FUJITSU

FUJITSU Software
Technical Computing Suite V4.0L20

Development Studio
数学ライブラリの利用手引

J2UL-2571-01Z0(03)
2022年9月

まえがき

本マニュアルでは本製品で提供する数学ライブラリの使い方について説明しています. 対象となる機能には以下があります.

SSL II
SSL II スレッド並列機能
C-SSL II
C-SSL II スレッド並列機能
SSL II/MPI
BLAS
LAPACK
ScaLAPACK
高速 4 倍精度基本演算ライブラリ(以降 fast_dd と呼びます)

SSL II, C-SSL II は 1 コア上で実行する逐次版です. なお, 本書では SSL II, SSL II スレッド並列機能, C-SSL II, C-SSL II スレッド並列機能, SSL II/MPI を総称して SSL II 系数学ライブラリと呼びます.

本書の構成

本書の構成は以下のとおりです.

1. SSL II 系数学ライブラリ

1.1 概説
SSL II 系数学ライブラリの各コンポーネントの概要について説明します.

1.2 呼び出し方法
各ルーチンの呼び出し方法を記述した使用手引書について説明しています.

1.3 SSL II および C-SSL II の使用例
SSL II および C-SSL II の使用例を載せています. SSL II および C-SSL II は, スレッドセーフな作りになっています. 複数のスレッドに別々のデータを与えて call することにより同時に計算するという使い方が可能です. 1.3 ではその原理を幾つかの使用例を使って説明しております.スレッドセーフ性を利用した使い方をされる場合は是非, 参照してください.

1.4 翻訳・結合・実行方法
SSL II 系数学ライブラリを呼び出しているプログラムの翻訳・結合・実行の方法を説明します.

2. BLAS, LAPACK, ScaLAPACK

2.1 概説
BLAS, LAPACK, ScaLAPACK の概要について説明します.

2.2 翻訳・結合・実行方法
ユーザプログラムを BLAS, LAPACK, ScaLAPACK と結合し, 実行する方法を説明します.

2.3 注意事項
BLAS, LAPACK, ScaLAPACK を利用するにあたり, 注意すべき事項について説明します.

3. 高速 4 倍精度基本演算ライブラリ

3.1 概説
fast_dd の概要について記述しています.

3.2 呼び出し方法
各ルーチンの呼び出し方法を記述した使用手引書について記述しています.

3.3 翻訳・結合・実行方法
fast_dd を使用するユーザプログラムの翻訳・結合・実行方法について説明します.

3.4 注意事項
fast_dd を利用するにあたり, 注意すべき事項について説明します.

各ルーチンの使用方法については1.2, 2.1, 3.2 で示す使用手引書またはドキュメントを参照してください. 本書ではシステム固有の情報を記述しています.

本書の内容を理解するための参考資料としては, 以下に示すマニュアルがあります.

“Fortran 使用手引書”
“C 言語使用手引書”
“C++言語使用手引書”
“MPI 使用手引書”

OpenMP の仕様については, 以下で公開されている仕様書を参照してください.

http://www.openmp.org/

謝辞：
BLAS, LAPACK 及び ScaLAPACK は Prof. Dongarra (テネシー大学) を中心とした研究グループで開発され, Netlib で公開されています.

輸出管理規制について

本ドキュメントを輸出または第三者へ提供する場合は, お客様が居住する国および米国輸出管理関連法規等の規制をご確認のうえ, 必要な手続きをおとりください.

出版年月および版数

版数	マニュアルコード
2022 年　9 月　第 1.3 版	J2UL-2571-01Z0(03)
2021 年　3 月　第 1.2 版	J2UL-2571-01Z0(02)
2020 年　6 月　第 1.1 版	J2UL-2571-01Z0(01)
2020 年　2 月　初版	J2UL-2571-01Z0(00)

変更履歴

変更内容	変更箇所	版数
-KNOSVE オプションが指定されたときにリンクされるライブラリの変更	1.4.2.1, 1.4.3.1, 1.4.4.1, 1.4.4.2, 1.4.5.1, 1.4.5.2, 1.4.6.1, 1.4.6.2, 2.2.2, 2.2.3, 2.2.4, 2.2.6.1, 2.2.6.2, 2.2.6.3, 3.3.1.2, 3.3.1.3	第 1.3 版
計算結果についての注意事項を追加	2.3.11	第 1.3 版
MRQ オーバーフローについての注意事項を追加	1.4.6.4, 2.3.10	第 1.2 版
製品のレベルアップに伴い, 体裁を変更	-	第 1.1 版
誤字の訂正	1.3.1, 1.3.2, 2.2.7.2, 2.2.7.5, 2.2.7.7	第 1.1 版
clang モードについての説明を追加	1.4.4.1, 1.4.4.2, 1.4.5.1, 1.4.5.2, 1.4.6.2, 2.2.6.1, 2.2.6.2, 2.2.6.3, 3.3.1.3	第 1.1 版

お願い

目次

表目次

1. SSL II 系数学ライブラリ

1.1 概説

SSL II 系数学ライブラリの各コンポーネントの概要について説明します.

1.1.1 SSL II

SSL II は, スカラ CPU 向けにチューニングしています. 特に A64FX CPU の性能を引き出すチューニングを行っております.

また, SSL II は, スレッドセーフなライブラリを提供します. すなわち, 逐次 Fortran プログラムからは勿論のこと, OpenMP Fortran で書かれたスレッド並列プログラムから call することもできます. 後者の場合, 複数のスレッドに別々のデータを与えて同じルーチンを call することにより同時に計算するという使い方が可能です.

1.1.2 SSL II スレッド並列機能

SSL II スレッド並列機能は, 共有メモリ型スカラ並列計算機システム用に開発された並列アルゴリズムのライブラリです. 特に A64FX CPU の性能を引き出すチューニングを行っております. 各サブルーチンは OpenMP Fortran 仕様で書かれており, ユーザの OpenMP Fortran プログラムを初め, 自動並列化されるプログラム, さらには従来の逐次 Fortran プログラムからも call することができます. これらのプログラムは, Fortran コンパイラにそれぞれに対応したオプションを指定することでコンパイルできます.

SSL II スレッド並列機能のサブルーチン名は従来の SSL II とは異なったものになっています. この狙いの一つは, ユーザプログラムの中で, 従来の SSL II サブルーチンとスレッド並列機能サブルーチンを混在して利用することを可能にするためです. さらに引数の並びも該当する SSL II サブルーチンのそれとは必ずしも同一ではありません.

SSL II スレッド並列機能では, 行列演算, 連立１次方程式(直接法), 連立１次方程式(反復法), 逆行列, 固有値問題, フーリエ変換, 乱数など, 大規模計算で必要となる分野であって, かつ, 並列化効果の見出されるサブルーチンを提供しています.

1.1.3 C-SSL II

C-SSL II は, スカラ CPU 向けにチューニングしています. 特に A64FX CPU の性能を引き出すチューニングを行っております.

また, C-SSL II は, スレッドセーフなライブラリを提供します. すなわち, 普通の逐次 C プログラムからは勿論のこと, OpenMP C で書かれたスレッド並列プログラムから call することもできます. 後者の場合, 複数のスレッドに別々のデータを与えて同じルーチンを call することにより同時に計算するという使い方が可能です.

1.1.4 C-SSL II スレッド並列機能

C-SSL II スレッド並列機能は, 共有メモリ型スカラ並列計算機システム用に開発された並列アルゴリズムのライブラリです. 特に A64FX CPU の性能を引き出すチューニングを行っております. 各サブルーチンは OpenMP C 仕様で書かれており, ユーザの OpenMP C プログラムを初め, 自動並列化されるプログラム, さらには従来の逐次 C 言語プログラムからも利用することができます. これらのプログラムは, C/C++コンパイラにそれぞれに対応したオプ

ションを指定することでコンパイルできます.
C-SSL II スレッド並列機能の関数名は従来の C-SSL II とは異なったものになっています. この狙いの一つは, ユーザプログラムの中で, 従来の C-SSL II 関数とスレッド並列機能関数を混在して利用することを可能にするためです. さらに引数の並びも該当する C-SSL II 関数のそれとは必ずしも同一ではありません.
C-SSL II スレッド並列機能では, 行列演算, 連立 1 次方程式(直接法), 連立 1 次方程式(反復法), 固有値問題, フーリエ変換, 乱数など, 大規模計算で必要となる分野であって, かつ, 並列化効果の見出されるサブルーチンを提供しています.

1.1.5 SSL II/MPI

SSL II/MPI は, 分散メモリ型並列計算機システム上で大規模な問題を効率良く計算するための計算機能を, 並列処理向けのアルゴリズムを採用して提供しています. 特に A64FX CPU の性能を引き出すチューニングを行っております.
SSL II/MPI の各機能は Fortran のサブルーチンの形で提供されており, CALL 文により利用できます.
なお, SSL II/MPI は単一プロセッサ用および SMP 並列向けの数値計算ライブラリ SSL II とは機能範囲, サブルーチン名, および呼出し方法が異なります.
SSL II/MPI では, 3 次元フーリエ変換を提供しています.

1.2 呼び出し方法

各ルーチンの呼び出し方法を記述した使用手引書について説明します.

- SSL II
  (1) 「富士通 SSL II 使用手引書」
  (2) 「FUJITSU SSL II 拡張機能使用手引書」
  (3) 「FUJITSU SSL II 拡張機能使用手引書 II」
  SSL II のマニュアルは, ボリュームの関係で, ご覧のとおり 3 分冊になっております.
  (2)と(3) は「拡張機能」という名前がついていますが意味はスーパーコンピュータ向けのサブルーチン群を表し, 機能的に大型問題 (大規模 FFT, スパース行列解法など) を扱っています. 一方, (1) に記述されているサブルーチンのいくつかについては, スーパーコンピュータ向きに改造したサブルーチンが(2) あるいは (3) に存在します (例：行列積). そうしたサブルーチンの場合は, 一般的には(2), (3) のほうを先に試すことをお奨めします. この意味でも, 3 冊全体に目を通して頂き, 目的に最適なサブルーチンをご使用頂きたくお願い致します.

- SSL II スレッド並列機能
  「FUJITSU SSL II スレッド並列機能 使用手引書」
  サブルーチン一覧表や, 概説 (ユーザプログラムからの呼出し方法, プログラミング上の要点, 一般的な使用例), そして個々のサブルーチンの使い方が書かれています. SSL II スレッド並列機能を知るには, 必ず参照してください.

- C-SSL II
  「FUJITSU C-SSL II 使用手引書」
  C-SSL II の概説, 各サブルーチンの使い方が書かれた, いわゆる使用手引書です.

- C-SSL II スレッド並列機能
  「FUJITSU C-SSL II スレッド並列機能 使用手引書」
  C-SSL II スレッド並列機能の概説, 各サブルーチンの使い方が書かれた, いわゆる使用手引書です.

- SSL II/MPI
  「FUJITSU SSL II/MPI 使用手引書」
  サブルーチン一覧表や, 概説(ユーザプログラムからの呼出し方法, プログラミング上の

要点, 一般的な使用例), そして個々のサブルーチンの使い方が書かれています. SSL II/MPI を知るには, 必ず参照してください.

1.3 SSL II および C-SSL II の使用例

1.3.1 SSL II の使用例

本製品で提供する SSL II はスレッドセーフな作りになっているため, 逐次 Fortran プログラムから call できるだけでなく, OpenMP Fortran で書かれたスレッド並列プログラムからも call できます. ここでは, 後者に焦点を当てて, スレッドセーフな SSL II を利用するときの使用例を述べます. 読者は「スレッド」の概念, および OpenMP Fortran の仕様についてある程度の予備知識があるものと想定します.

SSL II のスレッドセーフ性を利用する場合の前提条件は以下のとおりです.

(1) 共有メモリ・マルチコアまたはマルチプロセッサの環境で使う

(2) OpenMP Fortran で記述されたプログラムから利用する

(3) 富士通の Fortran コンパイラで翻訳して, 本製品と結合する

SSL II のスレッドセーフ性を利用する目的は, 独立な複数組のデータを与えて同時に計算させることでターンアラウンドを短縮することにあります. 例えば, 複数組のデータに対してあるサブルーチンを使う場合,スレッドセーフな SSL II では, 一つのスレッドがひと組みのデータを“担当”して, 複数のスレッドがサブルーチンの中を同時に実行することを許します. 発生させたスレッドの数に相当する分の並列処理が行われ, 時間短縮が図れます. 勿論, CPU またはコアが複数個, 備わっていることが条件です. 以下の例で具体的に説明します.

スレッドセーフな SSL II は, ひとつの問題を複数スレッドを使って並列に解くためのライブラリではありません. 例えば, ひとつの行列積の計算を複数スレッドで手分けして行うには並列アルゴリズムを備えたサブルーチンが必要です. SSL II は並列アルゴリズムを備えたライブラリではありません. あくまでも,「独立した複数の問題を複数スレッドで独立に, しかも同時に解く」ためのライブラリです.

（注）一方, 並列アルゴリズムを備えた SSL II のライブラリは,「SSL II スレッド並列機能」と呼ぶライブラリが別途あります.

以下の例は, OpenMP Fortran プログラムから SSL II のスレッドセーフ性を利用するときの原理を強調するために, 単純な問題を扱っています. 個々のサブルーチンの詳しい使用法についてはマニュアルを参照してください.

以下のプログラム例では, スレッドの個数について何も仮定していません. スレッドの個数は, 実行時に環境変数 OMP_NUM_THREADS で指定するものとします. ただし, SSL II では, スレッドの個数の上限を 128 としています.

CPU またはコアの個数は 1 つあれば実行には差し支えありませんが, 完全な並列実行を実現するにはスレッドの個数分必要になります. プログラムの翻訳・結合・実行の方法については, 1.4 翻訳・結合・実行方法を参照してください.

例 1 : 連立一次方程式

問題

n 元の連立一次方程式を考えます.

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{1.1}$$

ここで, $\boldsymbol{A}$ は n 次元の実行列, $\boldsymbol{b}$ は n 次の右辺ベクトル, そして $\boldsymbol{x}$ は n 次の解ベクトルです. 今, (1.2)のように右辺ベクトルが複数あってそれぞれの解ベクトルを求めたいとします.

$$\boldsymbol{A}\boldsymbol{x}_i = \boldsymbol{b}_i \quad i = 1, 2, \ldots , m \tag{1.2}$$

プログラム例

(1.2)の複数の問題を, SSL II のサブルーチン DVLAX を使って解きます. DVLAX は, 最初, 与えられた行列の LU 分解を行い, そのあとで後退代入, 前進代入により解 $\boldsymbol{x}$ を求めます. LU 分解の結果はもとの 2 次元配列 a に入れられ, 解 $\boldsymbol{x}$ は元の配列 b に入れられます. 右辺ベクトルを変えて call するときは, すでにある LU 分解の結果を使います. その際, 初回の call か 2 回目以降の call かを利用者がサブルーチンに伝える仕様になっており, そのための入力パラメタ `ISW` があります.

100 元の連立一次方程式を, 50 本の異なる右辺ベクトルに対して解くとします. SSL II のスレッドセーフ性を直感的に理解するためにまず, 逐次プログラム例を示します.

