User's Guide for Quantum ESPRESSO (version 4.1)

Contents

1 Introduction

This guide covers the installation and usage of QUANTUM ESPRESSO (opEn-Source Package for Research in Electronic Structure, Simulation, and Optimization), version 4.1.

The QUANTUM ESPRESSO distribution contains the following core packages for the calculation of electronic-structure properties within Density-Functional Theory, using a Plane-Wave basis set and pseudopotentials:

- PWscf (Plane-Wave Self-Consistent Field),
- CP (Car-Parrinello).

It also includes the following more specialized packages:

- PHonon: phonons with Density-Functional Perturbation Theory,
- PostProc: various utilities for data prostprocessing,
- PWcond: ballistic conductance,
- GIPAW (Gauge-Independent Projector Augmented Waves): EPR g-tensor and NMR chemical shifts,
- XSPECTRA: K-edge X-ray adsorption spectra,
- vdW: (experimental) dynamic polarizability,
- Wannier90: maximally localized Wannier functions.

Finally, the following auxiliary codes are included:

- PWgui (Graphical User Interface for PWscf): a graphical interface for producing input data files for PWscf,
- atomic: a program for atomic calculations and generation of pseudopotentials,
- iotk: an Input-Output ToolKit.

This guide documents PWscf, CP, PHonon, PostProc, PWcond. The remaining packages have separate documentation.

The QUANTUM ESPRESSO codes work on many different types of Unix machines, including parallel machines using Message Passing Interface (MPI). Running QUANTUM ESPRESSO on Mac OS X and MS-Windows is also possible: see section 2.2.

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

- the QUANTUM ESPRESSO Wiki (http://www.quantum-espresso.org/wiki/index.php/Main_Page) ;
- the Doc/ directory of the QUANTUM ESPRESSO distribution, containing a detailed description of all input data for all codes in the INPUT_* files (in .txt, .html, .pdf formats);
- the pw_forum mailing list (pw_forum@pwscf.org). You can subscribe to this list and browse and search its archives from the QUANTUM ESPRESSO web site (http://www.quantum-espresso.org/tools.php). Only subscribed users can post. Please search the archives before posting: your question may have already been answered.

This guide does not explain solid state physics and its computational methods. If you want to learn that, you should read a good textbook, such as e.g. the book by Richard Martin: *Electronic Structure: Basic Theory and Practical Methods*, Cambridge University Press (2004).

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

1.1 What can QUANTUM ESPRESSO do

PWscf can currently perform the following kinds of calculations:

- ground-state energy and one-electron (Kohn-Sham) orbitals
- atomic forces, stresses, and structural optimization
- molecular dynamics on the ground-state Born-Oppenheimer surface, also with variable cell
- Nudged Elastic Band (NEB) and Fourier String Method Dynamics (SMD) for energy barriers and reaction paths
- macroscopic polarization and finite electric fields via the modern theory of polarization (Berry Phases)

All of the above works for both insulators and metals, in any crystal structure, for many exchange-correlation functionals (including spin polarization, DFT+U, exact exchange), for norm-conserving (Hamann-Schluter-Chiang) pseudopotentials in separable form or Ultrasoft (Vanderbilt) pseudopotentials or Projector Augmented Waves (PAW) method. Non-collinear magnetism and spin-orbit interactions are also implemented. Finite electric fields are implemented also using a supercell approach.

PHonon can perform the following types of calculations:

- phonon frequencies and eigenvectors at a generic wave vector, using Density-Functional Perturbation Theory
- effective charges and dielectric tensors
- electron-phonon interaction coefficients for metals
- interatomic force constants in real space
- third-order anharmonic phonon lifetimes
- Infrared and Raman (nonresonant) cross section

PHonon can be used whenever PWscf can be used, with the exceptions of DFT+U and exact exchange. PAW is not implemented for higher-order response calculations. Furtrher calculations, in the Quasi-harmonic approximations, of the vibrational free energy can be performed using the QHA package.

PostProc can perform the following types of calculations:

- Scanning Tunneling Microscopy (STM) images;
- plots of Electron Localization Functions (ELF);
- Density of States (DOS) and Projected DOS (PDOS);
- Löwdin charges;
- planar and spherical averages;

plus interfacing with a number of graphical utilities and with external codes.

1.2 People

In the following, the cited affiliation is the one where the last known contribution was done and may no longer be valid.

The maintenance and further development of the QUANTUM ESPRESSO distribution is promoted by the DEMOCRITOS National Simulation Center of CNR-INFM (Italian Institute for Condensed Matter Physics) 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. Layla Martin-Samos (Democritos) is joining the team of coordinators.

The PWscf package (originally including PHonon and PostProc) was originally developed by Stefano Baroni, Stefano de Gironcoli, Andrea Dal Corso (SISSA), Paolo Giannozzi, and many others. We quote in particular:

- Matteo Cococcioni (MIT) for DFT+U implementation;
- David Vanderbilt's group at Rutgers for Berry's phase calculations;
- Michele Lazzeri (Paris VI) for the 2n+1 code and Raman cross section calculation with 2nd-order response;
- Ralph Gebauer (ICTP, Trieste) and Adriano Mosca Conte (SISSA, Trieste) for noncolinear magnetism;
- Andrea Dal Corso for spin-orbit interactions;
- Carlo Sbraccia (Princeton) for NEB, Strings method, Metadynamics, for improvements to structural optimization and to many other parts of the code.
- Paolo Umari (Democritos) for finite electric fields;
- Renata Wentzcovitch (Univ.Minnesota) for variable-cell molecular dynamics;
- Lorenzo Paulatto (SISSA) for PAW implementation, built upon previous work by Guido Fratesi (Univ.Milano Bicocca) and Riccardo Mazzarello (ETHZ-USI Lugano);
- Filippo Spiga (Univ. Milano Bicocca) for mixed SMP-OpenMP parallelization;
- Ismaila Dabo (INRIA, Palaiseau) for electrostatics with free boundary conditions;

- Andrea Dal Corso for Ultrasoft, noncollinear, spin-orbit extensions to PHonon.

The CP package is based on the original code written by Roberto Car and Michele Parrinello. CP was developed by Alfredo Pasquarello (IRRMA, Lausanne), Kari Laasonen (Oulu), Andrea Trave, Roberto Car (Princeton), Nicola Marzari (MIT), Paolo Giannozzi, and others. FPMD, later merged with CP, was developed by Carlo Cavazzoni, Gerardo Ballabio (CINECA), Sandro Scandolo (ICTP), Guido Chiarotti (SISSA), Paolo Focher, and others. We quote in particular:

- Carlo Sbraccia (Princeton) for NEB and Metadynamics;
- Manu Sharma (Princeton) and Yudong Wu (Princeton) for maximally localized Wannier functions and dynamics with Wannier functions;
- Paolo Umari (MIT) for finite electric fields and conjugate gradients;
- Paolo Umari and Ismaila Dabo for ensemble-DFT;
- Xiaofei Wang (Princeton) for META-GGA;
- The Autopilot feature was implemented by Targacept, Inc.

Other packages in QUANTUM ESPRESSO:

- PWcond was written by Alexander Smogunov (SISSA) and Andrea Dal Corso.
- GIPAW (http://www.gipaw.org) was written by Davide Ceresoli (MIT), Ari Seitsonen, Uwe Gerstmann, Francesco Mauri (Univ. Paris VI) .
- PWgui was written by Anton Kokalj (IJS Ljubljana) and is based on his GUIB concept (http://www-k3.ijs.si/kokalj/guib/).
- atomic was written by Andrea Dal Corso and it is the result of many additions to the original code by Paolo Giannozzi and others. Lorenzo Paulatto wrote the extension to PAW.
- iotk (http://www.s3.infm.it/iotk) was written by Giovanni Bussi (ETHZ and S3 Modena).
- Wannier90 (http://www.wannier.org/) was written by A. Mostofi, J. Yates, Y.-S Lee (MIT).

- XSPECTRA was written by Matteo Calandra (Univ. Paris VI) and collaborators.
- QHA was contributed by Eyvaz Isaev (Moscow Steel and Alloy Inst. and Linkoping and Uppsala Univ.)

Other relevant contributions to QUANTUM ESPRESSO:

- Gerardo Ballabio wrote the first `configure` for QUANTUM ESPRESSO
- Dispersions interaction in the framework of DFT-D have been contributed by Daniel Forrer (Padua Univ.) and Michele Pavone (Naples Univ. Federico II).
- The calculation of the finite (imaginary) frequency molecular polarizability using the approximated Thomas-Fermi + von Weizaecker scheme (VdW) was contributed by Huy-Viet Nguyen (SISSA).
- The calculation of RPA frequency-dependent complex dielectric function was contributed by Andrea Benassi (S3 Modena).
- The initial BlueGene porting was done by Costas Bekas and Alessandro Curioni (IBM Zurich).
- Audrius Alkauskas (IRRMA), Simon Binnie (Univ. College London), Davide Ceresoli (MIT), Andrea Ferretti (S3), Guido Fratesi, Axel Kohlmeyer (UPenn), Konstantin Kudin (Princeton), Sergey Lisenkov (Univ.Arkansas), Nicolas Mounet (MIT), William Parker (Ohio State Univ), Guido Roma (CEA), Gabriele Sclauzero (SISSA), Sylvie Stucki (IRRMA), Pascal Thibaudeau (CEA), answered questions on the mailing list, found bugs, helped in porting to new architectures, wrote some code.

An alphabetical list of further contributors includes: Dario Alfè, Alain Allouche, Francesco Antoniella, Francesca Baletto, Mauro Boero, Nicola Bonini, Claudia Bungaro, Paolo Cazzato, Gabriele Cipriani, Jiayu Dai, Cesar Da Silva, Alberto Debernardi, Gernot Deinzer, Martin Hilgeman, Yosuke Kanai, Nicolas Lacorne, Stephane Lefranc, Kurt Maeder, Andrea Marini, Pasquale Pavone, Mickael Profeta, Kurt Stokbro, Paul Tangney, Antonio Tilocca, Jaro Tobik, Malgorzata Wierzbowska, Silviu Zilberman, and let us apologize to everybody we have forgotten.

This guide was mostly written by Paolo Giannozzi. Gerardo Ballabio and Carlo Cavazzoni wrote the section on CP.

1.3 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 QE-forge web site: http://www.qe-forge.org/.

Announcements about new versions of QUANTUM ESPRESSO are available via a low-traffic mailing list Pw_users: pw_users@pwscf.org. You can subscribe (but not post) to this list from the QUANTUM ESPRESSO web site.

The recommended place where to ask questions about installation and usage of QUANTUM ESPRESSO, and to report bugs, is the Pw_forum mailing list: pw_forum@pwscf.org. Here you can obtain help from the developers and many knowledgeable users. You can browse and search its archive from the QUANTUM ESPRESSO web site, but you have to subscribe in order to post to the list. Please search the archives – using the search facility, accessible from the web site, or using google – before posting: many questions are asked over and over again.

Important notice: 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).

The Pw_forum mailing list is also the recommanded place where to contact the developers of QUANTUM ESPRESSO.

1.4 Terms of use

QUANTUM ESPRESSO is free software, released under the GNU General Public License: 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 this code will contain an explicit acknowledgment and the following reference:

P. Giannozzi et al., J. Phys.: Condens. Matter, in press

Note the form QUANTUM ESPRESSO for textual citations of the code. 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 Installing QUANTUM ESPRESSO

2.1 Download

Presently, QUANTUM ESPRESSO is only distributed in source form; some precompiled executables (binary files) are provided only for PWgui. Stable releases of the QUANTUM ESPRESSO source package (current version is 4.1) can be downloaded from this URL:
http://www.quantum-espresso.org/download.php .

