10.1 Installation on Unix

FFTW comes with a `configure` program in the GNU style. Installation can be as simple as:

```
./configure
make
make install
```

This will build the uniprocessor complex and real transform libraries along with the test programs. (We recommend that you use GNU `make` if it is available; on some systems it is called `gmake`.) The “`make install`” command installs the fftw and rfftw libraries in standard places, and typically requires root privileges (unless you specify a different install directory with the `--prefix` flag to `configure`). You can also type “`make check`” to put the FFTW test programs through their paces. If you have problems during configuration or compilation, you may want to run “`make distclean`” before trying again; this ensures that you don’t have any stale files left over from previous compilation attempts.

The `configure` script chooses the `gcc` compiler by default, if it is available; you can select some other compiler with:

```
./configure CC="<the name of your C compiler>"
```

The `configure` script knows good `CFLAGS` (C compiler flags) for a few systems. If your system is not known, the `configure` script will print out a warning. In this case, you should re-configure FFTW with the command

```
./configure CFLAGS="<write your CFLAGS here>"
```

and then compile as usual. If you do find an optimal set of `CFLAGS` for your system, please let us know what they are (along with the output of `config.guess`) so that we can include them in future releases.

`configure` supports all the standard flags defined by the GNU Coding Standards; see the `INSTALL` file in FFTW or the GNU web page. Note especially `--help` to list all flags and `--enable-shared` to create shared, rather than static, libraries. `configure` also accepts a few FFTW-specific flags, particularly:

- `--enable-float`: Produces a single-precision version of FFTW (`float`) instead of the default double-precision (`double`). See Precision.
- `--enable-long-double`: Produces a long-double precision version of FFTW (`long double`) instead of the default double-precision (`double`). The `configure` script will halt with an error message if `long double` is the same size as `double` on your machine/compiler. See Precision.
- `--enable-quad-precision`: Produces a quadruple-precision version of FFTW using the nonstandard `__float128` type provided by `gcc` 4.6 or later on x86, x86-64, and Itanium architectures, instead of the default double-precision (`double`). The `configure` script will halt with an error message if the compiler is not `gcc` version 4.6 or later or if `gcc`’s `libquadmath` library is not installed. See Precision.
- `--enable-threads`: Enables compilation and installation of the FFTW threads library (see Multi-threaded FFTW), which provides a simple interface to parallel transforms for SMP systems. By default, the threads routines are not compiled.
- `--enable-openmp`: Like `--enable-threads`, but using OpenMP compiler directives in order to induce parallelism rather than spawning its own threads directly, and installing an ‘`fftw3_omp`’ library rather than an ‘`fftw3_threads`’ library (see Multi-threaded FFTW). You can use both `--enable-openmp` and `--enable-threads` since they compile/install libraries with different names. By default, the OpenMP routines are not compiled.
- `--with-combined-threads`: By default, if `--enable-threads` is used, the threads support is compiled into a separate library that must be linked in addition to the main FFTW library. This is so that users of the serial library do not need to link the system threads libraries. If `--with-combined-threads` is specified, however, then no separate threads library is created, and threads are included in the main

FFTW library. This is mainly useful under Windows, where no system threads library is required and inter-library dependencies are problematic.

- `--enable-mpi`: Enables compilation and installation of the FFTW MPI library (see Distributed-memory FFTW with MPI), which provides parallel transforms for distributed-memory systems with MPI. (By default, the MPI routines are not compiled.) See FFTW MPI Installation.
- `--disable-fortran`: Disables inclusion of legacy-Fortran wrapper routines (see Calling FFTW from Legacy Fortran) in the standard FFTW libraries. These wrapper routines increase the library size by only a negligible amount, so they are included by default as long as the `configure` script finds a Fortran compiler on your system. (To specify a particular Fortran compiler *foo*, pass `F77=`*foo* to `configure`.)
- `--with-g77-wrappers`: By default, when Fortran wrappers are included, the wrappers employ the linking conventions of the Fortran compiler detected by the `configure` script. If this compiler is GNU `g77`, however, then *two* versions of the wrappers are included: one with `g77`'s idiosyncratic convention of appending two underscores to identifiers, and one with the more common convention of appending only a single underscore. This way, the same FFTW library will work with both `g77` and other Fortran compilers, such as GNU `gfortran`. However, the converse is not true: if you configure with a different compiler, then the `g77`-compatible wrappers are not included. By specifying `--with-g77-wrappers`, the `g77`-compatible wrappers are included in addition to wrappers for whatever Fortran compiler `configure` finds.
- `--with-slow-timer`: Disables the use of hardware cycle counters, and falls back on `gettimeofday` or `clock`. This greatly worsens performance, and should generally not be used (unless you don't have a cycle counter but still really want an optimized plan regardless of the time). See Cycle Counters.
- `--enable-sse` (single precision), `--enable-sse2` (single, double), `--enable-avx` (single, double), `--enable-avx2` (single, double), `--enable-avx512` (single, double), `--enable-avx-128-fma`, `--enable-kcvi` (single), `--enable-altivec` (single), `--enable-vsx` (single, double), `--enable-neon` (single, double on aarch64), `--enable-generic-simd128`, and `--enable-generic-simd256`:

  Enable various SIMD instruction sets. You need compiler that supports the given SIMD extensions, but FFTW will try to detect at runtime whether the CPU supports these extensions. That is, you can compile with`--enable-avx` and the code will still run on a CPU without AVX support.

  > - These options require a compiler supporting SIMD extensions, and compiler support is always a bit flaky: see the FFTW FAQ for a list of compiler versions that have problems compiling FFTW.
  > - Because of the large variety of ARM processors and ABIs, FFTW does not attempt to guess the correct `gcc` flags for generating NEON code. In general, you will have to provide them on the command line. This command line is known to have worked at least once:

  ```
  ./configure --with-slow-timer --host=arm-linux-gnueabi \
    --enable-single --enable-neon \
    "CC=arm-linux-gnueabi-gcc -march=armv7-a -mfloat-abi=softfp"
  ```

To force `configure` to use a particular C compiler *foo* (instead of the default, usually `gcc`), pass `CC=`*foo* to the `configure` script; you may also need to set the flags via the variable `CFLAGS` as described above.