逐次プログラムからの利用

```
      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=1,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
```

これに対してスレッド並列プログラムから利用すると, LU 分解と最初の右辺ベクトルに対する解を求めるまでは上と同じですが, 第 2 番目以降の右辺ベクトルに対しては, 複数のスレッドを使って同時に解を計算させることができます. このためには, 複数の右辺ベクトルをはじめから準備しておく必要があります. 以下ではそれを 2 次元配列 b に格納しておきます. さらに, サブルーチン内部で出力または変更される引数は, スレッド毎に別々のメモリ領域を準備する必要があります. そのような引数は ISW=2 の場合, icon だけであり, icon を配列化します. 他の引数,

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

は, ISW=2 の場合, サブルーチン内部で変更されないので, スレッド間で共有することができます.

(注意) DVLAX を ISW=2 で呼び出す場合は作業域(vw, ip)は変更されませんが一般的には作業域はサブルーチン内部で変更されるので, スレッド毎に別々のメモリ領域が必要です.

OpenMP Fortran のプログラム例を以下に示します. 解説はプログラムの後で述べます.

スレッド並列プログラムからの利用

```
      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
```

解説

(1) 翻訳指示行!$OMP PARALLEL と!$OMP END PARALLEL で囲まれたブロックが, 複数のスレッドで実行されます. !$OMP PARALLEL と書かれた位置が複数のスレッドの発生起点になります. ブロック内の変数／配列は, この例の場合, スレッド間で共有されます. すなわち, メモリには一つの実体しかありません.

PARALLEL 指示行にはオプションを書くことができ, ある変数 / 配列をスレッド毎に別領域にコピーを持たせることもできます. 詳しくは OpenMP の文法書(正確には, 公開されている “OpenMP Application Program Interface Version 3.0 May 2008”を指す) を参照してください.

(2) 翻訳指示行!$OMP DO SCHEDULE(STATIC)は, 直下の DO 文について, DO インデックスの各値に対して, DO 内の一連の実行文を独立に計算してよいことを示し, かつ, DO の繰返しをスレッド間で分担して処理する意味を持ちます. オプションの SCHEDULE(STATIC)部分は, 分担の仕方の一方法を示しています. すなわち上の例で, J=2,3,4,..., *m* のインデックス・レンジをスレッドの個数で(ほぼ) 等分割し, できた小レンジを順番にスレッドに静的に割り当てます.

(3) さらに特徴的なことは, 上のプログラムを, `-Kopenmp` オプションを指定せずに翻訳し, SSL II を結合すれば, 正常に動作し, 同じ計算結果を得ることができます. これは, 上のプログラムが!$OMP の翻訳指示行を除けば普通の逐次処理プログラムと同じであること, その翻訳指示行が`-Kopenmp` オプションを指定しない場合はコメント文として扱われるためです.
以下の例も, この意味で同じであり, 逐次処理プログラムと, スレッド並列プログラムを兼用していると言えます.

例 2 : 定積分

問題

以下の定積分,

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

を考え, p は, $p = i / 10$, $i = 1, 2, \ldots, 9$ と変えて, 各々の定積分を計算します.

プログラム例

(2.1)を, SSL II のサブルーチン DAQN9 を使って計算します. 被積分関数の値を計算する関数副プログラム FUN の中で p を使いますが, p の値はメインプログラムで変えています. このため FUN との間では COMMON ブロックで連携しています. ただし, p の為の変数は, スレッド毎に別々のメモリ領域を与える必要があります. こうしないと, 計算が早く進んだスレッドが, p の値を次の値に変えてしまい, 他のスレッドによる積分計算を乱すからです.

```
      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
```

上のプログラムを実行すると, WRITE 文による icon(*)の値がすべて 11000 と出ますが, これはサブルーチン DAQN9 が, 代数的特異点を検出して, それに相応しい計算を行い, 正しく結果を求めたことを意味します（マニュアルに詳しい記載があります）.

上の例のように, SSL II の内部から呼ばれる関数副プログラムを用意する場合に注意することが一つあります. それは, その関数副プログラムのオブジェクトコードがスレッドセーフになっている必要があります. スレッドセーフとは, その関数副プログラムを複数のスレッドが, 異なる引数の値(上で言うと fun(x)の x) で同時に実行しても, 各スレッドは正しい結果を計算できることです. 例えば次の関数副プログラムがあります.

```
      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
```

この副プログラムがスレッドセーフとなるためには, ローカル変数 s,term は, スレッド毎に別の領域が割り当てられる必要があります. それをコンパイラに指示する必要があります. Fortran コンパイラでは-Kopenmp というオプションがそれに当り, このオプションの指定により, s,term は自動変数となり, スタック領域にとられます. 複数のスレッドが実行する場合は, スレッド毎のスタック領域が使われます. ただし,

初期値を持った変数, SAVE 属性をもった変数, あるは EQUIVALENCE 変数

はスタック領域にとられません. 初期値を持った変数については, 実行中, 更新されないならば問題になることはありません. あとの二つはスレッドセーフにとって問題になるので, 適当な修正が必要です.

例 3 : 特殊関数

問題

第一種一次ベッセル関数 $J_1(x)$を, $0 \le x \le 10$ の区間を等間隔に 300 分割し, 各点で値を求めます.

プログラム例

SSL II のサブルーチン DBJ1 を使って計算します.

```
      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 C-SSL II の使用例

本製品で提供する C-SSL II はスレッドセーフな作りになっているため, 逐次 C プログラムから呼び出すことができるだけでなく, OpenMP C/C++で書かれたスレッド並列プログラムからも呼び出すことができます. ここでは, 後者に焦点を当てて, スレッドセーフな C-SSL II を利用するときの使用例を述べます. 読者は「スレッド」の概念, および OpenMP C/C++の仕様についてある程度の予備知識があるものと想定します.

C-SSL II のスレッドセーフ性を利用する場合の前提条件は以下のとおりです.

(1) 共有メモリ・マルチコアまたはマルチプロセッサの環境で使う

(2) OpenMP C/C++で記述されたプログラムから利用する

(3) 富士通の C/C++コンパイラで翻訳して, 本製品と結合する

C-SSL II のスレッドセーフ性を利用する目的は, 独立な複数組のデータを与えて同時に計算させることでターンアラウンドを短縮することにあります. 例えば, 複数組のデータに対してあるルーチンを使う場合, スレッドセーフな C-SSL II では, 一つのスレッドがひと組みのデータを“担当”して, 複数のスレッドがルーチンの中を同時に実行することを許します. 発生させたスレッドの数に相当する分の並列処理が行われ, 時間短縮が図れます. 勿論, CPU またはコアが複数個, 備わっていることが条件です. 以下の例で具体的に説明します.

スレッドセーフな C-SSL II は, ひとつの問題を複数スレッドを使って並列に解くためのライブラリではありません. 例えば, ひとつの行列積の計算を複数スレッドで手分けして行うには並列アルゴリズムを備えたルーチンが必要です. C-SSL II は並列アルゴリズムを備えたライブラリではありません. あくまでも, 「独立した複数の問題を複数スレッドで独立に, しかも同時に解く」ためのライブラリです.

以下の例は, OpenMP C/C++プログラムから C-SSL II のスレッドセーフ性を利用するときの原理を強調するために, 単純な問題を扱っています. 個々のルーチンの詳しい使用法については「FUJITSU C-SSL II 使用手引書」を参照してください.

以下のプログラム例では, スレッドの個数について何も仮定していません. スレッドの個数は, 実行時に環境変数 OMP_NUM_THREADS で指定するものとします. ただし, C-SSL II では, スレッドの個数の上限を 128 としています.

CPU またはコアの個数は 1 つあれば実行には差し支えありませんが, 完全な並列実行を実現するにはスレッドの個数分必要になります. プログラムの翻訳・結合・実行の方法については, 1.4 翻訳・結合・実行方法を参照してください.

例 1: 連立一次方程式

問題

n 元の連立一次方程式を考えます.

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{1.1}$$

ここで, $\boldsymbol{A}$ は n 次元の実行列, $\boldsymbol{b}$ は n 次の右辺ベクトル, そして $\boldsymbol{x}$ は n 次の解ベクトルです. 今, (1.2)式のように右辺ベクトルが複数あってそれぞれの解ベクトルを求めたいとします.

$$\boldsymbol{A}\boldsymbol{x}_i = \boldsymbol{b}_i \qquad i = 1, 2, \ldots, m \tag{1.2}$$

プログラム例

(1.2)式の複数の問題を, C-SSL II のルーチン c_dvlax を使って解きます. c_dvlax は, 最初, 与えられた行列の LU 分解を行い, そのあとで後退代入, 前進代入により解 $\boldsymbol{x}$ を求めます. LU 分解の結果はもとの 2 次元配列 a に入れられ, 解 $\boldsymbol{x}$ は元の配列 b に入れられます. 右辺ベクトルを変えて呼び出すときは, すでにある LU 分解の結果を使います. その際, 初回の呼び出しか 2 回目以降の呼び出しかを利用者がルーチンに伝える仕様になっており, そのための入力引数 isw があります.

100 元の連立一次方程式を, 50 本の異なる右辺ベクトルに対して解くとします. C-SSL II のスレッドセーフ性を直感的に理解するためにまず, 逐次プログラム例を示します.

逐次プログラムからの利用

```
#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);
        .......
      }
      .......
}
```

これに対してスレッド並列プログラムから利用すると, LU 分解と最初の右辺ベクトルに対する解を求めるまでは上と同じですが, 第 2 番目以降の右辺ベクトルに対しては, 複数のスレッドを使って同時に解を計算させることができます. このためには, 複数の右辺ベクトルをはじめから準備しておく必要があります. 以下ではそれを 2 次元配列 b に格納しておきます. さらに, ルーチン内部で出力または変更される引数は, スレッド毎に別々のメモリ領域を準備する必要があります. そのような引数は isw=2 の場合, icon だけであり, icon を配列化します. 他の引数,

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

は, isw=2 の場合, ルーチン内部で変更されないので, スレッド間で共有することができます.

(注意) c_dvlax を isw=2 で呼び出す場合は作業域(vw, ip)は変更されません. しかし, 一般的には作業域はルーチン内部で変更されるので, スレッド毎に別々のメモリ領域が必要です.

OpenMP C のプログラム例を以下に示します. 解説はプログラムの後で述べます.

スレッド並列プログラムからの利用

```
#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 right 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]);
      }
      .......
}
```

解説

(1) 指示子#pragma omp parallel の直後の構造ブロックが, 複数のスレッドで実行されます. ブロック内の変数／配列は, この例の場合, スレッド間で共有されます. すなわち, メモリには一つの実体しかありません.

parallel 指示行にはオプションを書くことができ, ある変数 / 配列をスレッド毎に別領域にコピーを持たせることもできます. 詳しくは OpenMP の文法書(正確には, 公開されている “OpenMP Application Program Interface Version 3.0 May 2008”を指す) を参照してください.

(2) 指示子#pragma omp for schedule(static)は, 直下の for 文について, for インデックスの各値に対して, for 内の一連の実行文を独立に計算してよいことを示し, スレッド間で手分けして計算させる意味を持ちます. オプションの schedule(static)部分は, 分担の仕方の一方法を示しています. すなわち上の例で, j=2,3,4,....,m のインデックス・レンジをスレッドの個数で(ほぼ)等分割し, できた小レンジを順番にスレッドに静的に割り当てます.

(3) さらに特徴的なことは, 上のプログラムを, `-Kopenmp` オプションを指定せずに翻訳し, C-SSL II を結合すれば, 正常に動作し, 同じ計算結果を得ることができます. これは, 上のプログラムが#pragma omp の指示子を除けば普通の逐次処理プログラムと同じであること, その指示子が`-Kopenmp` オプションを指定しない場合は無視されるためです.
以下の例も, この意味で同じであり, 逐次処理プログラムと, スレッド並列プログラムを兼用していると言えます.

例 2 : 定積分

問題

以下の定積分,

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

を考え, p は, $p = i / 10$, $i = 1, 2, \ldots, 9$ と変えて, 各々の定積分を計算します.

プログラム例

(2.1)を, C-SSL II のルーチン c_daqn9 を使って計算します. 被積分関数の値を計算する関数 fun の中で p を使いますが, p の値はメインプログラムで変えています. このため fun との間では外部変数で連携しています. ただし, p の為の変数は, スレッド毎に別々のメモリ領域を与える必要があります. こうしないと, 計算が早く進んだスレッドが, p の値を次の値に変えてしまい, 他のスレッドによる積分計算を乱すからです.

```
#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;
}
```

上のプログラムを実行すると, printf 文による icon[]の値がすべて 11000 と出ますが, これはルーチン c_daqn9 が, 代数的特異点を検出して, それに相応しい計算を行い, 正しく結果を求めたことを意味します（「FUJITSU C-SSL II 使用手引書」に詳しい記載があります）.

例 3 : 特殊関数

問題

第一種一次ベッセル関数 $J_1(x)$を, $0 \le x \le 10$ の区間を等間隔に 300 分割し, 各点で値を求めます.

プログラム例

C-SSL II のルーチン c_dbj1 を使って計算します.

```
#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 翻訳・結合・実行方法

1.4.1 準備

本製品を使用するためには以下の設定が必要です. “*製品インストールパス*”は, システム管理者にお問い合わせください.

- ログインノードでクロスコンパイラによりプログラムの翻訳および結合を行うために:
  環境変数 PATH に /*製品インストールパス*/bin を追加します.
- ネイティブコンパイラによりプログラムの翻訳および結合を行うために:
  環境変数 PATH に /*製品インストールパス* /bin を追加します.
- 計算ノードでプログラムの実行を行うために:
  環境変数 LD_LIBRARY_PATH に /*製品インストールパス*/lib64 を追加します.

また, SSL II/MPI を利用する場合には, 上記の他に以下の設定が必要です.

- 計算ノードでプログラムの実行を行うために:
  環境変数 PATH に /*製品インストールパス*/bin を追加します.
- ログインノードで manpages を使用するために:
  manpages は, SSL II の説明のために“ssl2(3)”, SSL II および SSL II スレッド並列機能の各サブルーチンのインターフェース説明のために“サブルーチン名(3F)”が利用できます. これらを利用するためには, 環境変数 MANPATH に SSL II の manpages のディレクトリ /*製品インストールパス*/man を追加設定する必要があります.

1.4.2 SSL II

SSL II は Fortran で記述された利用者プログラムに結合して使用することができます. 翻訳, 結合には Fujitsu Fortran コンパイラを使用します. ここではこの製品を利用するプログラムを翻訳, 結合及び実行する方法を説明します. なお, 翻訳から実行までの手順の詳細については“Fortran 使用手引書”を参照してください.

1.4.2.1 翻訳・結合

Fortran で記述されたユーザプログラムをクロスコンパイラにより翻訳し，SSL II を結合するためには，`frtpx` コマンド行に `-SSL2` を指定します.

SSL II は CPU の種類に応じたライブラリが用意されており, プログラムを実行する機種に合わせてオプションを指定することにより適切なライブラリが結合されます. `–KSVE` が有効な場合は SVE を使用して高速化したライブラリを結合し, `–KNOSVE` が有効な場合は汎用のライブラリを結合します. A64FX を搭載した計算ノードで実行する場合, どちらでも動作しますが, 高速に実行するためには SVE を使用したライブラリを推奨します.

