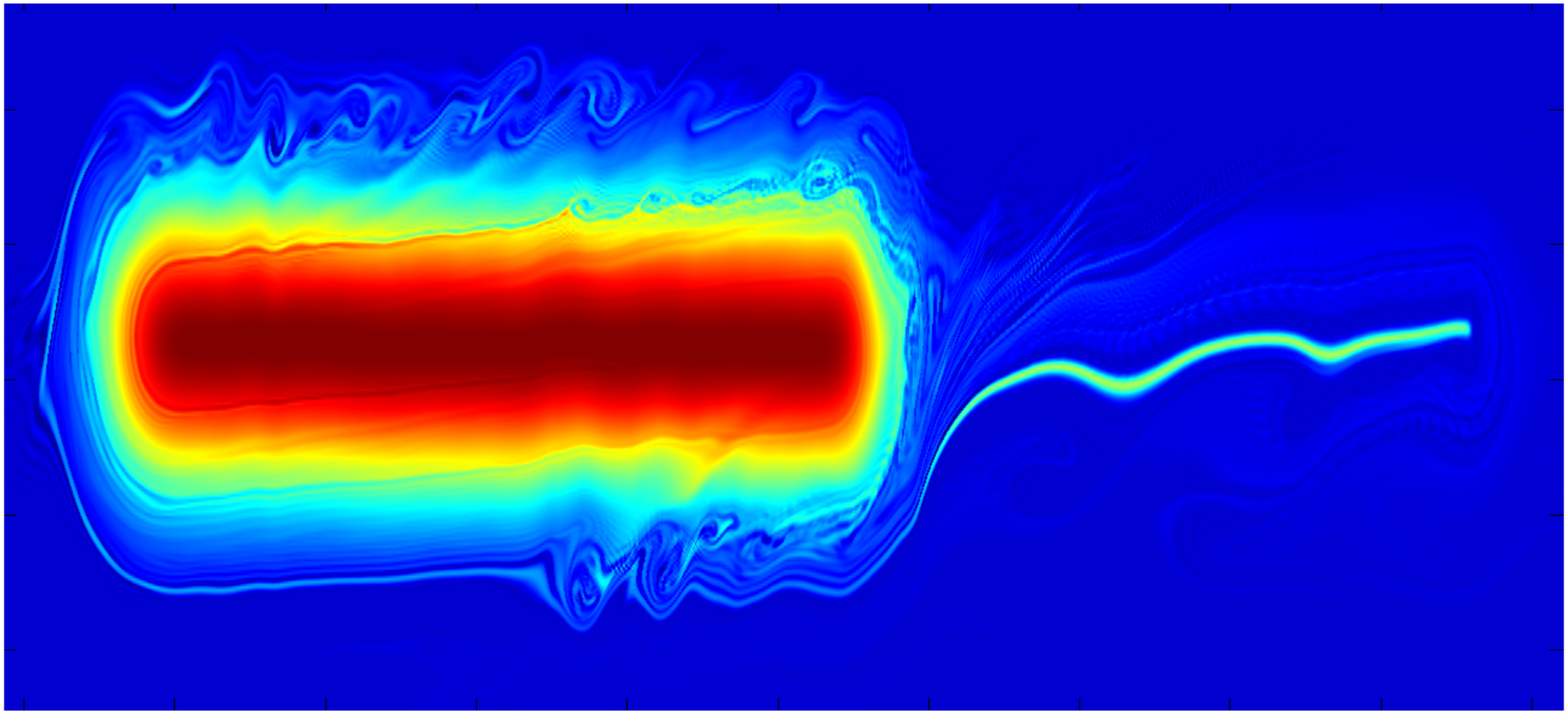
HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF



Chapter 6 - Libraries

Libraries

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

No need to re-invent the wheel...

- Libraries are a collection of functions or classes
- Many scientific libraries are open-source, i.e. we can get the source-code, look into the code, compile our own version of the library and use it for free
- There is a huge amount of interesting scientific/mathematical libraries out there. Examples are:
  - FFTW (Fourier Transform)
  - BLAS/LAPACK (Linear Algebra)
  - GSL (Collection of many scientific mathematical functions, ODE solvers,...)
  - PETSc (Linear Algebra / Non-linear Newton methods, fully parallel)
  - Eigen (Vector/Matrix library)
- Libraries that do not provide ready mathematical functions, but help us to implement parallel algorithms. Examples:
  - OpenMP (Multi-threading)
  - MPI (Communication between proccess running on multiple machines)

Libraries

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

From scratch to working library

- We will go through the necessary steps how to get a working library from its sources and use FFTW as an example. The steps for other libraries are very similar.

