The GROMOS Software for (Bio)Molecular Simulation

Volume 8: Installation Guide

January 9, 2021

Contents

CHAPTER 1

System requirements

GROMOS can be compiled on almost any operating system compatible with the POSIX standard[1]. Make sure that your system is powerful enough, that you have sufficient disk space, and that all required libraries are installed.

The hardware requirements are rather high for simulations, but nowadays most of the setup and analysis can be carried out on personal computers or even laptops.

Architecture: Intel or AMD x86, SUN SPARC or IBM PowerPC CPU.

Memory: This depends on the simulation you are running and the analysis you want to carry out. For most simulations and analyses you need just a few 100 MB of RAM but there are exceptions.

Diskspace: This also depends on the simulation. Typical are up to 5 GB for a big simulation. For analysis, up to 1 GB is sufficient.

Please have a look at the software requirements because these are usually not installed on an out-of-box operating system.

Operating System: POSIX compatible UNIX like Linux.

Build-Essentials: make, binutils (usually installed by default).

Compiler(C++): In principle it is possible to use any ISO C++ compiler but it is recommended to use the GNU Compiler Collection (GCC) `g++`.

Libraries(C++): There are a few of libraries that have to be installed on your system. Make sure the header files (-devel, -dev packages) are also installed. The libraries needed are:

- zlib compression library
- gsl GNU Scientific Library (see Appendix)
- for MD++: socket, nsl, fftw (3.3)

CHAPTER 2

Installation of required libraries

Some of the libraries required by GROMOS++ and MD++ are not available on standard operating systems and have to be installed manually. The installation of these libraries is discussed in this chapter. Make sure you only install the libraries that you really need for the subpart of GROMOS you want to install.

2.1. GNU scientific library

The GNU Scientific Library[2] is a C library for scientific calculations like complex number arithmetic, fast Fourier transformations, integration of functions etc. It is needed by MD++ and GROMOS++. Usually it can be installed via your operating systems package manager.

On Debian or Ubuntu Linux you can install it by typing `sudo apt-get install libgsl0 libgsl0-dev`. On Windows, install it via the CYGWIN setup.

If it is not distributed with your operating system or you are not super user you have to compile it from the source code. Fortunately this is rather easy and straightforward.

2.1.1. Installation from source. In your home create a directory where you want to install the library `/home/user/lib/gsl` and a working directory `/home/user/tmp`. Go to the GSL web page ftp://ftp.gnu.org/gnu/gsl/ and download the latest version into your working directory[1]. In a command line shell `cd /home/user/tmp` to the working directory and untar the package:

```
~/tmp> tar zxf gsl-2.6.tar.gz
~/tmp> cd gsl-2.6
```

Now you need to `./configure` for the compilation and installation. As a prefix give the directory where you want to install the library. After successful configuration make and install the GSL.

```
~/tmp/gsl-2.6> ./configure --prefix=/home/user/lib/gsl
~/tmp/gsl-2.6> make
~/tmp/gsl-2.6> make install
```

After a successful installation you can delete the working directory

```
~/tmp/gsl-2.6> cd ..
~/tmp> rm -rf gsl-2.6*
```

The GSL is now successfully installed in your home.

2.2. FFTW 3

The Fastest Fourier Transform in the West library[3] is a C library used to carry out fast Fourier transformations. It is needed by MD++. Usually the version available from package managers is too old to be of any use. Make sure that you install version 3. MD++ uses MPI parallelization and thus it is important that an MPI version of FFTW is installed.

On Debian or Ubuntu Linux you can install it by typing `sudo apt-get install libfftw3-3 libfftw3-dev`. However, this will not install the MPI version of the library.

If it is not distributed with your operating system or you are not super user you have to compile it from the source code.

[1] In the example below the version 2.6 was used but make sure the most recent version is used

2.2.1. Installation from source. In your home create a directory where you want to install the library `/home/user/lib/fftw3` and a working directory `/home/user/tmp`. Go to the FFTW web page http://www.fftw.org and download the latest version into your working directory. In a command line shell go to your working directory and untar the package:

```
~/tmp> tar zxf fftw-3.3.8.tar.gz
~/tmp> cd fftw-3.3.8
```

Then configure FFTW. In this case we want to build an MPI version of the library. Thus one has to use the correct MPI compiler wrappers and enable MPI.

```
~/tmp/fftw-3.3.8> ./configure --prefix=/home/user/lib/fftw3 \
                              --enable-moi CC=mpicc CXX=mpiCC F77=mpif77 \
                              --enable-fortran --disabled-shared
~/tmp/fftw-3.3.8> make
~/tmp/fftw-3.3.8> make install
```

Like this the normal FFTW library, an MPI version, a Fortran binding and shared libraries are installed. On clusters with multiple MPI implemenations installed it is important to use the same compiler wrappers for FFTW and for MD++.

