User's Guide for Quantum ESPRESSO (v.7.2)

Contents

1 Introduction

This guide gives a general overview of the contents and of the installation of QUANTUM ESPRESSO (opEn-Source Package for Research in Electronic Structure, Simulation, and Optimization), version 7.2.

Important notice: due to the lack of time and of manpower, this manual does not cover many important aspects and may contain outdated information.

The QUANTUM ESPRESSO distribution contains the core packages `PWscf` (Plane-Wave Self-Consistent Field) and `CP` (Car-Parrinello) for the calculation of electronic-structure properties within Density-Functional Theory (DFT), using a Plane-Wave (PW) basis set and pseudopotentials. It also includes other packages for more specialized calculations:

- `PWneb`: energy barriers and reaction pathways through the Nudged Elastic Band (NEB) method.
- `PHonon`: vibrational properties with Density-Functional Perturbation Theory (DFPT).
- `PostProc`: codes and utilities for data postprocessing.
- `PWcond`: ballistic conductance.
- `XSPECTRA`: K-, L_1-, $L_{2,3}$-edge X-ray absorption spectra.
- `TD-DFPT`: spectra from Time-Dependent Density-Functional Perturbation Theory.
- `GWL`: electronic excitations within the GW approximation and with the Bethe-Salpeter Equation
- `EPW`: calculation of the electron-phonon coefficients, carrier transport, phonon-limited superconductivity and phonon-assisted optical processes;
- `HP`: calculation of Hubbard U parameters using DFPT;
- `QEHeat`: energy current in insulators for thermal transport calculations.
- `KCW`: quasiparticle energies of finite and extended systems using Koopmans-compliant functionals in a Wannier representation.

The following auxiliary packages are included as well:

- `PWgui`: a Graphical User Interface, producing input data files for `PWscf` and some `PostProc` codes.
- `atomic`: atomic calculations and pseudopotential generation.

A copy of required external libraries is either included or automatically downloaded from the net. Finally, several additional packages that exploit data produced by QUANTUM ESPRESSO or patch some QUANTUM ESPRESSO routines can be automatically installed using `make`:

- `Wannier90`: maximally localized Wannier functions.
- `WanT`: quantum transport properties with Wannier functions.
- `YAMBO`: electronic excitations within Many-Body Perturbation Theory, GW and Bethe-Salpeter equation.
- `D3Q`: anharmonic force constants.
- `GIPAW` (Gauge-Independent Projector Augmented Waves): NMR chemical shifts and EPR g-tensor.

For QUANTUM ESPRESSO with the self-consistent continuum solvation (SCCS) model, aka "Environ", see `http://www.quantum-environment.org/`.

Documentation on single packages can be found in the `Doc/` directory of each package. A detailed description of input data is available for most packages in files `INPUT_*.txt` and `INPUT_*.html`.

The QUANTUM ESPRESSO codes work on many different types of Unix machines, including parallel machines using both OpenMP and MPI (Message Passing Interface). QUANTUM ESPRESSO also runs on Mac OS X and MS-Windows machines (see section 2.2). Since Feb.2021 the main repository also works with NVidia GPU's.

Further documentation, beyond what is provided in this guide, can be found in:

- the `Doc/` and `examples/` directories of the QUANTUM ESPRESSO distribution;
- the web site `www.quantum-espresso.org`;
- the archives of the mailing list: See section 1.2, "Contacts", for more info.

People who want to contribute to QUANTUM ESPRESSO should read the Wiki pages on GitLab: `https://gitlab.com/QEF/q-e/-/wikis`.

This guide does not explain the basic Unix concepts (shell, execution path, directories etc.) and utilities needed to run QUANTUM ESPRESSO; it does not explain either solid state physics and its computational methods. If you want to learn the latter, you should first read a good textbook, such as e.g. the book by Richard Martin: *Electronic Structure: Basic Theory and Practical Methods*, Cambridge University Press (2004); or: *Density functional theory: a practical introduction*, D. S. Sholl, J. A. Steckel (Wiley, 2009); or *Electronic Structure Calculations for Solids and Molecules: Theory and Computational Methods*, J. Kohanoff (Cambridge University Press, 2006). Then you should consult the documentation of the package you want to use for more specific references.

1.1 People

The maintenance and further development of the QUANTUM ESPRESSO distribution is promoted by the QUANTUM ESPRESSO Foundation under the coordination of Paolo Giannozzi (Univ. Udine and IOM-CNR, Italy) and Pietro Delugas (SISSA Trieste) with a strong support from the MaX - Materials design at the Exascale EU Centre of Excellence. The GPU porting is mostly the work of Pietro Bonfà (Univ. Parma).

Contributors to QUANTUM ESPRESSO, beyond the authors of the papers mentioned in Sec.1.4, include:

- Victor Yu (Urbana-Champaign) for various bug fixes and optimizations;
- Alexandre Tkatchenko's group, in particular Szabolcs Goger (U. Luxembourg), and Robert DiStasio's group, in particular Hsin-Yu Ko (Cornell), for Many-Body Dispersion (MBD) correction;
- Federico Ficarelli and Daniele Cesarini (CINECA), with help from Ye Luo (Argonne) and Sebastian Gsänger, for CMake support;
- Ye Luo (Argonne) for many contributions to improved threading, GPU porting, CI (Continuous integration), and testing;
- Fabio Affinito and Sergio Orlandini (CINECA) for ELPA support, for contributions to the FFT library, and for various parallelization improvements;
- Sebastiano Caravati for direct support of GTH pseudopotentials in analytical form, Santana Saha and Stefan Goedecker (Basel U.) for improved UPF converter of newer GTH pseudopotentials;
- Axel Kohlmeyer for libraries and utilities to call QUANTUM ESPRESSO from external codes (see the `COUPLE` sub-directory), made the parallelization more modular and usable by external codes;
- Èric Germaneau for TB09 meta-GGA functional, using `libxc`;
- Guido Roma (CEA Saclay) for vdw-df-obk8 e vdw-df-ob86 functionals;
- Yves Ferro (Univ. Provence) for SOGGA and M06L functionals;
- Ikutaro Hamada (NIMS, Japan) for RPBE, OPTB86B-vdW, REV-vdW-DF2 functionals, fixes to pw2xsf utility;
- Daniel Forrer (Padua Univ.) and Michele Pavone (Naples Univ. Federico II) for dispersions interaction in the framework of DFT-D;
- Filippo Spiga (University of Cambridge, now at NVidia) for mixed MPI-OpenMP parallelization and for the first GPU-enabled version;
- Costas Bekas and Alessandro Curioni (IBM Zurich) for the initial BlueGene porting.

Contributors to specific QUANTUM ESPRESSO packages are acknowledged in the documentation of each package.

An alphabetic list of further contributors who answered questions on the mailing list, found bugs, helped in porting to new architectures, wrote some code, contributed in some way or another at some stage, follows:

> Åke Sandgren, Audrius Alkauskas, Alain Allouche, Francesco Antoniella, Uli Aschauer, Francesca Baletto, Gerardo Ballabio, Mauro Boero, Scott Brozell, Claudia Bungaro, Paolo Cazzato, Gabriele Cipriani, Jiayu Dai, Stefano Dal Forno, Cesar Da Silva, Alberto Debernardi, Gernot Deinzer, Alin Marin Elena, Francesco Filipponi, Prasenjit Ghosh, Marco Govoni, Thomas Gruber, Martin Hilgeman, Yosuke Kanai, Konstantin Kudin, Nicolas Lacorne, Hyungjun Lee, Stephane Lefranc, Sergey Lisenkov, Kurt Maeder, Andrea Marini, Giuseppe Mattioli, Nicolas Mounet, William Parker, Pasquale Pavone, Mickael Profeta, Chung-Yuan Ren, Kurt Stokbro, David Strubbe, Sylvie Stucki, Paul Tangney, Pascal Thibaudeau, Davide Tiana, Antonio Tilocca, Jaro Tobik, Malgorzata Wierzbowska, Vittorio Zecca, Silviu Zilberman, Federico Zipoli,

and let us apologize to everybody we have forgotten.

1.2 Contacts

The main entry point for QUANTUM ESPRESSO users is the web site:

> `http://www.quantum-espresso.org/`.

There you find the stable releases for download, general information and documentation.

The recommended place where to ask questions about installation and usage of QUANTUM ESPRESSO, and to report problems, is the mailing list `users@lists.quantum-espresso.org`. Here you can obtain help from the developers and from knowledgeable users. You have to be subscribed (see the "Contacts" section of the web site) in order to post to the users' list. Please check your spam folder if you do not get a confirmation message when subscribing.

Please read the guidelines for posting, section 1.3! PLEASE NOTE: only messages that appear to come from the registered user's e-mail address, in its *exact form*, will be accepted. In case of trouble, carefully check that your return e-mail is the correct one (i.e. the one you used to subscribe).

