最終更新日: 平成 19 年 10 月 31 日

FFTSS ライブラリ 3.0 版
利用の手引き

Contents

1　はじめに

FFTSS ライブラリはオープンソースソフトウェアとして配布される高速フーリエ変換 (FFT) ライブラリです. 本ソフトウェアは科学技術振興機構 (JST) 戦略的創造研究推進事業 (CREST) の「大規模シミュレーション向け基盤ソフトウェアの開発」プロジェクトの成果の一つです.

本ライブラリの目標は, 多様な計算環境において高い性能を達成することにあります. 本ライブラリのパッケージ内には多種の FFT カーネル用サブルーチンを同梱しています. それらのうち実行可能なものを全て実行し, それぞれの計算環境で最も高速に計算を完了したものを選択するという自動チューニング機能を備えています.

また, 本ライブラリでは FFTW ライブラリのバージョン 3 と類似した関数インターフェイスを採用しています. このため FFTW をお使いのユーザ様が非常に容易に本ライブラリをお使いいただくことができます. FFTW 用に作成されたアプリケーションをお持ちであれば, ごく僅かな修正だけで本ライブラリを使用できます.

2　離散フーリエ変換 (DFT)

FFTSS ライブラリは長さ n の 1 次元 DFT で以下の計算を行います.
順変換

$$Y_k = \sum_{j=0}^{n} X_j \cdot e^{-2\pi jk\sqrt{-1}/n}$$

逆変換

$$Y_k = \sum_{j=0}^{n} X_j \cdot e^{2\pi jk\sqrt{-1}/n}.$$

X は入力となる倍精度複素数データの配列で, Y は出力です. FFTSS ライブラリの計算結果は FFTW ライブラリ等と同様であり, $1/N$ 等でスケーリングされません.

多次元 DFT では多次元の入力データの各次元方向に対して 1 次元 DFT の計算を行います.

3　インストール方法

本ライブラリは基本的にソースコードの形態で配布されています. このためアプリケーションで利用する前に, 利用者またはシステム管理者がコンパイル及びインストール作業を行う必要があります.

3.1　UNIX 及び互換システム

UNIX 及び互換システムでインストールを行う場合以下の 3 つの作業が必要になります. 1. configure スクリプトを実行します.
2. make コマンドを実行します.
3. make install コマンドを実行します. (任意)

本ライブラリの configure スクリプトでは一般的なフラグに加えて以下のフラグを指定することができます.

`--without-simd`	SIMD 命令の利用を禁止します.
`--without-asm`	アセンブリ言語の利用を禁止します.
`--with-bg`	IBM Blue Gene 用のライブラリを構築します. (クロスコンパイル)
`--with-bg-compat`	一部の Blue Gene 用の FFT カーネルを他の環境でも有効にします.
`--with-recommended`	開発者が推奨する環境変数 CC 及び CFLAGS を使用します.
`--enable-openmp`	OpenMP 並列化を有効にします.
`--enable-mpi`	MPI 版のライブラリも構築します.

環境変数 CC 及び CFLAGS 等を設定したい場合には以下のようなオプションを指定してください.

```
$ ./configure CC=gcc CFLAGS='-O3 -msse2'
```

3.2 Microsoft Windows

Microsoft Windows システムではライブラリの構築方法が 3 種類あります.

3.2.1 Visual Studio

パッケージの win32 フォルダ内に fftss.sln というファイルがあります. このファイルは Visual Studio .NET 2003 用のソリューションファイルです. Visual Studio 2005 以降で利用する場合にはこのソリューションファイルを変換して使用してください. Visual Studio の 2003 より古いバージョンはサポートされません. 適宜各プロジェクトのプロパティページにて最適化レベル等を調整してください. Microsoft SDK (または Platform SDK) がインストールされている必要があります.

同じく win32 フォルダ内にある pfftss.sln ファイルは MPI 用のライブラリを含んでいます. これを利用するためには Microsoft Compute Cluster Pack SDK 等の MPI 開発環境が必要です.