CHAPTER 3

Installation of GROMOS

3.1. Installing MD++

MD++ is a C++ batch program for molecular simulation jobs. Because its execution time is critical you have to compile it on the machine you would like to use it to make sure that the maximum performance is obtained. Open a command line shell and find out where your home is. Then create a working and an installation directory (md++).

```
~> pwd
/home/user
~> mkdir temp
~> mkdir md++
```

Change the path `/home/user` to your home (the result of the pwd command).

Copy md++.tar.gz into your working directory. Change into the `temp` directory and unpack MD++.

```
~> cd temp
~/temp> tar zxf md++.tar.gz
~/temp> cd md++
```

In this directory now lies the source code of MD++ which is ready for compilation. The configuration is a bit tricky, because there are many options which are explained below:

1. You can specify the installation path using the `--prefix` directive.
2. If the GNU Scientific Library (gsl) was installed by the superuser `configure` will find it. If it is installed locally in your home you must specify this using the `--with-gsl` directive.
3. The same applies for the FFTW library. If it is installed locally in your home you must specify this using the `--with-fftw` directive.
4. On clusters where setup of library paths is not guaranteed, it is a good idea to link it statically. `--disable-shared` will disable the shared library and create statically linked executables.

Now you know what the options mean and you are ready to configure and run the compilation of MD++. On multicore CPU machines add the `-j` flag to `make` to boost the compilation.

```
~/temp/md++> ./configure --prefix=/path/to/install/md++ \
                         --with-gsl=/path/to/gsl \
                         --with-fftw=/path/to/fftw3 \
                         --disable-shared
~/temp/md++> make
~/temp/md++> make install
~/temp/md++> make check
~/temp/md++> touch doc/doxygen.conf.in
~/temp/md++> make doc
~/temp/md++> cp -r doc /path/to/install/md++
```

MD++ is now installed. You can test it by typing

```
~/temp/gromosxx> /path/to/install/md++/bin/md
```

If it prints the usage everything is fine.

Clean up the working directory because the static builds need a lot of diskspace and are not needed anymore.

```
~/temp/md++> cd ..
~/temp> rm -rf md++
```

3.1.1. Debug version of MD++. MD++ allows you to compile a special version for debugging. This version has additional debug statements and the code is not optimised by the compiler. You can enable debugging by giving `--enable-debug` as an argument to `configure`. Using this special version you can specify level of debug information you are interested in. See Sec. 6-1.1.2 for details.

3.1.2. Parallel version of MD++. In order to enable MD++ to use the full power of your computer's hardware you have to compile a parallel version. MD++ knows two kinds of parallelization:

OpenMP: This parallelization is straightforward and enables MD++ to run on multiple core CPUs, like all recent x86 CPUs are. No additional software is required. You can enable this by adding `--enable-openmp` to configure.

MPI: MPI is used for parallelization when no shared memory is available: the different CPUs you want to use for the calculations are located in different machines. This is the case in computer clusters.

To compile the MPI version you have to use a special set of compilers wrappers, which know which MPI version and implementation you have on the cluster. In general these compilers are called `mpicc` for the C compiler and `mpiCC` for the C++ compiler. You have to tell `configure` to use these compilers and to `--enable-mpi`:

```
~/temp/md++> ./configure CC=mpicc CXX=mpiCC \
                         --enable-mpi \
                         --disable-shared \
                         --with-gsl=/path/to/gsl \
                         --with-fftw=/path/to/fftw3 \
                         --prefix=/home/user/md++
```

You have to make sure that the binary of the FFTW library was compiled using the same compiler wrappers and is linked to the same MPI libraries. This can be achieved by compiling an own version of FFTW with MPI enabled.

```
~/temp/fftw-3.3.8> ./configure --enable-mpi CC=mpicc CXX=mpiCC F77=mpif77 \
                               --enable-fortran \
                               --prefix=/path/to/install/fftw3_mpi
~/temp/fftw-3.3.8> make
~/temp/fftw-3.3.8> make install
```

After successful configuration just `make` and `make install` it as usual. If the test call

```
~/temp/md++> /home/user/md++/md/md_mpi
```

does not tell you to enable MPI, everything is fine.