例 1 : ユーザプログラム a.f を翻訳し, SVE 版 SSL II ライブラリを結合します.

```
frtpx –KSVE a.f -SSL2
```

例 2 : ユーザプログラム a.f を翻訳し, 汎用版 SSL II ライブラリを結合します.

```
frtpx –KNOSVE a.f -SSL2
```

ユーザプログラムが OpenMP Fortran で書かれている場合は, オプションに `-Kopenmp` を指定します.

例 3 : OpenMP Fortran で書かれたユーザプログラム a.f を翻訳し, SVE 版 SSL II ライブラリを結合します.

```
frtpx –Kopenmp,SVE a.f -SSL2
```

ネイティブコンパイラによりユーザプログラムを翻訳し, SSL II を結合するためには frt コマンドを使用します.

例 4 : ユーザプログラム a.f を翻訳し, SVE 版 SSL II ライブラリを結合します.

```
frt –Kopenmp,SVE a.f -SSL2
```

1.4.2.2 注意事項

SSL II を使用する上で注意すべき事項について説明します.

- スレッド数について
  ユーザープログラムが複数スレッドを使用する場合, 実行時のスレッドの個数は環境変数 OMP_NUM_THREADS などで指定できます.
  SSL II では, スレッド数の上限 は 128 です.
- スタックサイズについて
  プログラムを-Kopenmp オプションを指定し翻訳すると, 配列等がスタック領域にとられる場合があります. このとき配列が大きいとスタック領域が不足する場合があります. このようなときは, ulimit コマンドでスタックサイズを拡張して実行してください.
- ライブラリのアーカイブに含まれるルーチンついて
  SSL II のアーカイブには, BLAS, LAPACK, C-SSL II 及び SSL II スレッド並列機能 のルーチンも含まれています. その他, ルーチン名の先頭が, SS_ または#L_(#は S,D,C,Z,I,X)で始まるスレーブルーチンが含まれています. 利用するときはルーチン名の重複に注意してください.
- セクタキャッシュに関する警告メッセージについて
  数学ライブラリでは A64FX CPU のセクタキャッシュ機能を使って高速化しているルーチンがあります. プログラムの実行方法によってセクタキャッシュを利用できない場合があり, 以下のような警告メッセージを出力する場合があります. この場合, セクタキャッシュが使われないため性能に影響があるかもしれませんが, 実行は継続され計算は正しく行われます.

  jwe1047i-w A sector cache couldn't be used.

1.4.3 SSL II スレッド並列機能

SSL II スレッド並列機能は OpenMP Fortran のサブルーチンとして提供されています. これらを利用するユーザプログラムを翻訳して SSL II スレッド並列機能を含む SSL II ライブラリを結合して OpenMP Fortran のロードモジュールとして実行する方法を説明します. なお, 翻訳から実行までの手順の詳細については“Fortran 使用手引書”を参照してください.

1.4.3.1 翻訳・結合

SSL II スレッド並列機能は以下のユーザプログラムから呼び出し, SSL II ライブラリと結合して OpenMP Fortran のロードモジュールとして実行することができます.

- OpenMP Fortran のプログラム
- Fortran プログラムを自動並列化オプションで翻訳したオブジェクトモジュール
- Fortran プログラムを逐次実行オプションで翻訳したオブジェクトモジュール

OpenMP Fortran のロードモジュールを作るため結合時に frtpx または frt コマンド行に -Kopenmp 及び-SSL2 を指定します.

-Kopenmp オプションのかわりに-Kparallelオプションを指定して結合することもできます.

-Nfjomplib オプションが指定されているかつ,-Kparallel オプションのみを指定して結合した場合,実行時に環境変数 PARALLEL が指定されると, OMP_NUM_THREADS の指定よりも PARALLEL の指定のほうが優先されますのでご注意ください.

SSL II スレッド並列機能は CPU の種類に応じたライブラリが用意されており, プログラムを実行する機種に合わせてオプションを指定することにより適切なライブラリが結合されます. –KSVE が有効な場合は SVE を使用して高速化したライブラリを結合し, –KNOSVE が有効な場合は汎用のライブラリを結合します. A64FX を搭載した計算ノードで実行する場合, どちらでも動作しますが, 高速に実行するためには SVE を使用したライブラリを推奨します.

例 1 : OpenMP Fortran で書かれたユーザプログラム a.f をクロスコンパイラにより翻訳し, SVE 版 SSL II ライブラリを結合します.

```
frtpx –Kopenmp,SVE a.f -SSL2
```

例 2 : OpenMP Fortran で書かれたユーザプログラム a.f をクロスコンパイラにより翻訳し, 汎用版 SSL II ライブラリを結合します.

```
frtpx –Kopenmp,NOSVE a.f -SSL2
```

例 3 : OpenMP Fortran で書かれたユーザプログラム a.f をネイティブコンパイラにより翻訳し, SVE 版 SSL II ライブラリを結合します.

```
frt –Kopenmp,SVE a.f -SSL2
```

例 4 : Fortran で書かれたユーザプログラム a.f を自動並列化して翻訳しオブジェクトモジュールを作り, SVE 版 SSL II ライブラリを結合して OpenMP Fortran のロードモジュールを作ります.

```
frtpx –Kparallel -c a.f
frtpx –Kparallel,openmp,SVE a.o -SSL2
```

例 5 : Fortran で書かれたユーザプログラム a.f を逐次実行するプログラムとして翻訳しオブジェクトモジュールを作り, SVE 版 SSL II ライブラリを結合して OpenMP Fortran のロードモジュールを作ります.

```
frtpx -c a.f
frtpx –Kopenmp,SVE a.o -SSL2
```

1.4.3.2 実行

翻訳・結合したユーザプログラムを並列実行するスレッド数 及び 各スレッドのスタック領域の大きさの設定について説明します.

- SSL II スレッド並列機能のサブルーチンを並列実行するスレッド数は, 以下の環境変数で指定することができます.

  環境変数 OMP_NUM_THREADS にスレッド数を指定します.

- 呼び出しているユーザプログラムに自動並列を指定して翻訳した場合, 自動並列化されたプログラムを並列実行するスレッド数は以下の環境変数で指定します.

  -Nfjomplib オプションが指定されていない場合, 環境変数 OMP_NUM_THREADS にスレッド数を指定します.

  -Nfjomplib オプションが指定されている場合, 環境変数 PARALLEL にスレッド数を指定します.

- SSL II スレッド並列機能のサブルーチンの内部で, 各スレッド毎の作業域を自動割付け配列で割り付けています. 生成するスレッドの数を NT, 利用できるメモリ量を M としたとき, 各スレッドのスタック領域の大きさとして環境変数 OMP_STACKSIZE に M/(5*NT) 程度を指定して実行することを勧めます. -Nfjomplib オプションが指定されている場合,環境変数 THREAD_STACK_SIZE にスタック領域の大きさを指定する事もできます.

1.4.3.3 注意事項

SSL II スレッド並列機能を使用する上で注意すべき事項について説明します.

- スレッド数の上限
  SSL II スレッド並列でのスレッド数の制限は特にありません.
- ライブラリのアーカイブに含まれるルーチンついて
  SSL II スレッド並列機能のサブルーチンは,DM_U 及び DL_ で始まるスレーブルーチンを利用しています.
  SSL II のアーカイブには, BLAS, LAPACK, C-SSL II 及び SSL II スレッド並列機能 のルーチンも含まれています. その他, ルーチン名の先頭が, SS_ または#L_(#は S,D,C,Z,I,X)で始まるスレーブルーチンが含まれています. 利用するときはルーチン名の重複に注意してください.
- セクタキャッシュに関する警告メッセージについて
  数学ライブラリでは A64FX CPU のセクタキャッシュ機能を使って高速化しているルーチンがあります. プログラムの実行方法によってセクタキャッシュを利用できない場合があり, 以下のような警告メッセージを出力する場合があります. この場合, セクタキャッシュが使われないため性能に影響があるかもしれませんが, 実行は継続され計算は正しく行われます.

  jwe1047i-w A sector cache couldn't be used.

1.4.4 C-SSL II

C-SSL II は C 言語で記述された利用者プログラムに結合して使用することができます. 翻訳, 結合には Fujitsu C コンパイラを使用します. ここではこの製品を利用するプログラムを翻訳, 結合及び実行する方法を説明します. なお, 翻訳から実行までの手順の詳細については“C 言語使用手引書”または“C++言語使用手引書”を参照してください.

1.4.4.1 C プログラムからの利用方法

1) 共通インクルードファイル
   C-SSL II では標準ヘッダファイルとして cssl.h をライブラリと一緒に提供しています. ソースプログラムの先頭で必ずこのヘッダファイルをインクルードしなければなりません. このヘッダファイルの中では, 全ての関数のプロトタイプ宣言や複素数(dcomplex)型などが宣言されています.

2) メイン関数の関数名
   メイン関数の名前は `main` または `MAIN__`(大文字の `MAIN` に続けて 2 つのアンダースコア)を使用します.

3) プログラムの翻訳・結合の方法
   C 言語で記述されたユーザプログラムをクロスコンパイラにより翻訳し, C-SSL II を結合するためには, `fccpx` コマンド行に `-SSL2` を指定します.

   C-SSL II は CPU の種類に応じたライブラリが用意されており, プログラムを実行する機種に合わせてオプションを指定することにより適切なライブラリが結合さ

れます. –KSVE が有効な場合は SVE を使用して高速化したライブラリを結合し, –KNOSVE が有効な場合は汎用のライブラリを結合します. A64FX を搭載した計算ノードで実行する場合, どちらでも動作しますが, 高速に実行するためには SVE を使用したライブラリを推奨します.

clang モードで翻訳する場合(-Nclang が有効)は-KSVE・-KNOSVE オプションの代わりに-march オプションに+sve または+nosve を有効にすることによりライブラリが選択されます.

例 1 : ユーザプログラム a.c を翻訳し, SVE 版 C-SSL II ライブラリを結合します.

```
fccpx –KSVE a.c -SSL2
```

例 2 : ユーザプログラム a.c を翻訳し, 汎用版 C-SSL II ライブラリを結合します.

```
fccpx –KNOSVE a.c -SSL2
```

ユーザプログラムが OpenMP C で書かれている場合は, オプションに-Kopenmp を指定します.

例 3 : OpenMP C で書かれたユーザプログラム a.c を翻訳し, SVE 版 C-SSL II ライブラリを結合します.

```
fccpx –Kopenmp,SVE a.c -SSL2
```

ネイティブコンパイラによりユーザプログラムを翻訳し, C-SSL II を結合するためには fcc コマンドを使用します.

例 4 : ユーザプログラム a.c を翻訳し, SVE 版 C-SSL II ライブラリを結合します.

```
fcc –KSVE a.c -SSL2
```

1.4.4.2 C++プログラムからの利用方法

C-SSL II は, C++言語で記述された利用者プログラムに結合して使用することができます. C++から C-SSL II を利用する場合, 以下の点に注意する必要があります.

1) 共通インクルードファイル
C-SSL II では標準ヘッダファイルとして cssl.h をライブラリと一緒に提供しています. ソースプログラムの先頭で必ずこのヘッダファイルをインクルードしなければなりません. このヘッダファイルの中では, C-SSL II の全ての関数を C 言語へのリンケージ指定(extern "C")付きで宣言しています. また, 複素数(dcomplex)型などが宣言されています.

2) メイン関数
メイン関数の名前は main または MAIN__ (大文字の MAIN に続けて 2 つのアンダースコア)を使用します.

3) プログラムの翻訳・結合の方法
C++言語で記述されたユーザプログラムをクロスコンパイラにより翻訳し, C-SSL II を結合するためには, FCCpx コマンド行に -SSL2 を指定します.

C-SSL II は CPU の種類に応じたライブラリが用意されており, プログラムを実行する機種に合わせてオプションを指定することにより適切なライブラリが結合されます. –KSVE が有効な場合は SVE を使用して高速化したライブラリを結合し, –KNOSVE が有効な場合は汎用のライブラリを結合します. A64FX を搭載した計算ノードで実行する場合, どちらでも動作しますが, 高速に実行するためには SVE を使用したライブラリを推奨します.

clang モードで翻訳する場合(-Nclang が有効)は-KSVE・-KNOSVE オプションの代わりに-march オプションに+sve または+nosve を有効にすることによりライブラリが選択されます.

例 1 : ユーザプログラム a.cc を翻訳し, SVE 版 C-SSL II ライブラリを結合します.

```
FCCpx –KSVE a.cc -SSL2
```

例 2 : ユーザプログラム a.cc を翻訳し, 汎用版 C-SSL II ライブラリを結合します.

```
FCCpx –KNOSVE a.cc -SSL2
```

ユーザプログラムが OpenMP C++ で書かれている場合は, オプションに -Kopenmp を指定します.

例 3 : OpenMP C++ で書かれたユーザプログラム a.cc を翻訳し, SVE 版 SSL II ライブラリを結合します.

```
FCCpx –Kopenmp,SVE a.cc -SSL2
```

ネイティブコンパイラによりユーザプログラムを翻訳し, SVE 版 C-SSL II を結合するためには FCC コマンドを使用します.

例 4 : ユーザプログラム a.cc を翻訳し, C-SSL II ライブラリを結合します.

```
FCC –KSVE a.cc -SSL2
```

1.4.4.3 注意事項

C-SSL II を使用する上で注意すべき事項について説明します.

- スタックサイズについて
  C, C++言語では関数の中で宣言した配列等がスタックに取られるため, 配列が大きい場合にスタック領域が不足する場合があります. このような場合, ulimit コマンドでスタックサイズを拡張して実行します.
- スレッドセーフについて
  C-SSL II ではスレッドセーフ性を実現するために, 一部のルーチンで OpenMP の機能を使っています. このため, システムのスレッドライブラリ(pthread など)を直接利用して並列化されたプログラムから呼び出した場合の動作は保証されません.
- スレッド数について
  ユーザープログラムが複数スレッドを使用する場合, 実行時のスレッドの個数は環境変数 OMP_NUM_THREADS などで指定できます.
  C-SSL II では, スレッド数の上限は 128 です.
- ライブラリのアーカイブに含まれるルーチンついて
  C-SSL II のアーカイブには, BLAS, LAPACK, SSL II および SSL II スレッド並列機能のルーチンも含まれています. その他, ルーチン名の先頭が, SS_または #L_(#は S, D, C, Z, I, X)で始まる Fortran で書かれたスレーブルーチンが含まれています. 利用するときはルーチン名の重複に注意してください.
- セクタキャッシュに関する警告メッセージについて
  数学ライブラリでは A64FX CPU のセクタキャッシュ機能を使って高速化しているルーチンがあります. プログラムの実行方法によってセクタキャッシュを利用できない場合があり, 以下のような警告メッセージを出力する場合があります. この場合, セクタキャッシュが使われないため性能に影響があるかもしれませんが, 実行は継続され計算は正しく行われます.

  jwe1047i-w A sector cache couldn't be used.

1.4.5 C-SSL II スレッド並列機能

C-SSL II スレッド並列機能は OpenMP C の関数として提供されています. これらを利用するユーザプログラムを翻訳して C-SSL II スレッド並列機能を含む C-SSL II ライブラリを結合して OpenMP C のロードモジュールとして実行する方法を説明します. なお, 翻訳から実行までの手順の詳細については“C 言語使用手引書”または“C++言語使用手引書”を参照してください.

