FUJITSU

FUJITSU Software Technical Computing Suite V2.0

Programmer's Guide for Usage of Mathematical Libraries (PRIMEHPC FX100)

J2UL-1900-02ENZ0(00)
September 2015

Preface

This manual describes the use of Mathematical Libraries offered by the products. The features of interest is as follows.

SSL II
SSL II Thread-Parallel Capabilities
C-SSL II
C-SSL II Thread-Parallel Capabilities
SSL II/MPI
BLAS
LAPACK
ScaLAPACK
Fast Basic Operations Library for Quadruple Precision (referred to as fast_dd, throughout this manual)

SSL II and C-SSL II are sequential version that are executed on one core. SSL II, SSL II Thread-Parallel Capabilities, C-SSL II, C-SSL II Thread-Parallel Capabilities and SSL II/MPI are collectively referred to as Mathematical Libraries.

Organization of This Manual

This manual is organized as follows:

1. SSL II Mathematical Libraries

1.1 Overview

This section describes overview of SSL II Mathematical Libraries components.

1.2 Documentation

This section describes manuals that describe the calling interfaces of subroutines.

1.3 Example program using SSL II

This section shows how to call subroutines from user's programs by taking important examples. SSL II and C-SSL II are created as a thread-safe library. When called from an OpenMP Fortran program, an SSL II or C-SSL II routine can be called from multiple threads concurrently with different input data being given from each thread. This section explains the way of such concurrent execution.

1.4 Compile, Link and Run

This section describes how to compile user's programs that contain calls to SSL II routines and link-edit with SSL II Mathematical Libraries.

2. BLAS, LAPACK, ScaLAPACK

2.1 Overview

This section describes overview of BLAS, LAPACK and ScaLAPACK.

2.2 Compile, Link and Run

This section describes how to compile user's programs that contain calls to BLAS, LAPACK or ScaLAPACK routines and link-edit with the libraries.

2.3 Notes

Several notes are provided on usage of the products.

3. Fast Basic Operations Library for Quadruple Precision

3.1 Overview

This section describes overview of Fast Basic Operations Library for Quadruple Precision.

3.2 Documentation

This section describes manuals that describes the calling interfaces of subroutines.

3.3 Compile and Link and Run

This section describes how to compile user's programs that contain calls to Fast Basic Operations Library for Quadruple Precision routines and link-edit with the libraries.

3.4 Notes

Several notes are provided on usage of the products.

Refer to the manuals or the documentations mentioned in 1.2, 2.1 and 3.2 for usage descriptions of the individual routines. This manual describes the system-specific information.

In addition to this manual, the following manuals are references that apply to this manual:

Fortran User's Guide

C User's Guide

C++ User's Guide

MPI User's Guide

For a detailed specification of OpenMP Fortran please refer *OpenMP Application Program Interface Version 3.0 May 2008* (http://openmp.org/wp).

Date of Publication and Version

Version	Manual code
September 2015, 3rd Version	J2UL-1900-02ENZ0(00)
February 2015, 2nd Version	J2UL-1900-01ENZ0(01)
October 2014, 1st Version	J2UL-1900-01ENZ0(00)

Update History

Changes	Location	Version
Support shared library of BLAS, LAPACK and ScaLAPACK	2.2.7	2nd Version
Support PLASMA.	2.	3rd Version
Item is added in Notes.	2.3	
Using fast_dd with Coarray feature is added.	3.4.3	

Acknowledgement

BLAS, LAPACK and ScaLAPACK are collaborative effort involving several institution and it is distributed on Netlib.

Contents

List of Tables

1. SSL II Mathematical Libraries

1.1 Overview

This section describes overview of SSL II Mathematical Libraries components.

1.1.1 SSL II

SSL II is tuned to a scalar processor. For SPARC64 XIfx processors respectively their special hardware features are used to extract their performance.

The SSL II library is provided as thread-safe library. Namely, the routines can be called not only from regular sequential Fortran programs but also from thread-parallel Fortran programs written with OpenMP Fortran API. In the latter case, a routine can be called from multiple threads concurrently with different input data being given from each thread.

1.1.2 SSL II Thread-Parallel Capabilities

The SSL II Thread-Parallel Capabilities is a collection of parallel algorithms for shared memory scalar parallel computers. For SPARC64 XIfx processors respectively their special hardware features are used to extract their performance. Each subroutine is written with the OpenMP Fortran API, and can be called from not only OpenMP Fortran programs but also ones automatically parallelized , and conventional sequential Fortran programs. These types of programs can be compiled by Fortran compiler with the corresponding option respectively.

The subroutine names are different from those of conventional SSL II so that user's programs can use the routines from the conventional SSL II and ones from this parallel libraries at the same time in an application program. Besides, argument list of a subroutine is not always identical between the two libraries.

SSL II Thread-Parallel Capabilities provide the routines with scalable performance over the fields like matrix operation , linear equation solvers (direct methods) , linear equation solvers (iterative solvers) , inverse of matrix , eigenvalue problems , Fourier transforms and random number generators which are required for number crunching calculation.

1.1.3 C-SSL II

C-SSLII is tuned to a scalar processor. For SPARC64 XIfx processors respectively their special hardware features are used to extract their performance.

The C-SSL II library is provided as thread-safe library. Namely, the routines can be called not only from regular sequential C programs but also from thread-parallel C programs written with OpenMP C API. In the latter case, a routine can be called from multiple threads concurrently with different input data being given from each thread.

1.1.4 C-SSL II Thread-Parallel Capabilities

The C-SSL II Thread-Parallel Capabilities is a collection of parallel algorithms for shared memory scalar parallel computers. Especially, the tuning in which it specializes hard is done for SPARC64 XIfx CPU. Each subroutine is written with the OpenMP C API, and can be called from not only OpenMP C programs but also ones automatically parallelized, and conventional sequential C programs. These types of programs can be compiled by C/C++ compiler with the corresponding option respectively.

The subroutine names are different from those of conventional C-SSL II so that user's programs can use the routines from the conventional C-SSL II and ones from this parallel libraries at the same time in an application program. Besides, argument list of a subroutine is not always identical between the two libraries.

C-SSL II Thread-Parallel Capabilities provide the routines with scalable performance over the fields like matrix operation , linear equation solvers (direct methods) , linear equation solvers (iterative solvers) , eigenvalue problems , Fourier transforms and random number generators which are required for number crunching calculation.

1.1.5 SSL II/MPI

SSL II/MPI provides the computational functionality to efficiently compute large-scale problems on a parallel computer with distributed memory system. The algorithms for parallel processing have been adopted.Especially, the SPARC64 XIfx binary version are the result of extensive source code tuning by taking into account the CPU architecture.

Each capabilities in SSL II/MPI are available in a Fortran subroutine which can be referred in a CALL statement.

Besides, the functional scope, the routine names and their argument sequence of SSL II/MPI are different from those of SSL II, a mathematical software library, for a uni-processor and a parallel computer in SMP architecture.

SSL II/MPI provides the capabilities of three dimensional Fourier transform.

1.2 Documentation

- SSL II
  (1) SSL II User's Guide
  (2) FUJITSU SSL II Extended Capabilities User's Guide
  (3) FUJITSU SSL II Extended Capabilities User's Guide II

  As you can see, the Online User's Guides are divided into three separate volumes, in consideration of the total amount.
  Documents (2) and (3) describes those subroutines which were intended to be used especially for supercomputers and at the same time for large scale problems including FFT or sparse matrix algebra. Furthermore, it is noted that some of subroutines in document (1) have almost equivalent ones in functionality in document (2) or (3) with vector-oriented data layout or algorithms implemented (one typical example is matrix multiply routine). The user is encouraged to try out the equivalent ones from (2) and (3) first, when available.
  Because of this structure of documents, we strongly recommend the user to go through the three documents to find the most appropriate subroutines to meet the purposes.

- SSL II Thread-Parallel Capabilities

  FUJITSU SSL II Thread-Parallel Capabilities User's Guide

  This document contains a complete list of subroutines provided, general explanation on how to use the subroutines in the user's source codes, example codes using the subroutines, and detailed description on each subroutine.

- C-SSL II

  FUJITSU C-SSL II User's Guide

  This document is the User's Guide where general rule on usage of C-SSL II and detailed description of each routine are documented.

- C-SSL II Thread-Parallel Capabilities

  FUJITSU C-SSL II Thread-Parallel Capabilities User's Guide

  This document is the User's Guide where general rule on usage of C-SSL II Thread-Parallel Capabilities and detailed description of each routine are documented.

- SSL II/MPI

  FUJITSU SSL II/MPI User's Guide

  This document contains a complete list of subroutines provided, general explanation on how to use the subroutines in the user's source codes, example codes using the subroutines, and detailed description on each subroutine.

1.3 Example program using SSL II

1.3.1 How to use SSL II

SSL II can be called not only from the sequential Fortran programs but also from thread-parallelized programs written in OpenMP Fortran. This document is intended to show several examples of using thread-safe SSL II from Fujitsu Fortran, so that the user can understand general rules for using the library. The reader is assumed to have an introductory knowledge of "thread" and the specification of OpenMP Fortran standards.

What are needed to use the thread-safety of SSL II can be summarized as follows. Namely,

(1) SSL II is to be used in the environment of multiple cores or multiple processors,

(2) SSL II is to be called from programs written in OpenMP Fortran, and

(3) SSL II is to be link-edited with object programs generated by Fujitsu Fortran compiler.

The purpose of using thread-safety of SSL II is to have a SSL II subroutine concurrently solve different problems that are independent from each other and so reduce the turnaround time necessary to solve all the problems. For example, when the user has several independent but similar problems and want to solve them, the user normally calls an appropriate subroutine with one problem at a time and repeats the subroutine call with next problem (this is in sequential manner). With the thread-safe library, however, the user can give several problems to a subroutine at a time using multiple threads, where one thread takes care of one problem and all the threads run concurrently. This way, the user can expect a parallel execution with the number of parallelism equal to the number of threads. Of course, multiple CPUs or cores have to be available for reduction of elapsed time. The user will soon see concrete examples below.

Note that SSL II is not a parallel library by which we mean a subroutine is designed to solve a single problem using multiple threads. For example, when the user wants to compute a single matrix multiplication of large size using multiple threads, the user would need a different subroutine that does the operation in parallel on multiple threads. SSL II is not designed for that purpose.

*Note: There is another library "SSL II Thread-Parallel Capabilities" which adopts parallel algorithms. Refer to the top page of the online manual for SSL II Thread-Parallel Capabilites for details.

The examples given below handles simple problems so that the reader can understand the principle of using the thread-safety. For the detailed functionality and meanings of arguments of individual subroutines, please refer to an appropriate SSL II User's Guide.

Also the examples below do not assume the numbers of threads available. The number of threads is to be specified at execution time via the environment variable OMP_NUM_THREADS. In the SSL II, however, the maximum number of threads the user can specify is 128.

It is noted that the examples below could run on a single core even with multiple threads being specified. In order to achieve real parallelism, the user needs as many CPUs or cores as threads.

As for the procedure of compile and link-edit the example programs, please refer to "1.4.2 SSL II".

Example 1 : System of Linear Equations

Problem Description

Think of a system of linear equations of order n.

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{1.1}$$

Here, $\boldsymbol{A}$ is a real matrix of order n, $\boldsymbol{b}$ a right hand side vector of order n, and $\boldsymbol{x}$ the solution vector of order n. Now, let assume we want to solve the equations for different right hand side vectors as shown below.

$$\boldsymbol{Ax}_i = \boldsymbol{b}_i \qquad i = 1, 2, \dots , m \tag{1.2}$$

Example programs

Let's solve (1.2) for different right hand side vectors by using subroutine DVLAX from SSL II. The subroutine DVLAX is designed to factor the given matrix $\boldsymbol{A}$ into LU, and then solve the LU system for $\boldsymbol{x}$ by backward and forward substitution. The factored LU is returned in the 2-D array "a" that contained the original matrix $\boldsymbol{A}$ on entry, and the solution $\boldsymbol{x}$ is returned in the array "b" that contained the right hand side vector $\boldsymbol{b}$**.** For the solution for next right hand side vector $\boldsymbol{b}$ the user sets up the next $\boldsymbol{b}$ in array "b" and calls the subroutine with array "a" unchanged. At the same time, the user is asked to tell the subroutine if the call is the initial one or subsequent one through the "`ISW`".