3.2.2 インテル C/C++ コンパイラ

win32 フォルダ内にはインテル C/C++ コンパイラ用のバッチスクリプトファイルが用意されています. コマンドプロンプト等から実行してください. 実行する際には必ず win32 フォルダ内で実行してください. コンパイルオプション等を変更したい場合にはバッチスクリプトファイルを編集してください.

icl-x86.bat	インテル IA-32 アーキテクチャ用. (32 ビット)
icl-amd64	インテル EM64T (Intel 64) または AMD64 アーキテクチャ用. (64 ビット)
icl-ia64	インテル IA-64 アーキテクチャ用. (64bit)

Visual Studio と組み合わせて利用する場合には, ソリューションファイルを開き, 各プロジェクトをインテルコンパイラ用のプロジェクトに変換してください.

3.2.3 MinGW 環境

MinGW 環境では UNIX 及び互換環境と同様に configure スクリプトを利用してください.

4 アプリケーションの作成

4.1 UNIX 及び互換環境

一般的には以下の設定が必要になります.

- CFLAGS に "-I/path/to/header/file" を追加してください

- LDFLAGS に ”-L/path/to/library/file -lfftss” を追加してください
- LDFLAGS に ”-lpfftss” を追加してください (MPI の場合のみ)

/path/to/header/file, /path/to/library/file はそれぞれ fftss.h, libfftss.a ファイルがあるディレクトリを指します. 標準的な場所にインストールした場合には指定する必要がない可能性があります.

4.2 Visual Studio

Visual Studio を使用してライブラリを構築した場合, win32/Release (または win32/Debug) フォルダ内に fftss.lib ファイルが生成されます. MPI 版を構築した場合には pfftss.lib ファイルも同じフォルダ内にあります. fftss.h や pfftss.h 等のヘッダファイルは include フォルダ内にあります. アプリケーションから FFTSS ライブラリを利用する場合にはこれらのファイルを適切な場所にコピーしてください.

5 制限事項

現在のバージョンでは FFTSS ライブラリは倍精度浮動小数点の複素数－複素数変換のみに対応しています. また入力データの各次元方向の長さは 2 のべき乗でなければなりません.

本ライブラリの FFT カーネルでは Stockham の自動ソートアルゴリズムを利用しているため out-of-place transform となります. in-place transform のフラグを指定した場合でもライブラリ内で out-of-place transform 用に作業領域が確保されます.

6 ライブラリ関数の一覧

6.1 記憶領域の確保

6.1.1 fftss_malloc

Syntax:
void *fftss_malloc(long size**);**

fftss_malloc() 関数は size バイトの記憶領域を確保し, その先頭アドレスへのポインタを返します. この関数が返すアドレスは常に 16 バイト境界にアラインメントされます.

6.1.2 fftss_free

Syntax:
void fftss_free(void ***ptr);**

fftss_free() 関数は ptr で指定された記憶領域を開放します. **fftss_malloc()** で確保された領域にのみ用いてください.

6.2 プラン作成

6.2.1 fftss_plan_dft_1d

Syntax:
fftss_plan fftss_plan_dft_1d(long n **, double ***in**, double ***out **, long** sign**, long** flags **);**

fftss_plan_dft_1d() 関数は
長さ n の倍精度複素数 1 次元 FFT 用のプランを作成します. 入力データの i 番目の要素の実部は in[i*2] に, 虚部は in[i*2+1] にそれぞれ格納してください.

6.2.2 fftss_plan_dft_2d

Syntax:
fftss_plan fftss_plan_dft_2d(long nx , long ny, long py , double *in, double *out , long sign, long flags);

fftss_plan_dft_2d() 関数はサイズ nx × ny の倍精度複素数 2 次元 FFT 用のプランを作成します. (x,y) 要素の実部は in[x*2+y*py*2] に, 虚部は in[x*2+y*py*2+1] にそれぞれ格納してください.

6.2.3 fftss_plan_dft_3d

Syntax:
fftss_plan fftss_plan_dft_3d(long nx , long ny, long nz , long py, long pz , double *in, double *out , long sign, long flags);

fftss_plan_dft_3d() 関数はサイズ nx × ny × nz の倍精度複素数 3 次元 FFT 用のプランを作成します. (x,y,z) 要素の実部は in[x*2+y*py*2+z*pz*2] に, 虚部は in[x*2+y*py*2+z*pz*2+1] にそれぞれ格納してください.

