Last Modified: 31/10/07

FFTSS Library Version 3.0
User's Guide

Contents

1 Introduction

The FFTSS library is a Fast Fourier Transform (FFT) library that is distributed as an open source software. The FFTSS library is one of the products of the "Development of Software Infrastructure for Large Scale Scientific Simulation" project, which is supported by Core Research for Evolutional Science and Technology (CREST) of the Japan Science and Technology Agency (JST).

The target of this library is to achieve high performance (high speed) in various computing environments. The software package of this library contains several types of FFT kernel subroutine. The library executes all of them and selects the fastest subroutine for the target computing environment.

The interfaces of the library are similar to those of the FFTW library version 3.x. This makes it easy for FFTW users to use the FFTSS library. Porting to the FFTSS library requires a very slight modification of the source code.

2 DFT

The one-dimensional DFT of length n in the FFTSS library actually computes
the forward transform

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

and the backward transform

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

X is the array of input complex data in double precision, and Y is the output. The results computed by the FFTSS library are **not scaled** and are compatible with the FFTW library.

The multi-dimensional transforms simply compute the 1-D transform along each dimension of the array.

3 Installation

The package of this library is distributed as source code. Users need to build the library before using from application programs.

3.1 UNIX and Compatible Systems

To build the library software from the source package on UNIX or compatible systems, the following three steps are required:

1. Run 'configure' script.
2. Run 'make'.
3. Run 'make install' (optional).

The configure script of this library accepts all generic flags. In addition, the flags listed below are also available.

`--without-simd`	Do not use SIMD instructions.
`--without-asm`	Do not use assembly codes.
`--with-bg`	Build for IBM Blue Gene system. (cross build)
`--with-bg-compat`	Enable FFT kernels for Blue Gene in compatible mode.
`--with-recommended`	Set recommended CC and CFLAGS variables.
`--enable-openmp`	Enable OpenMP.
`--enable-mpi`	Enable MPI.

You can also set CC and CFLAGS manually as follows.

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

3.2 Microsoft Windows

On Microsoft Windows systems, three methods are available for building the software.

3.2.1 Visual Studio

In the 'win32' folder of the package, a solution file 'fftss.sln' is provided for Visual Studio 2003.NET. Open this file, and edit the compiler setting as you like. If you have Visual Studio 2005 series, convert the solution file first. Older versions of Visual Studio are not supported.

3.2.2 Intel C/C++ Compiler

In the 'win32' folder of the package, batch scripts for Intel C/C++ Compiler are included. Run one of these scripts on the 'Command Prompt' or corresponding 'Build Environment' of the Intel C/C++ Compiler. The working directory must be the 'win32' folder. If you want to change compiler options, edit the batch files directly.

icl-x86.bat	For Intel IA-32 Architecture. (32bit)
icl-amd64	For Intel EM64T or AMD64 Architecture. (64bit)
icl-ia64	For Intel IA-64 Architecture. (64bit)

3.2.3 MinGW environment

In the MinGW environment, the configure script, as well as UNIX or compatible systems, is usable.

4 Building Applications

4.1 UNIX and compatible systems

In general, you need to

- add "-I/path/to/header/file" to CFLAGS
- add "-L/path/to/library/file -lfftss" to LDFLAGS
- add "-lpfftss" to LDFLAGS (for MPI only)

to build your application with the FFTSS library.

4.2 Visual Studio

After building the entire binary of the FFTSS library using Visual Studio, a library file '**fftss.lib**' is found in './libfftss/Release' (or './libfftss/Debug'). The header file 'fftss.h' is found in the 'include' folder. To use the FFTSS library with your application, you should use these files by editing the properties of Visual Studio project files.

5 Limitations

In the current version, the FFTSS library only includes complex-to-complex, double precision (of floating-point numbers) routines. The length of 1-D transforms must be a power of two. In case of the multi-dimensional transform, the size of each dimension must be a power of two.

Since this library uses Stockham's auto-sort algorithm, all of the FFT kernels included in this library perform out-of-place transforms. Even if you request in-place transform in FFTW style, the library allocates the buffer for out-of-place transform internally.