Suppose we have a system of linear equations of order 100 with 50 right hand side vectors. In order to understand thread-safety of SSL II intuitively, let's look at a sequential program at first. The reader should notice how the subsequent solutions are computed.

<u>Using from sequential program</u>

```
      implicit real*8 (a-h,o-z)
      parameter (k=100,n=100,m=50)
      real*8 a(k,n),b(n),vw(n)
C
      epsz=0.0d0
C ==================
C Define the matrix
C ==================
      do 10 i=1,n
      do 10 j=i,n
        a(i,j)=.......
   10 continue
C ==================================
C Define the first right hand vector
C ==================================
      do 20 i=1,n
        b(i)=.......
   20 continue
      isw=1
      call dvlax(a,k,n,b,epsz,isw,is,vw,ip,icon)
      if(icon.ne. 0) then
        print *,'The given problem seems to be not normal'
        stop
      endif
C ===================================================
C Here we have got the LU factors in array a and the
C solution in b(1:n) with respect to the first
C right hand vector.
C Now we continue to get solutions for the rest of
C right hand vectors.
C ===================================================
      isw=2
      do 40 j=2,m
```

```
C =================================
C Define the next right hand vector
C =================================
      do 30 i=1,n
        b(i) = .......
   30 continue
      call dvlax(a,k,n,b,epsz,isw,is,vw,ip,icon)
      .......
   40 continue
      .......
      end
```

Next, let's use thread-safe DVLAX. The procedure of obtaining LU factorization and the solution for the first right hand side vector is the same as before. However, the solutions for subsequent right hand side vectors can be computed concurrently using multiple threads. In order to do so, the set of right hand side vectors need to be set up in prior. In the sample program below, the set of right hand side vectors is put in a 2-D array "b." Furthermore, those arguments which are altered on output need to be assigned memory space for each thread. The parameter `"icon"` is the case when called with `ISW`=2. So, we should have an array for `icon`. Other arguments such as

`a`, `k`, `n`, `epsz`, `isw`, `is`, `vw`, `ip`

are not altered when `ISW`=2, and therefore can be shared among threads.

Note: Although subroutine DVLAX does not alter the contents of work arrays (`vw`, `ip`) when called with `ISW`=2, there is need in general that a work array should have different memory space for each thread because it is to be altered inside.

Below is an example OpenMP Fortran program using thread-safe DVLAX. Some explanations of the code will be given after the listing.

Using from thread-parallelized program

```
      implicit real*8 (a-h,o-z)
      parameter (k=100,n=100,m=50)
      real*8 a(k,n),b(n,m),vw(n)
      integer ip(n),icon(m)
C
      epsz=0.0d0
C ===================================================
C Define the matrix a and multiple right hand vectors
C ===================================================
      do 10 i=1,n
      do 10 j=i,n
        a(i,j)=.......
   10 continue
      do 20 i=1,n
      do 20 j=1,m
        b(i,j)=.......
   20 continue
      isw=1
      call dvlax(a,k,n,b,epsz,isw,is,vw,ip,icon)
      if(icon(1).ne. 0) then
        print *,'The given problem seems to be not normal'
        stop
      endif
C ====================================================
C Here we have got the LU factors in array a and the
C solution in b(1:n,1) with respect to the first
```

```
C right hand vector.
C Now we continue to get solutions for the rest of
C right hand vectors b(1:n,2),b(1:n,3),...,b(1:n,m)
C Solutions will be calculated concurrently.
C ==================================================
!$OMP PARALLEL
      isw=2
!$OMP DO SCHEDULE(STATIC)
      do 30 j=2,m
        call dvlax(a,k,n,b(1,j),epsz,isw,is,vw,ip,icon(j))
   30 continue
!$OMP END DO
!$OMP END PARALLEL
      do 40 i=1,m
        print *, icon(i)
   40 continue
      .......
      end
```

Explanations

(1) The block of code enclosed by the !$OMP PARALLEL and !$OMP END PARALLEL directive pair is executed by multiple threads in parallel. The !$OMP PARALLEL specifies that a team of threads be created at that point. All variables and arrays are shared among the threads in the team in this example code. That means that there exists only one object for each variable and the object can be altered or referenced by any thread.

The OpenMP Fortran allows the user to specify a clause on the PARALLEL directive that declares some variables to get private copies. For more information on data scope attributes, see an appropriate documentation on OpenMP (e.g. "OpenMP Application Program Interface, Version 2.5 May 2005", available from the Web).

(2) The !$OMP DO SCHEDULE(STATIC) directive specifies that execution of the iterations of the immediately following DO loop can be executed in parallel and so can be distributed across the threads of the team. The clause SCHEDULE(STATIC) specifies that iterations are divided into pieces of a size (nearly) equal to the iteration count divided by the number of thread, and the pieces are statistically assigned to threads in the team.

(3) What is one more important characteristic is that the above code can be compiled without OpenMP features, link-edited with SSL II libraries. The resulting executable program still woks and produces the same computational results on a processor while everything is serialized. This is because the above code is nothing but a regular sequential program when !$OMP directives are ignored. The !$OMP directives are treated as Fortran comment statements when compiled by a Fortran compiler without `-Kopenmp` option. The subsequent examples appearing below have the same nature and so can be regarded as both sequential code and parallel code depending on whether the !$OMP directives are activated.

Example 2 : Quadrature

Problem description

Let's compute the following quadrature,

$$\int_0^1 f(p,x)dx\,, \qquad f(p,x) = \frac{1}{x^p} + \sin px \tag{2.1}$$

for different values of parameter $p = i / 10$, $i = 1, 2, \ldots , 9$.

Example program

Subroutine DAQN9, one of quadrature routines from SSL II, can be used for (2.1). The FUNCTION subprogram of the name FUN to evaluate the integrand must use values of *p*, while those values are controlled in the main program. Because of this, parameter *p* is storage associated through a named common block. It is essential that the variable for *p* must be made private to each thread, so that evaluation of the definite integrals for different values of *p* can be done by multiple threads in parallel. In other words, it is obvious that if the variable were just shared among the threads it should be impossible for the threads to evaluate the integrals concurrently.

```
      implicit real*8 (a-h,o-z)
      parameter (ip=9)
      common /aqcom/p
!$OMP THREADPRIVATE(/aqcom/)
      real*8 s(ip),err(ip)
      integer n(ip),icon(ip)
      external fun
      a=0.0d0
      b=1.0d0
      eps=dmach(eps)
      epsa=dmax1(100.0*eps,1.0d-10)
      epsr=eps
      nmin=21
      nmax=2000
C ==================================================
C Now calculate definite integrals for ip different
C functions concurrently.
C ==================================================
!$OMP PARALLEL
!$OMP DO SCHEDULE(STATIC)
      do 10 i=1,ip
        p=dfloat(i)/10.0d0
        call daqn9(a,b,fun,epsa,epsr,nmin,nmax,
     +        s(i),err(i),n(i),icon(i))
   10 continue
!$OMP END DO
!$OMP END PARALLEL

C =================
C Print the results
C =================
      write(6,500) (i,icon(i),s(i),err(i),i=1,ip)
  500 format(3x,'No, icon, integral, estimated err'//
     + (3x,i2,3x,i5,3x,E25.15,3x,E10.3))
      end

      function fun(x)
      real*8 x,p,fun
      common /aqcom/p
!$OMP THREADPRIVATE(/aqcom/)
      fun=0.0d0
      if(x.gt.0.0d0) fun=x**(-p)+dsin(p*x)
      return
      end
```

Upon completion of the above code, it will be noticed that all the elements of `icon(*)` have value 11000. This is just because DAQN9 has ability of detecting algebraic singularities and has detected successfully them for the given integrands but computed the correct results. (for detailed information on the subroutine. See the User's Guide.)

There is one precaution when writing subprograms that are to be called from inside SSL II. That is, the subprograms need to be *thread-safe*, which means the subprograms can be executed by multiple threads in parallel with different argument values like `x` of `fun(x)` in the above example, but produces correct results. For comparison, let's think of the following subprogram.

```
      function fun(x)
      implicit real*8 (a-h,o-z)
      term = 1.0d0
      s = term
      do 10 i=1,5
        term = term*x/dfloat(i)
        s = s+ term
   10 continue
      fun = s
      return
      end
```

In order to assure this program is thread-safe, the local variables `s`, `term` need to get its own copy for each thread. With Fujitsu Fortran compiler, the compile option `-Kopenmp` assures this. If the option is specified the variables `s`, `term` are treated as automatic and allocated in the stack area. In the presence of multiple threads running in the subprogram, each thread is given private stack area. However, the following restriction applies.

Variables that have initial values or SAVE attribute or are equivalenced with others can not be automatic. If the variables having initial data are not altered in the program, there should be no problem. For the other cases, the user will have to consider appropriate modifications.

Example 3 : Special Functions

Problem description

As the final example, let's consider evaluating the first order Bessel function of the first kind $J_1(x)$ at 300 equally spaced points in the interval $0 \le x \le 10$.

Example program

In the following code, subroutine DBJ1 from SSL II is used. The code looks straightforward and needs no extra explanation.

```
      implicit real*8 (a-h,o-z)
      parameter (ip=301)
      real*8 x(ip),f(ip)
      integer icon(ip)
      delta=10.0d0/dfloat(ip-1)
!$OMP PARALLEL
!$OMP DO SCHEDULE(STATIC)
      do 10 i=1,ip
        x(i)= delta*dfloat(i-1)
        call dbj1(x(i),f(i),icon(i))
   10 continue
!$OMP END DO
!$OMP END PARALLEL
      write(6,100) (x(i),icon(i),f(i),i=1,ip)
  100 format(3x,'x, icon, J1(x)'//
     + (3x,E25.15,3x,I5,3x,E25.15))
      end
```

1.3.2 How to use C-SSL II

C-SSL II can be called not only from the sequential C and C++ programs but also from thread-parallelized programs written in OpenMP C/C++. This document is intended to show several examples of using thread-safe C-SSL II, so that the user can understand general rules for using the library. The reader is assumed to have an introductory knowledge of "thread" and the specification of OpenMP C/C++ standards.

What are needed to use the thread-safety of C-SSL II can be summarized as follows. Namely,

(1) C-SSL II is to be used in the environment of multiple cores or multiple processors,

(2) C-SSL II is to be called from programs written in OpenMP C/C++, and

(3) C-SSL II is to be link-edited with object programs generated by Fujitsu C/C++ compiler.

The purpose of using thread-safety of C-SSL II is to have a C-SSL II routine concurrently solve different problems that are independent from each other and so reduce the turnaround time necessary to solve all the problems. For example, when the user has several independent but similar problems and want to solve them, the user normally calls an appropriate routine with one problem at a time and repeats the routine call with next problem (this is in sequential manner). With the thread-safe library, however, the user can give several problems to a routine at a time using multiple threads, where one thread takes care of one problem and all the threads run concurrently. This way, the user can expect a parallel execution with the number of parallelism equal to the number of threads. Of course, multiple CPUs or cores have to be available for reduction of elapsed time. The user will soon see concrete examples below.

Note that C-SSL II is not a parallel library by which we mean a routine is designed to solve a single problem using multiple threads. For example, when the user wants to compute a single matrix multiplication of large size using multiple threads, the user would need a different routine that does the operation in parallel on multiple threads. C-SSL II is not designed for that purpose.

