Note that: to be precise, CMake is not a build system but rather it generates another system's build files. Although it supports quite a few build environments, only Make is supported by QE. The current page title is kept to avoid breaking links.

Pre-requisites

Need CMAKE 3.20 or later to compile QE.

If there is no internet access on the machine where the code is compiled, execute `initialize_external_repos.sh` under `external/` before transferring QE source code to that machine.

Make sure your QE source directory is clean. It should not contain any *.mod files from previous in-source builds

Getting started

In order to build *all* the targets provided by QE, an *out of source* build can be performed like this:

```
$ mkdir build
$ cd build
$ cmake -DCMAKE_C_COMPILER=mpicc -DCMAKE_Fortran_COMPILER=mpif90 <path to QE source directory>
$ make [-jN]
```

If you do the above steps from the QE root source directory, just use `..` as `<path to QE source directory>` above.

Compiling a subset of QE is also supported, for example `make pw ph cp`. Accepted targets are `all_currents, couple, cp, epw, gwl, hp, ld1, neb, ph, pp, pw, pwall, pwcond, tddfpt, upf, xspectra`.

Once the make step is completed successfully, it is recommended to run basic tests via CTest

To install QE to a specific destination which you have write access

```
$ cmake -DCMAKE_INSTALL_PREFIX=DESIRED_DESTINATION .
$ make install
```

`make install` implicitly does `make` and the whole QE will be built and installed. If installing only a subset is desired, users need to manually copy executables from `bin` after building the subset.

Build options

Notable options provided by CMake.

- `CMAKE_C_COMPILER`, `CMAKE_Fortran_COMPILER` C and Fortran compilers. **Although CMake has the capability to speculate compilers, it is strongly recommended to specify the intended compilers or preferably MPI compiler wrappers, when using MPI, as `CMAKE_Fortran_COMPILER` and `CMAKE_C_COMPILER`.**
- `CMAKE_C_FLAGS`, `CMAKE_Fortran_FLAGS` additional compiler flags applied globally.
- `BUILD_SHARED_LIBS` default OFF. Build QE internal libraries as shared libraries.
- `CMAKE_INSTALL_PREFIX` install path when `make install`
- `CMAKE_VERBOSE_MAKEFILE` default OFF. Print full compilation command lines. Massive output. It is better to use `make VERBOSE=1` when need to investigate an issue.
- `CMAKE_PREFIX_PATH` optional. When CMake failed to to find the intended library, use it to hint alternative search locations. Some packages may also support `_ROOT .

See full list at https://cmake.org/cmake/help/latest/manual/cmake-variables.7.html

Notable options provided by QE.

- `QE_ENABLE_MPI` default ON. Use MPI parallelization.
- `QE_ENABLE_MPI_MODULE` default OFF. use MPI via Fortran module instead of mpif.h header inclusion.
- `QE_ENABLE_OPENMP` default ON when QE_ENABLE_CUDA=ON, otherwise OFF. Use OpenMP threading.
- `QE_ENABLE_CUDA` default OFF. Use CUDA GPU acceleration on NVIDIA GPUs.

- `QE_ENABLE_OPENACC` default OFF. Use OpenACC acceleration.
- `QE_ENABLE_MPI_GPU_AWARE` default OFF. Use GPU aware MPI operations
- `QE_ENABLE_STATIC_BUILD` default OFF. Build all the QE executables as static binaries.
- `QE_ENABLE_DOC` default OFF. Build all the latex docs.
- `QE_ENABLE_PLUGINS` default empty. A semicolon-separated list of plugins being built together with QE. Currently supported plugins are `d3q`, `gipaw`, `legacy`, `pw2qmcpack`. To enable multiple plugins, pass them as a list `-DQE_ENABLE_PLUGINS="d3q;pw2qmcpack"`.
- `QE_CPP`. C preprocessor used for parsing LAXLIB header files. If not passed in via command line, environment variable CPP is used. If neither a command line argument nor the environment variable exists, the default value is cpp (GNU C preprocessor). The need of this option is extremely rare.

Notable options for controlling libraries used by QE

- BLAS/LAPACK. See `BLA_VENDOR` for library selection. Alternatively, set `QE_LAPACK_INTERNAL=ON` to build a vanilla Netlib LAPACK.
- FFT. `QE_FFTW_VENDOR` default AUTO. Select a specific host FFTW library [Intel_DFTI, Intel_FFTW3, ArmPL, IBMESSL, FFTW3, Internal]. GPU accelerated libraries are enabled in addition to host libraries when corresponding programming models are selected.
- ScaLAPACK. `QE_ENABLE_SCALAPACK` default OFF. Leverage ScaLAPACK to solve eigenvalues over MPI.
- ELPA. `QE_ENABLE_ELPA` default OFF. Use ELPA library as enhancement to ScaLAPACK.
- LibXC. `QE_ENABLE_LIBXC` default OFF. Use LibXC for more exchange correlation functionals. Optionally, `LIBXC_ROOT` can be used for locating an LibXC installation.
- HDF5. `QE_ENABLE_HDF5` default OFF. Use HDF5 file format for I/O of binary dataset. Optionally, `HDF5_ROOT` can be used for locating an hdf5 installation.
- Fox. `QE_ENABLE_FOX` default OFF. Use Fox library for XML I/O. If OFF, use an internal replacement.