The main entry point for developers is the GitLab web site: `https://gitlab.com/QEF/q-e`. If you need to contact the developers for *specific* questions about coding, proposals, offers of help, etc., you may either post an "Issue" to GitLab, or send a message to the developers' mailing list `developers@lists.quantum-espresso.org`. Please do not post general questions there: they will be ignored.

1.3 Guidelines for posting to the mailing list

Life for mailing list subscribers will be easier if everybody complies with the following guidelines:

- Before posting, *please*: browse or search the archives – links are available in the "Contacts" section of the web site. Most questions are asked over and over again. Also: make an attempt to search the available documentation, notably the FAQs and the User Guide(s). The answer to most questions is already there.

- Reply to both the mailing list and the author or the post, using "Reply to all".
- Sign your post with your name and affiliation.
- Choose a meaningful subject. Do not start a new thread by making a "reply" to an existing message: it will confuse the ordering of messages into threads that most mailers can do. In particular, do not use "Reply" to a Digest!!!
- Be short: no need to send 128 copies of the same error message just because this is what came out of your 128-processor run. No need to send the entire compilation log for a single error appearing at the end.
- Do not post large attachments: post a link to a place where the attachment(s) can be downloaded from, such as e.g. DropBox, GoogleDocs, or one of the various web temporary storage spaces.
- Avoid excessive or irrelevant quoting of previous messages. Your message must be immediately visible and easily readable, not hidden into a sea of quoted text.
- Remember that even experts cannot guess where a problem lies in the absence of sufficient information. One piece of information that must *always* be provided is the version number of QUANTUM ESPRESSO.
- Remember that the mailing list is a voluntary endeavor: nobody is entitled to an answer, even less to an immediate answer.
- Finally, please note that the mailing list is not a replacement for your own work, nor is it a replacement for your thesis director's work.

1.4 Terms of use

QUANTUM ESPRESSO is free software, released under the GNU General Public License. See `http://www.gnu.org/licenses/old-licenses/gpl-2.0.txt`, or the file License in the distribution).

We shall greatly appreciate if scientific work done using the QUANTUM ESPRESSO distribution will contain an acknowledgment to the following references:

> P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G. L. Chiarotti, M. Cococcioni, I. Dabo, A. Dal Corso, S. Fabris, G. Fratesi, S. de Gironcoli, R. Gebauer, U. Gerstmann, C. Gougoussis, A. Kokalj, M. Lazzeri, L. Martin-Samos, N. Marzari, F. Mauri, R. Mazzarello, S. Paolini, A. Pasquarello, L. Paulatto, C. Sbraccia, S. Scandolo, G. Sclauzero, A. P. Seitsonen, A. Smogunov, P. Umari, R. M. Wentzcovitch, J.Phys.: Condens.Matter 21, 395502 (2009)

and

> P. Giannozzi, O. Andreussi, T. Brumme, O. Bunau, M. Buongiorno Nardelli, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, M. Cococcioni, N. Colonna, I. Carnimeo, A. Dal Corso, S. de Gironcoli, P. Delugas, R. A. DiStasio Jr, A. Ferretti, A. Floris, G. Fratesi, G. Fugallo, R. Gebauer, U. Gerstmann, F. Giustino, T. Gorni, J Jia, M. Kawamura, H.-Y. Ko, A. Kokalj, E. Küçükbenli, M .Lazzeri, M. Marsili, N.

Marzari, F. Mauri, N. L. Nguyen, H.-V. Nguyen, A. Otero-de-la-Roza, L. Paulatto, S. Poncé, D. Rocca, R. Sabatini, B. Santra, M. Schlipf, A. P. Seitsonen, A. Smogunov, I. Timrov, T. Thonhauser, P. Umari, N. Vast, X. Wu, S. Baroni, J.Phys.: Condens.Matter 29, 465901 (2017)

Users of the GPU-enabled version should also cite the following paper:

P. Giannozzi, O. Baseggio, P. Bonfà, D. Brunato, R. Car, I. Carnimeo, C. Cavazzoni, S. de Gironcoli, P. Delugas, F. Ferrari Ruffino, A. Ferretti, N. Marzari, I. Timrov, A. Urru, S. Baroni, J. Chem. Phys. 152, 154105 (2020)

Note the form QUANTUM ESPRESSO for textual citations of the code. Please also see package-specific documentation for further recommended citations. Pseudopotentials should be cited as (for instance)

[ ] We used the pseudopotentials C.pbe-rrjkus.UPF and O.pbe-vbc.UPF from `http://www.quantum-espresso.org`.

2 Installation

2.1 Download

QUANTUM ESPRESSO is distributed in source form, but selected binary packages and virtual machines are also available. Stable and development releases of the QUANTUM ESPRESSO source package (current version is 7.2), as well as available binary packages, can be downloaded from the links listed in the "Download" section of `www.quantum-espresso.org`.

The Quantum Mobile virtual machine for Windows/Mac/Linux/Solaris provides a complete Ubuntu Linux environment, containing QUANTUM ESPRESSO and much more. Link and description in `https://www.materialscloud.org/work/quantum-mobile`.

For source compilation, uncompress and unpack compressed archives in the typical .tar.gz format using the command:

```
tar zxvf qe-X.Y.Z.tar.gz
```

(a hyphen before "zxvf" is optional) where `X.Y.Z` stands for the version number.

A few additional packages that are not included in the base distribution will be downloaded on demand at compile time, using either `make` or `CMake` (see Sec.2.7). Note however that this will work only if the computer you are installing on is directly connected to the internet and has either `wget` or `curl` installed and working. If you run into trouble, manually download each required package into subdirectory `archive/`, *not unpacking or uncompressing it*: command `make` will take care of this during installation.

The QUANTUM ESPRESSO distribution contains several directories. Some of them are common to all packages:

`Modules/`	Fortran modules and utilities used by all programs
`upflib/`	pseudopotential-related code, plus conversion tools
`include/`	files *.h included by fortran and C source files
`FFTXlib/`	FFT libraries
`LAXlib/`	Linear Algebra (parallel) libraries
`KS_Solvers/`	Iterative diagonalization routines
`UtilXlib/`	Miscellaneous timing, error handling, MPI utilites
`XClib/`	Exchange-correlation functionals (excepted van der Waals)
`MBD/`	Routines for many-body dispersions
`LR_Modules/`	Fortran modules and utilities used by linear-response codes
`install/`	installation scripts and utilities
`pseudo/`	pseudopotential files used by examples
`Doc/`	general documentation
`archive/`	external libraries in .tar.gz form
`external/`	external libraries downloaded by CMake
`test-suite/`	automated tests

while others are specific to a single package:

`PW/`	`PWscf` package
`EPW/`	`EPW` package
`NEB/`	`PWneb` package
`PP/`	`PostProc` package
`PHonon/`	`PHonon` package
`PWCOND/`	`PWcond` package
`CPV/`	`CP` package
`atomic/`	`atomic` package
`GUI/`	`PWGui` package
`HP/`	`HP` package
`QEHeat/`	`QEHeat` package
`KCW/`	`KCW` package

Finally, directory `COUPLE/` contains code and documentation that is useful to call QUANTUM ESPRESSO programs from external codes.

2.2 Prerequisites

To install QUANTUM ESPRESSO from source, you need first of all a minimal Unix environment, that is: a command shell (e.g., bash, sh) and utilities `make`, `awk`, `sed`. In order to install external libraries from scratch, you will also need `git` v.2.13 or later. For MS-Windows, see Sec.2.9.4. If you prefer installation with CMake (supported since v.6.7), you will need a recent (v.3.14 or later) working CMake software; otherwise you will need the `configure` command from autoconf and, for FoX compilation, `m4`.

Note that the scripts contained in the distribution assume that the local language is set to the standard, i.e. "C"; other settings may break them. Use `export LC_ALL=C` (sh/bash) or `setenv LC_ALL C` (csh/tcsh) to prevent any problem when running scripts (including installation scripts).

Second, you need a Fortran compiler compliant with F2008 standard. For parallel execution, you will also need MPI libraries and a parallel (i.e. MPI-aware) compiler. For massively parallel machines, or for simple multicore parallelization, an OpenMP-aware compiler and libraries are

also required. To compile for GPUs you need a recent version of the NVidia HPC SDK (software development kit), formerly PGI compiler, freely available for download.

As a rule, QUANTUM ESPRESSO tries to keep compatibility with older compilers, avoiding nonstandard extensions and newer features that are not widespread or stabilized. If however your compiler is older than a few ($\sim$ 5) years, it is likely that something will not work. The same applies to mathematical and MPI libraries. For GPU compilation, you need v.19.10 or later of the NVidia HPC SDK (previous versions are no longer supported).