1.4.5.1 C プログラムからの利用方法

1) 共通インクルードファイル
C-SSL II スレッド並列機能では標準ヘッダファイルとして cssl.h をライブラリと一緒に提供しています. ソースプログラムの先頭で必ずこのヘッダファイルをインクルードしなければなりません. このヘッダファイルの中では, 全ての関数のプロトタイプ宣言や複素数(dcomplex)型などが宣言されています.

2) メイン関数の関数名
メイン関数の名前は `main` または `MAIN__`(大文字の `MAIN` に続けて 2 つのアンダースコア)を使用します.

3) プログラムの翻訳・結合の方法
C-SSL II スレッド並列機能は以下のユーザプログラムから呼び出し, C-SSL II ライブラリと結合して OpenMP C のロードモジュールとして実行することができます.

- OpenMP C のプログラム
- C プログラムを自動並列化オプションで翻訳したオブジェクトモジュール
- C プログラムを逐次実行オプションで翻訳したオブジェクトモジュール

OpenMP C のロードモジュールを作るため結合時に `fccpx` または `fcc` コマンド行に `-Kopenmp` 及び`-SSL2` を指定します.

`-Kopenmp` オプションのかわりに`-Kparallel` オプションを指定して結合することもできます.

`-Nfjomplib` オプションが指定されているかつ,`-Kparallel` オプションのみを指定して結合した場合,実行時に環境変数 `PARALLEL` が指定されると, `OMP_NUM_THREADS` の指定よりも `PARALLEL` の指定のほうが優先されますのでご注意ください.

C-SSL II スレッド並列機能は CPU の種類に応じたライブラリが用意されており, プログラムを実行する機種に合わせてオプションを指定することにより適切なライブラリが結合されます. `–KSVE` が有効な場合は SVE を使用して高速化したライブラリを結合し, `–KNOSVE` が有効な場合は汎用のライブラリを結合します. A64FX を搭載した計算ノードで実行する場合, どちらでも動作しますが, 高速に実行するためには SVE を使用したライブラリを推奨します.

clang モードで翻訳する場合(`-Nclang` が有効)は`-KSVE`・`-KNOSVE` オプションの代わりに`-march` オプションに`+sve` または`+nosve` を有効にすることによりライブラリが選択されます.

例 1 : OpenMP C で書かれたユーザプログラム a.c をクロスコンパイラにより翻訳し, SVE 版 C-SSL II ライブラリを結合します.

```
fccpx –Kopenmp,SVE -SSL2 a.c
```

例 2 : OpenMP C で書かれたユーザプログラム a.c をクロスコンパイラにより翻訳し, 汎用版 C-SSL II ライブラリを結合します.

```
fccpx –Kopenmp,NOSVE -SSL2 a.c
```

例 3 : OpenMP C で書かれたユーザプログラム a.c をネイティブコンパイラにより翻訳し, SVE 版 C-SSL II ライブラリを結合します.

```
fcc –Kopenmp,SVE -SSL2 a.c
```

例 4 : C で書かれたユーザプログラム a.c を自動並列化して翻訳しオブジェクトモジュールを作り, SVE 版 C-SSL II ライブラリを結合して OpenMP C のロードモジュールを作ります.

```
fccpx -Kparallel -c a.c
fccpx –Kparallel,openmp,SVE -SSL2 a.o
```

例 5 : C で書かれたユーザプログラム a.c を逐次実行するプログラムとして翻訳しオブジェクトモジュールを作り, SVE 版 C-SSL II ライブラリを結合して OpenMP C のロードモジュールを作ります.

```
fccpx -c a.c
fccpx –Kopenmp,SVE -SSL2 a.o
```

1.4.5.2 C++プログラムからの利用方法

C-SSL II スレッド並列機能は, C++言語で記述された利用者プログラムに結合して使用することができます. C++から C-SSL II スレッド並列機能を利用する場合, 以下の点に注意する必要があります.

1) 共通インクルードファイル
C-SSL II スレッド並列機能では標準ヘッダファイルとして cssl.h をライブラリと一緒に提供しています. ソースプログラムの先頭で必ずこのヘッダファイルをインクルードしなければなりません. このヘッダファイルの中では, C-SSL II スレッド並列機能の全ての関数を C 言語へのリンケージ指定(extern "C")付きで宣言しています. また, 複素数(dcomplex)型などが宣言されています.

2) メイン関数
メイン関数の名前は `main` または `MAIN__` (大文字の `MAIN` に続けて 2 つのアンダースコア)を使用します.

3) プログラムの翻訳・結合の方法
C-SSL II スレッド並列機能は以下のユーザプログラムから呼び出し, C-SSL II ライブラリと結合して OpenMP C++ のロードモジュールとして実行することができます.

- OpenMP C++ のプログラム
- C++プログラムを自動並列化オプションで翻訳したオブジェクトモジュール
- C++プログラムを逐次実行オプションで翻訳したオブジェクトモジュール

OpenMP C++ のロードモジュールを作るため結合時に `FCCpx` または `FCC` コマンド行に `-Kopenmp` 及び`-SSL2` を指定します.

`-Kopenmp` オプションのかわりに`-Kparallel` オプションを指定して結合することもできます.

`-Nfjomplib` オプションが指定されているかつ,`-Kparallel` オプションのみを指定して結合した場合,実行時に環境変数 `PARALLEL` が指定されると, `OMP_NUM_THREADS` の指定よりも `PARALLEL` の指定のほうが優先されますのでご注意ください.

C-SSL II スレッド並列機能は CPU の種類に応じたライブラリが用意されており, プログラムを実行する機種に合わせてオプションを指定することにより適切なライブラリが結合されます. –KSVE が有効な場合は SVE を使用して高速化したライブラリを結合し, –KNOSVE が有効な場合は汎用のライブラリを結合します. A64FX を搭載した計算ノードで実行する場合, どちらでも動作しますが, 高速に実行するためには SVE を使用したライブラリを推奨します.

clang モードで翻訳する場合(-Nclang が有効)は-KSVE・-KNOSVE オプションの代わりに-march オプションに+sve または+nosve を有効にすることによりライブラリが選択されます.

例 1 : OpenMP C++ で書かれたユーザプログラム a.cc をクロスコンパイラにより翻訳し, SVE 版 C-SSL II ライブラリを結合します.

```
FCCpx –Kopenmp,SVE -SSL2 a.cc
```

例 2 : OpenMP C++ で書かれたユーザプログラム a.cc をクロスコンパイラにより翻訳し, 汎用版 C-SSL II ライブラリを結合します.

```
FCCpx –Kopenmp,NOSVE -SSL2 a.cc
```

例 2 : OpenMP C++ で書かれたユーザプログラム a.cc をネイティブコンパイラにより翻訳し, SVE 版 C-SSL II ライブラリを結合します.

```
FCC –Kopenmp,SVE -SSL2 a.cc
```

例 3 : C++ で書かれたユーザプログラム a.cc を自動並列化して翻訳しオブジェクトモジュールを作り, SVE 版 C-SSL II ライブラリを結合して OpenMP C++ のロードモジュールを作ります.

```
FCCpx -Kparallel -c a.cc
FCCpx –Kparallel,openmp,SVE -SSL2 a.o
```

例 4 : C++ で書かれたユーザプログラム a.cc を逐次実行するプログラムとして翻訳しオブジェクトモジュールを作り, SVE 版 C-SSL II ライブラリを結合して OpenMP C++ のロードモジュールを作ります.

```
FCCpx -c a.cc
FCCpx –Kopenmp,SVE -SSL2 a.o
```

1.4.5.3 実行

翻訳・結合したユーザプログラムを並列実行するスレッド数 及び 各スレッドのスタック領域の大きさの設定について説明します.

- C-SSL II スレッド並列機能のサブルーチンを並列実行するスレッド数は, 以下の環境変数で指定することができます.

  環境変数 OMP_NUM_THREADS にスレッド数を指定します.

- 呼び出しているユーザプログラムに自動並列を指定して翻訳した場合, 自動並列化されたプログラムを並列実行するスレッド数は以下の環境変数で指定します.

  -Nfjomplib オプションが指定されていない場合, 環境変数 OMP_NUM_THREADS にスレッド数を指定します.

  -Nfjomplib オプションが指定されている場合, 環境変数 PARALLEL にスレッド数を指定します.

- C-SSL II スレッド並列機能のルーチンの内部で, 各スレッド毎の作業域をスタック領域に割り付けています. 生成するスレッドの数を NT, 利用できるメモリ量を M としたとき, 各スレッドのスタック領域の大きさとして環境変数

OMP_STACKSIZE に M/(5*NT) 程度を指定して実行することを勧めます. -Nfjomplib オプションが指定されている場合,環境変数 THREAD_STACK_SIZE にスタック領域の大きさを指定する事もできます.

1.4.5.4 注意事項

C-SSL II スレッド並列機能を使用する上で注意すべき事項について説明します.

- スレッド数について
  C-SSL II スレッド並列でのスレッド数の制限は特にありません.
- ライブラリのアーカイブに含まれるルーチンついて
  C-SSL II のアーカイブには, BLAS, LAPACK, C-SSL II 及び SSL II スレッド並列機能 のルーチンも含まれています. その他, ルーチン名の先頭が, SS_または #L_(#は S,D,C,Z,I,X)で始まるスレーブルーチンが含まれています. 利用するときはルーチン名の重複に注意してください.
- セクタキャッシュに関する警告メッセージについて
  数学ライブラリでは A64FX CPU のセクタキャッシュ機能を使って高速化しているルーチンがあります. プログラムの実行方法によってセクタキャッシュを利用できない場合があり, 以下のような警告メッセージを出力する場合があります. この場合, セクタキャッシュが使われないため性能に影響があるかもしれませんが, 実行は継続され計算は正しく行われます.

  jwe1047i-w A sector cache couldn't be used.

1.4.6 SSL II/MPI

SSL II/MPI は MPI と OpenMP Fortran によるハイブリッド並列のサブルーチンとして提供されています. これらを利用するユーザプログラムを翻訳して SSL II/MPI 及びそれらが利用している SSL II スレッド並列機能の内部ルーチンを結合して, MPI を利用する Fortran のロードモジュールとして実行する方法を説明します. なお, 翻訳から実行までの手順の詳細については“Fortran 使用手引書”または“MPI 使用手引書”を参照してください.

1.4.6.1 翻訳・結合

翻訳, 結合には富士通製 Fortran コンパイラを使用します.

SSL II/MPI を使用するユーザプログラムをクロスコンパイラにより翻訳し SSL II/MPI を結合するためには, `mpifrtpx` コマンドを使用し, オプションとして `-Kopenmp`, `-SSL2MPI` 及び, `-SSL2` または`-SSL2BLAMP` オプションを指定します.

SSL II/MPI は MPI と OpenMP Fortran のハイブリッド並列向けサブルーチンであるため, 結合時には`-Kopenmp` の指定が必要です. SSL II/MPI の利用する SSL II スレッド並列機能の内部ルーチンは, `-SSL2` でも`-SSL2BLAMP` でも結合することができます. 利用する BLAS, LAPACK の種類により, `-SSL2` または`-SSL2BLAMP` オプションを選択します. SSLII/MPI と合わせて, ユーザプログラムで BLAS/LAPACK をお使いの場合, 逐次/スレッド並列に応じて, `-SSL2` または `-SSL2BLAMP` オプションを選択します.

`-Kopenmp` オプションのかわりに`-Kparallel` オプションを指定して結合することもできます.

`-Nfjomplib` オプションが指定されているかつ,`-Kparallel` オプションのみを指定して結合した場合,実行時に環境変数 `PARALLEL` が指定されると,

OMP_NUM_THREADS の指定よりも PARALLEL の指定のほうが優先されますのでご注意ください.

SSL II/MPI は CPU の種類に応じたライブラリが用意されており, プログラムを実行する機種に合わせてオプションを指定することにより適切なライブラリが結合されます. –KSVE が有効な場合は SVE を使用して高速化したライブラリを結合し, –KNOSVE が有効な場合は汎用のライブラリを結合します. A64FX を搭載した計算ノードで実行する場合, どちらでも動作しますが, 高速に実行するためには SVE を使用したライブラリを推奨します.

例 1 : MPI で書かれたユーザプログラム a.f を翻訳し, SVE 版 SSL II/MPI ライブラリ及び BLAS, LAPACK 逐次版を結合します.

```
mpifrtpx –Kopenmp,SVE -SSL2MPI -SSL2 a.f
```

例 2 : MPI で書かれたユーザプログラム a.f を翻訳し, 汎用版 SSL II/MPI ライブラリ及び BLAS, LAPACK 逐次版を結合します.

```
mpifrtpx –Kopenmp,NOSVE -SSL2MPI -SSL2 a.f
```

例 3 : MPI で書かれたユーザプログラム a.f を翻訳し, SVE 版 SSL II/MPI ライブラリ及び BLAS, LAPACK スレッド並列版を結合します.

```
mpifrtpx –Kopenmp,SVE -SSL2MPI -SSL2BLAMP a.f
```

例 4 : MPI で書かれたユーザプログラム a.f を翻訳してオブジェクトを作成し, SVE 版 SSL II/MPI ライブラリを結合します. BLAS, LAPACK はスレッド並列版を結合します.

```
mpifrtpx -c a.f
mpifrtpx –Kopenmp,SVE -SSL2MPI -SSL2BLAMP a.o
```

ネイティブコンパイラによりユーザプログラムを翻訳し, SSL II/MPI を結合するためには mpifrt コマンドを使用します.

例 5 : MPI で書かれたユーザプログラム a.f を翻訳し, SVE 版 SSL II/MPI ライブラリ及び BLAS, LAPACK 逐次版を結合します.

```
mpifrt –Kopenmp,SVE -SSL2MPI -SSL2 a.f
```

例 6: MPI で書かれたユーザプログラム a.f を翻訳し, SVE 版 SSL II/MPI ライブラリ及び BLAS, LAPACK スレッド並列版を結合します.

```
mpifrt –Kopenmp,SVE -SSL2MPI -SSL2BLAMP a.f
```

例 7 : MPI で書かれたユーザプログラム a.f を翻訳してオブジェクトを作成し, SVE 版 SSL II/MPI ライブラリを結合します. BLAS, LAPACK はスレッド並列版を結合します.

```
mpifrt -c a.f
mpifrt –Kopenmp,SVE -SSL2MPI -SSL2BLAMP a.o
```

1.4.6.2 C, C++プログラムからの利用方法

SSL II/MPI は C, C++で書かれたユーザプログラムから利用することもできます. 翻訳, 結合には富士通製 C, C++コンパイラを使用します.

SSL II/MPI を使用するユーザプログラムをクロスコンパイラにより翻訳し SSL II/MPI を結合するためには, mpifccpx または mpiFCCpx コマンドを使用し, オプションとして-Kopenmp, -SSL2MPI 及び, -SSL2 または-SSL2BLAMP オプションを指定します.

SSL II/MPI は MPI と OpenMP Fortran のハイブリッド並列向けサブルーチンであるため, 結合時には-Kopenmp の指定が必要です. SSL II/MPI の利用する SSL II スレッド並列機能の内部ルーチンは, -SSL2 でも-SSL2BLAMP でも結合することができます. 利用する BLAS, LAPACK の種類により, -SSL2 または-SSL2BLAMP オプションを選択します. SSLII/MPI と合わせて, ユーザプログラムで BLAS/LAPACK をお使いの場合, 逐次/スレッド並列に応じて, -SSL2 または -SSL2BLAMP オプションを選択します.