Uncompress and unpack the distribution using the command:

```
tar zxvf espresso-4.1.tar.gz
```

(a hyphen before "zxvf" is optional). If your version of "tar" doesn't recognize the "z" flag:

```
gunzip -c espresso-4.1.tar.gz | tar xvf -
```

A directory espresso-4.1/, containing the distribution, will be created. Occasionally, patches for the current version, fixing some errors and bugs, may be distributed as a "diff" file. In order to install a patch (for instance):

```
cd espresso-4.1/
patch -p1 < /path/to/the/diff/file/patch-file.diff
```

If more than one patch is present, they should be applied in the correct order.

Daily snapshots of the development version can be downloaded from the developers' site qe-forge.org: follow the link "Quantum ESPRESSO", then "SCM". Beware: the development version is, well, under development: use at your own risk! The bravest may access the development version via anonymous CVS (Concurrent Version System): see the developer manual, section "Using CVS".

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

```
Modules/    source files for modules that are common to all programs
include/    files *.h included by fortran and C source files
clib/       external libraries written in C
flib/       external libraries written in Fortran
iotk/       Input/Output Toolkit
install/    machine-dependent makefiles
pseudo/     pseudopotential files used by examples
upftools/   converters to unified pseudopotential format (UPF)
```

```
examples/   sample input and output files
tests/      automated tests
Doc/        documentation
```

while others are specific to a single package:

```
PW/         PWscf: source files for scf calculations (pw.x)
pwtools/    PWscf: source files for miscellaneous analysis programs
PP/         PostProc: source files for post-processing of pw.x data file
PH/         PHonon: source files for phonon calculations (ph.x) and analysis
Gamma/      PHonon: source files for Gamma-only phonon calculation (phcg.x)
D3/         PHonon: source files for third-order derivative calculations (d3.x
PWCOND/     PWcond: source files for conductance calculations (pwcond.x)
vdW/        VdW: source files for calculation of the molecular polarizability
            at finite (imaginary) frequency using approximated Thomas-Fermi
            + von Weizacker scheme
CPV/        CP: source files for Car-Parrinello code (cp.x)
atomic/     Source files for the pseudopotential generation package (ld1.x)
atomic_doc/ Documentation, tests and examples for atomic
GUI/        PWGui: Graphical User Interface
```

2.2 Installation

To install QUANTUM ESPRESSO from source, you need C and Fortran-95 compilers (Fortran-90 is not sufficient, but most "Fortran-90" compilers are actually Fortran-95-compliant). If you don't have a commercial Fortran-95 compilers, you may install the free g95 compiler: (http://www.g95.org/) or the GNU fortran compiler gfortran: (http://www.gfortran.org/). Note that both the C and the F90 compilers must be in your PATH, or else their full path must be explicitly given.

You also need a minimal Unix environment: basically, a command shell (e.g., bash or tcsh) and the utilities make, awk and sed. MS-Windows users need to have Cygwin (a UNIX environment which runs under Windows) installed: see http://www.cygwin.com/. 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
```

or

```
setenv LC_ALL C
```

to prevent any problem when running scripts (including installation scripts).

Instructions for the impatient:

```
cd espresso-4.1/
./configure
 make all
```

Executable programs (actually, symlinks to them) will be placed in the bin/ subdirectory.

If you have problems or would like to tweak the default settings, read the detailed instructions below.

2.3 `configure`

To install the QUANTUM ESPRESSO source package, run the `configure` script. It 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), PC clusters, IBM SP machines, SGI Origin and Altix, some HP-Compaq Alpha machines, NEC SX, Cray X1-XD-XT machines, Mac OS X, MS-Windows PCs. It may work with some assistance also on other architectures (see below).

For cross-compilation, you have to specify the target machine with the –host option (see below). This feature has not been extensively tested, but we had at least one successful report (compilation for NEC SX6 on a PC).

Specifically, `configure` generates the following files:

```
make.sys:           compilation rules and flags
configure.msg:      a report of the configuration run
include/fft_defs.h: defines fortran variable for C pointer
include/c_defs.h:   defines C to fortran calling convention
                    and a few more things used by C files
```

`configure.msg` is only used by `configure` to print its final report and is not needed for compilation. NOTA BENE: unlike previous versions, `configure` no longer runs the `makedeps.sh` shell script that updates dependencies. If you modify the program sources, run `./makedeps.sh` or type `make depend` to update files `make.depend` in the various subdirectories.

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 tricky part is to get external libraries recognized and used: see Sec. for details and hints.

Environment variables may be set in any of these ways:

```
export VARIABLE=value               # sh, bash, ksh
./configure
setenv VARIABLE value               # csh, tcsh
./configure
./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 95, Fortran 77, and C compilers
MPIF90:       name of parallel Fortran 95 compiler (using MPI)
CPP:          source file preprocessor (defaults to $CC -E)
LD:           linker (defaults to $MPIF90)
CFLAGS, FFLAGS, F90FLAGS, CPPFLAGS, LDFLAGS: compilation flags
LIBDIRS:      extra directories to search for libraries (see below)
```

For example, the following command line:

```
./configure MPIF90=mpf90 FFLAGS="-O2 -assume byterecl" \
            CC=gcc CFLAGS=-O3 LDFLAGS=-static
```

instructs `configure` to use `mpf90` as Fortran 95 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 environmental 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) may need to define the correct ARCH even if they are apparently recognized, because `configure` cannot figure out that cross-compilation is needed. Try the one that looks more similar to your machine type; you'll probably have to do some additional tweaking. 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
aix:    IBM AIX machines
mips:   SGI MIPS machines
alpha:  HP-Compaq alpha machines
alinux: HP-Compaq alpha running Linux
```

```
sparc:  Sun SPARC machines
solaris:PC's running SUN-Solaris
crayx1: Cray X1 machines
crayxt4: Cray XT4/5 machines
macppc: Apple PowerPC machines running Mac OS X
mac686: Apple Intel machines running Mac OS X
cygwin: MS-Windows PCs with Cygwin
necsx:  NEC SX-6 and SX-8 machines
ppc64:  Linux PowerPC machines, 64 bits
ppc64-mn:as above, with IBM xlf compiler
```

Note: x86_64 replaces amd64 since v.4.1. Finally, `configure` recognizes the following command-line options:

```
--disable-parallel: compile serial code, even if parallel compiler available
--host=target :     specify target machine for cross-compilation.
                    target is a string identifying the architecture you want
                    to compile for; run config.guess on the target machine
--disable-shared:   don't use shared libraries: generate static executables
--enable-shared:    use shared libraries
```

The latter two options actually work only in a few specific cases. If you want to modify the `configure` script (advanced users only!), see the Developer Manual. You will need GNU Autoconf (http://www.gnu.org/software/autoconf/) installed.

2.3.1 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 can be used instead: this often yields huge performance gains.

BLAS and LAPACK QUANTUM ESPRESSO can use the following architecture- specific replacements for BLAS and LAPACK:

```
MKL for Intel Linux PCs
ACML for AMD Linux PCs
ESSL for IBM machines
complib.sgimath for SGI Origin
SCSL for SGI Altix
SUNperf for Sun
cxml for HP-Compaq Alphas.
```

If none of these is available, we suggest that you use the optimized ATLAS library (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. They can be freely downloaded (but not redistributed): http://www.cs.utexas.edu/users/flame/goto/

FFT Quantum ESPRESSO can use the following vendor-specific FFT libraries:

```
IBM ESSL
SGI SCSL
SUN sunperf
NEC ASL
AMD ACML
```

If none of the above is available, you should use FFTW, choosing before compilation whether to load the built-in copy of FFTW or an external v.3 FFTW library. `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. Appropriate precompiling options will be set in all cases:

```
__FFTW      internal FFTW
__FFTW3     external FFTW v.3
__SCSL      SGI SCSL
__SUNPERF   SUN sunperf
__ESSL      IBM ESSL
  ASL       NEC ASL
```

If you have recent versions of MKL installed, you may try the FFTW interface provided with MKL. You will have to compile them (they come in source form

with the MKL library) and to modify the make.sys accordingly (MKL must linked *after* the FFTW-MKL interface)

If everything else fails, you'll have to modify the make.sys file manually: see the section on Manual configuration.

MPI libraries For parallel execution, QUANTUM ESPRESSO uses the MPI libraries. In well-configured machine, `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.2.6.6.

Other libraries QUANTUM ESPRESSO can use the MASS vector math library from IBM, if available (only on AIX).

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.sys, 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.

Please note: If you change any settings 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.

2.3.2 Manual configuration

If `configure` stops before the end, and you don't find a way to fix it, you have to write working "make.sys", "include/fft_defs.h" and "include/c_defs.h" files. For the latter two files, follow the explanations in "include/defs.h.README".

If `configure` has run till the end, you should need only to edit `make.sys`. A few templates (each for a different machine type) are provided in the install/ directory: they have names of the form `Make.system`, where "system" is a string identifying the architecture and compiler.

Obsolete: if you have the Intel compiler ifc v.6 or earlier, you will have to run the script `./ifcmods.sh`.

Most likely (and even more so if there isn't an exact match to your machine type), you'll have to tweak make.sys by hand. In particular, you must specify the full list of libraries that you intend to link to.

NOTA BENE: If you modify the program sources, run the `makedeps.sh` script or type `make depend` to update files `make.depend` in the various subdirectories.

2.4 Compilation

There are a few adjustable parameters in `Modules/parameters.f90`. The present values will work for most cases. All other variables are dynamically allocated: you do not need to recompile your code for a different system.

At your option, 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. Here is a list:

- `make pw` produces PW/pw.x
  pw.x calculates electronic structure, structural optimization, molecular dynamics, barriers with NEB.

- `make ph` produces PH/ph.x
  ph.x calculates phonon frequencies and displacement patterns, dielectric tensors, effective charges (uses data produced by pw.x).

- `make d3` produces D3/d3.x
  d3.x calculates anharmonic phonon lifetimes (third-order derivatives of the energy), using data produced by pw.x and ph.x (Ultrasoft pseudopotentials not supported).

- `make gamma` produces Gamma/phcg.x
  phcg.x is a version of ph.x that calculates phonons at q = 0 using conjugate-gradient minimization of the density functional expanded to second-order. Only the Γ (q = 0) point is used for Brillouin zone integration. It is faster and takes less memory than ph.x, but does not support Ultrasoft pseudopotentials.

- `make pp` produces several codes for data postprocessing, in PP/ (see list below).
- `make tools` produces several utility programs in pwtools/ (see list below).
- `make pwcond` produces PWCOND/pwcond.x
  for ballistic conductance calculations.
- `make pwall` produces all of the above.
- `make ld1` produces code atomic/ld1.x
  for pseudopotential generation (see specific documentation in atomic_doc/).
- `make upf` produces utilities for pseudopotential conversion in directory upftools/ (see section 4, "Pseudopotentials").
- `make cp` produces the Car-Parrinello code CP in CPV/cp.x and the postprocessing code CPV/cppp.x.
- `make all` produces all of the above.

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 the CVS sources, see the GUI/README file instead.

