User's Guide for Quantum ESPRESSO (v.6.3)

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 6.3.

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notice: due to the lack of time and of manpower, this manual does not cover many important aspects, may contain outdated information.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

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 also included. 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.
- `PLUMED` (v.1.3 only): calculation of free-energy surface through metadynamics.

- `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). A GPU-enabled version is available on `https://github.com/fspiga/qe-gpu`.

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 Developer Manual: `Doc/developer_man.pdf`.

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.

All trademarks mentioned in this guide belong to their respective owners.

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, Italy) with the strong support of the CINECA National Supercomputing Center in Bologna under the responsibility of Carlo Cavazzoni.

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

- Ye Luo (Argonne) for improved FFT threading and miscellaneous contributions and optimization;
- Pietro Bonfà (CINECA) for multiple contributions to optimization and maintenance;
- Hsin-Yu Ho and Marcos Calegari Andrade (U.Princeton) for SCAN and other meta-GGA functionals with `libxc`;

- Fabio Affinito (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 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, UK) for mixed MPI-OpenMP parallelization and for the 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, Samuel Poncé, 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 web site for QUANTUM ESPRESSO is `http://www.quantum-espresso.org/`. Releases and patches can be downloaded from this site or following the links contained in it. The main entry point for developers is the GitLab web site: `https://gitlab.com/QEF/q-e`.

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 read the guidelines for posting, section 1.3! NOTA BENE: only messages that appear to come from the registered user's e-mail address, in its *exact form*, will be accepted. Messages "waiting for moderator approval" are automatically deleted with no further processing (sorry, too much spam). In case of trouble, carefully check that your return e-mail is the correct one (i.e. the one you used to subscribe).

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 use "reply" to start a new thread: 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: point a linker to a place where the attachment(s) can be downloaded from, such as e.g. DropBox, Google Docs, or one of the various web temporary storage spaces (e.g.: ExpireBox, File.Town, Uguu.se).
- 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)

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 are also available. Stable and development releases of the QUANTUM ESPRESSO source package (current version is 6.3), as well as available binary packages, can be downloaded from the links listed in the "Download" section of `www.quantum-espresso.org`.

Uncompress and unpack the base distribution 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. If your version of `tar` doesn't recognize the "z" flag:

```
gunzip -c qe-X.Y.Z.tar.gz | tar xvf -
```

A directory `qe-X.Y.Z/` will be created.

Additional packages that are not included in the base distribution will be downloaded on demand at compile time, using `make` (see Sec.2.5). 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
`include/`	files *.h included by fortran and C source files
`clib/`	libraries and utilities written in C
`FFTXlib/`	FFT libraries
`LAXlib/`	Linear Algebra (parallel) libraries
`KS_Solvers/`	Iterative diagonalization routines
`UtilXlib/`	Miscellaneous timing, error handling, MPI utilites
`install/`	installation scripts and utilities
`pseudo/`	pseudopotential files used by examples
`upftools/`	converters to unified pseudopotential format (UPF)
`Doc/`	general documentation
`archive/`	external libraries in .tar.gz form
`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

Finally, directory `COUPLE/` contains code and documentation that is useful to call QUANTUM ESPRESSO programs from external codes; directory `LR_Modules/` contains source files for modules that are common to all linear-response 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`. For MS-Windows, see Sec.2.7.7.

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 C and Fortran compilers, compliant with C89 and F2003 standards. 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.

As a rule, QUANTUM ESPRESSO tries to keep compatibility with older compilers, avoiding nonstandard extensions and newer features that are not widespread or stabilized. No warranty, however, if your compiler is older than ~ 5 years or so. The same applies to mathematical and MPI libraries.

Big machines with specialized hardware (e.g. IBM SP, CRAY, etc) typically have a Fortran compiler with MPI and OpenMP libraries bundled with the software. Workstations or "commodity" machines, using PC hardware, may or may not have the needed software. If not, you need either to buy a commercial product (e.g Intel, NAG, PGI) or to use an open-source compiler like gfortran from the gcc distribution. Some commercial compilers (e.g. PGI) may be available free of charge under some conditions (e.g. academic or personal usage).