Big computing centers typically provide a Fortran compiler complete with all needed libraries. Workstations or "commodity" machines using PC hardware, may or may not have the needed software. If not, you need to buy a commercial compiler or to use the open-source gfortran compiler from the gcc distribution (and possibly MPI libraries and run-time software). Note that most commercial compilers are also available free of charge under some conditions (e.g. academic or personal usage, no support) and may provide MPI libraries and run-time software as well.

2.3 Building with CMake

See `https://gitlab.com/QEF/q-e/-/wikis/Developers/CMake-build-system`.

2.4 Building with `make`

To install the QUANTUM ESPRESSO source package using `make`, run the `configure` script. This is actually a wrapper to the true `configure`, located in the `install/` subdirectory (`configure` -h for help). `configure` will (try to) detect compilers and libraries available on your machine, and set up things accordingly. Presently it supports all "common" computers, that is: based on Intel, AMD, ARM CPUs, running Linux, Mac OS X, MS-Windows. QUANTUM ESPRESSO is known to work on many more kinds of machines but may requires some tweaking, especially for the hardware of large HPC centers. Detailed but sometimes outdated installation instructions for specific HPC machines may be found in files `install/README.`*sys*, where *sys* is the machine name.

Instructions for the impatient:

```
cd qe-X.Y.Z/
./configure
 make all
```

This will (try to) produce parallel (MPI) executable if a proper parallel environment is detected, serial executables otherwise. For OpenMP executables, specify `./configure --enable-openmp`. Symlinks to executable programs will be placed in the `bin/` subdirectory. Note that both C and Fortran compilers must be in your execution path, as specified in the PATH environment variable.

`configure` generates the following files:

`make.inc`	compilation rules and flags (used by `Makefile`)
`install/configure.msg`	a report of the configuration run (not needed for compilation)
`install/config.log`	detailed log of the configuration run (useful for debugging)
`include/qe_cdefs.h`	(previously: `include/c_defs.h`) a few definitions used by C files
`include/configure.h`	optional: info on compilation flags (to enable it, uncomment `#define __HAVE_CONFIG_INFO` in Modules/environment.f90)

In addition, `configure` generates (since v.7) files `make.depend`, containing dependencies upon

modules, in the various subdirectories. If you add/remove/move/rename modules, or change the list of objects in any `Makefile`, type `make depend`, or run `./install/makedeps.sh`, to update files `make.depend`.

It is convenient to use "parallel make" to speed up compilation: `make -jN` compiles in parallel on N processors. Note that if you interrupt `make`, you may run into trouble the next time you type `make`(for instance, if `make` is interrupted while unpacking and compiling an external library). If so, run `make clean`, or even `make distclean`, before running `make` again.

You should always be able to compile the QUANTUM ESPRESSO suite of programs without having to edit any of the generated files. However you may have to tune `configure` by specifying appropriate environment variables and/or command-line options. Usually the tricky part is to get external libraries recognized and used: see Sec.2.5 for details and hints. In most cases, you may simply edit file `make.inc`.

Environment variables may be set in any of these ways:

```
export VARIABLE=value; ./configure            # sh, bash, ksh
setenv VARIABLE value; ./configure            # csh, tcsh
env VARIABLE=value ./configure                # any shell
./configure VARIABLE=value                    # any shell
```

As a rule: do not define environment variables for `configure` unless you have a good reason to. Try `configure` with no options as a first step. Some environment variables that are relevant to `configure` are:

`ARCH`	label identifying the machine type (see below)
`F90, CC`	names of Fortran and C compilers
`MPIF90`	name of parallel Fortran 90 compiler (using MPI)
`CPP`	source file preprocessor (defaults to $CC -E)
`LD`	linker (defaults to $MPIF90)
`(C,F,F90,CPP,LD)FLAGS`	compilation/preprocessor/loader flags
`LIBDIRS`	extra directories where to search for libraries

(note that `F90` is an "historical" name – we actually use Fortran 2008 – and that it should be used only together with option `--disable-parallel`. In fact, the value of F90 must be consistent with the parallel Fortran compiler which is determined by `configure` and stored in the `MPIF90` variable).

For example, the following command line:

```
./configure MPIF90=mpif90 FFLAGS="-O2 -assume byterecl" \
            CC=gcc CFLAGS=-O3 LDFLAGS=-static
```

instructs `configure` to use `mpif90` as Fortran compiler with flags `-O2 -assume byterecl`, `gcc` as C compiler with flags `-O3`, and to link with flag `-static`. Note that the value of `FFLAGS` must be quoted, because it contains spaces. NOTA BENE: passing the complete path to compilers (e.g., `F90=/path/to/f90xyz`) may lead to obscure errors during compilation.

If your machine type is unknown to `configure`, you may use the `ARCH` variable to suggest an architecture among supported ones. Some parallel machines using a front-end may actually need it, or else `configure` will correctly recognize the front-end but not the specialized compilation environment of those machines. In some cases, cross-compilation requires to specify the target machine with the `--host` option. This feature has not been extensively tested, but we had at least one successful report (compilation for NEC SX6 on a PC). Currently supported architectures are:

`x86_64`	Intel and AMD 64-bit running Linux
`arm`	ARM machines (with gfortran or armflang)
`craype`	Cray machines using Cray PE
`mac686`	Apple Intel machines running Mac OS X
`mingw32`	Cross-compilation for MS-Windows, using mingw, 32 bits
`mingw64`	As above, 64 bits
`cygwin`	MS-Windows PCs with Cygwin
`ppc64`*	Linux PowerPC machines, 64 bits
`ppc64-le`*	as above, with IBM xlf compiler
`ppc64-bg`*	IBM BlueGene
`ppc64-bgq`*	IBM BlueGene Q
`necsx`*	NEC SX-6 and SX-8 machines
`ia32`*	Intel 32-bit machines (x86) running Linux
`ia64`*	Intel 64-bit (Itanium) running Linux

Note: `x86_64` replaces `amd64` since v.4.1. Cray Unicos machines, SGI machines with MIPS architecture, HP-Compaq Alphas are no longer supported since v.4.2; PowerPC Macs are no longer supported since v.5.0; IBM machines with AIX are no longer supported since v.6.0; all architectures marked with a * are to be considered obsolescent or obsolete.

Finally, `configure` recognizes the following command-line options. Not all of them are implemented for all compilers, though. Default value is between bracket:

`--enable-parallel`	compile for parallel (MPI) execution if possible (yes)
`--enable-openmp`	compile for OpenMP execution if possible (no)
`--enable-static`	produce static executables, arger but more portable (no)
`--enable-shared`	produce objects that are suitable for shared libraries (no)
`--enable-debug`	compile with debug flags (no)
`--enable-pedantic`	compile with gfortran pedantic flags on (no)
`--enable-signals`	enable signal trapping (no)

and the following optional packages:

`--with-fox`	Use official FoX library instead of built-in replacement (default:no)
`--with-scalapack`	(yes|no|intel) Use scalapack if available. Set to `intel` to use Intel MPI and BLACS (default: use OpenMPI)
`--with-elpa-include`	Specify full path of ELPA include and modules headers (no)
`--with-elpa-lib`	Specify full path of the ELPA library (no)
`--with-elpa-version`	Specify ELPA API version: 2015 for ELPA releases 2015.x and 2016.05; 2016 for ELPA releases 2016.11, 2017.x and 2018.05; 2018 for ELPA releases 2018.11 and beyond (2018)
`--with-hdf5`	(no | yes | `<path>`) Compile HDF5 support (no). If "yes", configure assumes a valid v. >= 1.8.16 HDF5 installation with h5cc and h5fc in the default executable search path. If `<path>` is specified, it must be the root folder of a standalone hdf5 installation.
`--with-hdf5-libs`	Specify the link options and libraries needed to link HDF5, if configure fails to detect them. These options are usually composed by many substrings and must be enclosed into quotes.
`--with-hdf5-include`	Specify full path the HDF5 include folder containing module and headers files. Use it if configure fails to find the include folder.
`--with-libxc`	Enable support for the `libxc` library (no)
`--with-libxc-prefix`	directory where `libxc` is installed
`--with-libxc-include`	directory where `libxc` Fortran headers reside

In order to compile the code for NVidia GPU's you will need a recent version (v.19.10 or later: the more recent, the better) of the NVidia HPC software development kit (SDK). OpenMP should be enabled. Enabling faster communications between GPUs, via NVlink or Infiniband RDMA, is essential for optimal performance. If your MPI library is built to be CUDA-aware, then enable it with `--with-cuda-mpi=yes`. The following `configure` options are available:

`--with-cuda=value`	enable compilation of GPU-accelerated subroutines. `value` should point the path where the CUDA toolkit is installed, e.g. `/opt/cuda` (default: no)
`--with-cuda-cc=value`	sets the compute capabilities for the compilation of the accelerated subroutines. `value` must be consistent with the hardware and the NVidia driver installed on the workstation or on the compute nodes of the HPC facility (default: 35)
`--with-cuda-runtime=value`	sets the version of the CUDA toolkit used for the compilation of the accelerated code. `value` must be consistent with the CUDA Toolkit installed on the workstation or available on the compute nodes of the HPC facility.
`--with-cuda-mpi=value`	enable usage of a CUDA-aware MPI library (default: no).