6.2.4 利用可能なフラグの一覧

以下は 1 次元, 2 次元, 3 次元 FFT のプラン作成時に指定可能なフラグの一覧です.

- FFTSS_VERBOSE このフラグは幾つかの情報の出力を有効にします. 通常アプリケーションユーザがこのフラグを使用する必要はありません. 各環境でどの FFT カーネルが選択されたか等の情報を得ることができます.
- FFTSS_MEASURE

  プラン作成時に利用可能な FFT カーネルを全て実行し, その中から最も高速なものを選択するモードを指定します. (標準)
- FFTSS_ESTIMATE

  プラン作成時に最適な FFT カーネルを 推測 します. 実際に各 FFT カーネルは実行されないため, 最適でないカーネルが選択される可能性があります.
- FFTSS_PATIENT

  現在は FFTSS_MEASURE と同値です.
- FFTSS_EXHAUSTIVE

  現在は FFTSS_MEASURE と同値です.
- FFTSS_NO_SIMD

  このフラグは SIMD (及び SIMOMD) 命令の使用を禁止します. 最速の FFT カーネルを選択する際にこれらが使用されているカーネルが選択肢から除外されます.
- FFTSS_UNALIGNED

  このフラグは入力データが 16 バイト境界にアラインメントされていない場合に指定してください. これにより一部のアラインメントを要求するカーネルが候補から除外されます. 通常プラン作成時に指定された入力バッファ

と出力バッファに関してはアラインメントのチェックを行い, アラインメントされていない場合には自動的にこのフラグが追加されます. しかし **fftss_execute_dft()** 関数によってプラン作成時とは異なる, アラインメントされていない可能性のある入出力バッファを用いる場合にはこのフラグを明示的に指定する必要があります.

一般的にはアラインメントされている場合により高い性能を実現できるため, **fftss_malloc()** 関数によって確保されたアラインメントされた領域を用いることを推奨します.

- FFTSS_DESTROY_INPUT

  このフラグは入力バッファのデータを破壊してもよい場合に指定してください. (標準で破壊します)

- FFTSS_PRESERVE_INPUT

  このフラグは入力バッファのデータを破壊したくない場合に指定してください. 作業用のバッファが別途確保されます.

- FFTSS_INOUT

  このフラグが指定された場合には, 出力データは入力バッファ in に出力されます. out は作業用のバッファとして利用されます.

6.3 プランの実行

6.3.1 fftss_execute

Syntax:
void fftss_execute(fftss_plan p **);**

fftss_execute() 関数はプラン p を実行します.

6.3.2 fftss_execute_dft

Syntax:
void fftss_execute_dft(fftss_plan p **, double ***in**, double ***out**);**

fftss_execute() 関数はプラン p を実行します. 入力バッファ in と出力バッファ out がプラン p の作成時に指定されたものの代わりに使用されます.

6.4 プランの破壊

6.4.1 fftss_destroy_plan

Syntax:
void fftss_destroy_plan(fftss_plan p **);**

fftss_destroy_plan() 関数はプラン p と, このプランのために確保された領域を開放します.

6.5 タイマー

6.5.1 fftss_get_wtime

Syntax:
double fftss_get_wtime(void);

fftss_get_wtime() 関数は現在のタイムスタンプを秒単位で返します.

6.6 マルチスレッド

6.6.1 fftss_init_threads

Syntax:
int fftss_init_threads(void);

fftss_init_threads() 関数はなにもしません. この関数は FFTW ライブラリとの互換性のためにのみ存在します.

6.6.2 fftss_plan_with_nthreads

Syntax:
void fftss_plan_with_nthreads(int nthreads**);**

fftss_plan_with_nthreads() 関数は計算に利用するスレッド数を指定します. FFTSS ライブラリは OpenMP を用いたマルチスレッド並列化のみをサポートしており, この関数は単純に **omp_set_num_threads()** 関数を用いて OpenMP のスレッド数を指定しています.