2.3 `configure`

To install the QUANTUM ESPRESSO source package, 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 is expected to work on most Linux 32- and 64-bit PCs (all Intel and AMD CPUs) and PC clusters, IBM BlueGene machines, NEC SX, Cray XT machines, Mac OS X, MS-Windows PCs. 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. Additional instructions for special machines:

`./configure ARCH=crayxt4`	for CRAY XT machines
`./configure ARCH=necsx`	for NEC SX machines
`./configure ARCH=ppc64-mn`	PowerPC Linux + xlf (Marenostrum)
`./configure ARCH=ppc64-bg`	IBM BG/P (BlueGene)

`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/c_defs.h`	a few definitions used by C files
`include/configure.h`	info on compilation flags (not used: in Modules/environment.f90 uncomment `#define __HAVE_CONFIG_INFO` to enable its usage)

NOTA BENE: `configure` no longer updates dependencies. If you modify the sources, run `./install/makedeps.sh` or type `make depend` to update files `make.depend` in the various subdirectories.
NOTA BENE 2: `make.inc` used to be called `make.sys` until v.6.0. The change of name is due to frequent problems with mailers assuming that whatever ends in `.sys` is a suspect virus.

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.2.4 for details and hints.

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
```

Some environment variables that are relevant to `configure` are:

`ARCH`	label identifying the machine type (see below)
`F90, F77, CC`	names of Fortran, Fortran-77, 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 2003 – 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: do not pass compiler names with the leading path included. `F90=f90xyz` is ok, `F90=/path/to/f90xyz` is not. Do not use environment variables with `configure` unless they are needed! try `configure` with no options as a first step.

If your machine type is unknown to `configure`, you may use the `ARCH` variable to suggest an architecture among supported ones. Some large parallel machines using a front-end (e.g. Cray XT) will 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:

`ia32`	Intel 32-bit machines (x86) running Linux
`ia64`	Intel 64-bit (Itanium) running Linux
`x86_64`	Intel and AMD 64-bit running Linux - see note below
`crayxt4`	Cray XT4/XT5/XE machines
`mac686`	Apple Intel machines running Mac OS X
`cygwin`	MS-Windows PCs with Cygwin
`mingw32`	Cross-compilation for MS-Windows, using mingw, 32 bits
`mingw64`	As above, 64 bits
`necsx`	NEC SX-6 and SX-8 machines
`ppc64`	Linux PowerPC machines, 64 bits
`ppc64-mn`	as above, with IBM xlf compiler
`ppc64-bg`	IBM BlueGene
`arm`	ARM machines (with gfortran)

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. Finally, `configure` recognizes the following command-line options:

`--enable-parallel`	compile for parallel (MPI) execution if possible (default: yes)
`--enable-openmp`	compile for OpenMP execution if possible (default: no)
`--enable-shared`	use shared libraries if available (default: yes; "no" is implemented, untested, in only a few cases)
`--enable-debug`	compile with debug flags (only for selected cases; default: no)
`--enable-signals`	enable signal trapping (default: disabled)

and the following optional packages:

`--with-internal-blas`	compile with internal BLAS (default: no)
`--with-internal-lapack`	compile with internal LAPACK (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 (default: no)
`--with-elpa-lib`	Specify full path of the ELPA library (default: no)
`--with-elpa-version`	Specify ELPA version, only year (2015 or 2016, default: 2016)
`--with-hdf5`	(no | `<path>`) Write portable binary files using HDF5. A valid `<path>` for HDF5 library must be specified (default: no)

The following options are available for the CUDA Fortran accelerated version (currently in a separate package, not yet included in the main distribution):

`--with-cuda=value`	enables compilation of the CUDA Fortran 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. PGI compilers currently accept 7.5, 8.0 or 9.0 (default: 8.0)

Please note that in order to compile the CUDA Fortran code you need ... the CUDA Fortran code (it is not yet available in the main distribution)! you also need a recent version of the PGI Fortran compilers (at least 17.4). OpenMP must be enabled, and you may want to use a CUDA-aware MPI distribution to optimize the data transfer between the processes.

If you want to modify `configure` (advanced users only!), see the Developer Manual.

2.3.1 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/c_defs.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`.