- Note: With many Linux distributions you can also install libraries via the packet manager from the repositories. Usually you want to have full control about what version of the library you are working with and the precise settings that have been used generating it. Hence, we generate the library ourselfs.

- *1. Step:* Download the sources.
  Sources usually come in a file with filename extension `*.tgz` on Linux. `*.tgz` means: This is a tar-ball (usuall ending: `*.tar`) but it is additionally zipped (i.e. compressed) using GZip.
- *2. Step:* Unpack the sources
  If the file from step 1 has the name `fftw.tgz`, unpack with the `tar` command in the console:
  `tar xvfz fftw.tgz`

From scratch to working library

- *Step 3:* Prepare to compile the library
  Once you have unpacked the tar-file, you have all source code of the library readily available. Still, it is of no real use to us. We need to compile the sources into a binary object, the library. Compilation often requires a setup which defines what compiler options will be used, where the final output will be stored and sets library depending options.
  This setup process is handled via the `configure` tool. In almost all unpacked sources you will find an executable script called `configure`. Run `configure --help` to see all available options.

  A configure call for FFTW might look like:
  ```
  ./configure --prefix=$HOME/libs/fftw --enable-sse2 --disable-fortran
  ```

  The –prefix option is very common and almost always present. It tells the compiler to where the eventually generated library object will be copied.

  Once configure is done Makefiles are generated. These are the files which contain all receipes the compiler has to know to compile the library.

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

From scratch to working library

- *Step 4:* Compiling the library
  Just type "`make`" (get some coffee, call some friends, check facebook, or sleep)
  Type "`make -j N`" to compile on N cores in your machine (less time for coffee,...)

- *Step 5:* Install the library
  Type "`make install`" to have the library copied to the place we specified with the `--prefix` flag of configure.

- Step 6: Done!
  Check out the install directory and you will find at least two sub-folders in it. One is called "`lib`", the other "`include`". In the lib directory you will find the binary object which is the actual library. For FFTW this is the file "libfftw3.a". In the include directory you will find all the header files you need to use the library.

Using the library in our project

- When we use the library in our project, there are two places we have to pay attention:

- In our source code we will have to include the header-file(s) from the library.
  - When we compile our source code, using functions from the library, the compiler will have to check if there are any functions around with these names and the appropriate interfaces.
    The compiler will read the header file of the library and be satisfied. We will have to tell the compiler where to look for the header files of the library.

- Once our own program is compiled the linker has to generate the final executable. Therefor, the compiler will take the object generated out of our program and the library object and "melt" them together.
  We just have to tell the compiler where to find the library and that it has to be included in the "melting" process.

Using the library in our project

- A short program which calculates a 1D complex to complex FFT using FFTW

```
#include <fftw3.h>


int main(){
    fftw_complex *in, *out;
    fftw_plan p_fwd, p_bck;
    const int N = 1024;

    in = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);
    out = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);

    p_fwd = fftw_plan_dft_1d(N, in, out, FFTW_FORWARD, FFTW_ESTIMATE);
    p_bck = fftw_plan_dft_1d(N, in, out, FFTW_BACKWARD, FFTW_ESTIMATE);

    fftw_execute(p_fwd); /* repeat as needed */
    fftw_execute(p_bck);

    fftw_destroy_plan(p_fwd);
    fftw_destroy_plan(p_bck);
    fftw_free(in);
    fftw_free(out);
}
```

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Using the library in our project

- When compiling our program we have to add to the compiler options
  - the linker option to link libfftw3.a FFTW to our final file
  - the include directory to the include path
  - the library directory to the library path

```
g++ fftw_test.cxx -o fftw_test -L$HOME/libs/fftw/lib -I$HOME/libs/fftw/includes -lfftw3
```

- When linking libraries there is a convention for the name of the libraries. A library file name starts always with `lib`... (like `libfftw3.a`). In the compiler command line we always have to leave away the `lib`... and the extension.
- Real life is a bit more tricky. Usually there are *static* and *shared libraries.* We used a static library so far. Static libraries have a .a (for *archive*) extension, while dynamic libraries have .so (for *shared object*) as extension.

Extra: Installing FFTW3 in Ubuntu

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

- The VirtualBox Linux that we provided is an Ubuntu Linux

- You can use the FFTW3 provided from the Ubuntu package manager `apt`:
  - Open a terminal
  - Type: `sudo apt install libfftw3-dev`
  - Now you should be able to compile the example programs from the slides with:

```
g++ -o fft_test fft_test.cxx -lfftw3
```