-Kopenmp オプションのかわりに-Kparallel オプションを指定して結合することもできます.

-Nfjomplib オプションが指定されているかつ,-Kparallel オプションのみを指定して結合した場合,実行時に環境変数 PARALLEL が指定されると, OMP_NUM_THREADS の指定よりも PARALLEL の指定のほうが優先されますのでご注意ください.

SSL II/MPI は CPU の種類に応じたライブラリが用意されており, プログラムを実行する機種に合わせてオプションを指定することにより適切なライブラリが結合されます. –KSVE が有効な場合は SVE を使用して高速化したライブラリを結合し, –KNOSVE が有効な場合は汎用のライブラリを結合します. A64FX を搭載した計算ノードで実行する場合, どちらでも動作しますが, 高速に実行するためには SVE を使用したライブラリを推奨します.

clang モードで翻訳する場合(-Nclang が有効)は-KSVE･-KNOSVE オプションの代わりに-march オプションに+sve または+nosve を有効にすることによりライブラリが選択されます.

例 1 : C 言語で書かれたユーザプログラム a.c を翻訳し, SVE 版 SSL II/MPI ライブラリ及び BLAS, LAPACK 逐次版を結合します.

```
mpifccpx –Kopenmp,SVE -SSL2MPI -SSL2 a.c
```

例 2 : C 言語で書かれたユーザプログラム a.c を翻訳し, 汎用版 SSL II/MPI ライブラリ及び BLAS, LAPACK 逐次版を結合します.

```
mpifccpx –Kopenmp,NOSVE -SSL2MPI -SSL2 a.c
```

例 3 : C 言語で書かれたユーザプログラム a.c を翻訳し, SVE 版 SSL II/MPI ライブラリ及び BLAS, LAPACK スレッド並列版を結合します.

```
mpifccpx –Kopenmp,SVE -SSL2MPI -SSL2BLAMP a.c
```

例 4 : C 言語で書かれたユーザプログラム a.c を翻訳してオブジェクトを作成し, SVE 版 SSL II/MPI ライブラリを結合します. BLAS, LAPACK はスレッド並列版を結合します.

```
mpifccpx -c a.c
mpifccpx –Kopenmp,SVE -SSL2MPI -SSL2BLAMP a.o
```

例 5 : C++で書かれたユーザプログラム a.cpp を翻訳し, SVE 版 SSL II/MPI ライブラリ及び BLAS, LAPACK スレッド並列版を結合します.

```
mpiFCCpx –Kopenmp,SVE -SSL2MPI -SSL2BLAMP a.cpp
```

ネイティブコンパイラによりユーザプログラムを翻訳し, SVE 版 SSL II/MPI を結合するためには mpifcc または mpiFCC コマンドを使用します.

例 6 : C 言語で書かれたユーザプログラム a.c を翻訳し, SVE 版 SSL II/MPI ライブラリ及び BLAS, LAPACK 逐次版を結合します.

```
mpifcc –Kopenmp,SVE -SSL2MPI -SSL2 a.c
```

例 7 : C 言語で書かれたユーザプログラム a.c を翻訳し, SVE 版 SSL II/MPI ライブラリ及び BLAS, LAPACK スレッド並列版を結合します.

```
mpifcc –Kopenmp,SVE -SSL2MPI -SSL2BLAMP a.c
```

例 8 : C 言語で書かれたユーザプログラム a.c を翻訳してオブジェクトを作成し, SVE 版 SSL II/MPI ライブラリを結合します. BLAS, LAPACK はスレッド並列版を結合します.

```
mpifcc -c a.c
mpifcc –Kopenmp,SVE -SSL2MPI -SSL2BLAMP a.o
```

例 9 : C++で書かれたユーザプログラム a.cpp を翻訳し, SVE 版 SSL II/MPI ライブラリ及び BLAS, LAPACK スレッド並列版を結合します.

```
mpiFCC –Kopenmp,SVE -SSL2MPI -SSL2BLAMP a.cpp
```

SSL II/MPI は Fortran ルーチンであり, Fortran と C の言語間結合規則に従うため, 以下の注意が必要です.

結合規則については, “C 言語使用手引書”を参照してください.

- メイン関数の名前は main または MAIN__ (大文字の MAIN に続けて 2 つのアンダースコア)を使用します.
- SSL II/MPI のルーチン名の末尾には _ (アンダースコア)を付加します.
- 配列以外の変数パラメタには先頭に&を付加します.
- Fortran と C では, 多次元配列における各配列要素の記憶順序が異なります. 多次元配列を扱うルーチンの場合, Fortran の記憶順序で計算を行うため注意が必要です.
- SSL II/MPI ルーチンの引数として渡すコミュニケータは Fortran のコミュニケータにしてください. C のコミュニケータは MPI 関数の MPI_Comm_c2f で Fortran のコミュニケータに変換してください.

1.4.6.3 実行

SSL II/MPI ライブラリを結合したユーザプログラムは, mpiexec コマンドを使って実行します. 使用するスレッド数は環境変数 OMP_NUM_THREADS で指定します. プロセス数が n, スレッド数が s なら, CORE 数として n×s が必要です.

プロセス並列のみの実行時には, スレッド数 1 と設定する必要があります.

例 : プロセス数に 2 を, スレッド数に 4 を指定して a.out を実行します.

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

SSL II スレッド並列機能のサブルーチンの内部で, 各スレッド毎の作業域を自動割付け配列で割り付けています. 生成するスレッドの数を NT, 利用できるメモリ量を M としたとき, 各スレッドのスタック領域の大きさとして環境変数 OMP_STACKSIZE に M/(5*NT) 程度を指定して実行することを勧めます. -Nfjomplib オプションが指定されている場合, 環境変数 THREAD_STACK_SIZE にスタック領域の大きさを指定する事もできます.

1.4.6.4 注意事項

SSL II/MPI を使用する上で注意すべき事項について説明します.

- ライブラリのアーカイブに含まれるルーチンついて
  SSL II/MPI のサブルーチンは, DS_U または SS_U で始まるスレーブルーチンを利用しています.
  SSL II のアーカイブには, BLAS, LAPACK, ScaLAPACK, C-SSL II 及び SSL II スレッド並列機能のルーチンも含まれています. その他, ルーチン名の先頭が, SS_または#L_(#は S, D, C, Z, I, X)で始まるスレーブルーチンが含まれています. 利用するときはルーチン名の重複に注意してください.

- MRQ オーバーフローについて
  SSL II/MPI を使用したプログラムで, 1 つのプロセスが他の大量のプロセスと一度に通信を行うような処理を実行した場合に, 実行時に [mpi::common-tofu::tofu-mrq-overflow]から始まる MPI のエラーメッセージが出力される場合があります.
  このときは, MPI の MCA パラメタ common_tofu_num_mrq_entries で完了キューのエントリ数を増やして実行してください. エラーメッセージおよび MCA パラメタの詳細は"MPI 使用手引書"を参照してください.

2. BLAS, LAPACK, ScaLAPACK

2.1 概説

BLAS, LAPACK, ScaLAPACK は, Prof. Dongarra (テネシー大学) を中心とした研究グループで開発され Netlib (http://www.netlib.org/)で公開されている BLAS (Basic Linear Algebra Subprograms), LAPACK (Linear Algebra PACKage)及び ScaLAPACK (Scalable LAPACK)を, 当社サーバ向けに高度に性能チューニングを行い提供するものです.

弊社 BLAS, LAPACK, ScaLAPACK は以下のような特徴があります.

- BLAS はスカラ CPU 向けにチューニングしています. 特に A64FX CPU の性能を引き出すチューニングを行っております. BLAS の高速化により, これを call している LAPACK, さらには ScaLAPACK もその分, 高速化されたと言えます.
- BLAS, LAPACK については, 逐次版とスレッド並列版を提供いたします.
- BLAS, LAPACK 逐次版は, スレッドセーフな作りになっています. これにより, 従来の逐次 Fortran プログラムからは勿論のこと, OpenMP Fortran で記述されたスレッド並列プログラムからも call できます. 後者では, 複数のスレッドに別々のデータを与えて同じルーチンを call するという使い方が可能です.
- BLAS, LAPACK スレッド並列版は, OpenMP Fortran によって並列化されたルーチンを提供いたします. 大規模な問題を複数 CPU を使って並列計算することにより, 計算時間を短縮することができます. スレッド並列版もスレッドセーフな作りになっています.
- LAPACK を pthread で並列化した PLASMA を提供します. ルーチンによっては OpenMP で並列化した LAPACK よりも高速に実行されるものがあります. PLASMA はスレッドセーフではありませんのでご注意ください.
- ScaLAPACK はスレッド並列版 BLAS, LAPACK と組み合わせて利用することができます. これにより, ScaLAPACK レイヤーでは MPI で並列に計算し, ScaLAPACK から呼び出す BLAS, LAPACK ではマルチスレッドにより並列計算することができます.

サポートしているルーチンの一覧及び使用例については“BLAS LAPACK ScaLAPACK 使用手引書”を参照してください.

2.2 翻訳・結合・実行方法

BLAS, LAPACK は Fortran/C/C++で記述された利用者プログラムに結合して使用することができます. 翻訳, 結合には富士通製 Fortran/C/C++コンパイラを使用します. また, ScaLAPACK は Fortran/C/C++で記述された利用者プログラムに結合して使用することができます. ここではこれらの製品を利用するプログラムを翻訳, 結合及び実行する方法を説明します. プログラムの翻訳方法及び実行方法の詳細については“Fortran 使用手引書”または“MPI 使用手引書”を参照してください.

ScaLAPACK を使用する利用者プログラムが BLAS または LAPACK ルーチンを呼び出している場合は, “2.2.4 ScaLAPACK”で述べる結合方法に従えば, 一緒に結合される BLAS, LAPACK ルーチンを使用することができます.

2.2.1 準備

本製品を使用するためには以下の設定が必要です. “*製品インストールパス*”は, システム管理者にお問い合わせください.

- ログインノードでクロスコンパイルによりプログラムの翻訳および結合を行うために

  環境変数 PATH に/*製品インストールパス*/bin を追加します.

- ネイティブコンパイラによりプログラムの翻訳および結合を行うために

  環境変数 PATH に/*製品インストールパス*/bin を追加します.

- 計算ノードでプログラムの実行を行うために

  環境変数 LD_LIBRARY_PATH に/*製品インストールパス*/lib64 を追加します.

また, ScaLAPACK を利用する場合には, 上記の他に以下の設定が必要です.

- 計算ノードでプログラムの実行を行うために

  環境変数 PATH に/*製品インストールパス*/bin を追加します.

2.2.2 BLAS, LAPACK 逐次版

a) ライブラリの結合方法

ユーザプログラムをクロスコンパイラにより翻訳し, BLAS 及び LAPACK 逐次版を結合するためには, `frtpx` コマンド行に`-SSL2` を指定します.

BLAS, LAPACK 逐次版は CPU の種類に応じたライブラリが用意されており, プログラムを実行する機種に合わせてオプションを指定することにより適切なライブラリが結合されます. `–KSVE` が有効な場合は SVE を使用して高速化したライブラリを結合し, `–KNOSVE` が有効な場合は汎用のライブラリを結合します. A64FX を搭載した計算ノードで実行する場合, どちらでも動作しますが, 高速に実行するためには SVE を使用したライブラリを推奨します.

例 1：ユーザプログラム `a.f` を翻訳し, SVE 版 BLAS・LAPACK アーカイブを結合します.

```
frtpx –KSVE a.f -SSL2
```

例 2：ユーザプログラム `a.f` を翻訳し, 汎用版 BLAS・LAPACK アーカイブを結合します.

```
frtpx –KNOSVE a.f -SSL2
```

ユーザプログラムが OpenMP Fortran で書かれている場合は, オプションに `-Kopenmp` を指定します.

例 3：OpenMP Fortran で書かれたユーザプログラム `a.f` を翻訳し, SVE 版 BLAS・LAPACK アーカイブを結合します.

```
frtpx –Kopenmp,SVE a.f -SSL2
```

ネイティブコンパイラによりユーザプログラムを翻訳し, BLAS 及び LAPACK 逐次版を結合するためには, frt コマンドを使用します.

例 4：ユーザプログラム `a.f` を翻訳し, SVE 版 BLAS・LAPACK アーカイブを結合します.

```
frt –KSVE a.f -SSL2
```

b) スタックサイズに関する注意

プログラムを-Kopenmp オプションを指定して翻訳すると, メインプログラム以外のローカルな配列等がスタック上に割り付けられます. また, BLAS, LAPACK のルーチンにもスタック上の作業域を使うものがあります. このため, 実行時にスタックが不足しないよう“2.3.4 BLAS, LAPACKルーチンが使用する作業域について”を参照して, スタックサイズを拡張してプログラムを実行します.

c) スレッド数の指定方法

ユーザプログラムが複数スレッドを使用する場合, 実行時にプログラムが使用するスレッドの個数は, 環境変数 OMP_NUM_THREADS などで指定できます.

2.2.3 BLAS, LAPACK スレッド並列版

a) ライブラリの結合方法

ユーザプログラムを翻訳し, BLAS 及び LAPACK のスレッド並列版(PLASMA を含む)を結合するためには, frtpx または frt コマンド行に-Kopenmp 及び -SSL2BLAMP を指定します.

-Kopenmp オプションのかわりに-Kparallel オプションを指定することもできます. このとき, “2.3.6 スレッド並列版BLAS, LAPACKを-Kparallelオプションで結合する場合の注意”に示す注意事項があります.

-SSL2BLAMP と-SSL2 は排他的です. 誤って, 2 つとも指定した場合は, 後ろのほうが有効になるので注意してください.

BLAS, LAPACK スレッド並列版は CPU の種類に応じたライブラリが用意されており, プログラムを実行する機種に合わせてオプションを指定することにより適切なライブラリが結合されます. –KSVE が有効な場合は SVE を使用して高速化したライブラリを結合し, –KNOSVE が有効な場合は汎用のライブラリを結合します. A64FX を搭載した計算ノードで実行する場合, どちらでも動作しますが, 高速に実行するためには SVE を使用したライブラリを推奨します.

例 1: OpenMP Fortran で書かれたユーザプログラム a.f をクロスコンパイラにより翻訳し, SVE 版 BLAS・LAPACK アーカイブを結合します.

```
frtpx –Kopenmp,SVE a.f -SSL2BLAMP
```

例 2: OpenMP Fortran で書かれたユーザプログラム a.f をクロスコンパイラにより翻訳し, 汎用版 BLAS・LAPACK アーカイブを結合します.

```
frtpx –Kopenmp,NOSVE a.f -SSL2BLAMP
```

例 3: OpenMP Fortran で書かれたユーザプログラム a.f をネイティブコンパイラにより翻訳し, SVE 版 BLAS・LAPACK アーカイブを結合します.

```
frt –Kopenmp,SVE a.f -SSL2BLAMP
```

例 4: Fortran で書かれたユーザプログラム a.f を逐次実行プログラムとして翻訳し, SVE 版 BLAS・LAPACK アーカイブを結合します.

```
frtpx -c a.f
frtpx –Kopenmp,SVE a.o -SSL2BLAMP
```

例 5: Fortran で書かれたユーザプログラム a.f を自動並列化して翻訳し, SVE 版 BLAS・LAPACK アーカイブを結合します.

```
frtpx -c -Kparallel a.f
```

```
frtpx –Kparallel,openmp,SVE a.o -SSL2BLAMP
```

b) スタックサイズに関する注意

プログラムを-Kopenmp オプションを指定して翻訳すると, メインプログラム以外のローカルな配列等がスタック上に割り付けられます. また, BLAS, LAPACK のルーチンにもスタック上の作業域を使うものがあります. このため, 実行時にスタックが不足しないよう“2.3.4 BLAS, LAPACKルーチンが使用する作業域について”を参照して, スタックサイズを拡張してプログラムを実行します.

c) スレッド数の指定方法

実行時に BLAS, LAPACK が使用するスレッドの個数は, 環境変数 OMP_NUM_THREADS で指定します.