6.6.3 fftss_cleanup_threads

Syntax:
void fftss_cleanup_threads(void);

fftss_cleanup_threads() 関数は何もしません. FFTW ライブラリとの互換性のためだけに存在します.

6.7 MPI

FFTSS ライブラリバージョン 3 では pfftss_ から始まる MPI 用の関数を用意しています.

6.7.1 pfftss_plan_dft_2d

Syntax:
pfftss_plan pfftss_plan_dft_2d(long nx**, long** ny**, long** py**, long** oy**, long** ly**, double *inout, long** sign**, long** flags**, MPI_Comm** comm**);**

pfftss_plan_dft_2d() 関数は MPI 環境においてサイズ nx × ny の倍精度複素数 2 次元 FFT 用のプランを作成します. 指定されたバッファ inout は入力及び出力に用いられるため, 入力データは必ず失われます. 各ノードは oy 行から始まる ly 行分のデータを持ち, py × ly の 2 次元配列に格納してください. (言語的には 1 次元配列 double inout[2*py*ly]). データはノード間で, comm 内のランク順に行ブロック分割されていなければなりません.

nx は少なくともノード数以上である必要があります. ly は全てのノードで 0 でない必要があります. この関数では内部で入出力データに相当するサイズの領域を二つ確保します.

6.7.2 pfftss_execute

Syntax:
void pfftss_execute(pfftss_plan p**);**

pfftss_execute() 関数はプラン p を実行します.

6.7.3 pfftss_execute_dft

Syntax:
void pfftss_execute_dft(pfftss_plan p, double *inout);

pfftss_execute_dft() 関数はプラン p を実行します. 入出力バッファとしてプラン作成時に指定したものの代わりに inout を使用します.

6.7.4 pfftss_destroy_plan

Syntax:
void pfftss_destroy_plan(pfftss_plan p);

pfftss_destroy_plan() 関数はプラン p 及びこのプランに必要であった記憶領域を開放します.

7 マルチスレッド

現バージョンの FFTSS ライブラリは OpenMP を用いたマルチスレッドに対応しています. マルチスレッド用のライブラリを構築するためには, OpenMP に対応したコンパイラを用い, OpenMP を有効にするためのコンパイラオプションを指定する必要があります.

例えばインテル C/C++ コンパイラで構築する場合は以下のとおりです. コンパイラオプションとして -openmp を追加し, -xP オプションで SSE3 命令のサポートを有効にしています. $ ./configure CC=icc CFLAGS='-O3 -openmp -xP'

スレッド数の指定は通常環境変数 'OMP_NUM_THREADS' で行います. 指定しない場合のスレッド数は環境に依存しますが, 最大 CPU コア数または 1 となる場合が多いでしょう. また **omp_set_num_threads()** 関数等を用いてプログラム実行中に設定することもできます. 'fftss_plan_with_nthreads()' という関数を FFTW ライブラリとの互換性のために用意していますが, この関数は単純に **omp_set_num_threads()** 関数を呼び出しているだけです.

プラン作成時にスレッド数に応じた作業領域が確保されます. このため使用するスレッド数はプラン作成より前に指定されている必要があります. ベンチマーク目的等においてスレッド数を変えて同じプランを実行するような場合, プラン作成時にはスレッド数の最大値を指定してください.

```
max_threads = omp_get_num_procs();
fftss_plan_with_nthreads(max_threads);
plan = fftss_plan_dft_2d(nx, ny, py, vin, vout,
   FFTSS_FORWARD, FFTSS_MEASURE);

{  /*  配列の初期化.  */  }

for (nthreads = 1; nthreads <= max_threads; nthreads ++) {
   fftss_plan_with_nthreads(nthreads);
   t = fftss_get_wtime();
   fftss_execute(plan);
   t = fftss_get_wtime() - t;
   printf("%d スレッドで実行した場合 %lf 秒.\n", nthreads, t);
}
```

MPI 版でもさらに各ノード内の計算を OpenMP 並列化することができます. –enable-mpi と –enable-openmp の両方のオプションを指定することで MPI と OpenMP のハイブリッド並列化されたライブラリを構築することができます.