The codes for data postprocessing in PP/ are:

- pp.x extracts the specified data from files produced by pw.x, prepares data for plotting by writing them into formats that can be read by several plotting programs.
- bands.x extracts and reorders eigenvalues from files produced by pw.x for band structure plotting
- projwfc.x calculates projections of wavefunction over atomic orbitals, performs Lẅdin population analysis and calculates projected density of states. These can be summed using auxiliary code sumpdos.x.
- dipole.x calculates the dipole moment for isolated systems (molecules) and the Makov-Payne correction for molecules in supercells (beware: meaningful results only if the charge density is completely contained into the Wigner-Seitz cell)
- plotrho.x produces PostScript 2-d contour plots

- plotband.x reads the output of bands.x, produces band structure PostScript plots
- average.x calculates planar averages of quantities produced by pp.x (potentials, charge, magnetization densities,...)
- voronoy.x divides the charge density into Voronoy polyhedra (obsolete, use at your own risk)
- dos.x calculates electronic Density of States (DOS)
- pw2wan.x: interface with code WanT for calculation of transport properties via Wannier functions: see http://www.wannier-transport.org/
- pmw.x generates Poor Man's Wannier functions, to be used in DFT+U calculations
- pw2casino.x: interface with CASINO code for Quantum Monte Carlo calculation (http://www.tcm.phy.cam.ac.uk/~mdt26/casino.html).

The utility programs in pwtools/ are:

- dynmat.x applies various kinds of Acoustic Sum Rule (ASR), calculates LO-TO splitting at q = 0 in insulators, IR and Raman cross sections (if the coefficients have been properly calculated), from the dynamical matrix produced by ph.x
- q2r.x calculates Interatomic Force Constants (IFC) in real space from dynamical matrices produced by ph.x on a regular q-grid
- matdyn.x produces phonon frequencies at a generic wave vector using the IFC file calculated by q2r.x; may also calculate phonon DOS
- fqha.x for quasi-harmonic calculations
- lambda.x calculates the electron-phonon coefficient λ and the function $\alpha^2F(\omega)$
- dist.x calculates distances and angles between atoms in a cell, taking into account periodicity
- ev.x fits energy-vs-volume data to an equation of state
- kpoints.x produces lists of k-points

- pwi2xsf.sh, pwo2xsf.sh process respectively input and output files (not data files!) for pw.x and produce an XSF-formatted file suitable for plotting with XCrySDen, a powerful crystalline and molecular structure visualization program (http://www.xcrysden.org/). BEWARE: the pwi2xsf.sh shell script requires the pwi2xsf.x executables to be located somewhere in your $PATH.

- band_plot.x: undocumented and possibly obsolete

- bs.awk, mv.awk are scripts that process the output of pw.x (not data files!). Usage:

  ```
  awk -f bs.awk < my-pw-file > myfile.bs
  awk -f mv.awk < my-pw-file > myfile.mv
  ```

  The files so produced are suitable for use with xbs, a very simple X-windows utility to display molecules, available at:
  http://www.ccl.net/cca/software/X-WINDOW/xbsa/README.shtml

- path_int.sh/path_int.x: utility to generate, starting from a path (a set of images), a new one with a different number of images. The initial and final points of the new path can differ from those in the original one. Useful for NEB calculations.

- kvecs_FS.x, bands_FS.x: utilities for Fermi Surface plotting using XCrySDen

Other utilities VdW/ contains the sources for the calculation of the finite (imaginary) frequency molecular polarizability using the approximated Thomas-Fermi + von Weizäcker scheme, contributed by H.-V. Nguyen (Sissa and Hanoi University). Compile with `make vdw`, executables in VdW/vdw.x, no documentation yet, but an example in examples/example34.

2.5 Running examples

As a final check that compilation was successful, you may want to run some or all of the examples contained within the examples directory of the QUANTUM ESPRESSO distribution. Those examples try to exercise all the programs and features of the QUANTUM ESPRESSO distribution. A list of examples and of what each example does is contained in examples/README. For details, see the README file in each example's directory. If you find that

any relevant feature isn't being tested, please contact us (or even better, write and send us a new example yourself !).

To run the examples, you should follow this procedure:

1. Go to the examples directory and edit the environment variables file, 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
```

If you have downloaded the full QUANTUM ESPRESSO distribution, you may set BIN_DIR=$TOPDIR/bin and PSEUDO_DIR=$TOPDIR/pseudo, where =$TOPDIR is the root of the QUANTUM ESPRESSO source tree. In order to be able to run all the examples, the PSEUDO_DIR directory must contain all the needed pseudopotentials. If any of these are missing, you can download them (and many others) from the Pseudopotentials Page of the QUANTUM ESPRESSO web site (http://www.quantum-espresso.org/pseudo.php). TMP_DIR must be a directory 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).

2. 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 driver program (such as poe or mpiexec) and the number of processors: see section "Running on parallel machines' 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 < file.in > file.out
```

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

```
poe pw.x -procs 4 < 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 below through the batch queueing system installed on that machine. Ask your system administrator for instructions.

3. To run a single example, go to the corresponding directory (for instance, example/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 several minutes depending on your system. To run all the examples in one go, execute:

```
./run_all_examples
```

from the examples directory. On a single-processor machine, this typically takes a few hours. The make_clean script cleans the examples tree, by removing all the results subdirectories. However, if additional subdirectories have been created, they aren't deleted.

4. 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. Instead, you can run the check_example script in the examples directory:

```
./check_example example_dir
```

where example dir is the directory of the example that you want to check (e.g., ./check_example example01). You can specify multiple directories.

Note: check_example does only a fair job, and only for a few examples. For pw.x only, a much better series of automated tests is available in directory tests/. Edit variables PARA_PREFIX, PARA_POSTFIX (if needed) in file "check_pw.x.j"; explanations on how to run it are in the header of the file.

2.6 Installation tricks and problems

2.6.1 All architectures

Working fortran-95 and C compilers are needed in order to compile Quantum ESPRESSO. Most so-called "fortran-90" compilers implement the fortran-95 standard, but older versions may not be fortran-95 compliant.

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, 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 version.

If you get error messages at the loading phase that looks like "file XYZ.o: unknown (unrecognized, invalid, wrong, missing, ...) file type", or "file format not recognized for file XYZ.a", 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 does not stop at the first compilation error (it may happen in some software configurations). Remove file XYZ.o and look for the 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). If you did, but symbols are still missing, see below (for Linux PC). Remember: QUANTUM ESPRESSO is self-contained (with the exception of MPI libraries for parallel compilation). If system libraries are missing, the problem cannot be in QUANTUM ESPRESSO.

2.6.2 IBM AIX

On IBM machines with ESSL libraries installed, there is a potential conflict between a few LAPACK routines that are also part of ESSL, but with a different calling sequence. The appearence of run-time errors like

```
ON ENTRY TO ZHPEV  PARAMETER NUMBER  1 HAD AN ILLEGAL VALUE
```

is a signal that you are calling the bad routine. If you have defined -D__ESSL you should load ESSL before LAPACK: see variable LAPACK_LIBS in make.sys.

2.6.3 Linux PC

The web site of Axel Kohlmeyer contains a very informative section on compiling and running CPMD on Linux. Most of its contents applies to the QUANTUM ESPRESSO codes as well:
http://www.theochem.rub.de/~axel.kohlmeyer/cpmd-linux.html. In particular, there is a set of ATLAS libraries, containing all of LAPACK and no external reference to fortran libraries:
http://www.theochem.rub.de/~axel.kohlmeyer/cpmd-linux.html#atlas

It is convenient to create semi-statically linked executables (with only libc/libm/libpthread linked dynamically). 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).

If you get errors like

```
IPO Error: unresolved : __svml_cos2
```

at the linking stage, your compiler is optimized to use the SSE version of sine, cosine etc. contained in the SVML library. Append '-lsvml' to the list of libraries in your make.sys file (info by Axel Kohlmeyer, oct.2007).

Linux PCs with Portland Group compiler (pgf90) QUANTUM ESPRESSO does not work reliably, or not at all, with many versions of the Portland Group compiler (in particular, v.5.2 and 6.0). Version 5.1 used to work, v.6.1 is reported to work (info from Paolo Cazzato). Use the latest version of each release of the compiler, with patches if available: see the Portland Group web site, http://www.pgroup.com/faq/install.htm#release info

Linux PCs with Pathscale compiler Version 2.99 of the Pathscale compiler works and is recognized by `configure`, but the preprocessing step:

```
pathcc -E
```

causes a mysterious error in compilation of iotk and should be replaced by

```
/lib/cpp -P --traditional
```

The MVAPICH parallel environment with Pathscale compilers also works. (info by Paolo Giannozzi, July 2008)

Linux PCs with gfortran Recent versions of gfortran (e.g. v.4.1 and later) are supported, but only the basic functionalities have been tested. More advanced ones may or may not work. In particular: reading files produced by previous versions of QUANTUM ESPRESSO may not work, apparently due to a gfortran bug.

Linux PCs with Intel compiler (ifort, formerly ifc) 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.

Starting from the latests v 8.1 patchlevels, the recommended way to build semi-statically linked binaries is to use the -i-static flag; for multi-threaded libraries the linker flag would be -i-static -openmp (linking libguide is no longer needed and the compiler will pick the correct one). The warning: "feupdateenv is not implemented and will always fail", showing up in recent versions, can be safely ignored. For previous versions, try -static-libcxa (this will give an incomplete semi-static link on newer versions).

Each major release of the Intel compiler differs a lot from the previous one. Do not mix compiled objects from different releases: they may be incompatible.

In case of trouble, update your version with the most recent patches, available via Intel Premier support (registration free of charge for Linux): http://developer.intel.com/software/products/support/#premier.

ifort v.10: on 64-bit AMD CPUs, at least some versions of ifort 10.1 miscompile subroutine write_rho_xml in Module/xml_io_base.f90 with -O2 options. Using -O1 instead solves the problem (info by Carlo Cavazzoni, March 2008).

"The intel compiler version 10.1.008 miscompiles a lot of codes (I have proof for CP2K and CPMD) and needs to be updated in any case" (info by Axel Kohlmeter, May 2008).

ifort v.9: The latest (July 2006) 32-bit version of ifort 9.1 works flawlessy. Earlier versions yielded "Compiler Internal Error".

At least some versions of ifort 9.0 have a buggy preprocessor that either prevents compilation of iotk, or produces runtime errors in cft3. Update to a more patched version, or modify make.sys to explicitly perform preprocessing using /lib/cpp, as in the following example (courtesy from Sergei Lisenkov):

```
.f90.o:
        $(CPP) $(CPPFLAGS) $< -o $*.F90
        $(MPIF90) $(F90FLAGS) -c $*.F90 -o $*.o