The examples given below handles simple problems so that the reader can understand the principle of using the thread-safety. For the detailed functionality and meanings of arguments of individual routines, please refer to "FUJITSU C-SSL II User's Guide."

Also the examples below do not assume the numbers of threads available. The number of threads is to be specified at execution time via the environment variable OMP_NUM_THREADS. In the C-SSL II, however, the maximum number of threads the user can specify is 128.

It is noted that the examples below could run on a single core even with multiple threads being specified. In order to achieve real parallelism, the user needs as many CPUs or cores as threads.

As for the procedure of compile and link-edit the example programs, please refer to "1.4.4 C-SSL II".

Example 1 : System of Linear Equations

Problem Description

Think of a system of linear equations of order n.

$$\boldsymbol{Ax} = \boldsymbol{b} \qquad (1.1)$$

Here, $\boldsymbol{A}$ is a real matrix of order n, $\boldsymbol{b}$ a right hand side vector of order n, and $\boldsymbol{x}$ the solution vector of order n. Now, let assume we want to solve the equations for different right hand side vectors as shown below.

$$\boldsymbol{Ax}_i = \boldsymbol{b}_i \qquad i = 1, 2, \dots , m \qquad (1.2)$$

Example programs

Let's solve (1.2) for different right hand side vectors by using c_dvlax from C-SSL II. The routine c_dvlax is designed to factor the given matrix ***A*** into LU, and then solve the LU system for ***x*** by backward and forward substitution. The factored LU is returned in the 2-D array "a" that contained the original matrix ***A*** on entry, and the solution ***x*** is returned in the array "b" that contained the right hand side vector ***b*.** For the solution for next right hand side vector ***b*** the user sets up the next ***b*** in array "b" and calls the routine with array "a" unchanged. At the same time, the user is asked to tell the routine if the call is the initial one or subsequent one through the "`isw`".

Suppose we have a system of linear equations of order 100 with 50 right hand side vectors. In order to understand thread-safety of C-SSL II intuitively, let's look at a sequential program at first. The reader should notice how the subsequent solutions are computed.

<u>Using from sequential program</u>

```
#include <stdio.h>
#include <math.h>
#include "cssl.h"

#define N 100
#define K 100
#define M  50

MAIN__()
{
      double a[N][K],b[N],vw[N];
      int ip[N];
      double epsz;
      int n,k,m,i,j,ie,isw,is,icon,n,k;

      n=N;
      k=K;
      m=M;
      epsz=0.0;
/* ================== */
/* Define the matrix  */
/* ================== */
      for(i=0;i<n;i++)
        for(j=0;j<n;j++)
          a[i][j]=....;
/* ================================== */
/* Define the first right hand vector */
/* ================================== */
      for(i=0;i<n;i++)
        b[i]=.......;
      isw=1;
      ie=c_dvlax((double *)a,k,n,b,epsz,isw,&is,vw,ip,&icon);
      if(icon!=0) {
        printf("The given problem seems to be not normal¥n");
        exit(1);
      }
/* ================================================= */
/* Here we have got the LU factors in array a and the  */
/* solution in b[1:n] with respect to the first        */
/* right hand vector.                                  */
/* Now we continue to get solutions for the rest of    */
/* right hand vectors.                                 */
/* ================================================= */
      isw=2;
      for(j=1;j<m;j++) {
/* ================================= */
/* Define the next right hand vector */
/* ================================= */
        for(i=0;i<n;i++) {
          b[i] = .......;
        }
        ie=c_dvlax((double *)a,k,n,b,epsz,isw,&is,vw,ip,&icon);
        .......
      }
      .......
}
```

Next, let's use thread-safe c_dvlax. The procedure of obtaining LU factorization and the solution for the first right hand side vector is the same as before. However, the solutions for subsequent right hand side vectors can be computed concurrently using multiple threads. In order to do so, the set of right hand side vectors need to be set up in prior. In the sample program below, the set of right hand side vectors is put in a 2-D array "b." Furthermore, those arguments which are altered on output need to be assigned memory space for each thread. The parameter `"icon"` is the case when called with `isw`=2. So, we should have an array for `icon`. Other arguments such as

`a`, `k`, `n`, `epsz`, `isw`, `is`, `vw`, `ip`

are not altered when `isw`=2, and therefore can be shared among threads.

Note: Although routine c_dvlax does not alter the contents of work arrays (`vw`, `ip`) when called with `isw`=2, there is need in general that a work array should have different memory space for each thread because it is to be altered inside.

Below is an example OpenMP C program using thread-safe c_dvlax. Some explanations of the code will be given after the listing.

Using from thread-parallelized program

```
#include <stdio.h>
#include <math.h>
#include "cssl.h"

#define K 100
#define N 100
#define M 50

MAIN__()
{
      double a[N][K],b[M][N],vw[N];
      int ip[N];
      double epsz;
      int n,k,m,i,j,ie,isw,is,icon[M];

      n=N;
      k=K;
      m=M;
      epsz=0.0;
/* ================================================== */
/* Define the matrix a and multiple righ hand vectors */
/* ================================================== */
      for(i=0;i<n;i++)
        for(j=0;j<n;j++)
          a[i][j]=....;
      for(i=0;i<m;i++)
        for(j=0;j<n;j++)
           b[i][j]=.......;
      isw=1;
      ie=c_dvlax((double *)a,k,n,b[0],epsz,isw,&is,vw,ip,&icon[0]);
      if(icon[0]!=0) {
        printf("The given problem seems to be not normal¥n");
        exit(1);
      }
/* ================================================== */
/* Here we have got the LU factors in array a and the  */
/* solution in b[1:n] with respect to the first        */
/* right hand vector.                                  */
/* Now we continue to get solutions for the rest of    */
/* right hand vectors.                                 */
/* ================================================== */
      isw=2;
#pragma omp parallel
#pragma omp for schedule(static)
      for(j=1;j<m;j++) {
        ie=c_dvlax((double *)a,k,n,b[j],epsz,isw,&is,vw,ip,&icon[j]);
        .......
      }
      for(j=1;j<m;j++) {
        printf("icon[%d]=%d¥n",j,icon[j]);
      }
      .......
}
```

Explanations

(1) The structured block after `#pragma omp parallel` is executed by multiple threads in parallel. The `#pragma omp parallel` specifies that a team of threads be created at that point. All variables and arrays are shared among the threads in the team in this example code. That means that there exists only one object for each variable and the object can be altered or referenced by any thread.

The OpenMP C allows the user to specify a clause on the "parallel" directive that declares some variables to get private copies. For more information on data scope attributes, see an appropriate documentation on OpenMP (e.g. “OpenMP Application Program Interface, Version 2.5 May 2005”, available from the Web).

(2) The `#pragma omp for schedule(static)` directive specifies that execution of the iterations of the immediately following `for` loop can be executed in parallel and so can be distributed across the threads of the team. The clause `schedule(static)` specifies that iterations are divided into pieces of a size (nearly) equal to the iteration count divided by the number of thread, and the pieces are statically assigned to threads in the team.

(3) What is interesting also is that the above code could be compiled without OpenMP features, link-edited with C-SSL II library, then can work correctly producing the same results, while execution takes longer. This is because the above code is nothing but a regular sequential program when `#pragma omp` directives are ignored. The subsequent examples appearing below have the same nature and so can be regarded as both sequential code and parallel code depending on whether the `#pragma omp` directives are activated.

Example 2 : Quadrature

problem description

Let's compute the following quadrature,

$$\int_0^1 f(p,x)dx\,, \qquad f(p,x) = \frac{1}{x^p} + \sin px \tag{2.1}$$

for different values of parameter $p = i / 10$, $i = 1, 2, \ldots, 9$.

Example program

Subroutine c_daqn9, one of quadrature routines from C-SSL II, can be used for (2.1). The function of the name "fun" to evaluate the integrand must use values of *p*, while those values are controlled in the main program. Because of this, parameter *p* is storage associated through a named common block. It is essential that the variable for *p* must be made private to each thread, so that evaluation of the definite integrals for different values of *p* can be done by multiple threads in parallel. In other words, it is obvious that if the variable were just shared among the threads it should be impossible for the threads to evaluate the integrals concurrently.

```
#include <stdio.h>
#include <math.h>
#include "cssl.h"

#define IP 9

double p;
#pragma omp threadprivate(p)

MAIN__()
{
      double a,b,eps,epsa,epsr;
      double s[IP],err[IP];
      int n[IP],icon[IP];
      int nmin,nmax,i,j;
      double fun(double);

      a=0.0;
      b=1.0;
      eps=c_dmach();
      epsa=(100.0*eps>1.0e-10)?100.0*eps:1.0e-10;
      epsr=eps;
      nmin=21;
      nmax=2000;
      /*=======================================================*/
      /*  Now calculate definite integrals for IP different  */
      /*  functions concurrently.                            */
      /*=======================================================*/
#pragma omp parallel
#pragma omp for schedule(static)
      for(i=0;i<IP;i++) {
          p=(double)(i+1)/10.0;
          c_daqn9(a,b,fun,epsa,epsr,nmin,nmax,
                  &s[i],&err[i],&n[i],&icon[i]);
      }

      /*====================*/
      /*  Print the results  */
      /*====================*/
      printf    ("   No   icon       integral               estimated err¥n");
      for(i=0;i<IP;i++) {
          printf("   %2d   %5d   %25.15le   %10.3le¥n",
                       i,    icon[i],s[i],    err[i]);
      }
}

double fun(double x)
{
      double f;
      f=0.0;
      if(x>0.0) f=pow(x,-p)+sin(p*x);
      return f;
}
```

Upon completion of the above code, it will be noticed that all the elements of `icon[*]` have value 11000. This is just because c_daqn9 has ability of detecting algebraic singularities and has detected successfully them for the given integrands but computed the correct results. (for detailed information on the routine. See "FUJITSU C-SSL II User's Guide".)

Example 3 : Special Functions

Problem description

As the final example, let's consider evaluating the first order Bessel function of the first kind $J_1(x)$ at 300 equally spaced points in the interval $0 \le x \le 10$.

Example program

In the following code, routine c_dbj1 from C-SSL II is used. The code looks straightforward and needs no extra explanation.

```
#include <stdio.h>
#include "cssl.h"

#define IP 301

MAIN__()
{
      double x[IP],f[IP];
      double delta;
      int    icon[IP];
      int    i;
      delta=10.0/(double)(IP-1);
#pragma omp parallel
#pragma omp for schedule(static)
      for(i=0;i<IP;i++) {
          x[i]=delta*(double)i;
          c_dbj1(x[i],&f[i],&icon[i]);
      }
      printf("    x                           icon     J1(x)¥n");
      for(i=0;i<IP;i++) {
          printf("%25.15le   %5d    %25.15le¥n",x[i],icon[i],f[i]);
      }
}
```

1.4 Compile, Link and Run

1.4.1 Setting-up

The user is requested to set up environment variables as follows prior to using the library.

- To compile and link-edit using cross compiler on login node :

  Add /opt/FJSVmxlang/bin to the environment variable PATH.

- To compile and link-edit using own compiler :

  Add /opt/FJSVmxlang/bin to the environment variable PATH.

- To execute on calculation node :

  Add /opt/FJSVmxlang/lib64 to the environment variable LD_LIBRARY_PATH.

When using SSL II/MPI, the following additional set-up is reguired.

- To execute on calculation node :

  Add /opt/FJSVmxlang/bin to the environment variable PATH.

- To use manpages on login node :

  The ssl2(3) and subroutine names(3F) can be used for explanation as manpages. To use these the environment variable MANPATH into which the directory /opt/FJSVmxlang/man must be concatenated.

1.4.2 SSL II

SSL II can be used from user's programs written in Fortran.

This section describes the procedures from compiling the user's program that calls SSL II subroutines to link-editing it with the library. Fujitsu Fortran compiler needs to be used in any case. For details about compilation and link-edit, refer to "Fortran User's Guide".