6 List of library functions

6.1 Memory Allocation

6.1.1 fftss_malloc

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

fftss_malloc() allocates size bytes and returns a pointer to the allocated memory. The address returned by the function is always aligned to a 16-byte boundary.

6.1.2 fftss_free

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

fftss_free() frees the memory space pointed by ptr, which must have been returned by a previous call to **fftss_malloc()**.

6.2 Creating Plans

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() creates a plan for computing complex-to-complex double precision one-dimensional transforms of length n. The real part of the i-th element of the input sequence must be stored in in[i*2], and the imaginary part must be stored in in[i*2+1].

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() creates a plan for computing complex-to-complex double precision nx by ny two-dimensional transforms. The real part of element(x,y) must be stored in in[x*2+y*py*2], and the imaginary part must be stored in in[x*2+y*py*2+1].

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() creates a plan for computing complex-to-complex double precision nx by ny by nz three-dimensional transforms. The real part of element(x,y,z) must be stored in in[x*2+y*py*2+z*pz*2], and the imaginary part must be stored in in[x*2+y*py*2+z*pz*2+1].

6.2.4 List of Flags

The following lists the available flags for creating plans for 1-D, 2-D, and 3-D transforms.

- FFTSS_VERBOSE

  This flag enables verbose mode. In general, however, application users do not require verbose mode. Using this flag, the name of the selected FFT kernel is shown in standard output.

- FFTSS_MEASURE

  This is the default setting. In creating plans, the library executes all FFT kernels available in the computing environment and select the best kernel.

- FFTSS_ESTIMATE

  The library estimates the best FFT kernel for the computing environment without any executions.

- FFTSS_PATIENT

  Same as FFTSS_MEASURE.

- FFTSS_EXHAUSTIVE

  Same as FFTSS_MEASURE.

- FFTSS_NO_SIMD

  This flag disables the use of SIMD (or SIMOMD) instructions. Try this flag if you have some trouble with the FFT kernels using SIMD instructions.

- FFTSS_UNALIGNED

  Specify whether the input array is not aligned to a 16-byte boundary. The alignments of the input and the output buffers are checked when creating plans. If they are not aligned, this flag is added automatically. Therefore, this flag is usually not necessary. This flag must be specified only if all of the conditions listed below are satisfied.

1. Aligned buffers are given when creating plans.
2. Unaligned buffers are used with **fftss_execute_dft()**.
3. The selected FFT kernel requires alignment.

Since the input and output buffers should be aligned from the aspect of performance, the use of **fftss_malloc()** is strongly recommended for memory allocation.

- FFTSS_DESTROY_INPUT

  This flag allows destruction of data in the input buffer. (default)

- FFTSS_PRESERVE_INPUT

  The data in the input buffer is preserved, and working space will be allocated by the library function for out-of-place transforms.

- FFTSS_INOUT

  When this flag is specified, the results of the transforms are returned to the input buffer in. The output buffer out must also be specified because it will be used for working space.

6.3 Executing Plans

6.3.1 fftss_execute

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

fftss_execute() executes a plan p.

6.3.2 fftss_execute_dft

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

fftss_execute() executes a plan p. The input buffer in and output buffer out are used than the value specified when creating the plan p.

6.4 Destroying Plans

6.4.1 fftss_destroy_plan

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

fftss_destroy_plan() deallocates the plan p.

6.5 Timer

6.5.1 fftss_get_wtime

Syntax:
double fftss_get_wtime(void);

fftss_get_wtime() returns the current timestamp in seconds.

6.6 Multi-Threading

6.6.1 fftss_init_threads

Syntax:
int fftss_init_threads(void);

The **fftss_init_threads()** function does nothing and exists only for reasons of compatibility with FFTW3.

6.6.2 fftss_plan_with_nthreads

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

fftss_plan_with_nthreads() sets the number of threads used for computation. Since the FFTSS library supports only parallelization with OpenMP, this function simply sets the number of OpenMP threads using **omp_set_num_threads()**.

6.6.3 fftss_cleanup_threads

Syntax:
void fftss_cleanup_threads(void);

The **fftss_cleanup_threads()** function does nothing and exists only for reasons of compatibility with FFTW3.

6.7 MPI

The FFTSS library version 3 includes MPI interfaces with prefix 'pfftss_'.