CPP       = /lib/cpp
CPPFLAGS  = -P -C -traditional $(DFLAGS) $(IFLAGS)
```

On some versions of RedHat Linux, you may get an obscure error: IPO link: can not find "(" ... , due to a bad system configuration. Add option -no-ipo to LDFLAGS in file make.sys.

ifort v.8: Some releases of ifort 8 yield "Compiler Internal Error". Update to a more patched version: 8.0.046 for v. 8.0, 8.1.018 for v. 8.1.

There is a well known problem with ifort 8 and pthreads (that are used both in Debian Woody and Sarge) that causes "segmentation fault" errors (info from Lucas Fernandez Seivane). Version 7 did not have this problem.

ifc v.7: Some releases of ifc 7.0 and 7.1 yield "Compiler Internal Error". Update to the last version (should be 7.1.41).

Warnings "size of symbol ... changed ..." are produced by ifc 7.1 a the loading stage. These seem to be harmless, but they may cause the loader to stop, depending on your system configuration. If this happen and no executable is produced, add the following to LDFLAGS: -Xlinker –noinhibit-exec.

Linux distributions using glibc 2.3 or later (such as e.g. RedHat 9) may be incompatible with ifc 7.0 and 7.1. The incompatibility shows up in the form of messages "undefined reference to 'errno' " at linking stage. A workaround is available: see http://newweb.ices.utexas.edu/misc/ctype.c.

Linux PCs with MKL libraries On Intel CPUs it is very convenient to use Intel MKL libraries. They can be also used for AMD CPU, selecting the appropriate machine-optimized libraries, and also together with non-Intel compilers. If `configure` doesn't find MKL, try `configure` –enable-shared. Note that ifort 8 fails to load with MKL v. 5.2 or earlier versions, because some symbols that are referenced by MKL are missing There is a fix for this (info from Konstantin Kudin): add libF90.a from ifc 7.1 at the linking stage, as the last library. Note that some combinations of not-so-recent versions of MKL and ifc may yield a lot of "undefined references" when statically loaded: use `configure` –enable-shared, or remove the -static option in make.sys. Note that pwcond.x works only with recent versions (v.7 or later) of MKL.

MKL contains optimized FFT routines and a FFTW interface, to be separately compiled. For 64-bit Intel Core2 processors, they are slightly faster than FFTW (MKL v.10, FFTW v.3 fortran interface, reported by P. Giannozzi, November 2008).

Important: for parallel (MPI) execution on multiprocessor (SMP) machines, set the environmental variable OMP_NUM_THREADS to 1 ! See the section "Running on parallel machines" for more info on this and on the difference between MPI and OpenMP parallelization.

Fun with precompiled libraries Since there is no standard fortran compiler for Linux, different compilers have different ideas about the right way to call external libraries. As a consequence you may have a mismatch between what your compiler calls ("symbols") and the actual name of the required library call. Use the nm command to determine the name of a library call, as in the following examples:

```
nm /usr/local/lib/libblas.a | grep -i 'T daxpy'
nm /usr/local/lib/liblapack.a | grep -i 'T zhegv'
```

where typical location and name of libraries is assumed. Most precompiled libraries have lowercase names with one or two underscores (_) appended. `configure` should select the appropriate preprocessing options in make.sys, but in case of trouble, be aware that:

- the Absoft compiler is case-sensitive (like C and unlike other Fortran compilers) and does not add an underscore to symbol names (note that if your libraries contain uppercase or mixed case names, you are out of luck: You must either recompile your own libraries, or change the #define's in include/f_defs.h);
- both Portland compiler (pgf90) and Intel compiler (ifort/ifc) are case insensitive and add an underscore to symbol names.

Another potential source of trouble is the incompatibility between I/O libraries used by different fortran compilers. This manifests itself under the form of missing routines with strange names (like s_wsfe, do_fio...) at linking stage. Possible workarounds include

- loading the missing routines; it is often sufficient to load -lg2c (sometimes -lm may also be needed); or
- (better) to replace the BLAS routine xerbla (it should be the only one containing I/O calls) with a recompiled object.

If you choose the latter workaround, locate the library containing this routine using nm, for instance:

```
nm /path/to/your/libs/libf77blas.a | grep 'T xerbla'
```

and replace the object xerbla.o in the library with the one you will compile. In flib/:

```
make xerbla.o
ar rv /path/to/your/libs/libf77blas.a xerbla.o
```

If nothing works, you may need to recompile the libraries with your fortran compiler, or to use the internal (slow) copy.

2.6.4 AMD 32-bit CPUs

AMD Athlon CPUs can be basically treated like Intel Pentium CPUs. You can use the Intel compiler and MKL with Pentium-3 optimization.

Konstantin Kudin reports that the best results in terms of performances are obtained with ATLAS optimized BLAS/LAPACK libraries, using AMD

Core Math Library (ACML) for the missing libraries. ACML can be freely downloaded from AMD web site. Beware: some versions of ACML – i.e. the GCC version with SSE2 – crash PWscf. The "nosse2" version appears to be stable. Load first ATLAS, then ACML, then -lg2c, as in the following example (replace what follows -L with something appropriate to your configuration):

```
-L/location/of/fftw/lib/ -lfftw \
-L/location/of/atlas/lib -lf77blas -llapack -lcblas -latlas \
-L/location/of/gnu32_nosse2/lib -lacml -lg2c
```

2.6.5 64-bit CPUs

64-bit CPUs like the AMD Opteron and the Intel Itanium are supported and should work both in 32-bit emulation and in 64-bit mode. Both the Portland and the Intel compiler (v8.1 EM64T-edition, available via Intel Premier support) should work. 64-bit executables can address a much larger memory space, but apparently they are not especially faster than 32-bit executables. eThe Intel compiler has been reported to be more reliable and to produce faster executables wrt the Portland compiler. You may also try g95 or gfortran.

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 will not give you any advantage and you may run into trouble.

2.6.6 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. The number of possible configurations, in terms of type and version of the MPI libraries, kernels, system libraries, compilers, is very large. Quantum ESPRESSO compiles and works on all non-buggy, properly configured hardware and software combinations. You may have to recompile MPI libraries: not all MPI installations contain support for the fortran-90 compiler of your choice (or for any fortran-90 compiler at all!). See Axel Kohlmeyer's web site for precompiled versions of the MPI libraries. Very useful step-by-step instructions can be found in the following post by Javier Antonio Montoya:
http://www.democritos.it/pipermail/pw_forum/2008April/008818.htm .

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 section "Running on parallel machines" and Axel Kohlmeyer's web site for more info.

If you are dissatisfied with the performances in parallel execution, read the section on "Parallelization issues". See also the following post from Axel Kohlmeyer:
http://www.democritos.it/pipermail/pw_forum/2008-April/008796.html

2.6.7 Intel Mac OS X

Newer Mac OS-X machines with Intel CPUs are supported by `configure`, with gcc4+g95, gfortran, and the Intel compiler ifort with MKL libraries.

Intel Mac OS X with ifort "Uninstall darwin ports, fink and developer tools. The presence of all of those at the same time generates many spooky events in the compilation procedure. I installed just the developer tools from apple, the intel fortran compiler and everything went on great" (Info by Riccardo Sabatini, Nov. 2007)

Intel Mac OS X 10.4 and 10.5 with g95 and gfortran The stable and unstable versions of g95 are known to work. Recent gfortran versions also work, but they may require an updated version of Developer Tools (XCode 2.4.1 or 2.5), that can be downloaded from Apple. Some tests fails with mysterious errors, that disappear if fortran BLAS are linked instead of system Atlas libraries. Use:

```
BLAS_LIBS      = ../flib/blas.a -latlas
```

(Info by Paolo Giannozzi, jan.2008)

3 Running on parallel machines

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

1. a launcher program, such as poe, mpirun, mpiexec, with or without appropriate options
2. the number of processors, typically as an option to the launcher program, but in some cases to be specified after the program to be executed;

3. the program to be executed, with the proper path if needed: for instance, pw.x, or ./pw.x, or $HOME/bin/pw.x, or whatever applies;

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

   - the number of "pools" into which processors are to be grouped (pw.x only);
   - the number of "task groups" into which processors are to be grouped;
   - the number of "images" used by NEB calculations;
   - the number of processors performing iterative diagonalization (for pw.x) or orthonormalization (for cp.x).

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: see section "Understanding Parallelism" for the meaning of the various options.

For illustration, here is how to run pw.x on 16 processors partitioned into 8 pools (2 processors each), for several typical cases. For convenience, we also give the corresponding values of PARA_ PREFIX, PARA_POSTFIX to be used in running the examples distributed with QUANTUM ESPRESSO (see section "Run examples".

IBM SP machines, batch:

```
pw.x -npool 8 < input
PARA_PREFIX="", PARA_POSTFIX="-npool 8"
```

This should also work interactively, with environment variables NPROC set to 16, MP HOSTFILE set to the file containing a list of processors.

IBM SP machines, interactive, using poe:

```
poe pw.x -procs 16 -npool 8 < input
PARA_PREFIX="poe", PARA_POSTFIX="-procs 16 -npool 8"
```

PC clusters using mpiexec:

```
mpiexec -n 16 pw.x -npool 8 < input
PARA_PREFIX="mpiexec -n 16", PARA_POSTFIX="-npool 8"
```

SGI Altix and PC clusters using mpirun:

```
mpirun -np 16 pw.x -npool 8 < input
PARA_PREFIX="mpirun -np 16", PARA_POSTFIX="-npool 8"
```

IBM BlueGene using mpirun:

```
mpirun -np 16 -exe /path/to/executable/pw.x -args "-npool 8" \
  -in /path/to/input -cwd /path/to/work/directory
```

3.1 Understanding Parallelism in Quantum ESPRESSO

Quantum ESPRESSO uses MPI parallelization. Data structures are distributed across processors organized in a hierarchy of groups, which are identified by different MPI communicators level. The groups hierarchy is as follow:

```
world _ images _ pools _ task groups
              \_ ortho groups
```

world: is the group of all processors (MPI_COMM_WORLD).

images: Processors can then be divided into different "images", corresponding to a point in configuration space (i.e. to a different set of atomic positions). Such partitioning is used when performing Nudged Elastic band (NEB), Meta-dynamics and Laio-Parrinello simulations.

pools: When k-point sampling is used, each image group can be subpartitioned into "pools", and k-points can distributed to pools. Within each pool, reciprocal space basis set (plane waves) and real-space grids are distributed across processors. This is usually referred to as "plane-wave parallelization". All linear-algebra operations on array of plane waves / 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).

task groups: In order to allow good parallelization of the 3D FFT when the number of processors exceeds the number of FFT planes, data can be redistributed to "task groups" so that each group can process several wavefunctions at the same time.

ortho group: A further level of parallelization, independent on plane-wave (pool) parallelization, is the parallelization of subspace diagonalization (pw.x) or iterative orthonormalization (cp.x). Both operations required the diagonalization of arrays whose dimension is the number of Kohn-Sham states (or a small multiple). All such arrays are distributed block-like across the

"ortho group", a subgroup of the pool of processors, organized in a square 2D grid. 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).

Communications: Images and pools are loosely coupled and processors communicate between different images and pools only once in a while, whereas processors within each pool are tightly coupled and communications are significant. This means that Gigabit ethernet (typical for cheap PC clusters) is ok up to 4-8 processors per pool, but *fast* communication hardware (e.g. Mirynet or comparable) is absolutely needed beyond 8 processors per pool.

Choosing parameters: To control the number of images, pools and task groups, command line switch: -nimage -npools -ntg can be used. The dimension of the ortho group is set to the largest value compatible with the number of processors and with the number of electronic states. The user can choose a smaller value using the command line switch -ndiag (pw.x) or -northo (cp.x) . As an example consider the following command line:

```
mpirun -np 4096 ./pw.x -nimage 8 -npool 2 -ntg 8 -ndiag 144 -input my.input
```

This execute the PWscf code on 4096 processors, to simulate a system with 8 images, each of which is distributed across 512 processors. K-points are distributed across 2 pools of 256 processors each, 3D FFT is performed using 8 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: -nimage 1 -npool 1 -ntg 1 ; ndiag is chosen by the code as the fastest n^2 (n integer) that fits into the size of each pool.

Massively parallel calculations For very large jobs (i.e. O(1000) atoms or so) 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 both the "task group" and the "ortho group" parallelization. Without a judicious choice of parameters, large jobs will find a stumbling block in either memory or CPU requirements. In particular, the "ortho group" parallelization is used in the diagonalization of matrices in the subspace of Kohn-Sham states (whose dimension is as a strict minumum equal to the number of occupied states). These are stored as block-distributed matrixes (distributed across processors) and diagonalized using custom-taylored diagonalization algorithms that work on block-distributed matrixes.

Since v.4.1, Scalapack can be used to diagonalize block distributed matrixes, yielding better speed-up than the default algorithms for large (> 1000

) matroces, when using a large number of processors (> 512). If you want to test scalapack you have to compile adding -D__SCALAPACK to DFLAGS in make.sys and you have to modify the LAPACK_LIBS variable like in the following (works on CINECA BCX machine):