NOTA BENE: 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. `configure` will clean object and executables, unless you use option `--save`.

2.4 Libraries

Quantum ESPRESSO makes use of the following external libraries:

- 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

A copy of the needed routines is provided with the distribution. However, when available, optimized vendor-specific libraries should be used: this often yields huge performance gains.

BLAS and LAPACK QUANTUM ESPRESSO can use any architecture-optimized BLAS and LAPACK replacements, like those contained e.g. in the following libraries:

MKL for Intel CPUs
ACML for AMD CPUs
ESSL for IBM machines

If none of these is available, we suggest that you use the optimized ATLAS library: see `http://math-atlas.sourceforge.net/`. 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 QUANTUM ESPRESSO has an internal 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,
- `-D__ARM_LIB` for ARM Performance library,
- `-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.2.7.6.

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 such as `cublas, cufft, cusolver` and the eigensolver library explicitly developed for QUANTUM ESPRESSO by NVidia and distributed at `https://github.com/NVIDIA/Eigensolver_gpu`.

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

The user may need to install this library, compiling it with options `--enable-fortran`, `--enable-fortran-2003`, and `--enable-parallel` (see below). These options must be passed to the `configure` script of the library, not of QUANTUM ESPRESSO.

One can use either the `1.10` or `1.8` version of the library. For the latter the user has to download a version at least as new as `1.8.16`.

The path to the root directory of the library (the one containing `bin/`, `include/` and `lib/` directories) has to be passed to the `configure` script of QUANTUM ESPRESSO via the `--with-hdf5=...` option.

It is possible to use a library with disabled parallelism, but one has to add manually the flag `-D__HDF5_SERIAL` to the `MANUAL_DFLAGS` in the `make.inc` file.

The HDF5 packages provided by many LINUX distributions may also work, but the configure script fails if includes and libraries are not placed under the same root directory. In this case the user should manually set the correct paths in the `make.inc` file.

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.

2.5 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. `make` with no arguments yields a list of valid compilation targets:

- `make pw` compiles the self-consistent-field package `PWscf`
- `make cp` compiles the Car-Parrinello package `CP`
- `make neb` compiles the `PWneb` package. All executables are linked in main `bin` directory
- `make ph` compiles the `PHonon` package. All executables are linked in main `bin` directory
- `make pp` compiles the postprocessing package `PostProc`
- `make pwcond` compiles the ballistic conductance package `PWcond`. All executables are linked in main `bin` directory
- `make pwall` produces all of the above.
- `make ld1` compiles the pseudopotential generator package `atomic`. All executables are linked in main `bin` directory
- `make xspectra` compiles the package `XSpectra`. All executables are linked in main `bin` directory
- `make upf` produces utilities for pseudopotential conversion in directory `upftools/`
- `make all` produces all of the above
- `make epw` compiles package `EPW`
- `make plumed` unpacks `PLUMED`, patches several routines in `PW/`, `CPV/` and `clib/`, recompiles `PWscf` and `CP` with `PLUMED` support
- `make w90` downloads `wannier90`, unpacks it, copies an appropriate `make.inc` file, produces all executables in `W90/wannier90.x` and in `bin/`
- `make want` downloads `WanT`, unpacks it, runs its `configure`, produces all executables for `WanT` in `WANT/bin`.
- `make yambo` downloads `yambo`, unpacks it, runs its `configure`, produces all `yambo` executables in `YAMBO/bin`
- `make gipaw` downloads `GIPAW`, unpacks it, runs its `configure`, produces all `GIPAW` executables in `GIPAW/bin` and in main `bin` directory.

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).

2.6 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.

2.6.1 Test-suite

Automated tests give a "pass/fail" answer. All tests run quickly (less than a minute 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`. 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).

2.6.2 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):