Compiler specific options

- `NVFORTRAN_CUDA_VERSION` optionally select `nvfortran` underlying CUDA toolkit version. If not specified, `nvfortran` uses its own default.
- `NVFORTRAN_CUDA_CC` optionally select the compute capability used by `nvfotran`. For example, `70` for V100 and `80` for A100. If not specified, `nvfortran` builds all its known compute capabilities and takes more time.

Notable options for interacting with tools.

All off by default.

- `QE_ENABLE_TRACE`. For execution tracing output.
- `QE_ENABLE_PROFILE_NVTX`. Use NVIDIA NVTX profiler plugin.
- `QE_ENABLE_SANITIZER`. default none. Support GNU address and thread sanitizer [none,asan,tsan]. When using the thread sanitizer, `export TSAN_OPTIONS='ignore_noninstrumented_modules=1 halt_on_error=1'` removes false positive failure from the OpenMP runtime and halts execution after the first warning in each test run. GCC is required to be compiled with `--disable-linux-futex` option to prevent false positive warning from the thread sanitizer.
- `QE_CLOCK_SECONDS` default OFF. Prints program time in seconds, otherwise days-hours-minutes[-seconds].

In order to set a specific option's value you can ask the `cmake` command; e.g.: to enable CUDA accelerated code paths on NVIDIA GPUs :

```
$ cmake -DQE_ENABLE_CUDA=ON ..
```

If you would like to inspect the available build options and their values for the current build, just do:

```
$ cd <QE source directory>
$ cd build
$ cmake -LH ..

-- Cache values
// Choose the type of build.
CMAKE_BUILD_TYPE:STRING=Release

// Install path prefix, prepended onto install directories.
CMAKE_INSTALL_PREFIX:PATH=/Users/fmontag/qe-install

// enable distributed execution support via MPI
```

```
QE_ENABLE_MPI:BOOL=ON

// enable distributed execution support via OpenMP
QE_ENABLE_CUDA:BOOL=OFF

...
...
```

CMake command line for installs on HPC clusters

Galileo100 @CINECA

1. Default install with oneapi 2021 - compiler, mpi, mkl (21 May 2023)

```
module load intel/oneapi-2021--binary intelmpi/oneapi-2021--binary mkl/oneapi-2021--binary cmake/3.21.4 git
```

```
cmake -DCMAKE_C_COMPILER=icc -DCMAKE_CXX_COMPILER=icpc \
-DCMAKE_Fortran_COMPILER=mpiifort \
-DCMAKE_C_FLAGS:STRING=-xCORE-AVX512 -DCMAKE_Fortran_FLAGS:STRING=-xCORE-AVX512 \
-DQE_ENABLE_OPENMP=ON -DCMAKE_BUILD_TYPE:STRING=RELWITHDEBINFO ../
```

Marconi @CINECA

1. Default install with intel 2020 - compiler, mpi, mkl (21 May 2023)

```
module load intel/pe-xe-2020--binary intelmpi/2020--binary mkl/2020--binary cmake git
```

```
cmake -DCMAKE_C_COMPILER=icc -DCMAKE_C_FLAGS:STRING=-xCORE-AVX512 \
-DCMAKE_CXX_COMPILER=icpc -DCMAKE_Fortran_COMPILER=mpiifort \
-DCMAKE_Fortran_FLAGS:STRING=-xCORE-AVX512 -DQE_ENABLE_OPENMP=ON \
-DQE_ENABLE_SCALAPACK=ON -DCMAKE_BUILD_TYPE:STRING=RELWITHDEBINFO ../
```

Summit @ORNL

```
module load nvhpc hdf5 netlib-lapack essl cmake fftw

git clone https://gitlab.com/QEF/q-e.git
cd q-e
mkdir build
cd build

cmake -DCMAKE_C_COMPILER=mpicc -DCMAKE_Fortran_COMPILER=mpif90 \
-DQE_ENABLE_CUDA=ON -DNVFORTRAN_CUDA_CC=70 -DQE_ENABLE_HDF5=ON \
-DLAPACK_LIBRARIES="-L$OLCF_ESSL_ROOT/lib64 -lessl $OLCF_NETLIB_LAPACK_ROOT/lib64/liblapack.a" ..

make -j8
```

Frontera @TACC (courtesy of Hyungjun Lee, UT Austin)

Use of the system default collection of modules (Intel toolchains) (29 May 2023)

```
module reset

cd <path to QE source directory>
mkdir build
```

```
cd build

cmake -DCMAKE_C_COMPILER=icc -DCMAKE_C_FLAGS:STRING=-xCORE-AVX512 \
-DCMAKE_Fortran_COMPILER=mpiifort \
-DCMAKE_Fortran_FLAGS="-xCORE-AVX512 -assume byterecl -traceback" \
-DQE_ENABLE_OPENMP=ON -DQE_ENABLE_SCALAPACK=ON -DCMAKE_BUILD_TYPE:STRING=RELWITHDEBINFO \
-DMPIEXEC_EXECUTABLE=$(which ibrun) ..

make [-jN]
```