3.1.3. Compiling MD++ using the CUDA solvent-solvent interaction evaluation acceleration. In order to make use of the CUDA solvent-solvent interaction evaluation acceleration library (*cukernel*) MD++ has to be compiled using an additional path pointing to the directory containing the CUDA libraries and header file.

```
~/temp/md++> ./configure --disable-shared \
                         --with-gsl=/path/to/gsl \
                         --with-fftw=/path/to/fftw3 \
                         --with-cuda=/path/to/cuda \
                         --prefix=/home/user/md++
```

If necessary the appropriate compiler and flags can be set by adding the appropriate variables, e.g.:

```
NVCC=nvcc
NVCCFLAGS='-arch sm_30'
NVCC_CFLAGS='-O2 -D DNDEBUG -lcuda -lcudart '
```

3.1.4. What is installed. After successful compilation,

1. the program binaries are in `bin/`. If you used `--prefix` option, you will find `bin/` there. Note that for MPI support you should use the binary `md_mpi` (or `repex_mpi` for replica exchange).
2. in the `include/` and `lib/` subdirectories are the files needed for programming with MD++.

3.2. Installing GROMOS++

GROMOS++ is a collection of command line programs needed to setup a simulation or to analyze the results of a simulation. In order to use these programs you have to compile and install them on your machine. Open a command line shell and find out where your home is.

```
~> pwd
/home/nschmid
```

Change the path `/home/user` to your home (the result of the pwd command).

Unpack GROMOS++.

```
~> tar xfz gromos++.tar.gz
~> cd gromos++
```

You are now in the GROMOS++ source directory and can start to configure and make GROMOS++. Tell configure where it has to look for the GNU Scientific Library (gsl) and the Fastest Fourier Transform in the West library (fftw) using the `--with-gsl` and `--with-fftw` directives. Usually these libraries are installed in /usr/local and configure will find it without telling, but if you have installed them in your home you have to tell configure using the `--with-gsl` and `--with-fftw`directives. Some programs in GROMOS++ make use of algorithms of MD++. In order to use these programs one has to specify the location of the MD++ libraries and header files using the `--with-mdpp` directive. Debugging should be disabled in order to make use of compiler optimizations. Some operating systems require static linking which can be controlled by the `--disable-shared` directive. A few computationally demanding programs can be run in parallel on shared memory machines using OpenMP. OpenMP can be enabled by the `--enable-openmp` directive. On multicore CPU machines add the `-j` flag to `make` to speed up the compilation.

```
~/gromos++> ./configure --with-gsl=/path/to/gsl --with-fftw=/path/to/fftw
~/gromos++> make
~/gromos++> make install
```

3.2.1. Generating the documentation. If doxygen is installed on your machine you can generate documentation directly from the source code. Go to the gromos++ directory and type

```
~/gromos++> touch doc/doxygen.conf.in
~/gromos++> make doc
```

In the `doc` directory you will find html documentation. Open a web browser and open `file:///home/<username>/<path to gromos++>/gromos++/doc/html/index.html`. Under `available` you find the documentation of the various GROMOS++ programs.

After the successful installation you should clean up the working directory.

```
~/gromos++> make clean
```

3.2.2. Adding it to the path. In order to use the programs without specifying the full path you can add them to your PATH veriable. Add the following two lines to your `~/.bashrc`:

```
export PATH="${PATH}:/path/to/gromos++/bin"
export LD_LIBRARY_PATH="${LD_LIBRARY_PATH}:/path/to/gromos++/lib"
```

3.2.3. What is installed. After successful compilation,

1. the program binaries are in `bin/`. If you used `--prefix` option, you will find `bin/` there.
2. in the `include/` and `lib/` subdirectories are the files needed for programming with GROMOS++
3. `share/` is home of useful files (called libraries) and examples.

Bibliography

[1] IEEE Standards Department. *IEEE 1003.1-2008, Standard for Information Technology - Portable Operating System Interface (POSIX®)*. Institute of Electrical and Electronics Engineers (IEEE), 2008.

[2] M. Galassi, J. Daviesm, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi. *Gnu Scientific Library: Reference Manual*. Network Theory Ltd., 2003.

[3] F. Matteo and S. G. Johnson. The Design and Implementation of FFTW3. *Proceedings of the IEEE*, 93:216–231, 2005.

Index