1.4.2.1 Compile and Link-edit

Use `frtpx` command with `-SSL2` options in order to compile users programs written in Fortran and link-edit with SSL II.

Example 1:

Compile a user's program a.f by cross compiler, and link-edit it with the library of SSL II.

```
frtpx a.f -SSL2
```

When the user's program is written in OpenMP Fortran, specify the option `-Kopenmp` also.

Example 2:

Compile a user's program a.f written in OpenMP Fortran.

```
frtpx -Kopenmp a.f -SSL2
```

Use frt command in order to compile user programs and link-edit with the library of SSL II by using own compiler.

Example 3:

Compile a user's program a.f by own compiler, and link-edit it with the library of SSL II.

```
frt a.f -SSL2
```

1.4.2.2 Notes

- Number of threads

  When the user program is written in OpenMP Fortran using multiple threads, the number of threads can be specified at execution time through the environment variable OMP_NUM_THREADS.

  The maximum number of threads in SSL II is 128.

- Stack size

  When compiled with `-Kopenmp` option, the users data such as arrays are allocated on stack area in certain situations. In that case it might happen that the stack area runs out and the user needs to expand the stack area at execution time. Use the command "`ulimit`", for example, to expand the stack area.

- Address boundaries of arguments

  When using SSL II library in any case, actual arguments of types double precision real and double precision complex which are passed to the SSL II routines are assumed to be aligned on 8 byte address boundaries. So, the user program calling SSL II should be compiled without the option `-AA`.

- Routine names in archives

  Routines of BLAS, LAPACK, C-SSL II and SSL II Thread-Parallel Capabilities are also included in the archives. Besides that, the archives include slave routines having names beginning with SS_ or #L_ (# means S,D,C,Z,I or X). The user is asked to be careful not to duplicate subroutine names with them.

1.4.3 SSL II Thread-Parallel Capabilities

SSL II Thread-Parallel Capabilities are provided as subroutines written in OpenMP Fortran.

This section describes the procedures for compiling the user's program that calls these subroutines, link-editing it with the library which includes SSL II Thread-Parallel Capabilities and executing it as an OpenMP Fortran load module. Fujitsu Fortran compiler needs to be used in any case. For details about compilation and link-edit, refer to "Fortran User's Guide".

1.4.3.1 Compile and Link-edit

SSL II Thread-Parallel Capabilities can be called from the following three types of user's programs. After they are link-edited with the SSL II library, the programs can be executed as OpenMP Fortran load modules.

- OpenMP Fortran programs
- Fortran object programs generated with the automatic parallelization option of the Fortran compiler
- Fortran object programs generated without any parallelizing option, i.e. sequential object programs

The option `-Kopenmp` and `-SSL2` must be specified in the `frtpx` command line for the linker to generate OpenMP Fortran load modules.

`-Kparallel` option can be specified instead of `-Kopenmp` option. Note that, when a user program linkedited with this option is executed, environment variable PARALLEL has priority over OMP_NUM_THERADS.

Example 1:

Compile a user's program written in OpenMP Fortran a.f by cross compiler and link-edit it with the library of SSL II.

```
frtpx -Kopenmp a.f -SSL2
```

Example 2:

Compile a user's program written in OpenMP Fortran a.f by own compiler and link-edit it with the library of SSL II.

```
frt -Kopenmp a.f -SSL2
```

Example 3:

Compile a Fortran program with the automatic parallelization option specified and link-edit it with the library of SSL II to generate an OpenMP Fortran load module.

```
frtpx -Kparallel -c a.f
frtpx -Kopenmp a.o -SSL2
```

Example 4:

Compile a Fortran program without any parallelizing option specified and link-edit it with the library of SSL II to generate an OpenMP Fortran load module.

```
frtpx -c a.f
frtpx -Kopenmp a.o -SSL2
```

1.4.3.2 Run

The number of threads and the stack size per thread can be set at execution time as follows.

- The number of threads which execute in subroutines of SSL II Thread-Parallel Capabilities in parallel can be assigned through the following environment variable.

  Set the environment variable OMP_NUM_THREADS to the number of threads.

- When user's programs calling the subroutines are compiled with the automatic parallelization option specified, the number of threads which execute the automatically parallelized programs can be assigned through the following environment variable.

  Set the environment variable PARALLEL to the number of threads.

- Within a subroutine of SSL II Thread-Parallel Capabilities, a work area for each thread is allocated as automatic allocatable arrays.

  Suppose that the number of threads to be generated is NT and the total amount of the available memory is M, it is recommended that the environment variable THREAD_STACK_SIZE indicating the size of stack area for each thread should be set to about M/(5xNT).

1.4.3.3 Note

- Number of threads

  There is no restriction regarding the number of threads for the parallel execution of SSL II Thread-Parallel subroutines.

- Address boundaries of arguments

  When using SSL II library in any case, actual arguments of types double precision real and double precision complex which are passed to the SSL II routines are assumed to be aligned on 8 byte address boundaries. So, the user program calling SSL II should be compiled without the option `-AA`.

- Routine names in archives

  The subroutines in SSL II Thread-Parallel Capabilities use the internal subroutines, names of which are prefixed by DM_U or DL_.

  Routines of BLAS, LAPACK, and C-SSL II are also included in the archives. Besides that, the archives include slave routines having names beginning with SS_ or #L_ (# means S,D,C,Z,I or X). The user is asked to be careful not to duplicate subroutine names with them.

1.4.4 C-SSL II

C-SSL II can be used from user's programs written in C and C++ language.

This section describes the procedures from compiling the user's program that calls C-SSL II routines to link-editing it with the library. Fujitsu C/C++ compiler needs to be used in any case. For details about compilation and link-edit, refer to "C User's Guide" or "C++ User's Guide".

1.4.4.1 Using C-SSL II from C programs

1) Standard header file
   The C-SSL II is provided with a header file cssl.h which contains prototypes for all the user-callable functions, and other information such as the dcomplex data type definition. Every user program which calls the C-SSL II library must include this header file.

2) Name for the main program
   The function name of the user main program is `main` or `MAIN__` (two underscores after `MAIN`).

3) Compilation and linkage
   Use fccpx command with -SSL2 options in order to compile users programs written in C language and link-edit with C-SSL II.

   Example 1:
   Compile a user's program a.c by cross compiler and link-edit it with the library of C-SSL II.

```
fccpx a.c -SSL2
```

   When the user's program is written in OpenMP C, specify the option `-Kopenmp` also.

   Example 2:
   Compile a user's program a.c written in OpenMP C and link-edit it with the library of C-SSL II.

```
fccpx -Kopenmp a.c -SSL2
```

   Use `fcc` command in order to compile user programs and link-edit with the library of C-SSL II by using own compiler.

   Example 3:
   Compile a user's program a.c by own compiler, and link-edit it with the library of C-SSL II.

```
fcc a.c -SSL2
```

1.4.4.2 Using C-SSL II from C++ programs

1) Standard header file
   Every C++ user program which calls the C-SSL II library must include this header file cssl.h as is the cause with C programs (See previous section).

2) Name for the main program
   The function name of the user main program is `main` or `MAIN__` (two underscores after `MAIN`).

3) Compilation and linkage
   Use `FCCpx` command with `-SSL2` options in order to compile users programs written in C++ language and link-edit with C-SSL II.

Example 1:

Compile a user's program a.cc by cross compiler, and link-edit it with the library of C-SSL II.

```
FCCpx a.cc -SSL2
```

When the user's program is written in OpenMP C, specify the option `-Kopenmp` also.

Example 2:

Compile a user's program a.cc written in OpenMP C and link-edit it with the library of C-SSL II.

```
FCCpx -Kopenmp a.cc -SSL2
```

Use `FCC` command in order to compile user programs and link-edit with the library of C-SSL II by using own compiler.

Example 3:

Compile a user's program a.cc by own compiler, and link-edit it with the library of C-SSL II.

```
FCC a.cc -SSL2
```

1.4.4.3 Note

- Stack size

  The users data such as arrays are allocated on stack area in certain situations. In that case it might happen that the stack area runs out and the user needs to expand the stack area at execution time. Use the command "ulimit", for example, to expand the stack area.

- Thread-safe

  All the C-SSL II routines are thread-safe. Some routines of C-SSL II, however, use an OpenMP C/C++ library routine to make them thread-safe. As a results, the user cannot use C-SSL II routines from C programs parallelized directly using system threads library routine like pthread.

- Number of threads

  When the user program is written in OpenMP C/C++ using multiple threads, the number of threads can be specified at execution time through the environment variable OMP_NUM_THREADS.

  The maximum number of threads in C-SSL II is 128.

- Address boundaries of arguments

  When using C-SSL II library in any case, actual arguments of types double precision real and double precision complex which are passed to the C-SSL II routines are assumed to be aligned on 8 byte address boundaries.

- Routine names in archives

  Routines of BLAS, LAPACK, SSL II and SSL II Thread-Parallel Capabilities are also included in the archives. Besides that, the archives include slave routines having names beginning with SS_ or #L_ (# means S,D,C,Z,I or X). The user is asked to be careful not to duplicate subroutine names with them.

1.4.5 C-SSL II Thread-Parallel Capabilities

C-SSL II Thread-Parallel Capabilities are provided as subroutines written in OpenMP C.

This section describes the procedures for compiling the user's program that calls these routines, link-editing it with the library which includes C-SSL II Thread-Parallel Capabilities and executing it as an OpenMP C load module. Fujitsu C/C++ compiler needs to be used in any case. For details about compilation and link-edit, refer to "C User's Guide" or "C++ User's Guide".

1.4.5.1 Using C-SSL II from C programs

1) Standard header file
The C-SSL II Thread-Parallel capabilities is provided with a header file cssl.h which contains prototypes for all the user-callable routines, and other information such as the dcomplex data type definition. Every user program which calls the C-SSL II Thread-Parallel capabilities library must include this header file.

2) Name for the main program
The function name of the user main program is `main` or `MAIN__` (two underscores after `MAIN`).

3) Compilation and linkage
C-SSL II Thread-Parallel Capabilities can be called from the following three types of user's programs. After they are link-edited with the C-SSL II library, the programs can be executed as OpenMP C load modules.

- OpenMP C programs
- C object programs generated with the automatic parallelization option of the C compiler
- C object programs generated without any parallelizing option, i.e. sequential object programs

When uniting to make the OpenMP C load modules, `-Kopenmp` and `-SSL2` is specified for the `fccpx` or `fcc` command line.

`-Kparallel` option can be specified instead of `-Kopenmp` option. Note that, when a user program linkedited with this option is executed, environment variable PARALLEL has priority over OMP_NUM_THERADS.

Example 1:
Compile a user's program written in OpenMP C a.c by cross compiler and link-edit it with the library of C-SSL II.

```
fccpx -Kopenmp -SSL2 a.c
```

Example 2:
Compile a user's program written in OpenMP C a.c by own compiler and link-edit it with the library of C-SSL II.

```
fcc -Kopenmp -SSL2 a.c
```

Example 3:
Compile a C program with the automatic parallelization option specified and link-edit it with the library of C-SSL II to generate an OpenMP C load module.

```
fccpx -Kparallel -c a.c
fccpx -Kopenmp -SSL2 a.o
```

Example 4:
Compile a C program without any parallelizing option specified and link-edit it with the library of C-SSL II to generate an OpenMP C load module.

```
fccpx -c a.c
fccpx -Kopenmp -SSL2 a.o
```

1.4.5.2 Using C-SSL II from C++ programs

1) Standard header file
Every C++ user program which calls the C-SSL II Thread-Parallel library must include this header file cssl.h as is the cause with C programs (See above).

2) Name for the main program
The function name of the user main program is `main` or `MAIN__` (two underscores after `MAIN`).

3) Compilation and linkage
C-SSL II Thread-Parallel Capabilities can be called from the following three types of user's programs. After they are link-edited with the C-SSL II library, the programs can be executed as OpenMP C++ load modules.