Juwels Booster @FZ-JUELICH

- Bare install with NVHPC and OpenMPI

```
module load Stages/2022 NVHPC/22.3 OpenMPI/4.1.2 CUDA/11.5 FFTW/3.3.10 CMake

cmake -DCMAKE_C_COMPILER=nvc -DCMAKE_Fortran_COMPILER=mpif90 \
 -DQE_ENABLE_CUDA=ON -DQE_ENABLE_OPENACC=ON \
-DNVFORTRAN_CUDA_CC=80 -DQE_FFTW_VENDOR=FFTW3 ../

make [-jN]
```

- Install for Score-P instrumentation

```
module load Stages/2022 NVHPC/22.3 OpenMPI/4.1.2 FFTW/3.3.10 CUDA/11.5 Score-P/7.1 CMake

SCOREP_WRAPPER=off cmake -DCMAKE_C_COMPILER=scorep-pgcc -DCMAKE_Fortran_COMPILER=scorep-mpif90 \
-DQE_ENABLE_CUDA=ON -DQE_ENABLE_OPENACC=ON \
-DNVFORTRAN_CUDA_CC=80 -DQE_FFTW_VENDOR=FFTW3 ../

make SCOREP_WRAPPER_INSTRUMENTER_FLAGS="--mpp=mpi --thread=none --noopenmp --openacc --cuda --user" [-jN]
```

Leonardo-booster @CINECA

- OpenMPI software stack

```
module load fftw/3.3.10--openmpi--4.1.4--nvhpc--23.1
module load openblas/0.3.21--nvhpc--23.1
module load openmpi/4.1.4--nvhpc--23.1-cuda-11.8
module load nvhpc/23.1
module load cmake
module load git
```

```
cmake -DCMAKE_C_COMPILER=nvc -DCMAKE_Fortran_COMPILER=mpif90 \
-DQE_ENABLE_MPI_GPU_AWARE=ON -DQE_ENABLE_CUDA=ON -DQE_ENABLE_OPENACC=ON \
-DQE_FFTW_VENDOR=FFTW3 -DNVFORTRAN_CUDA_VERSION=11.8 -DNVFORTRAN_CUDA_CC=80 ../
```

```
make -j all
```

! Do not load cuda/11.8 module, it gives issues at runtime (14 July 2023)

- Alternative based on mkl

```
module purge
module load profile/archive
```

```
module load intel-oneapi-mkl/2022.2.1--gcc--11.3.0
module load nvhpc/23.1
module load openmpi/4.1.4--nvhpc--23.1-cuda-11.8

cmake -DCMAKE_C_COMPILER=nvc -DCMAKE_Fortran_COMPILER=mpif90 -DQE_ENABLE_PROFILE_NVTX=ON -DQE_ENABLE_CUDA=ON -D
make -j all
```

- HPCX-MPI software stack

```
module purge
module load profile/candidate
module load binutils/2.42
module load openblas/0.3.26--nvhpc--24.5
module load fftw/3.3.10--hpcx-mpi--2.19--nvhpc--24.5
module load nvhpc/24.5
module load hpcx-mpi/2.19
module load cmake/3.27.9
```

```
cmake -DCMAKE_C_COMPILER=nvc -DCMAKE_Fortran_COMPILER=mpif90 -DQE_ENABLE_CUDA=ON -DQE_ENABLE_OPENACC=ON -DQE_EN
```

```
make -j all
```

Leonardo-dcgp @ CINECA

- Portable install with Intel software stack (works on login or compute)

```
module purge
module load git/2.38.1
module load cmake/3.24.3
module load intel-oneapi-mpi/2021.7.1
module load intel-oneapi-mkl/2022.2.1
module load intel-oneapi-compilers@2023.0.0
```

```
cmake -DCMAKE_C_COMPILER=icc -DCMAKE_C_FLAGS:STRING="-xCORE-AVX512" -DCMAKE_Fortran_COMPILER=mpiifort -DCMAKE_F
```

```
make -j all
```

Karolina @ IT4I

```
module purge
module load FFTW/3.3.10-NVHPC-24.3-CUDA-12.3.0
module load NVHPC/24.3-CUDA-12.3.0
module load OpenMPI/4.1.6-NVHPC-24.3-CUDA-12.3.0
module load  CMake/3.27.6-GCCcore-13.2.0
```

```
cmake -DCMAKE_C_COMPILER=nvc -DCMAKE_Fortran_COMPILER=mpif90 \
-DQE_ENABLE_MPI_GPU_AWARE=ON -DQE_ENABLE_CUDA=ON -DQE_ENABLE_OPENACC=ON -DQE_ENABLE_OPENMP=ON \
-DQE_FFTW_VENDOR=FFTW3 -DNVFORTRAN_CUDA_VERSION=12.3 -DNVFORTRAN_CUDA_CC=80 ../
```

```
make -j all
```