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() creates a plan for computing complex-to-complex double precision nx by ny two-dimensional transforms for distributed parallel computers with MPI library. The two-dimensional array is used for both input and output, and therefore the input array inout is always overwritten. Each node has ly rows from the oy-th row of the nx by ny array, and the nodes are stored in a py by ly array (double inout[2*py*ly]). The data must be block-row distributed between nodes in the order of their MPI rank in comm.

nx must be not less than the number of nodes, and ly must be non-zero on all nodes. This function internally allocates memory for two arrays of approximately the same size as the input array.

6.7.2 pfftss_execute

Syntax:
void pfftss_execute(pfftss_plan p);

pfftss_execute() executes a plan p using MPI library.

6.7.3 pfftss_execute_dft

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

pfftss_execute_dft() executes a plan p using MPI library. inout is used for the input and output array rather than the value specified when creating the plan.

6.7.4 pfftss_destroy_plan

Syntax:
void pfftss_destroy_plan(pfftss_plan p);

pfftss_destroy_plan() destroys a plan p, and deallocates all memory areas allocated for the plan.

7 Multi-Threading

The current version of the FFTSS library supports multi-threading with OpenMP. To build a library for multi-threading, an OpenMP compiler is required, and the compiler options required for OpenMP support must be added.

For example, a build with Intel C/C++ Compiler is described below. The option '-openmp' enables support for OpenMP, and '-xP' enables support for Intel SSE3 instructions.
$ ./configure CC=icc CFLAGS='-O3 -openmp -xP'

To specify the number of threads, the environment variable 'OMP_NUM_THREADS' is used, as well as other OpenMP applications. If this variable is not set, the number of threads depends on the computing environment and typically is equal to one or the number of processors. In addition, you can set the number of threads with the **omp_set_num_threads()** function. We provide a library function 'fftss_plan_with_nthreads()' for compatibility with the FFTW library. This library function simply calls the **omp_set_num_threads()** function.

The number of threads must be set before creating plans because the work space for each thread is allocated. If you need to use a variable number of threads, the maximum number of threads must be set when creating plans.

```
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);

{  /*  Initialize arrays.  */  }
```

```
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("%lf sec with %d thread(s).\n", t, nthreads);
}
```

You can use OpenMP multi-threading with the MPI library if you specified both '–enable-mpi' and '–enable-openmp' options. The library is then built for MPI+OpenMP hybrid parallelization.

8 Compatibility with the FFTW library

The interfaces of this library are similar to those of the FFTW library version 3.

The FFTW users can use the FFTSS library in compatible mode. Application programs written for the FFTW library include header file 'fftw3.h'. To use the FFTSS library, users need only change the name of the header file to 'fftw3compat.h'.

In the header file 'fftw3compat.h', a number of macros are defined in order to convert the source code for the FFTW library, and the header file 'fftss.h' is also included in this file.

In the current version, only a subset of the FFTW3 library is available, which is defined or declared in the header file 'fftw3compat.h'. The MPI version has no compatibility.

8.1 Compatible Functions

- **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 Compatible Flags

- **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 Kernels

In this section, the available FFT kernels are listed. The name of the selected kernel can be displayed by using the **FFTSS_VERBOSE** flag.

- normal
  Normal implementation.
- FMA
  An implementation of FFT kernels optimized for Fused Multiply-Add (FMA) instructions.
- SSE2 (1)
  An implementation with Intel SSE2 instructions.
- SSE2 (2)
  An implementation with Intel SSE2 instructions. (UNPCKHPD/UNPCKLPD)
- SSE3
  An implementation with Intel SSE3 instructions. (ADDSUBPD)
- SSE3 (H)
  An implementation with Intel SSE3 instructions. (HADDPD/HSUBPD)
- C99 Complex
  An implementation using the C99 Complex data type.
- Blue Gene
  An implementation for IBM Blue Gene.
- Blue Gene (PL)
  An implementation for IBM Blue Gene (Software pipelined).
- Blue Gene asm
  An implementation in assembly language for IBM Blue Gene.

- IA-64 asm
  An implementation in assembly language for the Intel IA-64 architecture.

10 Tested Platforms

The FFTSS library has been tested on the following platforms.

Processor	OS	Compiler
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