```
poe pw.x -procs 4 -i file.in > file.out
```

you should set `PARA_PREFIX="poe"`, `PARA_POSTFIX="-procs 4"`. 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. They were generated on a Linux PC using the Intel compiler. On different architectures the precise numbers could be slightly different, 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.

2.7 Installation tricks and problems

2.7.1 All architectures

- Working Fortran and C compilers, compliant with F2003 and C89 standards respectively, are needed in order to compile QUANTUM ESPRESSO. Most recent Fortran compilers will do the job.

  C and Fortran compilers must be 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 mysterious *Segmentation fault* and the like errors in the provided tests and examples: your compiler, or your mathematical libraries, or MPI libraries, or a combination thereof, is very likely buggy, or there is some form of incompatibility (see below). 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.

2.7.2 Intel Xeon Phi

For Intel Xeon CPUs with Phi coprocessor, there are three ways of compiling:

- *offload* mode, executed on main CPU and offloaded onto coprocessor "automagically";
- *native* mode, executed completely on coprocessor;
- *symmetric* mode, requiring creation of both binaries.

"You can take advantage of the offload mode using the `libxphi` library. This library offloads the BLAS/MKL functions on the Xeon Phi platform hiding the latency times due to the communication. You just need to compile this library and then to link it dynamically. The library works with any version of QE. Libxphi is available from `https://github.com/cdahnken/libxphi`. Some documentation is available therein.

Instead, if you want to compile a native version of QE, you just need to add the `-mmic` flag and cross compile. If you want to use the symmetric mode, you need to compile twice: with and without the `-mmic` flag". "[...] everything, i.e. code+libraries, must be cross-compiled with the `-mmic` flag. In my opinion, it's pretty unlikely that native mode can outperform the execution on the standard Xeon cpu. I strongly suggest to use the Xeon Phi in offload mode, for now" (info by Fabio Affinito, March 2015).

2.7.3 Cray machines

(This section is likely obsolete)

For Cray XE machines:

```
$ module swap PrgEnv-cray PrgEnv-pgi
$ ./configure --enable-openmp --enable-parallel --with-scalapack
$ vim make.inc
```

then manually add `-D__IOTK_WORKAROUND1` at the end of `DFLAGS` line.

"Now, 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. So my recipe should work, 99% of the cases." (info by Filippo Spiga)

For Cray XT machines, use `./configure ARCH=crayxt4` or else `configure` will not recognize the Cray-specific software environment.

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

2.7.4 IBM BlueGene

The current `configure` was working on the machines at CINECA and at Jülich. For other machines, you may need something like