To modify or extend `configure` (advanced users only!), see the Wiki pages on GitLab: `https://gitlab.com/QEF/q-e/-/wikis`.

2.4.1 Manual configuration

If `configure` stops before the end, and you don't find a way to fix it, you have to write a working `make.inc` file (optionally, `include/qe_cdefs.h`). The template used by `configure`

is `install/make.inc.in` and contains explanations of the meaning of the various variables. Note that you may need to select appropriate preprocessing flags in conjunction with the desired or available libraries (e.g. you need to add `-D__FFTW` to `DFLAGS` if you want to link internal FFTW). For a correct choice of preprocessing flags, refer to the documentation in `include/defs.h.README`.

Even if `configure` works, you may need to tweak the `make.inc` file. It is very simple, but please note that a) you must know what you are doing, and b) if you change any settings (e.g. preprocessing, compilation flags) after a previous, successful or failed, compilation, you must run `make clean` before recompiling, unless you know exactly which routines are affected by the changed settings and how to force their recompilation. Running `configure` again cleans objects and executables, unless you use option `--save`.

2.5 Libraries

QUANTUM ESPRESSO downloads a copy of the following external libraries if needed:

- FoX for reading and writing xml files;
- BLAS (`http://www.netlib.org/blas/`) and LAPACK (`http://www.netlib.org/lapack/`) for linear algebra;
- FFTW (`http://www.fftw.org/`) for Fast Fourier Transforms.

Optimized vendor-specific libraries often yield huge performance gains with respect to compiled libraries and should be used whenever possible. `configure` always try to locate the best mathematical libraries.

BLAS and LAPACK QUANTUM ESPRESSO can use any architecture-optimized BLAS and LAPACK replacements, like those contained e.g. in the MKL for Intel and AMD CPUs, or ESSL for IBM Power machines.

If no optimized libraries are available, one may try the ATLAS library: `http://math-atlas.sourcef` Note that ATLAS is not a complete replacement for LAPACK: it contains all of the BLAS, plus the LU code, plus the full storage Cholesky code. Follow the instructions in the ATLAS distributions to produce a full LAPACK replacement.

Sergei Lisenkov reported success and good performances with optimized BLAS by Kazushige Goto. The library is now available under an open-source license: see the GotoBLAS2 page at `http://www.tacc.utexas.edu/tacc-software/gotoblas2/`.

FFT The FFTXlib of QUANTUM ESPRESSO contains a copy of an old FFTW library. It also supports the newer FFTW3 library and some vendor-specific FFT libraries. `configure` will first search for vendor-specific FFT libraries; if none is found, it will search for an external FFTW v.3 library; if none is found, it will fall back to the internal copy of FFTW. `configure` will add the appropriate preprocessing options:

- `-D__LINUX_ESSL` for ESSL on IBM Linux machines,
- `-DASL` for NEC ASL library on NEC machines (obsolete?),

- `-D__DFTI` for DFTI (Intel MKL library),
- `-D__FFTW3` for FFTW3 (external),
- `-D__FFTW` for FFTW (internal library),

to `DFLAGS` in the `make.inc` file. If you edit `make.inc` manually, please note that one and only one among the mentioned preprocessing option must be set.

If you have MKL libraries, you may either use the provided FFTW3 interface (v.10 and later), or directly link FFTW3 from MKL (v.12 and later) or use DFTI (recommended).

MPI libraries MPI libraries are usually needed for parallel execution, unless you are happy with OpenMP-only multicore parallelization. In well-configured machines, `configure` should find the appropriate parallel compiler for you, and this should find the appropriate libraries. Since often this doesn't happen, especially on PC clusters, see Sec.2.9.3.

Note: since v.6.1, MPI libraries implementing v.3 of the standard (notably, non-blocking broadcast and gather operations) are required.

Libraries for accelerators The accelerated version of the code uses standard CUDA libraries `cublas, cufft, curand, cusolver`, available from the NVidia HPC SDK.

HDF5 The HDF5 library (`https://www.hdfgroup.org/downloads/hdf5/`), v.1.8.16 or later, can be used to perform binary I/O using the HDF5 format.

If compiling the HDF5 library from sources, attention must be paid to pass options: `--enable-fortran`, `--enable-fortran2003`, and `--enable-parallel` (see below), to the `configure` script of `HDF5` (not of QUANTUM ESPRESSO).

To use HDF5 is usually sufficient to specify the path to the fortran compiler wrapper for HDF5 (`h5fc` of `h5pfc` with the `--with-hdf5=` option of configure. If the wrapper is in the default path, just use `--with-hdf5=yes`. The configure script is usually able to extract the linker options and the include directory path from the output of the wrapper. If it fails, the user can provide configure options `--with-hdf5-libs=<options>` and `--with-hdf5-include=<path>` for the linker options and include path respectively. These options are often needed when using the HDF5 packages provided by many LINUX distributions. In this case you may first try the `--with-hdf5=yes` option. If it fails, just type command `h5fc --show` (or `h5pfc` if you are using parallel HDF5): the command will print out the linker and include options to be passed manually to the configure script.

The configure script is able to determine whether one is linking to a serial or parallel HDF5 library, and will set the flag `-D__HDF5_SERIAL` in the `make.inc` file accordingly.

Other libraries QUANTUM ESPRESSO can use the MASS vector math library from IBM, if available (only on machines with XLF compiler: likely obsolete).

If optimized libraries are not found The `configure` script attempts to find optimized libraries, but may fail if they have been installed in non-standard places. You should examine the final value of `BLAS_LIBS, LAPACK_LIBS, FFT_LIBS, MPI_LIBS` (if needed), `MASS_LIBS` (IBM only), either in the output of `configure` or in the generated `make.inc`, to check whether it found all the libraries that you intend to use.

If some library was not found, you can specify a list of directories to search in the environment variable LIBDIRS, and rerun configure; directories in the list must be separated by spaces. For example:

```
./configure LIBDIRS="/opt/intel/mkl70/lib/32 /usr/lib/math"
```

If this still fails, you may set some or all of the *_LIBS variables manually and retry. For example:

```
./configure BLAS_LIBS="-L/usr/lib/math -lf77blas -latlas_sse"
```

Beware that in this case, configure will blindly accept the specified value, and won't do any extra search.

2.6 Libxc library

QUANTUM ESPRESSO is compatible with libxc version 4.3.0 or later (compatibility with older versions is not guaranteed).
The libxc functionals are available for LDA, GGA and metaGGA, however, not all of them are straightforwardly usable. Some of them may depend on specific external parameters and some others may provide as output the energy or the potential, but not both. Therefore some attention has to be paid when using libxc. Warning messages should appear in the output when particular cases whose correct operation in QUANTUM ESPRESSO is not guaranteed are chosen.

2.6.1 Linking in QUANTUM ESPRESSO

Once installed libxc, the linking with QUANTUM ESPRESSO can be enabled directly through the configuration script by adding the two switches --with-libxc and --with-libxc-prefix, e.g.:

```
./configure --with-libxc --with-libxc-prefix='/path/to/libxc/'
```

By adding the first switch only an automatic search for the libxc folder will be attempted, but its success is not guaranteed. It is always preferable to specify the second switch too. Optionally, a third switch can be added, namely --with-libxc-include='/path/to/libxc/include', which specifies the path to the Fortran headers (usually it is not necessary).

Alternatively, the link to libxc can be activated after the configuration of QUANTUM ESPRESSO by modifying the make.inc file in the main folder in this way:

- add -D__LIBXC to DFLAGS
- add -I/path/to/libxc/include/ to IFLAGS
- set LD_LIBS=-L/path/to/libxc/lib/ -lxcf03 -lxc

Then QUANTUM ESPRESSO can be compiled as usual.

Note for version 5.0.0: if the version of `libxc` is 5.0.0, the last point must be replaced by:

- set `LD_LIBS=-L/path/to/libxc/lib/ -lxcf90 -lxc`

since the `f03` interfaces are no longer available. They have been restored in following releases. Version 5.0.0 is still usable, but, before compiling QUANTUM ESPRESSO, a string replacement is necessary, namely '`xc_f03`' must be replaced with '`xc_f90`' everywhere in the `XClib` folder.