- OpenMP C++ programs
- C++ object programs generated with the automatic parallelization option of the C++ compiler
- C++ object programs generated without any parallelizing option, i.e. sequential object programs

When uniting to make the OpenMP C load modules, `-Kopenmp` and `-SSL2` is specified for the `FCCpx` or `FCC` command line.

`-Kparallel` option can be specified instead of `-Kopenmp` option. Note that, when a user program linkedited with this option is executed, environment variable PARALLEL has priority over OMP_NUM_THERADS.

Example 1:
Compile a user's program written in OpenMP C++ a.cc by cross compiler and link-edit it with the library of C-SSL II.

```
FCCpx -Kopenmp -SSL2 a.cc
```

Example 2:
Compile a user's program written in OpenMP C++ a.cc by own compiler and link-edit it with the library of C-SSL II.

```
FCC -Kopenmp -SSL2 a.cc
```

Example 3:
Compile a C++ program with the automatic parallelization option specified and link-edit it with the library of C-SSL II to generate an OpenMP C++ load module.

```
FCCpx -Kparallel -c a.cc
FCCpx -Kopenmp -SSL2 a.o
```

Example 4:
Compile a C++ program without any parallelizing option specified and link-edit it with the library of C-SSL II to generate an OpenMP C++ load module.

```
FCCpx -c a.cc
FCCpx -Kopenmp -SSL2 a.o
```

1.4.5.3 Run

The number of threads and the stack size per thread can be set at execution time as follows.

- The number of threads which execute in subroutines of C-SSL II Thread-Parallel Capabilities in parallel can be assigned through the following environment variable.

Set the environment variable OMP_NUM_THREADS to the number of threads.

- When user's programs calling the subroutines are compiled with the automatic parallelization option specified, the number of threads which execute the automatically parallelized programs can be assigned through the following environment variable.

  Set the environment variable PARALLEL to the number of threads.

- Within a subroutine of C-SSL II Thread-Parallel Capabilities , a work area for each thread is allocated as automatic allocatable arrays.

  Suppose that the number of threads to be generated is NT and the total amount of the available memory is M, it is recommended that the environment variable THREAD_STACK_SIZE indicating the size of stack area for each thread should be set to about M/(5xNT).

1.4.5.4 Note

- Number of threads

  There is no restriction regarding the number of threads for the parallel execution of C-SSL II Thread-Parallel routines.

- Address boundaries of arguments

  When using C-SSL II library in any case, actual arguments of types double precision real and double precision complex which are passed to the C-SSL II routines are assumed to be aligned on 8 byte address boundaries.

- Routine names in archives

  Routines of BLAS, LAPACK, SSL II and SSL II Thread-Parallel Capabilities are also included in the archives. Besides that, the archives include slave routines having names beginning with SS_ or #L_ (# means S,D,C,Z,I or X). The user is asked to be careful not to duplicate subroutine names with them.

1.4.6 SSL II/MPI

SSL II/MPI provides hybrid parallel subroutines parallelized in use of not only MPI but also OpenMP Fortran.

It is necessary to compile a user's program calling SSL II/MPI subroutines, linkedit with the internal routines in SSL II Thread-parallel capabilities and execute as a Fortran load module using MPI. This procedure is explained below.

Regarding the detail procedure for compilation to execution, also refer to "Fortran User's Guide" and "MPI User's Guide".

1.4.6.1 Compile and Link-edit

Compilation should be done using Fujitsu Fortran compiler.

An mpifrtpx command is used in order to compile a user's program using SSL II/MPI and linkedit.

Options of `-Kopenmp`, `-SSL2MPI` and either `-SSL2` or `-SSL2BLAMP` must be specified in an mpifrt command line.

And `-Kopenmp` option is needed for linkediting because SSL II/MPI routines are hybrid-parallel routines in use of MPI and OpenMP Fortran.

The internal routines in SSL II Thread-parallel capabilities used in SSL II/MPI can be linkedited specifying either `-SSL2` or `-SSL2BLAMP`. The option, either -SSL2 or -SSL2BLAMP should be selected depending upon the choice of either sequential BLAS and LAPACK or parallelized BLAS and LAPACK. Namely when a routine in BLAS and LAPACK in used together with that in SSL II/MPI in a user's program, please select `-SSL2` or `-SSL2BLAMP` to linkedit sequential or parallelized BLAS and LAPACK.

`-Kparallel` option can be specified instead of `-Kopenmp` option. Note that, when a user program linkedited with this option is executed, environment variable PARALLEL has priority over OMP_NUM_THERADS.

Example 1:

A user program using MPI is compiled and linkedited with the library for SSL II/MPI library and sequential version of BLAS and LAPACK.

```
mpifrtpx -Kopenmp -SSL2MPI -SSL2 a.f
```

Example 2:

A user program using MPI is compiled and linkedited with the library for SSL II/MPI library and thread-parallel version of BLAS and LAPACK.

```
mpifrtpx -Kopenmp -SSL2MPI -SSL2BLAMP a.f
```

Example 3:

After a user program using MPI is compiled to generate an object module, it is linkedited with the library for SSL II/MPI. Also it is linkedited with thread-parallel version of BLAS and LAPACK.

```
mpifrtpx -c a.f
mpifrtpx -Kopenmp -SSL2MPI -SSL2BLAMP a.o
```

Use mpifrt command in order to compile user programs and link-edit with the library of SSL II/MPI by using own compiler.

Example 4:

A user program using MPI is compiled and linkedited with the library for SSL II/MPI library and sequential version of BLAS and LAPACK.

```
mpifrt -Kopenmp -SSL2MPI -SSL2 a.f
```

Example 5:

A user program using MPI is compiled and linkedited with the library for SSL II/MPI library and thread-parallel version of BLAS and LAPACK.

```
mpifrt -Kopenmp -SSL2MPI -SSL2BLAMP a.f
```

Example 6:

After a user program using MPI is compiled to generate an object module, it is linkedited with the library for SSL II/MPI. Also it is linkedited with thread-parallel version of BLAS and LAPACK.

```
mpifrt -c a.f
mpifrt -Kopenmp -SSL2MPI -SSL2BLAMP a.o
```

1.4.6.2 Using SSL II/MPI from C, C++ programs

SSL II/MPI can be used from user's program written in C/C++ language. Compilation should be done using Fujitsu C/C++ compiler.

mpifccpx or mpiFCCpx command is used in order to compile a user's program using SSL II/MPI and linkedit.

Options of `-Kopenmp`, `-SSL2MPI` and either `-SSL2` or `-SSL2BLAMP` must be specified in an mpifcc/mpiFCC command line.

And `-Kopenmp` option is needed for linkediting because SSL II/MPI routines are hybrid-parallel routines in use of MPI and OpenMP Fortran.

The internal routines in SSL II Thread-parallel capabilities used in SSL II/MPI can be linkedited specifying either `-SSL2` or `-SSL2BLAMP`. The option, either `-SSL2` or `-SSL2BLAMP` should be selected depending upon the choice of either sequential BLAS and LAPACK or parallelized BLAS and LAPACK. Namely when a routine in BLAS and LAPACK in used together with that in SSL II/MPI in a user's program, please select `-SSL2` or `-SSL2BLAMP` to linkedit sequential or parallelized BLAS and LAPACK.

`-Kparallel` option can be specified instead of `-Kopenmp` option. Note that, when a user program linkedited with this option is executed, environment variable PARALLEL has priority over OMP_NUM_THERADS.

Example 1:
A user program using MPI is compiled and linkedited with the library for SSL II/MPI library and sequential version of BLAS and LAPACK.

```
mpifccpx -Kopenmp -SSL2MPI -SSL2 a.c
```

Example 2:
A user program using MPI is compiled and linkedited with the library for SSL II/MPI library and thread-parallel version of BLAS and LAPACK.

```
mpifccpx -Kopenmp -SSL2MPI -SSL2BLAMP a.c
```

Example 3:
After a user program using MPI is compiled to generate an object module, it is linkedited with the library for SSL II/MPI. Also it is linkedited with thread-parallel version of BLAS and LAPACK.

```
mpifccpx -c a.c
mpifccpx -Kopenmp -SSL2MPI -SSL2BLAMP a.o
```

Example 4:
A user program written in C++ using MPI is compiled and linkedited with the library for SSL II/MPI library and thread-parallel version of BLAS and LAPACK.

```
mpiFCCpx -Kopenmp -SSL2MPI -SSL2BLAMP a.cpp
```

Use mpifcc or mpiFCC command in order to compile user programs and link-edit with the library of SSL II/MPI by using own compiler.

Example 5:
A user program using MPI is compiled and linkedited with the library for SSL II/MPI library and sequential version of BLAS and LAPACK.

```
mpifcc -Kopenmp -SSL2MPI -SSL2 a.c
```

Example 6:
A user program using MPI is compiled and linkedited with the library for SSL II/MPI library and thread-parallel version of BLAS and LAPACK.

```
mpifcc -Kopenmp -SSL2MPI -SSL2BLAMP a.c
```

Example 7:
After a user program using MPI is compiled to generate an object module, it is linkedited with the library for SSL II/MPI. Also it is linkedited with thread-parallel version of BLAS and LAPACK.

```
mpifcc -c a.c
mpifcc -Kopenmp -SSL2MPI -SSL2BLAMP a.o
```

Example 8:

A user program written in C++ using MPI is compiled and linkedited with the library for SSL II/MPI library and thread-parallel version of BLAS and LAPACK.

```
mpiFCC -Kopenmp -SSL2MPI -SSL2BLAMP a.cpp
```

Note the points below. These are required so that a C program can be properly linked with Fortran using current C and Fortran compilers.

See "C User's Guide" for detail.

- The function name of the user main program is `main` or `MAIN__` (two underscores after `MAIN`).
- An underscore must be added to the end of the subroutine name of SSL II/MPI.
- Variable parameters except for arrays must begin with an ampersand.
- For arrays of multi dimensions, the storage sequence of each array element is different between Fortran and C. For SSL II/MPI capabilities that handle array of multi dimensions, the arrays are calculated using the storage sequence of Fortran.

1.4.6.3 Run

A user's program linkedited with SSL II/MPI is executed in use of an `mpiexec` command. The number of threads for parallel execution is specified in an environment variable OMP_NUM_THREADS. The number of cores needed is assumedly n*s when the number of processes n and that of threads s are specified.

The number of threads must be set to one when the routine is executed in process parallel only.

Example:

An a.out is executed setting the number of processes to two and that of threads to four.

```
setenv OMP_NUM_THREADS 4
mpiexec -n 2 a.out
```

Within a subroutine of SSL II Thread-Parallel Capabilities, a work area for each thread is allocated as automatic allocatable arrays.

Suppose that the number of threads to be generated is NT and the total amount of the available memory is M, it is recommended that the environment variable THREAD_STACK_SIZE indicating the size of stack area for each thread should be set to about M/(5*NT).

1.4.6.4 Note

- Address boundaries of arguments

  When using SSL II library in any case, actual arguments of types double precision real and double precision complex which are passed to the SSL II routines are assumed to be aligned on 8 byte address boundaries. So, the user program calling SSL II should be compiled without the option `-AA`.

- Routine names in archives

  The subroutines in SSL II/MPI use the internal subroutines, names of which are prefixed by DS_U.

  Routines of BLAS, LAPACK, C-SSL II, and SSL II Thread-Parallel Capabilities are also included in the archives. Besides that, the archives include slave routines having names beginning with SS_ or #L_ (# means S,D,C,Z,I or X). The user is asked to be careful not to duplicate subroutine names with them.

2. BLAS, LAPACK, ScaLAPACK

2.1 Overview

BLAS, LAPACK and ScaLAPACK are the implementations of the public domain BLAS (Basic Linear Algebra Subprograms), LAPACK (Linear Algebra PACKage) and ScaLAPACK (Scalable LAPACK) , which have been developed by groups of people such as Prof. Jack Dongarra, University of Tennessee, USA and all published on the WWW (URL: http://www.netlib.org/).