```
SCALAPACK_LIBS = \
/cineca/prod/libraries/SCALAPACK/1.8.0/openmpi--1.2.5--intel--10.1/libscalapack.
BLACS_LIBS     = \
/cineca/prod/libraries/BLACS/1.1/openmpi--1.2.5--intel--10.1/libblacs.a
BLACS_INI      = \
/cineca/prod/libraries/BLACS/1.1/openmpi--1.2.5--intel--10.1/libblacsF77init.a
LAPACK_LIBS = $(SCALAPACK_LIBS) $(BLACS_LIBS) $(BLACS_INI) $(BLACS_LIBS) \
                  /cineca/prod/acml/4.1.0/ifort64/lib/libacml.a
```

(info by Carlo Cavazzoni, Oct. 2008)

3.2 Tricks and problems

Trouble with MKL and OpenMP parallelization QUANTUM ESPRESSO uses a parallelization paradigm based on message-passing: a copy of the executable runs on each CPU, each copy living in a different world, communicating with other copies via calls to MPI (Message-Passing Interface) libraries. OpenMP is a different parallelization paradigm: a single executable spawn subprocesses (threads) that perform some specific tasks. OpenMP can be implemented via compiler directives or else via OpenMP-aware libraries such as ESSL or MKL. It is typically convenient for SMP (multiprocessor) machines, with one CPU running the master process and threads started on the other CPUs. Presently (nov. 2008) MPI and OpenMP parallelization *must not be active at the same time*. If you notice very bad parallel performances, OpenMP and MPI may be conflicting for the same CPUs. This is a frequent problem with MKL, that by default perform OpenMP autoparallelization.

When running in parallel with MPI (using "mpirun" for instance) on multiprocessor machines, you should set the environmental variable OMP_NUM_THREADS to 1. This is *crucial* if your code is linked with recent versions of MKL. 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".

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. Very useful step-by-step instructions to compile and install MPI libraries can be found in the following post by Javier Antonio Montoya:
http://www.democritos.it/pipermail/pw_forum/2008-April/008818.htm .

On a Xeon quadriprocessor cluster, erratic crashes in parallel execution have been reported, apparently correlated with ifort 10.1 (info by Nathalie Vast and Jelena Sjakste, May 2008).

Understanding parallel I/O In parallel exeution, each processor has its own slice of wavefunctions, to be written to temporary files during the calculation. The way wavefunctions are written by pw.x is governed by variable wf_collect, in namelist control. If wf_collect=.true., the final wavefunctions are collected into a single directory, written by a single processor, whose format is independent on the number of processors. If wf_collect=.false. (default) each processor writes its own slice of the final wavefunctions to disk in the internal format used by PWscf.

The former case requires more disk I/O and disk space, but produces portable data files; the latter case requires less I/O and disk space, but the data so produced can be read only by a job running on the same number of processors and pools, 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 on processors will follow the same pattern for different jobs).

cp.x instead always collects the final wavefunctions into a single directory. Files written by pw.x can be read by cp.x only if wf_collect=.true. (and if produced for k=0 case).

With the new file format (v.3.1 and later) all data (except wavefunctions in pw.x if wf_collect=.false.) is written to and read from a single directory outdir/prefix.save. A copy of pseudopotential files is also written there. If some processor cannot access outdir/prefix.save, it reads the pseudopotential files 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!

Avoid I/O to network-mounted disks (via NFS) as much as you can! Ideally the scratch directory (ESPRESSO_TMPDIR) 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 option "disk_io='minimal'", or even 'none', if you run into trouble (or angry system managers) with eccessive I/O with pw.x. The code will store wavefunctions into RAM during the calculation. Note however that this will increase your memory usage and may limit or prevent restarting from interrupted runs.

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 -in (or -inp or -input), followed by the input file name. Example:

```
pw.x -in inputfile npool 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 -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.

Cray XT3 On the cray xt3 there is a special hack to keep files in memory instead of writing them without changes to the code. You have to do a:

module load iobuf before compiling and then add liobuf at link time. If you run a job you set the environment variable IOBUF_PARAMS to proper numbers and you can gain a lot. Here is one example:

```
env IOBUF_PARAMS='*.wfc*:noflush:count=1:size=15M:verbose,\
*.dat:count=2:size=50M:lazyflush:lazyclose:verbose,\
*.UPF*.xml:count=8:size=8M:verbose' pbsyod =\
\~{}/pwscf/pwscfcvs/bin/pw.x npool 4 in si64pw2x2x2.inp > & \
si64pw2x2x232moreiobuf.out &
```

This will ignore all flushes on the *wfc* (scratch files) using a single i/o buffer large enough to contain the whole file ($\sim$ 12 Mb here). this way they are actually never(!) written to disk. The *.dat files are part of the restart, so needed, but you can be 'lazy' since they are writeonly. .xml files have a lot of accesses (due to iotk), but with a few rather small buffers, this can be handled as well. You have to pay attention not to make the buffers too large, if the code needs a lot of memory, too and in this example there is a lot of room for improvement. After you have tuned those parameters, you can remove the 'verboses' and enjoy the fast execution. Apart from the i/o issues the cray xt3 is a really nice and fast machine. (Info by Axel Kohlmeyer, maybe obsolete)

4 Using PWscf

Input files for the PWscf codes may be either written by hand (the good old way), or produced via the "PWgui" graphical interface by Anton Kokalj, included in the Quantum ESPRESSO distribution. See PWgui-x.y.z/INSTALL (where x.y.z is the version number) for more info on PWgui, or GUI/README if you are using CVS sources.

You may take the examples distributed with Quantum ESPRESSO as templates for writing your own input files: see section 2.3, "Run examples". In the following, whenever we mention "Example N", we refer to those. Input files are those in the results directories, with names ending in .in (they will appear after you have run the examples).

Note about exchange-correlation: the type of exchange-correlation used in the calculation is read from PP files. All PP's must have been generated using the same exchange-correlation.

4.1 Electronic and ionic structure calculations

Electronic and ionic structure calculations are performed by program pw.x.

Input data The input data is organized as several namelists, followed by other fields introduced by keywords.

The namelists are

```
&CONTROL: general variables controlling the run
&SYSTEM: structural information on the system under investigation
&ELECTRONS: electronic variables: self-consistency, smearing
&IONS (optional): ionic variables: relaxation, dynamics
&CELL (optional): variable-cell dynamics
&PHONON (optional): information required to produce data for
phonon calculations
```

Optional namelist may be omitted if the calculation to be performed does not require them. This depends on the value of variable calculation in namelist &CONTROL. Most variables in namelists have default values. Only the following variables in &SYSTEM must always be specified:

```
ibrav (integer): bravais-lattice index
celldm (real, dimension 6): crystallographic constants
nat (integer): number of atoms in the unit cell
ntyp (integer): number of types of atoms in the unit cell
ecutwfc (real): kinetic energy cutoff (Ry) for wavefunctions.
```

For metallic systems, you have to specify how metallicity is treated by setting variable occupations. If you choose occupations='smearing', you have to specify the smearing width degauss and optionally the smearing type smearing. If you choose occupations='tetrahedra', you need to specify a suitable uniform k-point grid (card K_POINTS with option automatic). Spin-polarized systems must be treated as metallic system, except the special case of a single k-point, for which occupation numbers can be fixed (occupations='from input' and card OCCUPATIONS).

Explanations for the meaning of variables ibrav and celldm are in file INPUT_PW. Please read them carefully. There is a large number of other variables, having default values, which may or may not fit your needs.

After the namelists, you have several fields introduced by keywords with self-explanatory names:

```
ATOMIC_SPECIES
ATOMIC_POSITIONS
K_POINTS
CELL_PARAMETERS (optional)
OCCUPATIONS (optional)
CLIMBING_IMAGES (optional)
```

The keywords may be followed on the same line by an option. Unknown fields (including some that are specific to CP package) are ignored by PWscf. See file Doc/INPUT_PW for a detailed explanation of the meaning and format of the various fields.

Note about k points: The k-point grid can be either automatically generated or manually provided as a list of k-points and a weight in the Irreducible Brillouin Zone only of the Bravais lattice of the crystal. The code will generate (unless instructed not to do so: see variable nosym) all required k-point and weights if the symmetry of the system is lower than the symmetry of the Bravais lattice. The automatic generation of k-points follows the convention of Monkhorst and Pack.

4.1.1 Typical cases

We may distinguish the following typical cases for pw.x:

Single-point (fixed-ion) SCF calculation Set calculation='scf'. Namelists &IONS and &CELL need not to be present (this is the default).
See Example 01.

Band structure calculation First perform a SCF calculation as above; then do a non-SCF calculation by specifying calculation='bands' or calculation='nscf', with the desired k-point grid and number nbnd of bands. If you are interested in calculating only the Kohn-Sham states for the given set of k-points, use calculation='bands'. If you are interested in further processing of the results of non-SCF calculations (for instance, in DOS calculations) use calculations='nscf'. Specify nosym=.true. to avoid generation of additional k-points in low symmetry cases. Variables prefix and outdir, which determine the names of input or output files, should be the same in the two runs.
See Example 01.

Important: until v.4.0, atomic positions for a non scf calculations were read from input, while the scf potential was read from the data file of the scf calculation. Since v.4.1, both atomic positions and the scf potential are read from the data file so that consistency is guaranteed.

Structural optimization Specify calculation='relax' and add namelist &IONS.

All options for a single SCF calculation apply, plus a few others. You may follow a structural optimization with a non-SCF band-structure calculation (since v.4.1, you do not need any longer to update the atomic positions in

the input file for non scf calculation).
See Example 03.

Molecular Dynamics Specify calculation='md' and time step dt.

Use variable ion dynamics in namelist &IONS for a fine-grained control of the kind of dynamics. Other options for setting the initial temperature and for thermalization using velocity rescaling are available. Remember: this is MD on the electronic ground state, not Car-Parrinello MD.
See Example 04.

Polarization via Berry Phase See Example 10, its README, and the documentation in the header of PW/bp_c_phase.f90.

Nudged Elastic Band calculation Specify calculation='neb' and add namelist &IONS.

All options for a single SCF calculation apply, plus a few others. In the namelist &IONS the number of images used to discretize the elastic band must be specified. All other variables have a default value. Coordinates of the initial and final image of the elastic band have to be specified in the ATOMIC POSITIONS card. A detailed description of all input variables is contained in the file Doc/INPUT PW. See also Example 17.

4.1.2 Data files

The output data files are written in the directory specified by variable outdir, with names specified by variable prefix (a string that is prepended to all file names, whose default value is: prefix='pwscf'). The "iotk" toolkit is used to write the file in a XML format, whose definition can be found in the Developer Manual. In order to use the data directory on a different machine, you need to convert the binary files to formatted and back, using the "bin/iotk" script.

The execution stops if you create a file "prefix.EXIT" in the working directory. IMPORTANT NOTE: this is the directory where the program is executed, NOT the directory "outdir" defined in input, where files are written. Note that with some versions of MPI, the "working directory" is the directory where the pw.x executable is! The advantage of this procedure is that all files are properly closed, whereas just killing the process may leave data and output files in unusable state.