2.6.2 Usage

In order to use `libxc` functionals, you can enforce them from input by including the `input_dft` string in the `system` namelist. Starting from v7.0 of QUANTUM ESPRESSO the only allowed notation for DFTs that include Libxc terms is the index one. For example, to use the `libxc` version of the PBE functional (both exchange and correlation):

```
input_dft = 'XC-000I-000I-101L-130L-000I-000I'
```

The letters I or L next to each ID stand for Internal and `libxc`. This is equivalent to the old full-name notation:

```
input_dft = 'gga_x_pbe gga_c_pbe'   ***OLD***
```

The order must be the usual one, namely LDA exchange, LDA correlation, GGA exchange, GGA correlation, MGGA exchange, MGGA correlation. `libxc` exchange+correlation functionals can be put in the exchange or in the correlation slot with no difference.
The reason why the full-name notation has been disabled is to eliminate the risk of overlaps among different names (occurring especially when combinations of internal and `libxc` DFTs are used).
The complete list of `libxc` functionals (and their IDs) is available at:

```
https://www.tddft.org/programs/libxc/functionals/
```

Combinations of QUANTUM ESPRESSO and `libxc` functionals are allowed in `PW`, but some attention has to be paid to their reciprocal compatibility (see section below).
For example, the internal exchange term of PBE together with the correlation term of PBE in `libxc` is obtained by:

```
input_dft = 'XC-001I-000I-003I-130L-000I-000I'
```

which corresponds to the old:

```
input_dft = 'sla pbx gga_c_pbe'   ***OLD***
```

Note that when using GGA internal functionals you must always specify the LDA term too, while it is not the case for the `libxc` ones.
Abbreviations are allowed when zero tails are present. The above example is still valid by putting:

```
input_dft = 'XC-001I-000I-003L-130L'
```

since no MGGA terms are present.
Non-local terms can be included by just adding their name after the index notation, for example:

```
input_dft='XC-001i-004i-013i-vdw2'
```

2.6.3 Differences between Libxc and internal functionals

There are some differences between QUANTUM ESPRESSO functionals and `libxc` ones. In QUANTUM ESPRESSO the LDA and the GGA terms are separated and must be specified independently. In `libxc` the GGA functionals already include the LDA part (Slater exchange and Perdew&Wang correlation in most of the cases with the exception, for example, of Lee Yang Parr functionals).
The `libxc` metaGGA functionals may or may not need the LDA and GGA terms, depending on the cases, therefore a good check of the chosen functionals is recommended before doing expensive runs.
Some functionals in `libxc` incorporate the exchange part and the correlation one into one term only (e.g. the ones that include the '`_xc`' kind label in their name). In these cases the whole functional is formally treated as 'correlation only' by QUANTUM ESPRESSO. This does not imply any loss of information.

2.6.4 Special cases

A number of `libxc` functional routines need extra information on input and/or provide only partial information on output (e.g. the energy or the potential only). In these cases the use of such functionals may not be straightforward and, depending on the cases, may require some work on the QUANTUM ESPRESSO source code.

External parameters. Several functionals in `libxc` depend on one or more external parameters. Some of these can be recovered inside QUANTUM ESPRESSO, some others are not directly available. In all these cases a direct intervention on the QUANTUM ESPRESSO source code might be necessary in order to be able to properly use such functionals. However two routines have been defined in the XC library of QUANTUM ESPRESSO that ease the task of setting and recovering the external parameters in `libxc`:

- `get_libxc_ext_param`: this function receives as input the ID of the `libxc` functional and the index of the chosen parameter and returns its value. If the parameter has not been set before it returns its default value.

- `set_libxc_ext_param`: this routine receives as input the index of the functional family-type (from 1 to 6: lda-exch, lda-corr, gga-exch, ...), the index of the chosen `libxc` parameter and the value to set it to.

In order to see the available parameters for a given `libxc` functional and their corresponding indexes, the `xc_infos.x` program is available in `XClib` folder. For more details see Sec. 2.6.5.
The two routines can be called almost anywhere in QUANTUM ESPRESSO, however, as any other `XClib` setting routine, they must be declared through the `xc_lib` module.
Without setting the external parameters inside the code, their default value will be assumed. This could lead to results different from the expectations.
In any case, when external parameters are needed by the chosen functionals, a warning message will appear in the output of QUANTUM ESPRESSO. An example of Libxc parameter setting can be found in the `xclib_test.f90` code (see below).

Functionals with partial output. A few `libxc` functional routines provides the potential but not the energy. These functionals are available in QUANTUM ESPRESSO for all the families and their output energy is set to zero.

MGGA Functionals that depend on the Laplacian of the density. At present such functionals are formally usable in QUANTUM ESPRESSO , but their dependency on the Laplacian is ignored and the corresponding output term of the potential is set to zero. Since the Laplacian of the density is computable in QUANTUM ESPRESSO, they might be fully exploited with a limited intervention on the code.

Other functionals. Besides exchange (`_x`), correlation (`_c`) and exchange plus correlation (`_xc`), a fourth kind of functionals is available in `libxc`, the kinetic functionals (`_k`). At present, they are not usable in QUANTUM ESPRESSO.

2.6.5 XC test

A testing program, `xclib_test.x`, for the `XClib` library of QUANTUM ESPRESSO is available. The program is available for LDA, GGA and MGGA functionals (both QE and Libxc). It also tests the potential derivatives for LDA (`dmxc`) and GGA (`dgcxc`).
Another small program, `xc_infos.x`, is available in the `XClib` folder starting from v6.8. It receives as input the name of any DFT usable in QUANTUM ESPRESSO (both internal and `libxc`) and provides infos about their family, type, external parameters, limitations, references, etc.
See XClib/README.TEST file for further details on each of the two programs.

2.7 Compilation

The compiled codes can run with any input: almost all variables are dinamically allocated at run time. Only a few variables have fixed dimensions, set in file `Modules/parameters.f90`:

```
ntypx  = 10,     &! max number of different types of atom
npsx   = ntypx,  &! max number of different PPs (obsolete)
nsx    = ntypx,  &! max number of atomic species (CP)
npk    = 40000,  &! max number of k-points
lmaxx  = 3,      &! max non local angular momentum (l=0 to lmaxx)
lqmax= 2*lmaxx+1  ! max number of angular momenta of Q
```

These values should work for the vast majority of cases. In case you need more atomic types or more k-points, edit this file and recompile.

At your choice, you may compile the complete QUANTUM ESPRESSO suite of programs (with `make all`), or only some specific programs. All executables are linked in main `bin` directory. `make` with no arguments yields ain updated list of valid compilation targets.

For the setup of the GUI, refer to the `PWgui-X.Y.Z /INSTALL` file, where X.Y.Z stands for the version number of the GUI (should be the same as the general version number). If you are using sources from the git repository, see the `GUI/README` file instead.

If `make` refuses for some reason to download additional packages, manually download them into subdirectory `archive/`, *not* unpacking or uncompressing them, and try `make` again. Also see Sec.(2.1).

2.8 Running tests and examples

As a final check that compilation was successful, you may want to run some or all of the tests and examples. Notice that most tests and examples are devised to be run serially or on a small number of processors; do not use tests and examples to benchmark parallelism, do not try to run on too many processors.

2.8.1 Test-suite

Automated tests give a "pass/fail" answer. All tests run quickly (less than a minute each at most), but they are not meant to be realistic, just to test a specific case. Many features are tested but only for the following codes: `pw.x`, `cp.x`, `ph.x`, `epw.x`, `hp.x`. Instructions for the impatient:

```
cd test-suite
make run-tests
```

Instructions for all others: go to the `test-suite/` directory, read the `README` file, or at least, type `make`. You may need to edit the `run-XX.sh` shells, defining variables `PARA_PREFIX` and `PARA_POSTFIX` (see below for their meaning).

2.8.2 Examples

There are many examples and reference data for almost every piece of QUANTUM ESPRESSO, but you have to manually inspect the results.

In order to use examples, you should edit file `environment_variables`, setting the following variables as needed.

BIN_DIR: directory where executables reside
PSEUDO_DIR: directory where pseudopotential files reside
TMP_DIR: directory to be used as temporary storage area

The default values of BIN_DIR and PSEUDO_DIR should be fine, unless you have installed things in nonstandard places. TMP_DIR must be a directory where you have read and write access to, with enough available space to host the temporary files produced by the example runs, and possibly offering high I/O performance (i.e., don't use an NFS-mounted directory). NOTA BENE: do not use a directory containing other data: the examples will clean it!