The BLAS, LAPACK and ScaLAPACK library has the following features.

- The BLAS library has been improved in performance. Especially, the SPARC64 XIfx binary version are the result of extensive source code tuning by taking into account the CPU architecture. The performance-up of BLAS also has contributed to that of LAPACK and ScaLAPACK because they call BLAS routines.
- The BLAS and LAPACK include both sequential version and thread-parallel version.
- The routines of BLAS and LAPACK sequential version are thread-safe , and can be called not only from sequential Fortran programs but also from thread-parallel programs written with OpenMP Fortran API. In the latter case, a routine can be called from multiple threads concurrently with different input data being given from each thread.
- PLASMA, which is parallelized LAPACK by pthread, is provided. Some routines are faster than LAPACK parallelized by OpenMP. Note that PLASMA is not thread-safe.
- The routines of BLAS and LAPACK thread-parallel version are parallelized with the OpenMP Fortran. The parallel library is aimed at those subroutines which are likely to be used by large scale scientific applications and can be speeded up by parallel algorithms. The routines are thread-safe too.
- ScaLAPACK can be link-edited with BLAS and LAPACK thread-parallel version. When the thread-parallel version is linked, ScaLAPACK routines are executed in parallel by multiple processes of MPI in each of which BLAS and LAPACK routines that are called from ScaLAPACK routines are executed in parallel by multiple threads.

2.2 Compile, Link and Run

BLAS and LAPACK are to be called from a Fortran/C/C++ program. Compilation should be done using Fujitsu Fortran/C/C++ compiler. ScaLAPACK is to be used from a Fortran/C/C++ program with MPI library calls. This section describes procedures of compilation through execution for these parallel programs, while more details are available in manuals *Fortran User's Guide* or *MPI User's Guide*.

For users calling ScaLAPACK, BLAS and LAPACK at the same time, follow information given in “2.2.4 ScaLAPACK”, then BLAS or LAPACK routines link-edited with ScaLAPACK product can be used.

2.2.1 Preparation

Prior to using BLAS, LAPACK and ScaLAPACK products, the following set-up is required.

- To compile and link-edit using cross compiler on login node :

  Add /opt/FJSVmxlang/bin to the environment variable PATH.

- To compile and link-edit using own compiler :

Add /opt/FJSVmxlang/bin to the environment variable PATH.

- To execute on calculation node :

  Add /opt/FJSVmxlang/lib64 to the environment variable LD_LIBRARY_PATH.

When using ScaLAPACK, the following additional set-up is required.

- To execute on calculation node :

  Add /opt/FJSVmxlang/bin to the environment variable PATH.

2.2.2 BLAS, LAPACK Sequential version

a) Link-editing of libraries

Use `frtpx` command with `-SSL2` options in order to compile user programs written in Fortran and link-edit with sequential version of BLAS and LAPACK.

Example 1:

Compile a user's program a.f, and link-edit it with the archive of BLAS and LAPACK.

```
frtpx a.f –SSL2
```

When the user's program is written in OpenMP Fortran, specify the option `–Kopenmp` also.

Example 2:

Compile a user's program a.f written in OpenMP Fortran and link-edit it with the archive of BLAS and LAPACK.

```
frtpx –Kopenmp a.f –SSL2
```

Use `frt` command in order to compile user programs and link-edit with the archive of BLAS and LAPACK by using own compiler.

Example 3:

Compile a user's program a.f and link-edit it with the archive of BLAS and LAPACK by using own compiler.

```
frt a.f -SSL2
```

b) Notes on stacksize

When compiling with `-Kopenmp` option, local arrays other than the arrays of main program are allocated in stack area, and also the stack area is used for the internal work array of BLAS or LAPACK routine, so it might happen the stack area runs out of space. In that case, the user can expand the stack area by the command ulimit. For details about the internal work area used by BLAS or LAPACK routines, refer to “2.3.5 Internal work area used for BLAS, LAPACK.”

c) Specifying the number of threads

When the user program is written to use multiple threads, the number of threads to be created can be specified by the environment variable OMP_NUM_THREADS.

2.2.3 BLAS, LAPACK Thread-Parallel version

a) Link-editing of libraries

Use `frtpx` or `frt` command with `-Kopenmp`, `-SSL2BLAMP` options in order to compile user programs written in Fortran and link-edit with thread-parallel version of BLAS and LAPACK(include PLASMA).

`-Kparallel` option can be specified instead of `-Kopenmp` option. See “2.3.7 Notes on link-editing BLAS, LAPACK thread-parallel version with `-Kparallel` option”.

Example 1:

Compile a user's program a.f written in OpenMP Fortran by cross compiler, and link-edit it with the archive of BLAS and LAPACK.

```
frtpx -Kopenmp a.f -SSL2BLAMP
```

Example 2:

Compile a user's program a.f written in OpenMP Fortran by own compiler, and link-edit it with the archive of BLAS and LAPACK.

```
frt -Kopenmp a.f -SSL2BLAMP
```

Example 3:

Compile a user's program a.f written in Fortran as a sequential object module, and link-edit it with the archive of BLAS and LAPACK.

```
frtpx -c a.f

frtpx –Kopenmp a.o –SSL2BLAMP
```

Example 4:

Compile a user's program a.f written in Fortran using the automatic parallelization facility, and link-edit it with the archive of BLAS and LAPACK.

```
frtpx -c -Kparallel a.f
frtpx –Kopenmp a.o –SSL2BLAMP
```

b) Note on stacksize

When compiling with `-Kopenmp` option, local arrays other than the arrays of main program are allocated in the stack area, and also the stack area is used for internal work arrays of BLAS or LAPACK routine, so it might happen the stack area runs out of space. In that case, the user can expand the stack area by the command ulimit. For details about the internal work area used by BLAS or LAPACK, refer to "2.3.5 Internal work area used for BLAS, LAPACK."

c) Specifying the number of threads

The number of threads to be created by BLAS and LAPACK parallelized by OpenMP can be specified by the environment variable OMP_NUM_THREADS.

The number of threads to be created by PLASMA can be specified by the argument of the routine PLASMA_Init.

d) Environment variable to execute PLASMA

Set 1 to environment variable FLIB_PTHREAD in order to execute user program which calls PLASMA routines.

2.2.4 ScaLAPACK

a) Link-editing of libraries

ScaLAPACK can be used by linking it with user programs written in Fortran language. It is linked to the user program when `-SCALAPACK` and either `-SSL2` or `-SSL2BLAMP` are specified on the `mpifrtpx` or `mpifrt` command line. When `-SSL2BLAMP` is specified, `-Kopenmp` option is also needed.

Example 1:

Compile a user's program a.f by cross compiler and link-edit it with archives. The sequential version of BLAS and LAPACK are link-edited.

```
mpifrtpx a.f -SCALAPACK -SSL2
```

Example 2:

Compile a user's program a.f by own compiler and link-edit it with archives. The sequential version of BLAS and LAPACK are link-edited.

```
mpifrt a.f -SCALAPACK -SSL2
```

Example 3:

Compile a user's program a.f and link-edit it with archives. The thread-parallel version of BLAS and LAPACK are link-edited.

```
mpifrtpx -c a.f
mpifrtpx –Kopenmp a.o –SCALAPACK -SSL2BLAMP
```

b) Executing a program

Applications, which call ScaLAPACK, are regarded as MPI-based message passing programs. The way of executing such applications is documented in detail in the manual *MPI User's Guid*.

When thread-parallel version of BLAS, LAPACK archive is linked, the number of threads is to be specified at execution time via the environment variable OMP_NUM_THREADS. When the number of processes is n and the number of threads is s, $n \times s$ CPUs are needed.

Example 4:

Execute a program a.out on 4 processes.

```
mpiexec -n 4 a.out
```

Example 5:

Execute a program a.out. The number of processes is 2 and the number of threads is 4.

```
setenv OMP_NUM_THREADS 4

mpiexec -n 2 a.out
```

2.2.5 Using BLACS with C Interface

BLACS also includes C interface routines. C interface BLACS is linked to the user program when `-lCblacs` is specified on the `mpifccpx` command line.

Example:

```
mpifccpx a.c –lCblacs
```

The program that unites with BLACS in C interface is executed by the `mpiexec` command as well as the program that unites ScaLAPACK.

When BLACS in C interface is used, it is necessary to note the following.

- `MPI_Init`, which is an MPI initialize function, must be called in the user main program.
- C interface BLACS is used when all user programs are written in C. If some of user programs are written in Fortran, the user is just asked to follow description in “2.2.4 ScaLAPACK”.

2.2.6 Using BLAS, LAPACK and ScaLAPACK from C/C++ programs

BLAS, LAPACK and ScaLAPACK can also be used by linking it with user programs written in C/C++.

2.2.6.1 BLAS, LAPACK sequential version

Use `fccpx`, `FCCpx`, `mpifccpx`, `mpiFCCpx`, `fcc`, `FCC`, `mpifcc`, or `mpiFCC` command with `-SSL2` option in order to compile user programs written in C/C++ and link-edit with sequential version of BLAS and LAPACK.

Example 1:

Compile a user's program a.c by cross compiler and link-edit it with the archive of BLAS and LAPACK.

```
fccpx –SSL2 a.c
```

Example 2:

Compile a user's program a.c by own compiler and link-edit it with the archive of BLAS and LAPACK.

```
fcc –SSL2 a.c
```

Example 3:

Compile a user's program a.cpp written in C++ and link-edit it with the archive of BLAS and LAPACK.

```
FCCpx a.cpp –SSL2
```

-Xg option is needed in order to compile the C++ program which calls the C interface LAPACK routine.

Example 4:

Compile a user's program a.cpp written in C++ and link-edit it with the archive of BLAS and LAPACK. It calls C interface LAPACK routine.

```
FCCpx –Xg a.cpp –SSL2
```

2.2.6.2 BLAS, LAPACK thread-parallel version

Use `fccpx`, `FCCpx`, `mpifccpx`, `mpiFCCpx`, `fcc`, `FCC`, `mpifcc` or `mpiFCC` command with `-Kopenmp` and `-SSL2BLAMP` options in order to compile user programs written in C/C++ and link-edit with thread-parallel version of BLAS and LAPACK (include PLASMA).

The followings are examples of fccpx or fcc command. fccpx or fcc command can be replaced with other command which user want to use.

Example 1:

Compile a user's program a.c written in OpenMP C by cross compiler, and link-edit it with the archive of BLAS and LAPACK.

```
fccpx –Kopenmp a.c -SSL2BLAMP
```

Example 2:

Compile a user's program a.c written in OpenMP C by own compiler, and link-edit it with the archive of BLAS and LAPACK.

```
fcc –Kopenmp a.c -SSL2BLAMP
```

Example 3:

Compile a user's program a.c written in C as a sequential object module, and link-edit it with the archive of BLAS and LAPACK.

```
fccpx -c a.c
fccpx -Kopenmp a.o -SSL2BLAMP
```

Example 4:

Compile a user's program a.c written in C using the automatic parallelization facility, and link-edit it with the archive of BLAS and LAPACK.

```
fccpx -c -Kparallel a.c
fccpx -Kparallel,openmp a.o -SSL2BLAMP
```

Example 5:

Compile a user's program a.cpp written in C++ and link-edit it with the archive of BLAS and LAPACK.

```
FCCpx -Kopenmp a.cpp -SSL2BLAMP
```

-Xg option is needed in order to compile the C++ program which calls the C interface LAPACK routine.

Example 6:

Compile a user's program a.cpp written in C++ and link-edit it with the archive of BLAS and LAPACK.

```
FCCpx -Kopenmp –Xg a.cpp -SSL2BLAMP
```

`-Kparallel` option can be specified instead of `-Kopenmp` option. See "2.3.7 Notes on link-editing BLAS, LAPACK thread-parallel version with `-Kparallel` option".