4.2 Phonon calculations

Phonon calculation is presently a two-step process: First, you have to find the ground-state atomic and electronic configuration; Second, you can calculate phonons using Density-Functional Perturbation Theory. Further processing to calculate Interatomic Force Constants, to add macroscopic electric field and impose Acoustic Sum Rules at q=0 may be needed.

Since version 4 it is possible to safely stop execution of the phonon code using the same mechanism of the pw.x code, i.e. by creating a file prefix.EXIT in the working directory. Execution can be resumed by setting 'recover=.true.' in the subsequent input data.

Single-q calculation The phonon code ph.x calculates normal modes at a given q-vector, starting from data files produced by pw.x with a simple SCF calculation. NOTE: the alternative procedure in which a band-structure calculation with calculation='phonon' was performed as an intermediate step is no longer implemented since version 4.1. It is also no longer needed to specify lnscf=.true. for $q \neq 0$.

The output data file appear in the directory specified by variables outdir, with names specified by variable prefix. After the output file(s) has been produced (do not remove any of the files, unless you know which are used and which are not), you can run ph.x.

The first input line of ph.x is a job identifier. At the second line the namelist &INPUTPH starts. The meaning of the variables in the namelist (most of them having a default value) is described in file INPUT PH. Variables outdir and prefix must be the same as in the input data of pw.x. Presently you must also specify amass (real, dimension ntyp): the atomic mass of each atomic type.

After the namelist you must specify the q-vector of the phonon mode. This must be the same q-vector given in the input of pw.x.

Notice that the dynamical matrix calculated by ph.x at q = 0 does not contain the non-analytic term occuring in polar materials, i.e. there is no LO-TO splitting in insulators. Moreover no Acoustic Sum Rule (ASR) is applied. In order to have the complete dynamical matrix at q = 0 including the non-analytic terms, you need to calculate effective charges by specifying option epsil=.true. to ph.x.

Use program dynmat.x to calculate the correct LO-TO splitting, IR cross sections, and to impose various forms of ASR. If ph.x was instructed to calculate Raman coefficients, dynmat.x will also calculate Raman cross sections for a typical experimental setup.

A sample phonon calculation is performed in Example 02.

Calculation of interatomic force constants in real space First, dynamical matrices D(q) are calculated and saved for a suitable uniform grid of q-vectors (only those in the Irreducible Brillouin Zone of the crystal are needed). Although this can be done one q-vector at the time, a simpler procedure is to specify variable ldisp=.true. and to set variables nq1,nq2,nq3 to some suitable Monkhorst-Pack grid, that will be automatically generated, centered at q = 0. Do not forget to specify epsil=.true. in the input data of ph.x if you want the correct TO-LO splitting in polar materials.

Second, code q2r.x reads the D(q) dynamical matrices produced in the preceding step and Fourier-transform them, writing a file of Interatomic Force Constants in real space, up to a distance that depends on the size of the grid of q-vectors. Program matdyn.x may be used to produce phonon modes and frequencies at any q using the Interatomic Force Constants file as input. See Example 06.

Calculation of electron-phonon interaction coefficients The calculation of electron-phonon coefficients in metals is made difficult by the slow convergence of the sum at the Fermi energy. It is convenient to calculate phonons, for each q-vector of a suitable grid, using a smaller k-point grid, saving the dynamical matrix and the self-consistent first-order variation of the potential (variable fildvscf). Then a non-SCF calculation with a larger k-point grid is performed. Finally the electron-phonon calculation is performed by specifying elph=.true., trans=.false., and the input files fildvscf, fildyn. The electron-phonon coefficients are calculated using several values of gaussian broadening (see PH/elphon.f90) because this quickly shows whether results are converged or not with respect to the k-point grid and Gaussian broadening. See Example 07.

All of the above must be repeated for all desired q-vectors and the final result is summed over all q-vectors, using pwtools/lambda.x. The input data for the latter is described in the header of pwtools/lambda.f90.

4.3 Post-processing

There are a number of auxiliary codes performing postprocessing tasks such as plotting, averaging, and so on, on the various quantities calculated by pw.x. Such quantities are saved by pw.x into the output data file(s). Postprocessing codes are in the PP/ directory.

Plotting selected quantities The main postprocessing code pp.x reads data file(s), extracts or calculates the selected quantity, writes it into a format

that is suitable for plotting.

Quantities that can be read or calculated are:

```
charge density
spin polarization
various potentials
local density of states at $E_F$
local density of electronic entropy
STM images
selected squared wavefunction
ELF (electron localization function)
planar averages
integrated local density of states
```

Various types of plotting (along a line, on a plane, three-dimensional, polar) and output formats (including the popular cube format) can be specified. The output files can be directly read by the free plotting system Gnuplot (1D or 2D plots), or by code plotrho.x that comes with PWscf (2D plots), or by advanced plotting software XCrySDen and gOpenMol (3D plots).

See file INPUT_PP.* for a detailed description of the input for code pp.x. See example05/ in the examples/ directory for a charge density plot.

Band structure, Fermi surface The code bands.x reads data file(s), extracts eigenvalues, regroups them into bands (the algorithm used to order bands and to resolve crossings may not work in all circumstances, though). The output is written to a file in a simple format that can be directly read by plotting program plotband.x. Unpredictable plots may results if k-points are not in sequence along lines. See example05/ in the examples/ directory for a simple band plot.

The code bands.x performs as well a symmetry analysis of the band structure: see example01/.

The calculation of Fermi surface can be performed using kvecs_FS.x and bands_FS.x. The resulting file in .xsf format can be read and plotted using xcrysden. See example08/ for an example of Fermi surface visualization (Ni, including the spin-polarized case).

Projection over atomic states, DOS The code projwfc.x calculates projections of wavefunctions over atomic orbitals. The atomic wavefunctions are those contained in the pseudopotential file(s). The Löwdin population analysis (similar to Mulliken analysis) is presently implemented. The projected

DOS (or PDOS: the DOS projected onto atomic orbitals) can also be calculated and written to file(s). More details on the input data are found in file INPUT_PROJWFC.*. The auxiliary code sumpdos.x (courtesy of Andrea Ferretti) can be used to sum selected PDOS, by specifiying the names of files containing the desired PDOS. Type sumpdos.x -h or look into the source code for more details. The total electronic DOS is instead calculated by code PP/dos.x. See example08/ in the examples/ directory for total and projected electronic DOS calculations.

4.4 Tools

Code sumpdos.x is a small utility that sums the partial DOS of selected atoms. See the header of sumpdos.f90 for instructions.

The code path_int.x is intended to be used in the framework of NEB calculations. It is a tool to generate a new path (what is actually generated is the restart file) starting from an old one through interpolation (cubic splines). The new path can be discretized with a different number of images (this is its main purpose), images are equispaced and the interpolation can be also performed on a subsection of the old path. The input file needed by path_int.x can be easily set up with the help of the self-explanatory path_int.sh shell script.

5 Using CP

This section is intended to explain how to perform basic Car-Parrinello (CP) simulations using the CP package.

It is important to understand that a CP simulation is a sequence of different runs, some of them used to "prepare" the initial state of the system, and other performed to collect statistics, or to modify the state of the system itself, i.e. modify the temperature or the pressure.

To prepare and run a CP simulation you should:

1. Define the system:
   (a) atomic positions
   (b) system cell
   (c) pseudopotentials
   (d) number of electrons and bands
   (e) cut-offs

(f) FFT grids

2.

The first run, when starting from scratch, is always an electronic minimization, with fixed ions and cell, to bring the electronic system on the ground state (GS) relative to the starting atomic configuration.
Example of input file (Benzene Molecule):

```
&control
   title = 'Benzene Molecule',
   calculation = 'cp',
   restart_mode = 'from_scratch',
   ndr = 51,
   ndw = 51,
   nstep = 100,
   iprint = 10,
   isave = 100,
   tstress = .TRUE.,
   tprnfor = .TRUE.,
   dt    = 5.0d0,
   etot_conv_thr = 1.d-9,
   ekin_conv_thr = 1.d-4,
   prefix = 'c6h6',
   pseudo_dir='/scratch/benzene/',
   outdir='/scratch/benzene/Out/'
/
&system
   ibrav = 14,
   celldm(1) = 16.0,
   celldm(2) = 1.0,
   celldm(3) = 0.5,
   celldm(4) = 0.0,
   celldm(5) = 0.0,
   celldm(6) = 0.0,
   nat = 12,
   ntyp = 2,
   nbnd = 15,
   nelec = 30,
   ecutwfc = 40.0,
   nr1b= 10, nr2b = 10, nr3b = 10,
   xc_type = 'BLYP'
```

```
/
&electrons
   emass = 400.d0,
   emass_cutoff = 2.5d0,
   electron_dynamics = 'sd'
/
&ions
   ion_dynamics = 'none'
/
&cell
   cell_dynamics = 'none',
   press = 0.0d0,
 /
 ATOMIC_SPECIES
 C 12.0d0 c_blyp_gia.pp
 H 1.00d0 h.ps
 ATOMIC_POSITIONS (bohr)
 C     2.6 0.0 0.0
 C     1.3 -1.3 0.0
 C    -1.3 -1.3 0.0
 C    -2.6 0.0 0.0
 C    -1.3 1.3 0.0
 C     1.3 1.3 0.0
 H     4.4 0.0 0.0
 H     2.2 -2.2 0.0
 H    -2.2 -2.2 0.0
 H    -4.4 0.0 0.0
 H    -2.2 2.2 0.0
 H     2.2 2.2 0.0
```

You can find the description of the input variables in file INPUT_CP in the Doc/ directory.

5.1 Reaching the electronic ground state

The first step in a CP scheme is to reach the electronic ground state (GS), for a given set of nuclear positions. Sometimes a single run is not enough to reach the GS. In this case, you need to re-run the electronic minimization stage. Use the input of the first run, changing restart_mode = 'from_scratch' to restart_mode = 'restart'.

Important Unless you are already experienced with the system you are studying or with the internals of the code, you wil usually need to tune some input parameters, like emass, dt, and cut-offs. For this purpose, a few trial runs could be useful: you can perform short minimizations (say, 10 steps) changing and adjusting these parameters to fit your needs. You can specify the degree of convergence with these two thresholds:

etot_conv_thr: total energy difference between two consecutive steps

ekin_conv_thr: value of the fictitious kinetic energy of the electrons

Usually we consider the system on the GS when ekin conv_thr $< 10^{-5}$. You could check the value of the fictitious kinetic energy on the standard output (column EKINC).

Different strategies are available to minimize electrons, but the most used ones are:

- steepest descent: electron_dynamics = 'sd'
- damped dynamics: electron_dynamics = 'damp', electron_damping = a number typically ranging from 0.1 and 0.5

See the input description to compute the optimal damping factor.

5.2 Relax the system

Once your system is in the GS, depending on how you have prepared the starting atomic configuration:

- if you have set the atomic positions "by hand" and/or from a classical code, check the forces on atoms, and if they are large ($\sim 0.1 \div 1.0$ atomic units), you should perform an ionic minimization, otherwise the system could break up during the dynamics.
- if you have taken the positions from a previous run or a previous ab-initio simulation, check the forces, and if they are too small ($\sim 10^{-4}$ atomic units), this means that atoms are already in equilibrium positions and, even if left free, they will not move. Then you need to randomize positions a little bit (see below).

1.) Minimize ionic positions.