If you have compiled the parallel version of QUANTUM ESPRESSO (this is the default if parallel libraries are detected), you will usually have to specify a launcher program (such as `mpirun` or `mpiexec`) and the number of processors: see Sec.3 for details. In order to do that, edit again the `environment_variables` file and set the `PARA_PREFIX` and `PARA_POSTFIX` variables as needed. Parallel executables will be run by a command like this:

```
$PARA_PREFIX pw.x $PARA_POSTFIX -i file.in > file.out
```

For example, if the command line is like this (as for an IBM SP):

```
mpirun -np 4 pw.x -i file.in > file.out
```

you should set `PARA_PREFIX="mpirun -np 4"`, `PARA_POSTFIX=""`. Furthermore, if your machine does not support interactive use, you must run the commands specified above through the batch queuing system installed on that machine. Ask your system administrator for instructions. For execution using OpenMP on N threads, use `PARA_PREFIX="env OMP_NUM_THREADS=N ... "`.

To run an example, go to the corresponding directory (e.g. `PW/examples/example01`) and execute:

```
./run_example
```

This will create a subdirectory `results/`, containing the input and output files generated by the calculation. Some examples take only a few seconds to run, while others may require up to several minutes.

In each example's directory, the `reference/` subdirectory contains verified output files, that you can check your results against. You may get slightly different reesults on different machines, in particular if different FFT dimensions are automatically selected. For this reason, a plain diff of your results against the reference data doesn't work, or at least, it requires human inspection of the results.

The example scripts stop if an error is detected. You should look *inside* the last written output file to understand why.

2.9 Installation tricks and problems

2.9.1 All architectures

- Working Fortran and C compilers must be present in your PATH. If `configure` says that you have no working compiler, well, you have no working compiler, at least not in your PATH, and not among those recognized by `configure`.
- If you get *Compiler Internal Error* or similar messages: your compiler version is buggy. Try to lower the optimization level, or to remove optimization just for the routine that has problems. If it doesn't work, or if you experience weird problems at run time, try to install patches for your version of the compiler (most vendors release at least a few patches for free), or to upgrade to a more recent compiler version.
- If you get error messages at the loading phase that look like *file XYZ.o: unknown / not recognized/ invalid / wrong file type / file format / module version*, one of the following things have happened:

  1. you have leftover object files from a compilation with another compiler: run `make clean` and recompile.
  2. `make` did not stop at the first compilation error (it may happen in some software configurations). Remove the file *.o that triggers the error message, recompile, look for a compilation error.

  If many symbols are missing in the loading phase: you did not specify the location of all needed libraries (LAPACK, BLAS, FFTW, machine-specific optimized libraries), in the needed order. Note that Quantum ESPRESSO is self-contained (with the exception of MPI libraries for parallel compilation): if system libraries are missing, the problem is in your compiler/library combination or in their usage, not in Quantum ESPRESSO.

- If you get *Segmentation fault* or similar errors in the provided tests and examples: your compiler, or your mathematical libraries, or MPI libraries, or a combination thereof, is buggy, or there is some software incompatibility. Although one can never rule out the presence of subtle bugs in QUANTUM ESPRESSO that are not revealed during the testing phase, it is very unlikely that this happens on the provided tests and examples.
- If all test fails, look into the output and error files: there is some dumb reason for failure.
- If most test pass but some fail, again: look into the output and error files. A frequent source of trouble is complex function `zdotc`. See the "Linux PCs with gfortran compilers" paragraph.

2.9.2 Linux PC

Both AMD and Intel CPUs, 32-bit and 64-bit, are supported and work, either in 32-bit emulation and in 64-bit mode. 64-bit executables can address a much larger memory space than 32-bit executable, but there is no gain in speed. Beware: the default integer type for 64-bit machine is typically 32-bit long. You should be able to use 64-bit integers as well, but it is not guaranteed to work and will not give any advantage anyway.

Currently, `configure` supports gfortran and the Intel (ifort), NVidia (nvfortran, formerly PGI pgf90) compilers. ARM (armflang) and NAG (nagfor) compilers are supported but little tested and may or may not work. Pathscale, Sun Studio, AMD Open64, are no longer supported after v.6.2; g95, since v.6.1.

Intel MKL mathematical libraries are supported. ACML support must be considered as obsolete.

It is usually convenient to create semi-statically linked executables (with only libc, libm, libpthread dynamically linked). If you want to produce a binary that runs on different machines, compile it on the oldest machine you have (i.e. the one with the oldest version of the operating system).

Linux PCs with gfortran Gfortran v.4.8.5, still found on CentOS machines, no longer compiles QUANTUM ESPRESSO v.6.6 or later, due to a gfortran bug. You need at least gfortran v.4.9.X.

"There is a known incompatibility problem between different calling convention for Fortran functions that return complex values [...] If your code crashes during a call to `zdotc`, recompile QUANTUM ESPRESSO using the internal BLAS and LAPACK routines (using `configure` options `--with-internal-blas` and `--with-internal-lapack`) to see if the problem disappears; or, add the `-ff2c` flag" (info by Giovanni Pizzi, Jan. 2013). You may also consider replacing the offending calls to `zdotc` with fortran intrinsic `dot_product`.

If you want to use MKL libraries together with gfortran, link `-lmkl_gf_lp64`, not `-lmkl_intel_lp64`.

Linux PCs with Intel compiler (ifort) The Intel compiler ifort `http://software.intel.com/` produces fast executables, at least on Intel CPUs, but not all versions work as expected. In particular, ifort versions earlier than v.15 miscompile the new XML code in QE v.6.4 and later and should not be used any longer. In case of trouble, update your version with the most recent patches. Since each major release of ifort differs a lot from the previous one, compiled objects from different releases may be incompatible and should not be mixed.

If `configure` doesn't find the compiler, or if you get *Error loading shared libraries* at run time, you may have forgotten to execute the script that sets up the correct PATH and library path. Unless your system manager has done this for you, you should execute the appropriate script – located in the directory containing the compiler executable – in your initialization files. Consult the documentation provided by Intel.

The warning: *feupdateenv is not implemented and will always fail*, can be safely ignored. Complains about "recommended formats" may also be ignored.

Linux PCs with MKL libraries On Intel CPUs it is very convenient to use Intel MKL libraries (freely available at `https://software.intel.com/en-us/performance-libraries`). MKL libraries can be used also with non-Intel compilers. They work also for AMD CPU, selecting the appropriate machine-optimized libraries: see "Linux PCs with AMD processors".

`configure` properly detects only recent (v.12 or later) MKL libraries, as long as the $MKL-ROOT environment variable is set in the current shell. Normally this environment variable is set by sourcing the Intel MKL or Intel Parallel Studio environment script. By default the non-threaded version of MKL is linked, unless option `configure --with-openmp` is specified. In case of trouble, refer to the following web page to find the correct way to link MKL:
`http://software.intel.com/en-us/articles/intel-mkl-link-line-advisor/`.

For parallel (MPI) execution on multiprocessor (SMP) machines, set the environment variable OMP_NUM_THREADS to 1 unless you know how to run MPI+OpenMP. See Sec.3 for more info on this and on the difference between MPI and OpenMP parallelization.

If you get a mysterious "too many communicators" error and a subsequent crash: there is a bug in Intel MPI and MKL 2016 update 3. See this thread and the links quoted therein: `http://www.mail-archive.com/pw_forum@pwscf.org/msg29684.html`.

Linux PCs with AMD processors For AMD CPUs you may find convenient to link AMD acml libraries (can be freely downloaded from AMD web site). `configure` should recognize properly installed acml libraries. UPDATE (February 2020): acml have been discontinued. There are new libraries called AMD Optimizing CPU Libraries (AOCL), tuned for the AMD EPYC processor family. " Recently I played around with some AMD EPYC cpus and the bad thing is that I also saw some strange numbers when using libflame/aocl 2.1. (...) Since version 2020 the MKL performs rather well when using AMD cpus, however, if you want to get the best performance you have to additionally set:

```
export MKL_DEBUG_CPU_TYPE=5
```

which gives an additional 10-20% speedup with MKL 2020, while for earlier versions the speedup is greater than 200%. [...] Another note, there seem to be problems using FFTW interface of MKL with AMD cpus. To get around this problem, one has to additionally set

```
export MKL_CBWR=AUTO
```

" (Info by Tobias Klöffel, Feb. 2020)

2.9.3 Linux PC clusters with MPI

PC clusters running some version of MPI are a very popular computational platform nowadays. Quantum ESPRESSO is known to work with at least two of the major MPI implementations

(MPICH, LAM-MPI), plus with the newer MPICH2 and OpenMPI implementation. `configure` should automatically recognize a properly installed parallel environment and prepare for parallel compilation. Unfortunately this not always happens. In fact:

- `configure` tries to locate a parallel compiler in a logical place with a logical name, but if it has a strange names or it is located in a strange location, you will have to instruct `configure` to find it. If there is no parallel Fortran compiler (e.g., mpif90), you will have to install one.