2.2.6.3 ScaLAPACK

ScaLAPACK can be used by linking it with user programs written in C/C++ language. It is linked to the user program when `-SCALAPACK` and either `-SSL2` or `-SSL2BLAMP` are specified on the `mpifccpx`, `mpiFCCpx`, `mpifcc` or `mpiFCC` command line. When `-SSL2BLAMP` is specified, `-Kopenmp` option is also needed.

The followings are examples of `mpifccpx` or `mpifcc` command. `mpiFCCpx` command can be replaced with other command which user want to use.

Example 1:

Compile a user's program a.c by cross compiler and link-edit it with ScaLAPACK and sequential version of BLAS and LAPACK.

```
mpifccpx a.c -SSL2 –SCALAPACK
```

Example 2:

Compile a user's program a.c by own compiler and link-edit it with ScaLAPACK and sequential version of BLAS and LAPACK.

```
mpifcc a.c -SSL2 –SCALAPACK
```

Example 3:

Compile a user's program a.c and link-edit it with ScaLAPACK and thread-parallel version of BLAS and LAPACK.

```
mpifccpx -c a.c
mpifccpx –Kopenmp a.o -SSL2BLAMP –SCALAPACK
```

Example 4:

Compile a user's program a.cpp and link-edit it with ScaLAPACK and thread-parallel version of BLAS and LAPACK.

```
mpiFCCpx -Kopenmp a.cpp -SSL2BLAMP –SCALAPACK
```

2.2.6.4 Notes

Note the points below. These are required so that a C program can be properly linked with Fortran using current C and Fortran compilers.

See “C User's Guide” for detail.

- The function name of the user main program is `main` or `MAIN__` (two underscores after `MAIN`).
- An underscore must be added to the end of the subroutine name.
- Variable parameters except for arrays must begin with an ampersand.
- For arrays of two dimensions, the storage sequence of each array element is different between FORTRAN and C. For BLAS and LAPACK capabilities that handle array of two dimensions, the arrays are calculated using the storage sequence of FORTRAN.

A header file `fj_lapack.h`, which contains prototypes for the user-accessible routines, is provided. The following notes are for using it.

- It contains prototypes for all the BLAS routines (for XBLAS only C interface prototypes are included), all the driver and computational routines of LAPACK and some auxiliary routines of LAPACK.
- `fcomplex` and `dcomplex` are defined as the types corresponding to complex and double-precision complex.

```
typedef struct {
   float   re, im;
} fcomplex;


typedef struct {
   double  re, im;
} dcomplex;
```

- A two dimensional array must be recast as a pointer in calls to a library routines. For example, `(double *)` is needed before the argument name of double precision two dimensional array.
- If the actual argument is of type character, the length is passed after the address of the arguments.
- The function result is set with Fortran and the function of the complex type is set to the function value.

  Example

```
z=zdotu_(&n,x,&incx,y,&incy);
```

C interface routines to LAPACK are available. User program which calls C interface routines must include a header file `lapacke.h`. The prototypes for interfaces, type definitions for complex data types and array argumenst specifying whether the arrays are stored in row-major or column-major order are contained in it.

2.2.7 Using shared libraries

The libraries of BLAS, LAPACK and ScaLAPACK are statically linked in default.

Shared libraries are also provided. Statically linked libraries are useful for general purpose, but they can't be used in some situation e.g. the user program loads the libraries dynamically using dlopen and dlsym function. In such case, the shared libraries can be used. The library structure and the operation for link-editing of shared libraries are different from those of static library to use them by dynamic loading.

2.2.7.1 Using BLAS and LAPACK sequential version from Fortran program

Use `frtpx` or `frt` command with `-lfjlapack` options in order to compile user programs written in Fortran and link-edit with sequential version of BLAS and LAPACK. Write `-lfjlapack` option after the Fortran source and object names.

Example 1:

Compile a user's program a.f and link-edit it with the library of BLAS and LAPACK sequential version.

```
frtpx a.f -lfjlapack
```

Example 2:

Compile a user's program a.f written in OpenMP Fortran and link-edit it with the archive of BLAS and LAPACK.

```
frtpx -Kopenmp a.f -lfjlapack
```

Example 3:

Compile a user's program a.f and link-edit it with the archive of BLAS and LAPACK by using own compiler.

```
frt a.f -lfjlapack
```

2.2.7.2 Using BLAS and LAPACK thread-parallel version from Fortran program

Use `frtpx` or `frt` command with `-Kopenmp`, `-lfjlapackex` options in order to compile user programs written in Fortran and link-edit with thread-parallel version of BLAS and LAPACK(include PLASMA). Write `-lfjlapackex` option after the Fortran source and object names.

`-Kparallel` option can be specified instead of `-Kopenmp` option. See “2.3.7 Notes on link-editing BLAS, LAPACK thread-parallel version with `-Kparallel` option”.

Example 1:

Compile a user's program a.f written in OpenMP Fortran, and link-edit it with the BLAS and LAPACK library.

```
frtpx -Kopenmp a.f -lfjlapackex
```

Example 2:

Compile a user's program a.f written in OpenMP Fortran by own compiler, and link-edit it with the BLAS and LAPACK library.

```
frt -Kopenmp a.f -lfjlapackex
```

Example 3:

Compile a user's program a.f written in Fortran as a sequential object module, and link-edit it with the BLAS and LAPACK library.

```
frtpx -c a.f
frtpx -Kopenmp a.o -lfjlapackex
```

Example 4:

Compile a user's program a.f written in Fortran using the automatic parallelization facility, and link-edit it with the BLAS and LAPACK library.

```
frtpx -c -Kparallel a.f
frtpx -Kopenmp a.o -lfjlapackex
```

2.2.7.3 Using ScaLAPACK from Fortran program

ScaLAPACK can be used by linking it with user programs written in Fortran language. It is linked to the user program when `-lfjscalapack` and either `-lfjlapack` or `-lfjlapackex` are specified on the `mpifrtpx` or `mpifrt` command line.

Write `-lfjscalapack` option before `-lfjlapack` or `-lfjlapackex` option after the Fortran source and object names.

When `-lfjlapackex` is specified, `-Kopenmp` option is also needed.

Example 1:

Compile a user's program a.f and link-edit it with ScaLAPACK and sequential version of BLAS and LAPACK.

```
mpifrtpx a.f -lfjscalapack -lfjlapack
```

Example 2:

Compile a user's program a.f and link-edit it with ScaLAPACK and thread parallel version of BLAS and LAPACK.

```
mpifrtpx -Kopenmp a.f -lfjscalapack -lfjlapackex
```

Example 3:

Compile a user's program a.f by own compiller and link-edit it with ScaLAPACK and thread parallel version of BLAS and LAPACK.

```
mpifrt -Kopenmp a.f -lfjscalapack -lfjlapackex
```

2.2.7.4 Using BLAS and LAPACK sequential version from C/C++ programs

Use `fccpx`, `FCCpx` `fcc` or `FCC` command with `-lfjlapack` and either `-SSL2` or `-SSL2BLAMP` option in order to compile user programs written in C/C++ and link-edit with sequential version of BLAS and LAPACK.

Write `-lfjlapack` option after the Fortran source and object names.

Example 1:

Compile a user's program a.c written in C and link-edit it with the library of BLAS and LAPACK sequential version.

```
fccpx a.c -lfjlapack -SSL2
```

Example 2:

Compile a user's program a.cpp written in C++ and link-edit it with the library of BLAS and LAPACK sequential version.

```
FCCpx a.cpp -lfjlapack -SSL2
```

Example 3:

Compile a user's program a.c written in OpenMP C and link-edit it with the library of BLAS and LAPACK sequential version.

```
fccpx -Kopenmp a.c -lfjlapack -SSL2
```

Example 4:

Compile a user's program a.c by own compiler and link-edit it with the library of BLAS and LAPACK sequential version.

```
fcc -Kopenmp a.c -lfjlapack -SSL2
```

`-Xg` option is needed in order to compile the C++ program which calls the C interface LAPACK routine.

Example 5:

Compile a user's program a.cpp written in C++ and link-edit it with the library of BLAS and LAPACK sequential version. It calls C interface LAPACK routine.

```
FCCpx -Xg a.cpp -lfjlapack -SSL2
```

2.2.7.5 Using BLAS and LAPACK thread parallel version from C/C++ programs

Use `fccpx`, `FCCpx` `fcc` or `FCC` command with `-Kopenmp`, `-lfjlapack` and either `-SSL2` or `-SSL2BLAMP` option in order to compile user programs written in C/C++ and link-edit with thread parallel version of BLAS and LAPACK(include PLASMA). Write `-lfjlapack` option after the Fortran source and object names.

`-Kparallel` option can be specified instead of `-Kopenmp` option. See “2.3.7 Notes on link-editing BLAS, LAPACK thread-parallel version with `-Kparallel` option”.

Example 1:

Compile a user's program a.c written in C and link-edit it with the library of BLAS and LAPACK thread parallel version.

```
fccpx -Kopenmp a.c -lfjlapackex -SSL2
```

Example 2:

Compile a user's program a.cpp written in C++ and link-edit it with the library of BLAS and LAPACK thread parallel version.

```
FCCpx -Kopenmp a.cpp -lfjlapackex -SSL2
```

Example 3:

Compile a user's program a.c written in C as a sequential object module, and link-edit it with the library of BLAS and LAPACK thread parallel version.

```
fccpx -c a.c
fccpx -Kopenmp a.o -lfjlapackex -SSL2
```

Example 4:

Compile a user's program a.c written in C using the automatic parallelization facility, and linkedit it with the library of BLAS and LAPACK thread parallel version.

```
fccpx -c -Kparallel a.c
fccpx -Kparallel,openmp a.o -lfjlapackex -SSL2
```

Example 5:

Compile a user's program a.c written in C and link-edit it with the library of BLAS and LAPACK thread parallel version by own compiler.

```
fcc -Kopenmp a.c -lfjlapackex -SSL2
```

`-Xg` option is needed in order to compile the C++ program which calls the C interface LAPACK routine.

Example 6:

Compile a user's program a.cpp written in C++ and link-edit it with the library of BLAS and LAPACK thread parallel version.

```
FCCpx -Xg -Kopenmp a.cpp -lfjlapackex -SSL2
```

2.2.7.6 Using ScaLAPACK from C/C++ programs

ScaLAPACK can be used by linking it with user programs written in C/C++ language.

It is linked to the user program when `-lfjscalapack`, either `-lfjlapack` or `-lfjlapackex`, either `-SSL2` or `-SSL2BLAMP` and `-SCALAPACK` are specified on the `mpifccpx`, `mpiFCCpx`, `mpifcc` or `mpiFCC` command line. Write `-lfjlapack` option before `-lfjlapack` or `-lfjlapackex` after the Fortran source and object names. When `-lfjlapackex` is specified, `-Kopenmp` option is also needed.

Example 1:

Compile a user's program a.c and link-edit it with ScaLAPACK and sequential version of BLAS and LAPACK.

```
mpifccpx a.c -lfjscalapack -lfjlapack -SSL2 -SCALAPACK
```

Example 2:

Compile a user's program a.cpp writen in C++ and link-edit it with ScaLAPACK and thread-parallel version of BLAS and LAPACK.

```
mpiFCCpx -Kopenmp a.cpp -lfjscalapack -lfjlapackex -SSL2
-SCALAPACK
```

Example 3:

Compile a user's program a.c by own compiler and link-edit it with ScaLAPACK and thread parallel version of BLAS and LAPACK.

```
mpifcc -Kopenmp a.c -lfjscalapack -lfjlapackex -SSL2 -SCALAPACK
```

2.2.7.7 Using BLAS, LAPACK or ScaLAPACK by dynamic loading

When the BLAS, LAPACK or ScaLAPACK library is called from user program as dynamic loading using dlopen and dlsym functions, the file name of library is specified in parameter of dlopen function. They are shown in Table 1.