As we pointed out in 4) if the interatomic forces are too high, the system could "explode" if we switch on the ionic dynamics. To avoid that we need to relax the system. Again there are different strategies to relax the system, but the most used are again steepest descent or damped dynamics for ions

and electrons. You could also mix electronic and ionic minimization scheme freely, i.e. ions in steepest and electron in damping or vice versa.

(a) suppose we want to perform a steepest for ions. Then we should specify the following section for ions:

```
&ions
  ion_dynamics = 'sd'
/
```

Change also the ionic masses to accelerate the minimization:

```
ATOMIC_SPECIES
 C 2.0d0 c_blyp_gia.pp
 H 2.00d0 h.ps
```

while leaving other input parameters unchanged. *Note* that if the forces are really high (> 1.0 atomic units), you should always use steepest descent for the first (~ 100 relaxation steps.

(b) as the system approaches the equilibrium positions, the steepest descent scheme slows down, so is better to switch to damped dynamics:

```
&ions
  ion_dynamics = 'damp',
  ion_damping = 0.2,
  ion_velocities = 'zero'
/
```

A typical value of ion damping between 0.05 is good for many systems. It is also better to specify to restart with zero ionic and electronic velocities, since we have changed the masses.

Change further the ionic masses to accelerate the minimization:

```
ATOMIC_SPECIES
C 0.1d0 c_blyp_gia.pp
H 0.1d0 h.ps
```

(c) when the system is really close to the equilibrium, the damped dynamics slow down too, especially because, since we are moving electron and ions together, the ionic forces are not properly correct, then it is often better to perform a ionic step every N electronic steps, or to move ions only when electron are in their GS (within the chosen threshold).

This can be specified by adding, in the ionic section, the ion_nstepe parameter, then the ionic input section become as follows:

```
&ions
  ion_dynamics = 'damp',
  ion_damping = 0.2,
  ion_velocities = 'zero',
  ion_nstepe = 10
/
```

Then we specify in the control input section:

```
  etot_conv_thr = 1.d-6,
  ekin_conv_thr = 1.d-5,
  forc_conv_thr = 1.d-3
```

As a result, the code checks every 10 electronic steps whether the electronic system satisfies the two thresholds etot_conv_thr, ekin_conv_thr: if it does, the ions are advanced by one step. The process thus continues until the forces become smaller than forc_conv_thr.

Note that to fully relax the system you need many run, and different strategies, that you shold mix and change in order to speed-up the convergence. The process is not automatic, but is strongly based on experience, and trial and error.

Remember also that the convergence to the equilibrium positions depends on the energy threshold for the electronic GS, in fact correct forces (required to move ions toward the minimum) are obtained only when electrons are in their GS. Then a small threshold on forces could not be satisfied, if you do not require an even smaller threshold on total energy.

2. Randomization of positions.

If you have relaxed the system or if the starting system is already in the equilibrium positions, then you need to move ions from the equilibrium positions, otherwise they will nott move in a dynamics simulation. After the randomization you should bring electrons on the GS again, in order to start a dynamic with the correct forces and with electrons in the GS. Then you should switch off the ionic dynamics and activate the randomization for each species, specifying the amplitude of the randomization itself. This could be done with the following ionic input section:

```
&ions
  ion_dynamics = 'none',
  tranp(1) = .TRUE.,
  tranp(2) = .TRUE.,
  amprp(1) = 0.01
  amprp(2) = 0.01
/
```

In this way a random displacement (of max 0.01 a.u.) is added to atoms of species 1 and 2. All other input parameters could remain the same. Note that the difference in the total energy (etot) between relaxed and randomized positions can be used to estimate the temperature that will be reached by the system. In fact, starting with zero ionic velocities, all the diffrence is potential energy, but in a dynamics simulation, the energy will be equipartitioned between kinetic and potential, then to estimate the temperature take the difference in energy (de), convert it in Kelvins, divide for the number of atoms and multiply by 2/3. Randomization could be useful also while we are relaxing the system, especially when we suspect that the ions are in a local minimum or in an energy plateau.

5.3 CP dynamics

At this point after having minimized the electrons, and with ions dis- placed from their equilibrium positions, we are ready to start a CP dynamics. We need to specify 'verlet' both in ionic and electronic dynamics. The threshold in control input section will be ignored, like any parameter related to minimization strategy. The first time we perform a CP run after a minimization, it is always better to put velocities equal to zero, unless we have velocities, from a previous simulation, to specify in the input file. Restore the proper masses for the ions. In this way we will sample the microcanonical ensemble. The input section changes as follow:

```
&electrons
   emass = 400.d0,
   emass_cutoff = 2.5d0,
   electron_dynamics = 'verlet',
   electron_velocities = 'zero'
/
&ions
   ion_dynamics = 'verlet',
   ion_velocities = 'zero'
/
ATOMIC_SPECIES
C 12.0d0 c_blyp_gia.pp
H 1.00d0 h.ps
```

If you want to specify the initial velocities for ions, you have to set ion_velocities ='from_input', and add the IONIC_VELOCITIES card, after the ATOMIC_POSITION card, with the list of velocities in atomic units.

IMPORTANT: in restarting the dynamics after the first CP run, remember to remove or comment the velocities parameters:

```
&electrons
   emass = 400.d0,
   emass_cutoff = 2.5d0,
   electron_dynamics = 'verlet'
   ! electron_velocities = 'zero'
/
&ions
   ion_dynamics = 'verlet'
   ! ion_velocities = 'zero'
/
```

otherwise you will quench the system interrupting the sampling of the microcanonical ensemble.

Varying the temperature It is possible to change the temperature of the system or to sample the canonical ensemble fixing the average temperature, this is done using the Nosé thermostat. To activate this thermostat for ions you have to specify in the ions input section:

```
&ions
   ion_dynamics = 'verlet',
   ion_temperature = 'nose',
   fnosep = 60.0,
   tempw = 300.0
/
```

where fnosep is the frequency of the thermostat in THz, thatishould be chosen to be comparable with the center of the vibrational spectrum of the system, in order to excite as many vibrational modes as possible. tempw is the desired average temperature in Kelvin.

Note: to avoid a strong coupling between the Nosé thermostat and the system, proceed step by step. Don't switch on the thermostat from a completely relaxed configuration: adding a random displacement is strongly recommended. Check which is the average temperature via a few steps of a microcanonical simulation. Don't increase the temperature too much. Finally switch on the thermostat. In the case of molecular system, different modes have to be thermalized: it is better to use a chain of thermostat or equivalently running different simulations with different frequencies.

Nośe thermostat for electrons It is possible to specify also the thermostat for the electrons. This is usually activated in metals or in systems where we have a transfer of energy between ionic and electronic degrees of freedom. Beware: the usage of electronic thermostats is quite delicate. The following information comes from K. Kudin:

"The main issue is that there is usually some "natural" fictitious kinetic energy that electrons gain from the ionic motion ("drag"). One could easily quantify how much of the fictitious energy comes from this drag by doing a CP run, then a couple of CG (same as BO) steps, and then going back to CP. The fictitious electronic energy at the last CP restart will be purely due to the drag effect."

"The thermostat on electrons will either try to overexcite the otherwise "cold" electrons, or it will try to take them down to an unnaturally cold state where their fictitious kinetic energy is even below what would be just due pure drag. Neither of this is good."

"I think the only workable regime with an electronic thermostat is a mild overexcitation of the electrons, however, to do this one will need to know rather precisely what is the fictititious kinetic energy due to the drag."

Self-interaction Correction The self-interaction correction (SIC) included in the CP package is based on the Constrained Local-Spin-Density approach proposed my F. Mauri and coworkers (M. D'Avezac et al. PRB 71, 205210 (2005)). It was used for the first time in QUANTUM ESPRESSO by F. Baletto, C. Cavazzoni and S.Scandolo (PRL 95, 176801 (2005)).

This approach is a simple and nice way to treat ONE, and only one, excess charge (EC). It is moreover necessary to check a priori that the spin-up and spin-down eigenvalues are not too different, for the corresponding neutral system. working in the Local-Spin-Density Approximation (setting nspin = 2). If these two conditions are satisfied and you are interest in charged systems, you can apply the SIC. This approach is a on-the-fly method to correct the self-interaction with the excess charge with itself.

Briefly, both the Hartree and the exchange-correlation part have been corrected to avoid the interaction of the EC with tself.

For example, for the Boron atoms, where we have an even number of electrons (valence electrons = 3), the parameters for working with the SIC are:

```
&system
nbnd= 2,
nelec= 3,
nelup = 2,
```

```
neldw = 1,
sic_alpha = 1.d0,
sic_epsilon = 1.0d0,
sic = 'sic_mac',
force_pairing = .true.,

&ions
ion_dynamics = 'none',
ion_radius(1) = 0.8d0,
sic_rloc = 1.0,

ATOMIC_POSITIONS (bohr)
B 0.00 0.00 0.00 0 0 0 1
```

The two main parameters are:
"force_pairing = .true.", which forces the paired electrons to be the same;
"sic='sic_mac'," which instructs the code to use Mauri's correction.
Remember to add an extra-column in ATOMIC_POSITIONS with "1" to activate SIC for those atoms.

Warning: This approach has known problems for dissociation mechanism driven by excess electrons.

Comment 1: Two parameters, "sic_alpha" and "sic_epsilon", have been introduced following the suggestion of M. Sprik (ICR(05)) to treat the radical (OH)-H_2O. In any case, a complete ab-initio approach is followed using "sic_alpha=sic_epsilon=1".

Comment 2: When you apply this SIC scheme to a molecule or to an atom, which are neutral, remember to add the correction to the energy level as proposed by Landau: in a neutral system, subtracting the self-interaction, the unpaired electron feels a charged system, even if using a compensating positive background. For a cubic box, the correction term due to the Madelung energy is approx. given by $1.4186/L_{box} - 1.047/(L_{box})^3$, where L_{box} is the linear dimension of your box (=celldm(1)). The Madelung coefficient is taken from I. Dabo et al. PRB 77, 115139 (2007).

(info by F. Baletto, francesca.baletto@kcl.ac.uk)

5.4 Variable-cell MD

The variable-cell MD is when the Car-Parrinello technique is also applied to the cell. This technique is useful to study system at very high pressure.

5.5 Conjugate Gradient

This page is under construction.

ensemble-DFT The ensemble-DFT (eDFT) is a robust method to simulate the metals in the framework of "ab-initio" molecular dynamics. It was introduced in 1997 by Marzari et al.

The specific subroutines for the eDFT are in ensemble_dft.f90 where you define all the quantities of interest. The subroutine inner_loop_cold.f90 called by cg_sub.f90, control the inner loop, and so the minimization of the free energy A with respect to the occupation matrix.

To select a eDFT calculations, the user has to choice:

```
calculation = 'cp'
occupations= 'ensemble'
tcg = .true.
passop= 0.3
maxiter = 250
```

to use the CG procedure. In the eDFT it is also the outer loop, where thei energy is minimized with respect to the wavefunction keeping fixed the occupation matrix. While the specific parameters for the inner loop. Since eDFT was born to treat metals, keep in mind that we want to describe the broadening of the occupations around the Fermi energy. Below the new parameters in the electrons list, are listed.

- smearing: used to select the occupation distribution; there are two options: Fermi-Dirac smearing='fd', cold-smearing smearing='cs' (recommanded)
- degauss: is the electronic temperature; it controls the broadening of the occupation numbers around the Fermi energy.
- ninner: is the number of iterative cycles in the inner loop, done to minimize the free energy A with respect the occupation numbers. The typical range is 2-8.
- conv_thr: is the threshold value to stop the search of the 'minimum' free energy.
- niter_cold_restart: controls the frequency at which a full iterative inner cycle is done. It is in the range 1-ninner. It is a trick to speed up the calculation.