8 FFTW ライブラリとの互換性

本ライブラリの関数インターフェイスは FFTW ライブラリと類似しています. FFTW ライブラリの利用者は FFTSS ライブラリを互換モードで利用することができます. 通常 FFTW ライブラリ用のアプリケーションプログラムは fftw3.h というヘッダファイルをインクルードしていますが, これを fftw3compat.h と書き換えることによって FFTSS ライブラリを使うことができます.

fftw3compat.h ファイルには FFTW ライブラリ用のソースコードを FFTSS ライブラリ用のソースコードに変換するためにマクロ等が定義されており, fftss.h もこのファイルからインクルードされています.

現バージョンでは FFTW ライブラリの機能の中で fftw3compat.h に定義されている部分のみサポートされています. MPI 版に関しては一切互換性はありません.

8.1 互換性のある関数の一覧

- **fftw_malloc()**
- **fftw_free()**
- **fftw_plan_dft_1d()**
- **fftw_plan_dft_2d()**
- **fftw_plan_dft_3d()**
- **fftw_execute()**
- **fftw_execute_dft()**
- **fftw_destroy_plan()**
- **fftw_init_threads()**
- **fftw_plan_with_nthreads()**
- **fftw_cleanup_threads()**

8.2 互換性のあるフラグの一覧

- **FFTW_MEASURE**
- **FFTW_ESTIMATE**
- **FFTW_PATIENT**
- **FFTW_EXHAUSTIVE**
- **FFTW_NO_SIMD**
- **FFTW_PRESERVE_INPUT**

- **FFTW_DESTROY_INPUT**
- **FFTW_FORWARD**
- **FFTW_BACKWARD**

9 List of FFT カーネルの一覧

以下は本ライブラリに含まれる FFT カーネルの一覧です. プラン作成時に選択されたカーネルの名前は, **FFTSS_VERBOSE** フラグを指定した場合に表示されます.

- normal
  標準的な実装.
- FMA
  積和演算命令に適した FFT カーネルを用いた実装.
- SSE2 (1)
  Intel SSE2 拡張命令を用いた実装.
- SSE2 (2)
  Intel SSE2 拡張命令を用いた実装 (UNPCKHPD/UNPCKLPD).
- SSE3
  Intel SSE2 拡張命令を用いた実装 (ADDSUBPD).
- SSE3 (H)
  Intel SSE2 拡張命令を用いた実装 (HADDPD/HSUBPD).
- C99 Complex
  C99 規格の複素数型を用いた実装.
- Blue Gene
  IBM Blue Gene 用の実装.
- Blue Gene (PL)
  IBM Blue Gene 用の実装 (ソフトウェアパイプライン化).
- Blue Gene asm
  IBM Blue Gene 用の実装 (アセンブリ言語で記述).
- IA-64 asm
  Intel IA-64 アーキテクチャ用の実装 (アセンブリ言語で記述).

10　動作確認を行った環境

FFTSS ライブラリは以下のプラットフォームで動作確認を行っています.

プロセッサ	オペレーティングシステム	コンパイラ
UltraSPARC III	Sun Solaris 9	Sun ONE Studio 11
Itanium 2	Linux	Intel C/C++ Compiler 9.1, gcc 4.0.1
PowerPC G5	Mac OS X 10.4	IBM XL C Compiler 6.0, gcc 4.0
POWER5	Linux	IBM XL C Compiler 7.0, gcc 4.0.1
POWER4	AIX	IBM XL C Compiler 6.0
PA-RISC	HP-UX 11	Bundled C Compiler
PPC440FP2	Blue Gene CNK	IBM XL C Compiler 7.0/8.0
Opteron	Linux	gcc 3.3.3, gcc 4.0.1
Pentium 4	Solaris 9 (IA-32)	Sun ONE Studio 11, gcc 4.0.1
Xeon	Linux	Intel C/C++ Compiler 8.1/9.0/9.1, gcc
IA-32	Windows XP SP2	Visual Studio .NET 2003
IA-32	Windows XP SP2	Visual Studio 2005
IA-32	Windows XP SP2	Intel C/C++ Compiler 9.1
x64	Windows XP, 2003	Visual Studio .NET 2003
x64	Windows XP, 2003	Intel C/C++ Compiler for EM64T 9.1