Table 1 The file name of shared library

Library	File name
BLAS, LAPACK sequential version	libfjlapack.so
BLAS, LAPACK thread parallel version	libfjlapackex.so
ScaLAPACK	libfjscalapack.so

When the BLAS, LAPACK and ScaLAPACK library is used by dynamic loading `-lfjlapack`, `-lfjlapackex` or `-lfjscalapack` options aren't needed.

When thread parallel version of BLAS, LAPACK is used, `-Kopenmp` option is also needed.

`-SSL2` or `-SSL2BLAMP` option is needed, when `fccpx`, `FCCpx`, `mpifccpx` or `mpiFCCpx` is used to link-edit user program.

`-SCALAPACK` option is also needed, when ScaLAPACK library is link-edited with user program using `mpifccpx` or `mpiFCCpx`.

Example 1:

Compile and link-edit a user program a.c which load the library of BLAS, LAPACK sequential version dynamically using dlopen and dlsym function.

```
fccpx a.c -SSL2
```

Example 2:

Compile and link-edit a user program a.c which load the library of BLAS, LAPACK thread parallel version dynamically using dlopen and dlsym function.

```
fccpx -Kopenmp a.c -SSL2
```

Example 3:

Compile and link-edit a user program a.c which load the ScaLAPACK library and the library of BLAS, LAPACK sequential version dynamically using dlopen and dlsym function.

```
mpifccpx a.c -SSL2 -SCALAPACK
```

Example 4:

Compile and link-edit a user program a.c which load the ScaLAPACK library and the library of BLAS, LAPACK thread parallel version dynamically using dlopen and dlsym function.

```
mpifccpx -Kopenmp a.c -SSL2 -SCALAPACK
```

Example 5:

Compile a.c which load the library of BLAS, LAPACK sequential version dynamically using dlopen and dlsym function. Then compile and link-edit a Fortran program b.f which call it.

```
fccpx -c a.c
frtpx b.f a.o
```

Example 6:

Compile a.c which load the ScaLAPACK library and the library of BLAS, LAPACK thread parallel version dynamically using dlopen and dlsym function. Then compile and link-edit a Fortran program b.f which call it.

```
fccpx -c a.c
frtpx -Kopenmp b.f a.o
```

Example 7:

Compile a.c which load the the ScaLAPACK library and the library of BLAS, LAPACK sequential version dynamically using dlopen and dlsym function. Then compile and link-edit a Fortran program b.f which call it.

```
mpifccpx -c a.c
mpifrtpx b.f a.o
```

Example 8:

Compile a.c which load the ScaLAPACK library and the library of BLAS, LAPACK thread parallel version dynamically using dlopen and dlsym function. Then compile and link-edit a Fortran program b.f which call it.

```
mpifccpx -c a.c
mpifrtpx -Kopenmp b.f a.o
```

2.3 Notes

Following are notes the user should be aware of for producing right results from routines.

2.3.1 Maximum number of threads

When calling BLAS or LAPACK, the maximum number of threads that can enter a subroutine at a time is 128.

2.3.2 Data alignment

Subroutines of the products assume that arguments of double precision real and double precision complex to be passed have been aligned on 8 byte address boundaries, so specifying the compile option `-AA` should be avoided.

2.3.3 Infinity and NaN

In the LAPACK since version 3.0 from netlib, a set of subroutines have been added that expect infinities and NaN (not a number) defined in the IEEE standard, to be returned as results of zero-divide or overflow and not to terminate the computation.

Fujitsu Fortran fully conforms to the standard of such numbers. However, when the user specifies the option -NRtrap, error messages will come out. So, make sure to avoid using the option when compiling programs that call LAPACK routines.

2.3.4 Routines in the archive file

Routines of C-SSL II and SSL II are also included in BLAS, LAPACK archives. And also the library includes slave routines, the names of which starts SS_ or #L_(# means S, D, C, S, I or X). The user needs to be careful not to duplicate subroutine names with them.

2.3.5 Internal work area used for BLAS, LAPACK

Some subroutines of BLAS and LAPACK take large work area internally. The work area is allocated on “stack”, so the user needs to set an appropriate value to the stack size.

There are two kind of stack area, “stack area of a process” and “stack area of each thread”. Both or one of them needs to be expanded in some cases.

The stack size for a process can be set by the command ulimit. For the stack area of each thread, the same size as that of a process is allocated by default. If the user wants to acquire a different size from that of process, the user can use the environment variable THREAD_STACK_SIZE. More details are available in *Fortran User's Guide*.

2.3.5.1 Sequential version

Table 2 shows the stack size for BLAS and LAPACK sequential version. When the routines are called from sequential Fortran program, the stack size for a process needs to be greater than the value.

When the routines are called from inside parallelized potions of OpenMP Fortran program, the user needs to add the value to both of the stack size for a process and for each thread. When the user program uses work area on the stack, set to sum of the size in Table 2 and the size for user program. When the user program calls different precision routines , the largest value should be used.

Table 2 The size of internal work area for sequential version

Precision	Size of workspace
REAL(4)	2Mbyte
REAL(8)	4Mbyte
COMPLEX(4)	2Mbyte
COMPLEX(8)	4Mbyte

2.3.5.2 Thread-parallel version

a) Calls from a sequential program or from sequential potions of OpenMP Fortran program

The user need to add the value in Table 3 to stack size for a process and for each thread respectively.

Table 3 The size of internal work area for thread-parallel version (1)

Precision	Size of workspace	
	for process	for each thread
REAL(4)	11Mbyte	2Mbyte
REAL(8)	21Mbyte	4Mbyte
COMPLEX(4)	19Mbyte	2Mbyte
COMPLEX(8)	38Mbyte	4Mbyte

b) Calls from inside parallelized potions of OpenMP Fortran program

The user need to add the value in Table 4 to both the stack size for a process and for each thread.

Table 4 The size of internal work area for thread-parallel version (2)

Precision	Size of workspace
REAL(4)	11Mbyte
REAL(8)	21Mbyte
COMPLEX(4)	19Mbyte
COMPLEX(8)	38Mbyte

2.3.6 Size of local array used by ScaLAPACK routines

The number of elements of local array used by ScaLAPACK routines must not exceed 21474836478. Because some routines reffer the array as one dimensional array and the index is four byte integer.

Example :

```
REAL(8)::A(100000,100000)
```

This array A can't be used as parameter of ScaLAPACK routines because the size of A is 100000×100000=10000000000 and greater than 2147483647.

2.3.7 Notes on link-editing BLAS, LAPACK thread-parallel version with `-Kparallel` option

There are some notes to link-edit BLAS, LAPACK thread-parallel version with `-Kparallel` option instead of -Kopenmp option.

1) If environment variable PARALLEL as well as OMP_NUM_THREADS is set, PARALLEL has priority over OMP_NUM_THERADS.

2.3.8 The matrix size

There are formulas like M*N*k where M and N are matrix sizes and k is constant in some of the Netlib original source codes of BLAS, LAPACK and ScaLAPACK. They are calculated in use of four bytes integer. When M or N is too huge, the value of the expression may be incorrect through overflow and consequently the algorithm may not work well. It becomes the implicit limit of the matrix size of BLAS, LAPACK and ScaLAPACK routines. Most of the routines accept huge matrix size but some routines may not work correctly when the matrix size are huge such that M*N*k >= 2147483647. The user needs to be careful of the matrix size.

2.3.9 Module of PLASMA

Some modules are provided to use PLASMA in Fortran90 program. The module names are plasma.mod, plasma_#.mod (# is s, d, c, z, ds or zc). The user needs to be careful not to duplicate module names with them.

If compiler option -AU is specified when a user program written in Fortran is compiled, the module names and each routine name must be in lowercase characters.

3. Fast Basic Operations Library for Quadruple Precision

3.1 Overview

The fast_dd is a library in which a quadruple precision number is expressed in double-double format and arithmetic operations are performed on such formatted numbers. In the format, a quadruple precision number is stored in two double precision variables. Arithmetic operations on such quadruple precision numbers can be processed by using double precision hardware instructions provided on the processor. So, in most cases, it is significantly faster than the intrinsic quadruple precision in the compilers.

The fast_dd is tuned to a scalar processor. For SPARC64 XIfx processors respectively their special hardware features are used to extract their performance.

3.2 Documentation

The fast_dd library can be used in both Fortran and C++ programs. Check "Fast Basic Operations Library for Quadruple Precision User’s Guide" for a description of each function.

3.3 Compile, Link and Run

3.3.1 Compile user's programs and link-edit with fast_dd

fast_dd is to be called from a Fortran or C++ program. Compilation and link-editing should be done using Fujitsu Fortran or C++ compiler

3.3.1.1 Preparation

The user is requested to set up environment variables as follows prior to using the library.

- To compile and link-edit using cross compiler on login node :

  Add /opt/FJSVmxlang/bin to the environment variable PATH.

- To compile and link-edit using own compiler:

  Add /opt/FJSVmxlang/bin to the environment variable PATH.

- To execute on calculation node :

  Add /opt/FJSVmxlang/lib64 to the environment variable LD_LIBRARY_PATH.

3.3.1.2 Fortran version

Use frtpx or frt command with -SSL2 option in order to compile user programs written in Fortran and link-edit with fast_dd. When a routine in BLAS and LAPACK thread parallel version is used in a user's program, please use -SSL2BLAMP option instead of -SSL2 option.

Example 1: Compile a user's program a.f90, and link-edit it with the archive of fast_dd by using cross compiler.

```
frtpx a.f90 -SSL2
```

When the user's program uses thread parallel routines of fast_dd, specify the option –Kopenmp or –Kparallel also.

Example 2: Compile a user’s program b.f90 which calls thread parallel routine, and link-edit it with the archive of fast_dd.

```
frtpx b.f90 -SSL2 –Kopenmp
```

Use frt command in order to compile user programs and link-edit with the archive of fast_dd by using own compiler.

Example 3: Compile a user's program a.f90, and link-edit it with the archive of fast_dd by using own compiler.

```
frt a.f90 -SSL2
```

Use mpifrtpx, mpifrt command instead of frtpx, frt, when user’s program uses MPI routine.

Fast_dd can’t be used from XPFortran program.

3.3.1.3 C++ version

Use FCCpx or FCC command with -SSL2 option in order to compile user programs written in C++ and link-edit with fast_dd. When a routine in BLAS and LAPACK thread parallel version is used in a user's program, please use -SSL2BLAMP option instead of -SSL2 option. -Kopenmp, -Kparallel or -mt option is also needed, because the objects of fast_dd are thread-safe.

Example 1: Compile a user's program a.cpp, and link-edit it with the archive of fast_dd by using cross compiler.

```
FCCpx –Kopenmp a.cpp -SSL2
```

Use FCC command in order to compile user programs and link-edit with the archive of fast_dd by using own compiler.

Example 2: Compile a user's program a.cpp, and link-edit it with the archive of fast_dd by using own compiler.

```
FCC –Kopenmp a.cpp -SSL2
```

Use mpiFCCpx, mpiFCC command instead of FCCpx, FCC, when user’s program uses MPI routine.

3.4 Notes

3.4.1 Routines in the archive file

Routines of BLAS, LAPACK, C-SSL II and SSL II are also included in fast_dd archives. And also the library includes slave routines, the names of which starts SS_ or #L_(# means S, D, C, S, I or X). The user needs to be careful not to duplicate subroutine names with them.

3.4.2 Fortran compiler option -AU

If compiler option `-AU` is specified when a user program written in Fortran is compiled, the module name "fast_dd" and each routine name must be in lowercase characters.

3.4.3 Using fast_dd with Coarray feature

When dd_rean type and dd_complex type of fast_dd is used with Coarray feature of Fortran, they must satisfy following conditions.

- It must not be pointer.
- It must not be structure component.
- It must have ALLOCATABLE or SAVE attribute. It must not have AUTOMATIC attribute.

FUJITSU