```
./configure ARCH=ppc64-bg BLAS_LIBS=...  LAPACK_LIBS=... \
            SCALAPACK_DIR=... BLACS_DIR=..."
```

where the various *_LIBS and *_DIR "suggest" where the various libraries are located.

2.7.5 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 Intel (ifort), NAG (nagfor), PGI (pgf90) and gfortran compilers. Pathscale, Sun Studio, AMD Open64, are no longer supported after v.6.2: g95, since v.6.1.

Both Intel MKL and AMD acml mathematical libraries are supported, the former much better than the latter.

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 You need at least gfortran v.4.4 or later to properly compile QUANTUM ESPRESSO.

"There is a known incompatibility problem between the calling convention for Fortran functions that return complex values: there is the convention used by g77/f2c, where in practice the compiler converts such functions to subroutines with a further parameter for the return value; gfortran instead produces a normal function returning a complex value. If your system libraries were compiled using g77 (which may happen for system-provided libraries in not-too-recent Linux distributions), and you instead use gfortran to compile QUANTUM ESPRESSO, your code may crash or produce random results. This typically happens during calls to `zdotc`, which is one the most commonly used complex-returning functions of BLAS+LAPACK.

For further details see for instance this link:
`http://www.macresearch.org/lapackblas-fortran-106#comment-17071`
or read the man page of gfortran under the flag `-ff2c`.

If your code crashes during a call to `zdotc`, try to recompile QUANTUM ESPRESSO using the internal BLAS and LAPACK routines (using the `--with-internal-blas` and `--with-internal-la` parameters of the configure script) to see if the problem disappears; or, add the `-ff2c` flag" (info by Giovanni Pizzi, Jan. 2013).

Note that a similar problem with complex functions exists with MKL libraries as well: if you compile with gfortran, link `-lmkl_gf_lp64`, not `-lmkl_intel_lp64`, and the like for other architectures. Since v.5.1, you may use the following workaround: add preprocessing option `-Dzdotc=zdotc_wrapper` to `DFLAGS`.

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 (see below). 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.

The Intel compiler is no longer free for personal usage, but it is still for students and open-source contributors (`https://software.intel.com/en-us/qualify-for-free-software`).

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. Warnings on "bad preprocessing option" when compiling iotk and complains about "recommended formats" may also be ignored.

The following compiler releases are known to give segmentation faults in at least some cases of compilation of QUANTUM ESPRESSO v.6.0:

12.0.0.084 Build 20101006
12.0.1.107 Build 20101116
12.0.2.137 Build 20110112
12.0.4.191 Build 20110427
12.0.5.220 Build 20110719
16.0.1.150 Build 20151021

(Filippo Spiga, Aug. 2016)

ifort v.12: release 12.0.0 miscompiles iotk, leading to mysterious errors when reading data files. Workaround: increase the parameter BLOCKSIZE to e.g. 131072*1024 when opening files in `iotk/src/iotk_files.f90` (info by Lorenzo Paulatto, Nov. 2010).

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, but with reduced performances.

`configure` properly detects only recent (v.12 or later) MKL libraries, as long as the $MKLROOT 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 what you are doing. 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 ACML libraries For AMD CPUs, especially recent ones, you may find convenient to link AMD acml libraries (can be freely downloaded from AMD web site). `configure` should recognize properly installed acml libraries.

2.7.6 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. Note that in many PC clusters (Beowulf), there is no parallel Fortran compiler in default installations: you have to configure an appropriate script, such as mpif90.

- `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.3.4. If you are dissatisfied with the performances in parallel execution, see Sec.3 and in particular Sec.3.4.

2.7.7 Microsoft Windows

MS-Windows users may compile Quantum ESPRESSO on MinGW/MSYS. Download the installer from `https://osdn.net/projects/mingw/`, 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/`.

Finally, Windows-10 users may 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.

2.7.8 Mac OS

Mac OS-X machines (10.4 and later) with Intel CPUs are supported by `configure`, both with gfortran and with the Intel compiler ifort and MKL libraries. Parallel compilation with OpenMPI also works.

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, occurring when `zdotc` is called, are due to the same incompatibility of complex functions with some optimized BLAS as reported in the "Linux PCs with gfortran" paragraph. Workaround: add preprocessing option `-Dzdotc=zdotc_wrapper` to `DFLAGS`.

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 mathematical libraries, e.g.: ESSLSMP, ACML_MP, MKL (the latter is natively multithreading). 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.

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 half 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.

3.3.1 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.
- "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 distributed format for final data requires compilation with `-D__OLDXML` preprocessing option and will be removed in the next releases.

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 reduce the the amount of I/O done by `pw.x`. Since v.5.1, the dafault 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.

For very large `cp.x` runs, you may consider using `wf_collect=.false.`, `memory='small'` and `saverho=.false.` to reduce I/O to the strict minimum.

3.4 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 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.