PLASMA が使用するスレッドの個数は, 初期化ルーチン PLASMA_Init の引数で指定します.

d) PLASMA 実行時の環境変数について

-Nfjomplib オプションが指定されているかつ,PLASMA を使用するプログラムを実行する場合 環境変数 FLIB_PTHREAD に 1 を設定してください.

A64FX を搭載したシステムでは, PLASMA を呼び出すプログラムを実行する場合に環境変数 FLIB_SCCR_CNTL に FALSE を指定してください. A64FX 向け BLAS ではセクタキャッシュを使用して高速化しており, pthread で並列化した PLASMA から呼ばれた場合に正常に動作することができないためです.

2.2.4 ScaLAPACK

a) ライブラリの結合方法

ScaLAPACK を使用するユーザプログラムを翻訳し ScaLAPACK を結合するためには, mpifrtpx または mpifrt コマンドを使用し,オプションとして -SCALAPACK 及び, 使用する BLAS, LAPACK の種類に応じて-SSL2 または -SSL2BLAMP オプションを指定します. -SSL2BLAMP オプションを指定する場合, -Kopenmp オプションも一緒に指定します.

ScaLAPACK は CPU の種類に応じたライブラリが用意されており, プログラムを実行する機種に合わせてオプションを指定することにより適切なライブラリが結合されます. –KSVE が有効な場合は SVE を使用して高速化したライブラリを結合し, –KNOSVE が有効な場合は汎用のライブラリを結合します. A64FX を搭載した計算ノードで実行する場合, どちらでも動作しますが, 高速に実行するためには SVE を使用したライブラリを推奨します.

例 1: ユーザプログラム a.f をクロスコンパイラにより翻訳し, SVE 版アーカイブを結合します. BLAS, LAPACK は逐次版を結合します.

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

例 2: ユーザプログラム a.f をクロスコンパイラにより翻訳し, SVE 版アーカイブを結合します. BLAS, LAPACK は逐次版を結合します.

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

例 3: ユーザプログラム a.f をネイティブコンパイラにより翻訳し, SVE 版アーカイブを結合します. BLAS, LAPACK は逐次版を結合します.

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

例 4: ユーザプログラム a.f を翻訳してオブジェクトを作成し, SVE 版アーカイブを結合します. BLAS, LAPACK はスレッド並列版を結合します.

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

b) プログラムの実行方法

ScaLAPACK を結合したユーザプログラムは, mpiexec コマンドを使って実行します. また, ScaLAPACK ではノード間及びノード内の通信を使った並列計算が可能です. 実行方法の詳細については“MPI 使用手引書”を参照してください.

BLAS, LAPACK スレッド並列版を結合した場合, 使用するスレッド数は環境変数 OMP_NUM_THREADS で指定します. プロセス数が *n*, スレッド数が *s* なら, CPU 数として *n*×*s* が必要です.

例 1: a.out をプロセス数に 4 を指定して実行します.

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

例 2: プロセス数に 2 を, スレッド数に 4 を指定して a.out を実行します.

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

2.2.5 C インターフェースの BLACS の使用方法

ScaLAPACK のうち BLACS レイヤーのルーチンは C 言語プログラムからも使用することができます. C インターフェースの BLACS を使用する場合, mpifccpx コマンド行に-lCblacs を指定することにより, 利用者プログラムに結合されます.これは次の例に示す順番で結合します.

例：mpifccpx a.c -lCblacs

C インターフェースの BLACS と結合したプログラムは, ScaLAPACK を結合したプログラムと同様に mpiexec コマンドで実行します.

C インターフェースの BLACS を使用する場合, 次のことに注意する必要があります.

- C 言語のメインプログラム(main) の中で最初に MPI の関数 MPI_Init を呼び出します.
- この方法で結合するのは, 結合すべきプログラムがすべて C 言語のプログラムの場合に限られます. Fortran と C 言語で書かれたプログラムが混在する場合, “2.2.4 ScaLAPACK”で述べた方法で結合してください. このとき-lCblacs を指定する必要はありません.

2.2.6 BLAS, LAPACK, ScaLAPACK の C/C++プログラムからの利用方法

BLAS, LAPACK, ScaLAPACK は, C/C++言語で記述された利用者プログラムと結合して使用することもできます.

2.2.6.1 BLAS, LAPACK 逐次版

BLAS, LAPACK 逐次版と結合するためには fccpx, FCCpx, mpifccpx, mpiFCCpx, fcc, FCC, mpifcc, または mpiFCC コマンドを使用し, -SSL2 を指定します.

BLAS, LAPACK は CPU の種類に応じたライブラリが用意されており, プログラムを実行する機種に合わせてオプションを指定することにより適切なライブラリが結合されます. –KSVE が有効な場合は SVE を使用して高速化したライブラリを結合し, –KNOSVE が有効な場合は汎用のライブラリを結合します. A64FX を搭載した計算ノードで実行する場合, どちらでも動作しますが, 高速に実行するためには SVE を使用したライブラリを推奨します.

clang モードで翻訳する場合(-Nclang が有効)は-KSVE・-KNOSVE オプションの代わりに-march オプションに+sve または+nosve を有効にすることによりライブラリが選択されます.

例 1:ユーザプログラム a.c をクロスコンパイラにより翻訳し, SVE 版 BLAS, LAPACK アーカイブを結合します.

```
fccpx –KSVE –SSL2 a.c
```

例 2:ユーザプログラム a.c をクロスコンパイラにより翻訳し, 汎用版 BLAS, LAPACK アーカイブを結合します.

```
fccpx –KNOSVE –SSL2 a.c
```

例 3：ユーザプログラム a.c をネイティブコンパイラにより翻訳し, SVE 版 BLAS, LAPACK アーカイブを結合します.

```
fcc –KSVE –SSL2 a.c
```

例 4：C++で書かれたユーザプログラム a.cpp を翻訳し, SVE 版 BLAS, LAPACK アーカイブを結合します.

```
FCCpx –KSVE a.cpp –SSL2
```

2.2.6.2 BLAS, LAPACK スレッド並列版

BLAS, LAPACK スレッド並列版(PLASMA を含む)と結合するためには fccpx, FCCpx, mpifccpx, mpiFCCpx, fcc, FCC, mpifcc, または mpiFCC コマンドを使用し,オプションとして-Kopenmp 及び-SSL2BLAMP を指定します.

以下の例では, 代表して fccpx または fcc を使用した例を記述しています. 他のコマンドを使用する場合はそれぞれ置き換えて指定してください.

BLAS, LAPACK は CPU の種類に応じたライブラリが用意されており, プログラムを実行する機種に合わせてオプションを指定することにより適切なライブラリが結合されます. –KSVE が有効な場合は SVE を使用して高速化したライブラリを結合し, –KNOSVE が有効な場合は汎用のライブラリを結合します. A64FX を搭載した計算ノードで実行する場合, どちらでも動作しますが, 高速に実行するためには SVE を使用したライブラリを推奨します.

clang モードで翻訳する場合(-Nclang が有効)は-KSVE・-KNOSVE オプションの代わりに-march オプションに+sve または+nosve を有効にすることによりライブラリが選択されます.

例 1: OpenMP C で書かれたユーザプログラム a.c をクロスコンパイラにより翻訳し, SVE 版 BLAS, LAPACK アーカイブを結合します.

```
fccpx –Kopenmp,SVE a.c -SSL2BLAMP
```

例 2: OpenMP C で書かれたユーザプログラム a.c をクロスコンパイラにより翻訳し, 汎用版 BLAS, LAPACK アーカイブを結合します.

```
fccpx –Kopenmp,NOSVE a.c -SSL2BLAMP
```

例 3: OpenMP C で書かれたユーザプログラム a.c をネイティブコンパイラにより翻訳し, SVE 版 BLAS, LAPACK アーカイブを結合します.

```
fcc –Kopenmp,SVE a.c -SSL2BLAMP
```

例 4: C 言語で書かれたユーザプログラム a.c を逐次実行プログラムとして翻訳し, 汎用版 BLAS, LAPACK アーカイブを結合します.

```
fccpx -c a.c
fccpx –Kopenmp,SVE a.o -SSL2BLAMP
```

例 5: C 言語で書かれたプログラム a.c を自動並列化して翻訳し, SVE 版 BLAS, LAPACK アーカイブを結合します.

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

例 6: C++で書かれたプログラム a.cpp を翻訳し, SVE 版 BLAS, LAPACK アーカイブを結合します.

```
FCCpx –Kopenmp,SVE a.cpp –SSL2BLAMP
```

-Kopenmp オプションのかわりに-Kparallel オプションを指定することもできます.このとき, “2.3.6 スレッド並列版BLAS, LAPACKを-Kparallelオプションで結合する場合の注意”に示す注意事項があります.

2.2.6.3 ScaLAPACK

ScaLAPACK を使用するユーザプログラムを翻訳し ScaLAPACK を結合するためには, mpifccpx, mpiFCCpx, mpifcc,または mpiFCC コマンドを使用し,オプションとして-SCALAPACK 及び, 使用する BLAS, LAPACK の種類に応じて-SSL2 または -SSL2BLAMP オプションを指定します. -SSL2BLAMP オプションを指定する場合, -Kopenmp オプションも一緒に指定します.

以下の例では, 代表して mpifccpx または mpifcc を使用した例を記述しています. mpiFCCpx を使用する場合はそれぞれ置き換えて指定してください.

ScaLAPACK は CPU の種類に応じたライブラリが用意されており, プログラムを実行する機種に合わせてオプションを指定することにより適切なライブラリが結合されます. –KSVE が有効な場合は SVE を使用して高速化したライブラリを結合し, –KNOSVE が有効な場合は汎用のライブラリを結合します. A64FX を搭載した計算ノードで実行する場合, どちらでも動作しますが, 高速に実行するためには SVE を使用したライブラリを推奨します.

clang モードで翻訳する場合(-Nclang が有効)は-KSVE・-KNOSVE オプションの代わりに-march オプションに+sve または+nosve を有効にすることによりライブラリが選択されます.

例 1: ユーザプログラム a.c をクロスコンパイラにより翻訳し, SVE 版アーカイブを結合します. BLAS, LAPACK は逐次版を結合します.

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

例 2: ユーザプログラム a.c をクロスコンパイラにより翻訳し, 汎用版アーカイブを結合します. BLAS, LAPACK は逐次版を結合します.

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

例 3: ユーザプログラム a.c をネイティブコンパイラにより翻訳し, SVE 版アーカイブを結合します. BLAS, LAPACK は逐次版を結合します.

```
mpifcc –KSVE a.c -SSL2 -SCALAPACK
```

例 4: ユーザプログラム a.c を翻訳し, SVE 版アーカイブを結合します. BLAS, LAPACK はスレッド並列版を結合します.

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

例 5: ユーザプログラム a.cpp を翻訳し, SVE 版アーカイブを結合します. BLAS, LAPACK はスレッド並列版を結合します.

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

2.2.6.4 注意事項

BLAS, LAPACK, ScaLAPACK は Fortran ルーチンであり, Fortran と C の言語間結合規則に従うため, 以下の注意が必要です.
結合規則については, “C 言語使用手引書”を参照してください.

- メイン関数の名前は `main` または `MAIN__` (大文字の `MAIN` に続けて 2 つのアンダースコア)を使用します.
- ルーチン名の末尾には _ (アンダースコア)を付加します.
- 配列以外の変数パラメタには先頭に&を付加します.
- Fortran と C では, 2 次元配列における各配列要素の記憶順序が異なります.
  2 次元配列を扱うルーチンの場合, Fortran の記憶順序で計算を行うため注意が必要です.

BLAS, LAPACK については, C/C++プログラムから呼び出す場合のプロトタイプ宣言を記述したヘッダファイルとして `fj_lapack.h` が提供されています. このヘッダファイルを使う場合, 次のような注意が必要です.

- ヘッダファイルには, BLAS の全ルーチン(XBLAS は C インターフェースのみ), LAPACK のドライバルーチン・計算ルーチン及び一部の補助ルーチンを呼び出すためのプロトタイプ宣言が記述されています.
- 複素数型を構造体を使って表わし, 次のように `typedef` を使って `fcomplex` および `dcomplex` という名前で複素数型を宣言しています.

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

typedef struct {
   double  re, im;
```

```
} dcomplex;
```

- 2 次元配列 a はポインタに型変換して渡す必要があります. すなわち, double 型の 2 次元配列ならば, (double *) a と書きます.
- 文字型の引数を受け渡す場合は, 引数長も実引数として渡します. 引数長を表す実引数は, 指定されたすべての引数の後ろに, 文字型の引数の個数分だけ存在します.
- Fortran で関数結果が複素数型の関数は, 関数値に設定されます.

  例)

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

LAPACK については, C インターフェースで利用できるルーチンが利用できます. C インターフェースの LAPACK を使用する場合, lapacke.h をインクルードしてください. lapacke.h では,ルーチンのプロトタイプ宣言,複素数の型宣言および配列の格納順序を示す引数の値が定義されています.

2.2.7 共有ライブラリの利用方法

BLAS, LAPACK, ScaLAPACK は 2.2.2～2.2.6 で示した方法で結合すると, スタティックリンクになります. これとは別に, 共有ライブラリも提供しています. 一般的な利用ではスタティックリンク版で十分ですが, dlopen, dlsym 関数で動的ロードして使用する場合などスタティック版では対応できない場合には共有ライブラリをご利用ください. 共有ライブラリ版は動的ロードして使う場合を考慮し、アーカイブの構成および結合方法がスタティックリンク版と異なっています.

共有ライブラリは結合時に汎用版と SVE 版を選択することができます. A64FX を搭載した計算ノードで実行するときはどちらの場合も SVE を使って高速化したライブラリを使って実行されます.

さらに, BLAS, LAPACK では整数型および論理型の引数が 4 バイト整数である LP64 インターフェースのライブラリに加えて, 整数型および論理型の引数を 8 バイト整数とする ILP64 インターフェースのライブラリを使うことができます.

2.2.7.1 BLAS, LAPACK 逐次版の Fortran からの利用方法

Fortran で書かれたユーザプログラムを翻訳し, BLAS 及び LAPACK の逐次版の共有ライブラリを結合するためには, frtpx または frt コマンド行に表1に示すオプションを指定します. このオプションはユーザプログラムよりも後ろに指定します.

表1 BLAS, LAPACK **逐次版を結合するためのオプション**(Fortran)

	結合オプション
LP64, 汎用版	-lfjlapack
LP64, SVE 版	-lfjlapacksve
ILP64, 汎用版	-lfjlapack_ilp64
ILP64, SVE 版	-lfjlapacksve_ilp64

例 1: Fortran で書かれたユーザプログラム a.f を翻訳し, SVE 版 BLAS, LAPACK ライブラリを結合します.