- `configure` tries to locate libraries (both mathematical and parallel libraries) in the usual places with usual names, but if they have strange names or strange locations, you will have to rename/move them, or to instruct `configure` to find them. If MPI libraries are not found, parallel compilation is disabled.

- `configure` tests that the compiler and the libraries are compatible (i.e. the compiler may link the libraries without conflicts and without missing symbols). If they aren't and the compilation fails, `configure` will revert to serial compilation.

Apart from such problems, QUANTUM ESPRESSO compiles and works on all non-buggy, properly configured hardware and software combinations. In some cases you may have to recompile MPI libraries: not all MPI installations contain support for the Fortran compiler of your choice (or for any Fortran compiler at all).

If QUANTUM ESPRESSO does not work for some reason on a PC cluster, try first if it works in serial execution. A frequent problem with parallel execution is that QUANTUM ESPRESSO does not read from standard input, due to the configuration of MPI libraries: see Sec.3.5. If you are dissatisfied with the performances in parallel execution, see Sec.3 and in particular Sec.3.5.

2.9.4 Microsoft Windows

Since February 2020 QUANTUM ESPRESSO can be compiled on MS-Windows 10 using PGI 19.10 Community Edition (freely downloadable). `configure` works with the bash script provided by PGI but the `configure` of FoX fails: use script `install/build_fox_with_pgi.sh` to manually compile FoX.

Another option: use MinGW/MSYS. Download the installer from `https://osdn.net/projects/mi` install MinGW, MSYS, gcc and gfortran. Start a shell window; run "./configure"; edit `make.inc`; uncommenting the second definition of TOPDIR (the first one introduces a final "/" that Windows doesn't like); run "make". Note that on some Windows the code fails when checking that `tmp_dir` is writable, for unclear reasons.

Another option is Cygwin, a UNIX environment which runs under Windows: see `http://www.cygwin.com/`.

Windows-10 users may also enable the Windows Subsystem for Linux (see here: `https://docs.microsoft.com/en-us/windows/wsl/install-win10`), install a Linux distribution, compile QUANTUM ESPRESSO as on Linux. It works very well.

As a final option, one can use Quantum Mobile: `https://www.materialscloud.org/work/quantum-mobile`.

2.9.5 Mac OS

"I have had some success compiling pw.x on the newish apple hardware. Running run-tests-pw-parallel resulted in all but 3 tests passed (3 unknown). QE6.7 works out of the box:

- Install homebrew
- Using homebrew install gcc (11.2.0), open-mpi (4.1.1_2), fftw3 (3.3.10), and veclibfort (0.4.2_7)

To configure QE:

```
./configure FC=mpif90 CC=mpicc CPP=cpp-11 BLAS_LIBS="-L/opt/homebrew/lib
           -lveclibfort" LIBDIRS=/opt/homebrew/lib
```

Current develop branch needed two changes:

1. The script external/devxlib/config/config.sub is outdated, and needs to be adjusted to correctly parse the machine information. I pulled a more up-to-date version from iains/gcc-darwin-arm64 github repo

2. PW/src/efermig.f90 needed to be compiled without optimization -O0. No idea why at the moment."

(Info by John Vinson, NIST)

Mac OS-X machines with gfortran or with the Intel compiler ifort and MKL libraries should work, but "your mileage may vary", depending upon the specific software stack you are using. Parallel compilation with OpenMPI should also work.

If you get an error like

```
clang: error: no input files
make[1]: *** [laxlib.fh] Error 1
make: *** [libla] Error 1i
```

redefine `CPP` as `CPP=gcc -E` in `make.inc`.

Gfortran information and binaries for Mac OS-X here: `http://hpc.sourceforge.net/` and `https://wiki.helsinki.fi/display/HUGG/GNU+compiler+install+on+Mac+OS+X`.

Mysterious crashes in `zdotc` are due to a known incompatibility of complex functions with some optimized BLAS. See the "Linux PCs with gfortran" paragraph.

2.9.6 Cray machines

"... despite what people can imagine, every CRAY machine deployed can have different environment. For example on the machine I usually use for tests [...] I do have to unload some modules to make QE running properly. On another CRAY [...] there is also Intel compiler as option and the system is slightly different compared to the other." (info by Filippo Spiga)

For recent Cray machines, use `./configure ARCH=craype`. This selects the `ftn` compiler, that typically uses the `crayftn` compiler but may also use other ones, depending upon the site and personal environment. `ftn` v.15.0.1 compiles QE properly. With the `PrgEnv-cray` module v.6.0.10, `ftn` v.14.0.3, you run into the following problems:

- internal compiler error in `esm_stres_mod.f90`;
- crashes when writing the final xml data file.

Workaround: compile codes `esm_stres_mod.f90`, `Modules/qexsd*.f90`, `PW/src/pw_restart_new.f90` with reduced optimization, using -O0 or -O1 instead of the default -O3,fp3 optimization.

If you want to use the Intel compiler instead, try something like:

```
$ module swap PrgEnv-cray PrgEnv-intel
$ ./configure ARCH=craype [--enable-openmp --enable-parallel --with-scalapack]
```

Old Cray machines: T3D, T3E, X1, etc, are no longer supported.

3 Parallelism

3.1 Understanding Parallelism

Two different parallelization paradigms are currently implemented in QUANTUM ESPRESSO:

1. *Message-Passing (MPI).* A copy of the executable runs on each CPU; each copy lives in a different world, with its own private set of data, and communicates with other executables only via calls to MPI libraries. MPI parallelization requires compilation for parallel execution, linking with MPI libraries, execution using a launcher program (depending upon the specific machine). The number of CPUs used is specified at run-time either as an option to the launcher or by the batch queue system.

2. *OpenMP.* A single executable spawn subprocesses (threads) that perform in parallel specific tasks. OpenMP can be implemented via compiler directives (*explicit* OpenMP) or via *multithreading* libraries (*library* OpenMP). Explicit OpenMP require compilation for OpenMP execution; library OpenMP requires only linking to a multithreading version of the mathematical libraries. The number of threads is specified at run-time in the environment variable OMP_NUM_THREADS.

MPI is the well-established, general-purpose parallelization. In QUANTUM ESPRESSO several parallelization levels, specified at run-time via command-line options to the executable, are implemented with MPI. This is your first choice for execution on a parallel machine.

The support for explicit OpenMP is steadily improving. Explicit OpenMP can be used together with MPI and also together with library OpenMP. Beware conflicts between the various kinds of parallelization! If you don't know how to run MPI processes and OpenMP threads in a controlled manner, forget about mixed OpenMP-MPI parallelization.

3.2 Running on parallel machines

Parallel execution is strongly system- and installation-dependent. Typically one has to specify:

1. a launcher program such as `mpirun` or `mpiexec`, with the appropriate options (if any);

2. the number of processors, typically as an option to the launcher program;

3. the program to be executed, with the proper path if needed;

4. other QUANTUM ESPRESSO-specific parallelization options, to be read and interpreted by the running code.

Items 1) and 2) are machine- and installation-dependent, and may be different for interactive and batch execution. Note that large parallel machines are often configured so as to disallow interactive execution: if in doubt, ask your system administrator. Item 3) also depend on your specific configuration (shell, execution path, etc). Item 4) is optional but it is very important for good performances. We refer to the next section for a description of the various possibilities.

3.3 Parallelization levels

In QUANTUM ESPRESSO several MPI parallelization levels are implemented, in which both calculations and data structures are distributed across processors. Processors are organized in a hierarchy of groups, which are identified by different MPI communicators level. The groups hierarchy is as follow:

- **world**: is the group of all processors (MPI_COMM_WORLD).
- **images**: Processors can then be divided into different "images", each corresponding to a different self-consistent or linear-response calculation, loosely coupled to others.
- **pools**: each image can be subpartitioned into "pools", each taking care of a group of k-points.
- **bands**: each pool is subpartitioned into "band groups", each taking care of a group of Kohn-Sham orbitals (also called bands, or wavefunctions). Especially useful for calculations with hybrid functionals.
- **PW**: orbitals in the PW basis set, as well as charges and density in either reciprocal or real space, are distributed across processors. This is usually referred to as "PW parallelization". All linear-algebra operations on array of PW / real-space grids are automatically and effectively parallelized. 3D FFT is used to transform electronic wave functions from reciprocal to real space and vice versa. The 3D FFT is parallelized by distributing planes of the 3D grid in real space to processors (in reciprocal space, it is columns of G-vectors that are distributed to processors).
- **tasks**: In order to allow good parallelization of the 3D FFT when the number of processors exceeds the number of FFT planes, FFTs on Kohn-Sham states are redistributed to "task" groups so that each group can process several wavefunctions at the same time. Alternatively, when this is not possible, a further subdivision of FFT planes is performed.
- **linear-algebra group**: A further level of parallelization, independent on PW or k-point parallelization, is the parallelization of subspace diagonalization / iterative orthonormalization. Both operations required the diagonalization of arrays whose dimension is the number of Kohn-Sham states (or a small multiple of it). All such arrays are distributed block-like across the "linear-algebra group", a subgroup of the pool of processors, organized in a square 2D grid. As a consequence the number of processors in the linear-algebra group is given by n^2, where n is an integer; n^2 must be smaller than the number of processors in the PW group. The diagonalization is then performed in parallel using standard linear algebra operations. (This diagonalization is used by, but should not be confused with, the iterative Davidson algorithm). The preferred option is to use ELPA and ScaLAPACK; alternative built-in algorithms are anyway available.

