognize many user-

[1] Information about VAST® from Pacific Sierra Research Corporation can be found on the World Wide Web at `http://www.psrv.com/vast/vast_super.html`.

written matrix multiplications and replace them with a call to the matrix multiplication subroutine in Sun Performance Library.

User code that does not use subroutines from Sun Performance Library is passed into VAST	Pacific Sierra's VAST automatically transforms user code into equivalent libsunperf calls	Application uses Sun Performance Library for speed, but no source code changes are needed

Getting Peak Performance with Sun Performance Library

Getting peak performance from Sun Performance Library for single-CPU applications is simply a matter of identifying code constructs in an application that can be replaced by calls to subroutines in Sun Performance Library. MP applications can get additional speed by identifying opportunities for parallelization. Examples of each of these is shown in this section.

The easiest situation occurs when a block of user code exactly duplicates a capability of Sun Performance Library. 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 $y \leftarrow Ax + y$, which can be performed with the DGEMV subroutine. In other cases, a block of code may be equivalent to several Sun Performance Library calls or may contain a mixture of code that can be replaced together with code that has no natural replacement in Sun Performance Library.

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 Sun Performance Library 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 Sun Performance Library, so that code is simply pulled out of the outer loop. With that code removed to its own loop, the rest of the loop can be recognized as simply being a rank-1 update of the general matrix X, which can be accomplished using the DGER subroutine 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 may not bring a large payoff. Note also 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, 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 subroutine will probably bring a much larger payoff than a loop over calls to DGER.

All Sun Performance Library subroutines are MT-safe except that a few subroutines with 'LA' as the second and third letters of their name are not MT-safe when they are called from user-written code[2]. The 'LA' subroutines that are most likely to be used in user-written code are MT-safe including xLAMCH, xLAENV, and all of the vector and matrix norms. Because the subroutines are MT-safe, additional performance is possible on MP machines by using iMPact to parallelize loops that contain calls to Sun Performance Library. As always, users are responsible for making sure that there is no loop dependency in user-parallelized loops.

An example of an effective combination of a Sun Performance Library subroutine together with an iMPact parallelization directive is shown below. Sun Performance Library 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 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 subroutines are MT-safe, a user may use iMPact directives as shown below to instruct the compiler to parallelize this loop.

```
C$PAR DOALL
      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
```

Note that a user may 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[3] because many vendors have implemented those subroutines using code that is not MT-safe. However the

[2] Subroutines that compute matrix norms and that have 'LA' as the second and third letters of their names use a technique called ''reverse communication'' and are not MT-safe when they are called from user-written code using the standard LAPACK interface. Those subroutines are MT-safe when called by subroutines or functions in Sun Performance Library. Virtually no user applications call 'LA' subroutines directly.

[3] Loops containing calls to the expert drivers of the linear system solvers (subroutines whose names end in SVX) are usually not parallelizable with other implementations of LAPACK. The implementation of LAPACK in Sun Performance Library does allow parallelization of loops containing such calls.

versions in Sun Performance Library are MT-safe and so users of MP machines can get significant additional performance by parallelizing those loops.

Native C Interfaces in Sun Performance Library

Sun Performance Library contains native C interfaces for each of the subroutines and functions contained in LAPACK, BLAS, FFTPACK, VFFTPACK, and LINPACK. These improve on the CLAPACK available on Netlib, which is just an f2c translation of the standard libraries. The native interfaces in Sun Performance Library 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 can also use these interfaces by modifying their prototype declarations with the `extern ‘‘C’’` declaration.

Direct Support for C

The first thing to notice about the native 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 native interfaces in Sun Performance Library are intended for C programmers, so the underscore is not required.

Another important improvement in Sun Performance Library 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 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.

A third improvement involves workspace allocation. Because C can allocate workspace while ANSI FORTRAN 77 cannot, the native C interfaces do not have arguments related to workspace. Having an application manage workspace can sometimes convey a performance advantage, so the native C interfaces in Sun Performance Library have a mechanism for allowing a user to specify workspace, but it is never required. An example of user-allocated workspace is shown below.

```
float  *workspace;

/* Allocate the workspace and inform Sun Performance */
/* Library that user-allocated space is available.   */
workspace = malloc (sizeof (float) * (4 * n));
allocate_workspace (workspace, 4 * n);

sgeqrf (n, n, t, ldt, tau, &info);
if (info == 0) {
    sgesvx (fact, trans, n, 1, a, lda, af, ldaf, ipivot, &equed,
            row_scale, col_scale, b, n, x, n, row_cond, fwd_err,
            bkw_err, info);
```

Note that the workspace is used as a result of the call to `allocate_workspace` and it is not passed as an argument to the functions that will use it.

User-allocated workspace can boost performance in two ways, though it may degrade performance if not done carefully. In the example above, the workspace used for both `sgeqrf` and `sgesvx` is the same, which may improve cache performance over the case where the two workspaces are distinct. Having collocated workspaces is almost always an improvement over having distinct workspaces, though Sun Performance Library will often do that without user intervention. A second instance in which it may be better for a user to explicitly allocate workspace arises when solving very large problems. Note in the example above that the size of the workspace allocated by the application is 4`n` floats while the man page shows that the optimal size for `sgeqrf` is larger than that. If the matrix being factored is so large that the optimal workspace size exceeds the amount of available RAM, it may be better to use user allocation to force the use of a smaller workspace.

There are a few things to remember about user-allocated workspace. First, if a future release of Sun Performance Library increases the minimum workspace size to be bigger than the amount allocated by the user, the user-allocated space will be too small to be usable and the library will allocate its own workspace. The total allocated workspace will then be the sum of user- and library-allocated space, which may actually make things worse. Second, if the user runs the code on a machine with enough RAM to hold the optimal workspace, the size of the user-allocated workspace will remain the same and the code will run slower than it could because the size of the workspace used will be the smaller suboptimal value that the user has allocated. It is usually best to allow Sun Performance Library to do workspace allocation unless there is a good reason for doing otherwise.

Getting Peak Performance with Sun Performance Library

As with the FORTRAN examples in the previous section, 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 code sequence below 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] > biggest)
        largest = a[i];
    if (b[i] > biggest)
        largest = b[i];
}
```

There is no subprogram 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, 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 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 = 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];
```

All Sun Performance Library subroutines are MT-safe except that a few subroutines with 'LA' as the second and third letters of their name are not MT-safe when they are called from user applications[4]. As a result, additional speed can be had by parallelizing loops that contain Sun Performance Library calls. The easiest way to do this is to use iMPact parallelization directives as shown in the preceding section describing the FORTRAN interfaces to Sun Performance Library. However all of the subroutines are also safe when called from user-managed threads.

As an example of a program that uses Sun Performance Library 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 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] = 1;
   (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 probably 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;
}
```

[4] This is described in more detail in the preceding section about using Sun Performance Library from within a FORTRAN program.