```
frtpx a.f -lfjlapacksve
```

例 2: Fortran で書かれたユーザプログラム a.f を翻訳し, 汎用版 BLAS, LAPACK ライブラリを結合します.

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

例 3: Fortran で書かれたユーザプログラム a.f を翻訳し, ILP64 インターフェースの SVE版BLAS, LAPACKライブラリを結合します. ソースプログラムの基本整数型および基本論理型は`-CcdII8 -CcdLL8` オプションで 8 バイトに拡張します.

```
frtpx -CcdII8 -CcdLL8 a.f –lfjlapacksve_ilp64
```

例 4: OpenMP Fortran で書かれたユーザプログラム a.f を翻訳し, SVE 版 BLAS, LAPACK ライブラリを結合します.

```
frtpx -Kopenmp a.f -lfjlapacksve
```

例 5: Fortran で書かれたユーザプログラム a.f をネイティブコンパイラで翻訳し, SVE 版 BLAS, LAPACK ライブラリを結合します.

```
frt a.f -lfjlapacksve
```

2.2.7.2 BLAS, LAPACK スレッド並列版の Fortran からの利用方法

Fortran で書かれたユーザプログラムを翻訳し, BLAS 及び LAPACK のスレッド並列版(PLASMA を含む)の共有ライブラリを結合するためには, `frtpx` または `frt` コマンド行にオプション`-Kopenmp` に加えて表2に示すオプションを指定します. このオプションはユーザプログラムよりも後ろに指定します.

表2 BLAS, LAPACK **スレッド並列版を結合するためのオプション**(Fortran)

	結合オプション
LP64, 汎用版	`-lfjlapackex`
LP64, SVE 版	`-lfjlapackexsve`
ILP64, 汎用版	`-lfjlapackex_ilp64`
ILP64, SVE 版	`-lfjlapackexsve_ilp64`

`-Kopenmp` オプションのかわりに`-Kparallel` オプションを指定することもできます. このとき, "2.3.6 スレッド並列版BLAS, LAPACKを`-Kparallel`オプションで結合する場合の注意"に示す注意事項があります.

例 1: Fortran で書かれたユーザプログラム a.f を翻訳し, SVE 版 BLAS・LAPACK スレッド並列版ライブラリを結合します.

```
frtpx -Kopenmp a.f -lfjlapackexsve
```

例 2: Fortran で書かれたユーザプログラム a.f を翻訳し, 汎用版 BLAS・LAPACK スレッド並列版ライブラリを結合します.

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

例 3: Fortran で書かれたユーザプログラム a.f を翻訳し, ILP64 インターフェースの SVE 版 BLAS・LAPACK スレッド並列版ライブラリを結合します. ソースプログラムの基本整数型および基本論理型は`-CcdII8 -CcdLL8` オプションで 8 バイトに拡張します.

```
frtpx -Kopenmp -CcdII8 -CcdLL8 a.f –lfjlapackexsve_ilp64
```

例 4: Fortran で書かれたユーザプログラム a.f をネイティブコンパイラで翻訳し, SVE 版 BLAS, LAPACK スレッド並列版ライブラリを結合します.

```
frt -Kopenmp a.f -lfjlapackexsve
```

例 5: Fortran で書かれたユーザプログラム a.f を逐次実行プログラムとして翻訳し, SVE 版 BLAS・LAPACK スレッド並列版ライブラリを結合します.

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

例 6: Fortran で書かれたユーザプログラム a.f を自動並列化して翻訳し, SVE 版 BLAS・LAPACK スレッド並列版ライブラリを結合します.

```
frtpx -c -Kparallel a.f
frtpx -Kparallel,openmp a.o –lfjlapackexsve
```

2.2.7.3 ScaLAPACK の Fortran からの利用方法

ScaLAPACK を使用するユーザプログラムを翻訳し ScaLAPACK の共有ライブラリを結合するためには, `mpifrtpx` または `mpifrt` コマンドを使用し, 表3に示すオプション及び, 使用する BLAS, LAPACK の種類に応じて表1または表2に示すオプションを指定します.

表3 ScaLAPACK を結合するためのオプション(Fortran)

	結合オプション
LP64, 汎用版	`-lfjscalapack`
LP64, SVE 版	`-lfjscalapacksve`
ILP64, 汎用版	-
ILP64, SVE 版	-

ScaLAPACK は ILP64 版はサポートしていません.

ScaLAPACK を結合するオプションはユーザプログラムよりも後ろに指定し, BLAS・LAPACK を結合するためのオプションよりも前に指定します. スレッド並列版 BLAS・LAPACK を使用する場合, `-Kopenmp` オプションも一緒に指定します.

例 1: Fortran で書かれたユーザプログラム a.f を翻訳し, SVE 版 ScaLAPACK ライブラリを結合します. BLAS, LAPACK は逐次版を結合します.

```
mpifrtpx a.f -lfjscalapacksve -lfjlapacksve
```

例 2: Fortran で書かれたユーザプログラム a.f を翻訳し, 汎用版 ScaLAPACK ライブラリを結合します. BLAS, LAPACK は逐次版を結合します.

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

例 3: Fortranで書かれたユーザプログラム a.f を翻訳し, SVE版ScaLAPACKライブラリを結合します. BLAS, LAPACK はスレッド並列版を結合します.

```
mpifrtpx -Kopenmp a.f -lfjscalapacksve -lfjlapackexsve
```

例 4: Fortran で書かれたユーザプログラム a.f をネイティブコンパイラで翻訳し, SVE 版 ScaLAPACK ライブラリを結合します. BLAS, LAPACK はスレッド並列版を結合します.

```
mpifrt -Kopenmp a.f -lfjscalapacksve -lfjlapackexsve
```

2.2.7.4 BLAS, LAPACK 逐次版の C, C++からの利用方法

C, C++で書かれたユーザプログラムを翻訳し, BLAS 及び LAPACK の逐次版の共有ライブラリを結合するためには `fccpx`, `FCCpx`, `fcc` または `FCC` コマンド行に表4で示すオプションに加えて, `-SSL2` または`-SSL2BLAMP` を指定します. 結合オプションはユーザプログラムよりも後ろに指定します. `-SSL2` または`-SSL2BLAMP` オプションはどちらを指定してもかまいません.

ILP64 インターフェースのライブラリを使用する場合は, ILP64 に対応したヘッダファイルを使用するために`-I/`*製品インストールパス*`/include/lapack_ilp64` を翻訳時のオプションに指定します.

表4 BLAS, LAPACK 逐次版を結合するためのオプション(C/C++)

	結合オプション
LP64, 汎用版	`-lfjlapack`
LP64, SVE 版	`-lfjlapacksve`
ILP64, 汎用版	`-lfjlapack_ilp64`
ILP64, SVE 版	`-lfjlapacksve_ilp64`

例 1: C で書かれたユーザプログラム a.c を翻訳し, SVE 版 BLAS, LAPACK ライブラリを結合します.

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

例 2: C で書かれたユーザプログラム a.c を翻訳し, 汎用版 BLAS, LAPACK ライブラリを結合します.

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

例 3: C で書かれたユーザプログラム a.c を翻訳し, ILP64 インターフェースの SVE 版 BLAS, LAPACK ライブラリを結合します.

```
fccpx -I/製品インストールパス/include/lapack_ilp64 a.c
   –lfjlapacksve_ilp64 -SSL2
```

例 4: C++で書かれたユーザプログラム a.cpp を翻訳し, SVE 版 BLAS, LAPACK ライブラリを結合します.

```
FCCpx a.cpp -lfjlapacksve -SSL2
```

例 5: OpenMP C で書かれたユーザプログラム a.c を翻訳し, SVE 版 BLAS, LAPACK ライブラリを結合します.

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

例 6: OpenMP C で書かれたユーザプログラム a.c をネイティブコンパイラで翻訳し, SVE 版 BLAS, LAPACK ライブラリを結合します.

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

2.2.7.5 BLAS, LAPACK スレッド並列版の C, C++からの利用方法

ユーザプログラムを翻訳し, BLAS 及び LAPACK のスレッド並列版(PLASMA を含む)の共有ライブラリを結合するためには, `fccpx`, `FCCpx`, `fcc` または `FCC` コマンド行に`-Kopenmp` と表5に示すオプションに加えて, `-SSL2` または`-SSL2BLAMP` を指定します. 結合オプションはユーザプログラムよりも後ろに指定します. `-SSL2` または `-SSL2BLAMP` オプションはどちらを指定してもかまいません.

ILP64 インターフェースのライブラリを使用する場合は, ILP64 に対応したヘッダファイルを使用するために`-I/`*製品インストールパス*`/include/lapack_ilp64` を翻訳時のオプションに指定します.

表5 BLAS, LAPACK **スレッド並列版を結合するためのオプション**(C/C++)

	結合オプション
LP64, 汎用版	`-lfjlapackex`
LP64, SVE 版	`-lfjlapackexsve`
ILP64, 汎用版	`-lfjlapackex_ilp64`
ILP64, SVE 版	`-lfjlapackexsve_ilp64`

`-Kopenmp` オプションのかわりに`-Kparallel` オプションを指定することもできます. このとき, "2.3.6 スレッド並列版BLAS, LAPACKを`-Kparallel`オプションで結合する場合の注意"に示す注意事項があります.

例 1: C で書かれたユーザプログラム a.c を翻訳し, SVE 版 BLAS, LAPACK スレッド並列版ライブラリを結合します.

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

例 2: C で書かれたユーザプログラム a.c を翻訳し, 汎用版 BLAS, LAPACK スレッド並列版ライブラリを結合します.

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

例 3: C で書かれたユーザプログラム a.c を翻訳し, ILP64 インターフェースの SVE 版 BLAS, LAPACK スレッド並列版ライブラリを結合します.

```
fccpx -Kopenmp -I/製品インストールパス/include/lapack_ilp64
   a.c –lfjlapackexsve_ilp64 -SSL2
```

例 4: C++で書かれたユーザプログラム a.cpp を翻訳し, SVE 版 BLAS, LAPACK スレッド並列版ライブラリを結合します.

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

例 5: C で書かれたユーザプログラム a.c を逐次実行プログラムとして翻訳し, SVE 版 BLAS,LAPACK スレッド並列版ライブラリを結合します.

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

例 6: C で書かれたユーザプログラム a.c を自動並列化して翻訳し, SVE 版 BLAS, LAPACK スレッド並列版ライブラリを結合します.

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

例 7: C で書かれたユーザプログラム a.c をネイティブコンパイラで翻訳し, SVE 版 BLAS, LAPACK スレッド並列版ライブラリを結合します.

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

2.2.7.6 ScaLAPACK の C, C++からの利用方法

C, C++で書かれたユーザプログラムを翻訳し ScaLAPACK の共有ライブラリを結合するためには, `mpifccpx`, `mpiFCCpx`, `mpifcc` または `mpiFCC` コマンドを使用し, 表6に示すオプション及び, 使用する BLAS, LAPACK の種類に応じて表4または表5に示すオプションを指定します. さらに, `-SCALAPACK` に加えて `-SSL2` または `-SSL2BLAMP` を指定します. ScaLAPACK の結合オプションはユーザプログラムよりも後ろに指定し, BLAS, LAPACK を結合するためのオプションよりも前に指定します. `-SSL2` または`-SSL2BLAMP` オプションはどちらを指定してもかまいません.

表6 ScaLAPACK を結合するためのオプション(C/C++)

	結合オプション
LP64, 汎用版	`-lfjscalapack`
LP64, SVE 版	`-lfjscalapacksve`
ILP64, 汎用版	-
ILP64, SVE 版	-

ScaLAPACK は ILP64 版はサポートしていません.

スレッド並列版 BLAS・LAPACK を使用する場合, `-Kopenmp` オプションも一緒に指定します.

例 1: C で書かれたユーザプログラム a.c を翻訳し, SVE 版 ScaLAPACK ライブラリを結合します. BLAS, LAPACK は逐次版を結合します.

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

例 2: C で書かれたユーザプログラム a.c を翻訳し, 汎用版 ScaLAPACK ライブラリを結合します. BLAS, LAPACK は逐次版を結合します.

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

例 3: C で書かれたユーザプログラム a.c を翻訳し, SVE 版 ScaLAPACK ライブラリを結合します. BLAS, LAPACK は逐次版を結合します.

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

例 4: C++で書かれたユーザプログラム a.cpp を翻訳し, SVE 版 ScaLAPACK ライブラリを結合します. BLAS, LAPACK はスレッド並列版を結合します.

```
mpiFCCpx -Kopenmp a.cpp -lfjscalapacksve -lfjlapackexsve
   -SSL2 -SCALAPACK
```

例 5: C で書かれたユーザプログラム a.c をネイティブコンパイラで翻訳し, SVE 版 ScaLAPACK ライブラリを結合します. BLAS, LAPACK はスレッド並列版を結合します.