Note however that not all parallelization levels are implemented in all codes.

When a communicator is split, the MPI process IDs in each sub-communicator remain ordered. So for instance, for two images and $2n$ MPI processes, image 0 contains IDs $0, 1, ..., n-1$, image 1 contains IDs $n, n+1, .., 2n-1$.

About communications Images and pools are loosely coupled: inter-processors communication between different images and pools is modest. Processors within each pool are instead tightly coupled and communications are significant. This means that fast communication hardware is needed if your pool extends over more than a few processors on different nodes.

Choosing parameters : To control the number of processors in each group, command line switches: `-nimage`, `-npools`, `-nband`, `-ntg`, `-ndiag` or `-northo` (shorthands, respectively: `-ni`, `-nk`, `-nb`, `-nt`, `-nd`) are used. As an example consider the following command line:

```
mpirun -np 4096 ./neb.x -ni 8 -nk 2 -nt 4 -nd 144 -i my.input
```

This executes a NEB calculation on 4096 processors, 8 images (points in the configuration space in this case) at the same time, each of which is distributed across 512 processors. k-points are distributed across 2 pools of 256 processors each, 3D FFT is performed using 4 task groups (64 processors each, so the 3D real-space grid is cut into 64 slices), and the diagonalization of the subspace Hamiltonian is distributed to a square grid of 144 processors (12x12).

Default values are: `-ni 1 -nk 1 -nt 1` ; `nd` is set to 1 if ScaLAPACK is not compiled, it is set to the square integer smaller than or equal to the number of processors of each pool.

Massively parallel calculations For very large jobs (i.e. O(1000) atoms or more) or for very long jobs, to be run on massively parallel machines (e.g. IBM BlueGene) it is crucial to use in an effective way all available parallelization levels: on linear algebra (requires compilation with ELPA and/or ScaLAPACK), on "task groups" (requires run-time option "-nt N"), and mixed MPI-OpenMP (requires OpenMP compilation: `configure`–enable-openmp). Without a judicious choice of parameters, large jobs will find a stumbling block in either memory or CPU requirements. Note that I/O may also become a limiting factor.

3.4 Understanding parallel I/O

In parallel execution, each processor has its own slice of data (Kohn-Sham orbitals, charge density, etc), that have to be written to temporary files during the calculation, or to data files at the end of the calculation. This can be done in two different ways:

- "collected": all slices are collected by the code to a single processor that writes them to disk, in a single file, using a format that doesn't depend upon the number of processors or their distribution. This is the default since v.6.2 for final data.
- "portable": as above, but data can be copied to and read from a different machines (this is not guaranteed with Fortran binary files). Requires compilation with `-D__HDF5` preprocessing option and HDF5 libraries.

There is a third format, no longer used for final data but used for scratch and restart files:

- "distributed": each processor writes its own slice to disk in its internal format to a different file. The "distributed" format is fast and simple, but the data so produced is readable only by a job running on the same number of processors, with the same type of parallelization, as the job who wrote the data, and if all files are on a file system that is visible to all processors (i.e., you cannot use local scratch directories: there is presently no way to ensure that the distribution of processes across processors will follow the same pattern for different jobs).

The directory for data is specified in input variables `outdir` and `prefix` (the former can be specified as well in environment variable ESPRESSO_TMPDIR): `outdir/prefix.save`. A copy of pseudopotential files is also written there. If some processor cannot access the data directory, the pseudopotential files are read instead from the pseudopotential directory specified in input data. Unpredictable results may follow if those files are not the same as those in the data directory!

IMPORTANT: Avoid I/O to network-mounted disks (via NFS) as much as you can! Ideally the scratch directory `outdir` should be a modern Parallel File System. If you do not have any, you can use local scratch disks (i.e. each node is physically connected to a disk and writes to it) but you may run into trouble anyway if you need to access your files that are scattered in an unpredictable way across disks residing on different nodes.

You can use input variable `disk_io` to vary the amount of I/O done by `pw.x`. Since v.5.1, the default value is `disk_io='low'`, so the code will store wavefunctions into RAM and not on disk during the calculation. Specify `disk_io='medium'` only if you have too many k-points and you run into trouble with memory; choose `disk_io='none'` if you do not need to keep final data files.

3.5 Tricks and problems

Many problems in parallel execution derive from the mixup of different MPI libraries and runtime environments. There are two major MPI implementations, OpenMPI and MPICH, coming in various versions, not necessarily compatible; plus vendor-specific implementations (e.g. Intel MPI). A parallel machine may have multiple parallel compilers (typically, `mpif90` scripts calling different serial compilers), multiple MPI libraries, multiple launchers for parallel codes (different versions of `mpirun` and/or `mpiexec`). You have to figure out the proper combination of all of the above, which may require using command `module` or manually setting environment variables and execution paths. What exactly has to be done depends upon the configuration of your machine. You should inquire with your system administrator or user support (if available; if not, YOU are the system administrator and user support and YOU have to solve your problems).

Always verify if your executable is actually compiled for parallel execution or not: it is declared in the first lines of output. Running several instances of a serial code with `mpirun` or `mpiexec` produces strange crashes.

Trouble with input files Input files should be plain ASCII text. The presence of CRLF line terminators (may appear as ^M, Control-M, characters at the end of lines), tabulators, or non-ASCII characters (e.g. non-ASCII quotation marks, that at a first glance may look the same as the ASCII character) is a frequent source of trouble. Typically, this happens with files coming from Windows or produced with "smart" editors. Verify with command `file` and convert with command `iconv` if needed.

Some implementations of the MPI library have problems with input redirection in parallel. This typically shows up under the form of mysterious errors when reading data. If this happens, use the option `-i` (or `-in`, `-inp`, `-input`), followed by the input file name. Example:

```
pw.x -i inputfile -nk 4 > outputfile
```

Of course the input file must be accessible by the processor that must read it (only one processor reads the input file and subsequently broadcasts its contents to all other processors).

Apparently the LSF implementation of MPI libraries manages to ignore or to confuse even the `-i/in/inp/input` mechanism that is present in all QUANTUM ESPRESSO codes. In this case, use the `-i` option of `mpirun.lsf` to provide an input file.

Trouble with MKL and MPI parallelization If you notice very bad parallel performances with MPI and MKL libraries, it is very likely that the OpenMP parallelization performed by the latter is colliding with MPI. Recent versions of MKL enable autoparallelization by default on multicore machines. You must set the environment variable OMP_NUM_THREADS to 1 to disable it. Note that if for some reason the correct setting of variable OMP_NUM_THREADS does not propagate to all processors, you may equally run into trouble. Lorenzo Paulatto (Nov. 2008) suggests to use the `-x` option to `mpirun` to propagate OMP_NUM_THREADS to all processors. Axel Kohlmeyer suggests the following (April 2008): "(I've) found that Intel is now turning on multithreading without any warning and that is for example why their FFT seems faster than FFTW. For serial and OpenMP based runs this makes no difference (in fact the multi-threaded FFT helps), but if you run MPI locally, you actually lose performance. Also if you use the 'numactl' tool on linux to bind a job to a specific cpu core, MKL will still try to use all available cores (and slow down badly). The cleanest way of avoiding this mess is to either link with

`-lmkl_intel_lp64 -lmkl_sequential -lmkl_core` (on 64-bit: x86_64, ia64)
`-lmkl_intel -lmkl_sequential -lmkl_core` (on 32-bit, i.e. ia32)

or edit the `libmkl_'platform'.a` file. I'm using now a file `libmkl10.a` with:

```
GROUP (libmkl_intel_lp64.a libmkl_sequential.a libmkl_core.a)
```

It works like a charm". UPDATE: Since v.4.2, `configure` links by default MKL without multithreaded support.

Trouble with compilers and MPI libraries Many users of QUANTUM ESPRESSO, in particular those working on PC clusters, have to rely on themselves (or on less-than-adequate system managers) for the correct configuration of software for parallel execution. Mysterious and irreproducible crashes in parallel execution are sometimes due to bugs in QUANTUM ESPRESSO, but more often than not are a consequence of buggy compilers or of buggy or miscompiled MPI libraries.