```
mpifcc -Kopenmp a.c -lfjscalapacksve -lfjlapackexsve -SSL2
   -SCALAPACK
```

2.2.7.7 BLAS, LAPACK, ScaLAPACK を動的ロードして利用する方法

dlopen, dlsym 関数で動的ロードして BLAS, LAPACK, ScaLAPACK ライブラリを利用する場合, dlopen 関数にファイル名を指定してライブラリをロードします. ライブラリのファイル名を表7に示します.

表7 共有ライブラリのファイル名

	BLAS, LAPACK 逐次版	BLAS, LAPACK スレッド並列版	ScaLAPACK
LP64, 汎用版	libfjlapack.so	libfjlapackex.so	libfjscalapack.so
LP64, SVE 版	libfjlapacksve.so	libfjlapackexsve.so	libfjscalapacksve.so
ILP64, 汎用版	libfjlapack_ilp64.so	libfjlapackex_ilp64.so	-
ILP64, SVE 版	libfjlapacksve_ilp64.so	libfjlapackexsve_ilp64.so	-

dlopen, dlsym 関数を使って BLAS, LAPACK および ScaLAPACK を使用する場合, 結合時にライブラリを指定するオプションを指定する必要はありません.

BLAS, LAPACK スレッド並列版を使用するユーザプログラムを結合する場合は, 結合時のオプションに-Kopenmp を指定してください.

fccpx, FCCpx, mpifccpx または mpiFCCpx コマンドを使ってユーザプログラムを結合する場合, -SSL2 または-SSL2BLAMP オプションも一緒に指定してください.

mpifccpx または mpiFCCpx コマンドを使ってユーザプログラムと ScaLAPACK ライブラリを結合する場合, -SCALAPACK オプションも一緒に指定してください.

例 1: BLAS, LAPACK 逐次版を動的ロードする C プログラム a.c を翻訳・結合する.

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

例 2: BLAS, LAPACK スレッド並列版を動的ロードする C プログラム a.c を翻訳・結合する.

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

例 3: ScaLAPACK および BLAS, LAPACK 逐次版を動的ロードする C プログラム a.c を翻訳・結合する.

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

例 4: ScaLAPACK および BLAS, LAPACK スレッド並列版を動的ロードする C プログラム a.c を翻訳・結合する.

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

例 5: BLAS, LAPACK 逐次版を動的ロードする C プログラム a.c と、それを呼び出す Fortran プログラム b.f を翻訳・結合する.

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

例 6: BLAS, LAPACK スレッド並列版を動的ロードする C プログラム a.c と、それを呼び出す Fortran プログラム b.f を翻訳・結合する.

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

例 7: ScaLAPACK および BLAS, LAPACK 逐次版を動的ロードする C プログラムと a.c, それを呼び出す Fortran プログラム b.f を翻訳・結合する.

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

例 8: ScaLAPACK および BLAS, LAPACK スレッド並列版を動的ロードする C プログラム a.c と、それを呼び出す Fortran プログラム b.f を翻訳・結合する.

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

2.3 注意事項

ここでは, BLAS, LAPACK 及び ScaLAPACK を使用する上で注意すべき事項について説明します.

2.3.1 スレッド数の上限

BLAS, LAPACK では, スレッド数の上限は 128 です.

2.3.2 無限大や NaN (Not a Number)の扱いについて

NetlibのLAPACK version 3.0 以降では, ゼロ除算やオーバーフローした場合の計算結果としてIEEE規格で決められた値(無限大やNaN)が設定され, 計算を続行することを前提に書かれたルーチンがあります.

弊社 Fortran では IEEE 規格どおりの動作をしますが, 翻訳オプションに-NRtrap を指定すると, 上記のような演算でエラーメッセージが出力されます. このため, LAPACK を使用するプログラムを翻訳する場合に-NRtrap オプションを指定するのはさけてください.

2.3.3 ライブラリのアーカイブに含まれるルーチンについて

BLAS, LAPACK のアーカイブには, SSL II 及び C-SSL II のルーチンも含まれています. また, 本製品独自のスレーブルーチンとして, ルーチン名の先頭が, SS_または#L_(#はS,D,C,Z,I,X)で始まるルーチンが含まれています. 利用するときはルーチン名の重複に注意してください.

2.3.4 BLAS, LAPACK ルーチンが使用する作業域について

BLAS, LAPACK には性能を向上させるために内部で比較的大きな作業域を使用するルーチンがあります. この作業域はスタック上に割り付けられるため, スタックに十分なサイズを割り当てる必要があります.

スタックにはプロセスのスタック領域とスレッドごとのスタック領域があり, BLAS, LAPACK の利用方法に応じて一方または両方のサイズを増やします.

プロセスのスタックの制限値は ulimit コマンドで拡張することができます. OpenMP Fortran で並列化されたプログラムのスレッドごとのスタック領域の大きさは, デフォルトではプロセスのスタックと同じ大きさで確保されますが, 環境変数 OMP_STACKSIZE で変更することができます. -Nfjomplib オプションが指定されている場合, 環境変数 THREAD_STACK_SIZE にスタック領域の大きさを指定する事もできます. 指定方法の詳細については, “Fortran 使用手引書”を参照してください.

2.3.4.1 逐次版

表8に BLAS, LAPACK 逐次版で利用するスタックのサイズを示します.

一般の逐次 Fortran プログラムでは表8の値よりも大きい値をプロセスのスタックサイズに設定します.

OpenMP Fortran で並列化された部分から呼び出す場合は, プロセスのスタックとスレッドごとのスタックの両方に表8の値を加えて指定します. ユーザプログラムがスタックに必要とする作業域があれば, その値と加算します. 1 つのプログラムの中で複数の精度のルーチンを使用している場合は, そのうちの最大の値を加えます.

表8 BLAS, LAPACK 逐次版が使用する作業域の大きさ

使用するルーチンの型	作業域の大きさ
REAL(4)	2Mbyte
REAL(8)	4Mbyte
COMPLEX(4)	2Mbyte

COMPLEX(8)	4Mbyte

2.3.4.2 スレッド並列版

a) 逐次プログラムまたは OpenMP Fortran プログラムの逐次部分から呼び出す場合

プロセスのスタック及びスレッドごとのスタックのそれぞれに表9で示した値を加えて指定します.

表9 BLAS, LAPACK スレッド並列版が使用する作業域の大きさ(1)

使用するルーチンの型	作業域の大きさ	
	プロセス スタック	スレッドごとの スタック
REAL(4)	11Mbyte	2Mbyte
REAL(8)	21Mbyte	4Mbyte
COMPLEX(4)	19Mbyte	2Mbyte
COMPLEX(8)	38Mbyte	4Mbyte

b) パラレルリージョンの内側での call がある場合

プロセスのスタック及びスレッドごとのスタックの両方に表10で示した値を加えて指定します.

表10 BLAS, LAPACK スレッド並列版が使用する作業域の大きさ(2)

使用するルーチンの型	作業域の大きさ
REAL(4)	11Mbyte
REAL(8)	21Mbyte
COMPLEX(4)	19Mbyte
COMPLEX(8)	38Mbyte

2.3.5 ScaLAPACK のローカル配列のサイズ

ScaLAPACK を使用するプログラムではローカル配列の要素数は 2147483647 を越えないようにしてください. ScaLAPACK の内部で 2 次元配列を 1 次元配列として参照する部分があり, その添え字が 4 バイト整数型のため, 2147483647 を越えると正しく処理ができません.

例えば, `REAL(8)::A(100000,100000)`のように宣言された配列 A は要素数が 100000×100000=10000000000 ≥ 2147483647 のため ScaLAPACK ルーチンでは使用できません.

2.3.6 スレッド並列版 BLAS, LAPACK を -Kparallel オプションで結合する場合の注意

スレッド並列版 BLAS, LAPACK を -Kopenmp オプションを指定せず, -Kparallel オプションを指定して結合した場合, 次に示す注意事項があります.

1) –Nfjomplib オプションが指定されているかつ,-Kparallel オプションのみを指定して結合した場合,環境変数 PARALLEL が指定されると, OMP_NUM_THREADS の指定よりも PARALLEL の指定のほうが優先されますのでご注意ください.

2.3.7 行列のサイズについて

Netlib で公開されているオリジナルの BLAS, LAPACK および ScaLAPACK の一部のソースでは M×N の行列について M*N*k (k は定数)のような式が記述さているものがあります. この式が 4 バイト整数型で計算されるために, M や N が大きい値の場合, 式の値がオーバーフローして正しく計算できなくなり, 期待どおりの動作をしなくなります. これが BLAS, LAPACK および ScaLAPACK で問題を解くことができるサイズの暗黙の限界となっています. 多くのルーチンでは問題ありませんが, 一部のルーチンでは M*N*k >= 2147483647 のような大きなサイズの問題で正しく動作できません. ご利用の際には問題サイズにご注意ください.

2.3.8 PLASMA のモジュールについて

PLASMA を Fortran90 から使用するためのモジュールを提供しています. モジュール名は plasma.mod, plasma_#.mod (#は s,d,c,z,ds,zc)です. 利用するときはモジュール名の重複に注意してください.

Fortran プログラムで書かれたユーザプログラムを翻訳するときにオプションとして -AU を指定する場合, モジュール名とルーチン名は英小文字でなければなりません.

2.3.9 セクタキャッシュに関する警告メッセージについて

数学ライブラリでは A64FX CPU のセクタキャッシュ機能を使って高速化しているルーチンがあります. プログラムの実行方法によってセクタキャッシュを利用できない場合があり, 以下のような警告メッセージを出力する場合があります. この場合, セクタキャッシュが使われないため性能に影響があるかもしれませんが, 実行は継続され計算は正しく行われます.

jwe1047i-w A sector cache couldn't be used.

2.3.10 MRQ オーバーフローについて

ScaLAPACK を使用したプログラムで, 1 つのプロセスが他の大量のプロセスと一度に通信を行うような処理を実行した場合に, 実行時に [mpi::common-tofu::tofu-mrq-overflow]から始まる MPI のエラーメッセージが出力される場合があります. このときは, MPI の MCA パラメタ common_tofu_num_mrq_entries で完了キューのエントリ数を増やして実行してください. エラーメッセージおよび MCA パラメタの詳細は"MPI 使用手引書"を参照してください.

2.3.11 計算結果について

本製品の BLAS, LAPACK, ScaLAPACK は Netlib で公開されているソースを元に開発していますが, 高速化のため副作用のある最適化, 例えば演算の評価順序の変更, FMA 命令の利用, 逆数近似命令を使った除算・平方根の利用など, を行っています. 多くの場

合は正常に計算できますが, 一部の計算で Netlib のソースを副作用のないオプションでコンパイルした場合と結果が異なる場合があります.

例として, 条件の悪い問題で丸め誤差程度の値の違いが桁落ち等で拡大されて最終的な計算結果に大きく影響する場合があります. また, 計算途中に非正規化数のような特殊な値が現れた場合に結果の精度が変わったり, NaN が通知されたりする場合があります.

3. 高速 4 倍精度基本演算ライブラリ

3.1 概説

高速 4 倍精度基本演算ライブラリ(fast_dd) は, 4 倍精度の値を double-double 形式で表現し, 演算を行うライブラリです.

double-double 形式は倍精度実数型の変数 2 つで 4 倍精度変数を表現する形式です. 演算はプロセッサの持つ倍精度演算命令を使って行うため, 言語処理系が持つ 4 倍精度実数型に比べて高速に演算できます.

fast_dd はスカラ CPU 向けにチューニングしています. 特に A64FX CPU の性能を引き出すチューニングを行っております.

3.2 呼び出し方法

高速 4 倍精度基本演算ライブラリは Fortran または C++プログラムから利用できます.各機能の説明については「高速 4 倍精度基本演算ライブラリ使用手引書」を参照してください.

3.3 翻訳・結合・実行方法

3.3.1 ユーザプログラムの翻訳・結合方法

fast_dd は Fortran または C++で記述された利用者プログラムに結合して使用することができます. 翻訳, 結合には富士通製 Fortran/C++コンパイラを使用します.

3.3.1.1 準備

本製品を使用するためには以下の設定が必要です.“*製品インストールパス*”は,システム管理者にお問い合わせください.

- ログインノードでクロスコンパイラによりプログラムの翻訳および結合を行うために

  環境変数 PATH に/*製品インストールパス*/bin を追加します.
- ネイティブコンパイラによりプログラムの翻訳および結合を行うために

  環境変数 PATH に/*製品インストールパス*/bin を追加します.
- 計算ノードでプログラムの実行を行うために

  環境変数 LD_LIBRARY_PATH に/*製品インストールパス*/lib64 を追加します.

3.3.1.2 Fortran 版の結合方法

ユーザプログラムをクロスコンパイラにより翻訳し, fast_dd を結合するためには, frtpx コマンド行に-SSL2 を指定します. ユーザプログラムで BLAS/LAPACK のスレッド並列版をお使いの場合は, -SSL2 のかわりに-SSL2BLAMP オプションを指定します.

fast_dd は CPU の種類に応じたライブラリが用意されており, プログラムを実行する機種に合わせてオプションを指定することにより適切なライブラリが結合されます. –KSVE が有効な場合は SVE を使用して高速化したライブラリを結合し, –KNOSVE が有効な場合は汎用のライブラリを結合します. A64FX を搭載した計算ノードで実行する場合, どちらでも動作しますが, 高速に実行するためには SVE を使用したライブラリを推奨します.

例 1：ユーザプログラム a.f90 を翻訳し, SVE 版 fast_dd のアーカイブを結合します.

```
frtpx –KSVE a.f90 -SSL2
```

例 2：ユーザプログラム a.f90 を翻訳し, 汎用版 fast_dd のアーカイブを結合します.

```
frtpx –KNOSVE a.f90 -SSL2
```

fast_dd のスレッド並列化されたルーチンを使用する場合は, -Kopenmp または -Kparallel オプションを追加します.

例 3：スレッド並列ルーチンを使用するユーザプログラム b.f90 を翻訳し, fast_dd のアーカイブを結合します.

```
frtpx  -Kopenmp,SVE b.f90 -SSL2
```

ネイティブコンパイラによりユーザプログラムを翻訳し, fast_dd を結合するためには frt コマンドを使用します.

例 4：ユーザプログラム a.f90 を翻訳し, SVE 版 fast_dd のアーカイブを結合します.

```
frt –KSVE a.f90 -SSL2
```

プログラムが MPI を使用する場合は, frtpx, frt のかわりに mpifrtpx, mpifrt コマンドを使用します.

3.3.1.3 C++版の結合方法

ユーザプログラムをクロスコンパイラにより翻訳し, fast_dd を結合するためには, FCCpx コマンド行に-SSL2 を指定します. ユーザプログラムで BLAS/LAPACK のスレッド並列版をお使いの場合は, -SSL2 のかわりに-SSL2BLAMP オプションを指定します. また, fast_dd はスレッドセーフなオブジェクトとなっているため, -Kopenmp, -Kparallel または-mt のいずれかを指定してください.

fast_dd は CPU の種類に応じたライブラリが用意されており, プログラムを実行する機種に合わせてオプションを指定することにより適切なライブラリが結合されます. –KSVE が有効な場合は SVE を使用して高速化したライブラリを結合し, –KNOSVE が有効な場合は汎用のライブラリを結合します. A64FX を搭載した計算ノードで実行する場合, どちらでも動作しますが, 高速に実行するためには SVE を使用したライブラリを推奨します.

clang モードで翻訳する場合(-Nclang が有効)は-KSVE・-KNOSVE オプションの代わりに-march オプションに+sve または+nosve を有効にすることによりライブラリが選択されます.

例 1：ユーザプログラム a.cpp を翻訳し, SVE 版 fast_dd のアーカイブを結合します.

```
FCCpx –Kopenmp,SVE a.cpp -SSL2
```

例 2：ユーザプログラム a.cpp を翻訳し, 汎用版 fast_dd のアーカイブを結合します.

```
FCCpx –Kopenmp,NOSVE a.cpp -SSL2
```

ネイティブコンパイラによりユーザプログラムを翻訳し, fast_dd を結合するためには FCC コマンドを使用します.

例 3：ユーザプログラム a.cpp を翻訳し, SVE 版 fast_dd のアーカイブを結合します.

```
FCC –Kopenmp,SVE a.cpp -SSL2
```

プログラムが MPI を使用する場合は, FCCpx, FCC のかわりに mpiFCCpx, mpiFCC コマンドを使用します.

3.4 注意事項

3.4.1 ライブラリのアーカイブに含まれるルーチンについて

fast_dd のアーカイブには, SSL II, SSL II スレッド並列機能, C-SSL II, C-SSL II スレッド並列機能, BLAS 及び LAPACK のルーチンも含まれています. その他, ルーチン名の先頭が, SS_または#L_(#は S, D, C, Z, I, X)で始まるスレーブルーチンが含まれています. 利用するときはルーチン名の重複に注意してください.

3.4.2 Fortran コンパイラオプション-AU について

Fortran プログラムで書かれたユーザプログラムを翻訳するときにオプションとして -AU を指定する場合, モジュール名 fast_dd とルーチン名は英小文字でなければなりません.

3.4.3 Coarray 機能での利用について

Fortran の Coarray 機能で dd_real 型または dd_complex 型を使用する場合, 以下の制約があります.

- ポインタであってはなりません.
- 構造体の成分にはなれません.
- ALLOCATABLE か SAVE 属性のいずれかが必要です. 自動オブジェクトにすることはできません.