- lambda_cold: is the length step along the search line for the best value for A, when the iterative cycle is not performed. The value is close to 0.03, smaller for large and complicated metallic systems.

NOTE: degauss is in Hartree, while in PWscf is in Ry (!!!). The tyPical range is 0.01-0.02 Ha.

The input for an Al surface is:

```
&CONTROL
 calculation = 'cp',
 restart_mode = 'from_scratch',
 nstep  = 10,
 iprint = 5,
 isave  = 5,
 dt    = 125.0d0,
 prefix = 'Aluminum_surface',
 pseudo_dir = '~/UPF/',
 outdir = '/scratch/'
 ndr=50
 ndw=51
/
&SYSTEM
 ibrav=  14,
 celldm(1)= 21.694d0, celldm(2)= 1.00D0, celldm(3)= 2.121D0,
 celldm(4)= 0.0d0,   celldm(5)= 0.0d0, celldm(6)= 0.0d0,
 nat= 96,
 ntyp= 1,
 nspin=1,
 ecutwfc= 15,
 nbnd=160,
 nelec=291,
 xc_type = 'pbe'
 occupations= 'ensemble',
 smearing='cs',
 degauss=0.018,
/
&ELECTRONS
 orthogonalization = 'Gram-Schmidt',
 startingwfc = 'random',
 ampre = 0.02,
 tcg = .true.,
 passop= 0.3,
```

```
 maxiter = 250,
 emass_cutoff = 3.00,
 conv_thr=1.d-6
 n_inner = 2,
 lambda_cold = 0.03,
 niter_cold_restart = 2,
/
&IONS
 ion_dynamics  = 'verlet',
 ion_temperature = 'nose'
 fnosep = 4.0d0,
 tempw = 500.d0
/
ATOMIC_SPECIES
 Al 26.89 Al.pbe.UPF
```

NOTA1 remember that the time step is to integrate the ionic dynamics, so you can choose something in the range of 1-5 fs.
NOTA2 with eDFT you are simulating metals or systems for which the occupation number is also fractional, so the number of band, nbnd, has to be chosen such as to have some empty states. As a rule of thumb, start with an initial occupation number of about 1.6-1.8 (the more bands you considera, the more the calculation is accurate, but it also takes longer. The CPU time scales almost linearly with the number of bands.)
NOTA3 the parameter emass_cutoff is used during the preconditioning and it has a completely different meaning with respect to plain CP. It ranges between 4 and 7.

All the other parameters have the same meaning in the usual CP input, and they are discussed above.

5.6 About nr1b, nr2b, nr3b

ecutrho defines the resolution on the real space FFT mesh (as expressed by nr1, nr2 and nr3, that the code left on its own sets automatically). In the ultrasoft case we refer to this mesh as the "hard" mesh, since it is denser than the smooth mesh that is needed to represent the square of the non-norm-conserving wavefunctions.

On this "hard", fine-spaced mesh, you need to determine the size of the cube that will encompass the largest of the augmentation charges - this is what nr1b, nr2b, nr3b are.

So, nr1b is independent of the system size, but dependent on the size

of the augmentation charge (that doesn't vary that much) and on the real-space resolution needed by augmentation charges (rule of thumb: ecutrho is between 6 and 12 times ecutwfc).

In practice, nr1b et al. are often in the region of 20-24-28; testing seems again a necessity (unless the code started automagically to estimate these).

The core charge is in principle finite only at the core region (as defined by rcut) and vanishes out side the core. Numerically the charge is represented in a Fourier series which may give rise to small charge oscillations outside the core and even to negative charge density, but only if the cut-off is too low. Having these small boxes removes the charge oscillations problem (at least outside the box) and also offers some numerical advantages in going to higher cut-offs.

The small boxes should be set as small as possible, but large enough to contain the core of the largest element in your system. The formula for determining the box size is quite simple:

$$nr1b = (2 \times rcut)/Lx \times nr1$$

where rcut is the cut-off radius for the largest element and Lx is the physical length of your box along the x axis. You have to round your result to the nearest larger integer." (info by Nicola Marzari)

6 Performance issues (PWscf)

6.1 CPU time requirements

The following holds for code pw.x and for non-US PPs. For US PPs there are additional terms to be calculated, that may add from a few percent up to 30-403N_{at} modes requires a CPU time of the same order of that required by a self-consistent calculation in the same system. For cp.x, the required CPU time of each time step is in the order of the time $T_h + T_{orth} + T_{sub}$ defined below.

The computer time required for the self-consistent solution at fixed ionic positions, T_{scf} , is:

$$T_{scf} = N_{iter} T_{iter} + T_{init}$$

where N_{iter} = niter = number of self-consistency iterations, T_{iter} = CPU time for a single iteration, T_{init} = initialization time for a single iteration. Usually $T_{init} << N_{iter} T_{iter}$.

The time required for a single self-consistency iteration T_{iter} is:

$$T_{iter} = N_k T_{diag} + T_{rho} + T_{scf}$$

where N_k = number of k-points, T_{diag} = CPU time per hamiltonian iterative diagonalization, T_{rho} = CPU time for charge density calculation, T_{scf} = CPU time for Hartree and exchange-correlation potential calculation.

The time for a Hamiltonian iterative diagonalization T_{diag} is:

$$T_{diag} = N_h T_h + T_{orth} + T_{sub}$$

where N_h = number of $H\psi$ products needed by iterative diagonalization, T_h = CPU time per $H\psi$ product, T_{orth} = CPU time for orthonormalization, T_{sub} = CPU time for subspace diagonalization.

The time T_h required for a $H\psi$ product is

$$T_h = a_1 MN + a_2 MN_1N_2N_3 log(N_1N_2N_3) + a_3 MPN.$$

The first term comes from the kinetic term and is usually much smaller than the others. The second and third terms come respectively from local and nonlocal potential. a_1, a_2, a_3 are prefactors, M = number of valence bands, N = number of plane waves (basis set dimension), N_1, N_2, N_3 = dimensions of the FFT grid for wavefunctions ($N_1N_2N_3 \sim 8N$), P = number of projectors for PPs (summed on all atoms, on all values of the angular momentum l, and m = 1, . . . , 2l + 1)

The time T_{orth} required by orthonormalization is

$$T_{orth} = b_1 NM_x^2$$

and the time T_{sub} required by subspace diagonalization is

$$T_{sub} = b_2 M_x^3$$

where b_1 and b_2 are prefactors, M_x = number of trial wavefunctions (this will vary between M and a few times M , depending on the algorithm).

The time T_{rho} for the calculation of charge density from wavefunctions is

$$T_{rho} = c_1 MN_{r1}N_{r2}N_{r3} log(N_{r1}N_{r2}N_{r3}) + c_2 MN_{r1}N_{r2}N_{r3} + T_{us}$$

where c_1, c_2, c_3 are prefactors, N_{r1}, N_{r2}, N_{r3} = dimensions of the FFT grid for charge density ($N_{r1}N_{r2}N_r3 \sim 8N_g$, where $N_g¿$ = number of G-vectors for the charge density), and T_{us} = CPU time required by ultrasoft contribution (if any).

The time T_{scf} for calculation of potential from charge density is

$$T_{scf} = d_2 N_{r1}N_{r2}N_{r3} + d_3 N_{r1}N_{r2}N_{r3} log(N_{r1}N_{r2}N_{r3})$$

where d_1, d_2 are prefactors.

6.2 Memory requirements

A typical self-consistency or molecular-dynamics run requires a maximum memory in the order of O double precision complex numbers, where

$$O = mMN + PN + pN_1N_2N_3 + qN_{r1}N_{r2}N_{r3}$$

with m, p, q = small factors; all other variables have the same meaning as above. Note that if the $\Gamma-$point only (q = 0) is used to sample the Brillouin Zone, the value of N will be cut into half.

The memory required by the phonon code follows the same patterns, with somewhat larger factors m, p, q .

6.3 File space requirements

A typical pw.x run will require an amount of temporary disk space in the order of O double precision complex numbers:

$$O = N_kMN + qN_{r1}N_{r2}N_{r3}$$

where $q = 2\times$ mixing_ndim (number of iterations used in self-consistency, default value = 8) if disk_io is set to 'high' or not specified; q = 0 if disk_io='low' or 'minimal'.

6.4 Parallelization issues

pw.x and cp.x can run in principle on any number of processors. The effectiveness of parallelization is ultimately judged by the "scaling", i.e. how the time needed to perform a job scales with the number of processors, and depends upon:

- the size and type of the system under study;
- the judicious choice of the various levels of parallelization (detailed in the "Running on parallel machines" sections);
- the availability of fast interprocess communications (or lack thereof).

Ideally one would like to have linear scaling, i.e. $T_N \sim T_0/N_p$ for N_p processors. In addition, one would like to have linear scaling of the RAM per processor: $O_N \sim O_0/N_p$, so that large-memory systems fit into the RAM of each processor.

As a general rule, image parallelization:

- may give good scaling, but the slowest image will determine the overall performances ("load balancing" may be a problem);
- requires very little communications (suitable for ethernet communications);
- does not reduce the required memory per processor (unsuitable for large-memory jobs).

Parallelization on k-points:

- guarantees (almost) linear scaling if the number of k-points is a multiple of the number of pools;
- requires little communications (suitable for ethernet communications);
- does not reduce the required memory per processor (unsuitable for large-memory jobs).

Parallelization on plane-waves:

- yields good to very good scaling, especially if the number of processors in a pool is a divisor of N_3 and N_{r3} (the dimensions along the z-axis of the FFT grids, which may coincide);
- requires heavy communications (suitable for Gigabit ethernet up to 4, 8 CPUs at most, specialized communication hardware needed for 8 or more processors);
- yields almost linear reduction of memory per processor with the number of processors in the pool.

A note on scaling: optimal serial performances are achieved when the data are as much as possible kept into the cache. As a side effect, plane-wave parallelization may yield superlinear (better than linear) scaling, thanks to the increase in serial speed coming from the reduction of data size (making it easier for the machine to keep data in the cache).

For each system there is an optimal range of number of processors on which to run the job. A too large number of processors will yield performance degradation. If the size of pools is especially delicate: N_p should not exceed by much N_3 and N_{r3}. For large jobs, it is convenient to further subdivide a pool of processors into "task groups". When the number of processors exceeds the number of FFT planes, data can be redistributed to "task groups" so that each group can process several wavefunctions at the same time.

The optimal number of processors for the "ortho" (cp.x) or "ndiag" (pw.x) parallelization, taking care of linear algebra operations involving $M \times M$ matrices, is automatically chosen by the code.

Actual parallel performances will also depend a lot on the available software (MPI libraries) and on the available communication hardware. For Beowulf-style machines (clusters of PC) the newest version 1.1 and later of the OpenMPI libraries (http://www.openmpi.org/) seems to yield better performances than other implementations (info by Kostantin Kudin). Note however that you need a decent communication hardware (at least Gigabit ethernet) in order to have acceptable performances with PW parallelization. Do not expect good scaling with cheap hardware: plane-wave calculations are by no means an "embarrassing parallel" problem.

Also note that multiprocessor motherboards for Intel Pentium CPUs typically have just one memory bus for all processors. This dramatically slows down any code doing massive access to memory (as most codes in the Quantum ESPRESSO distribution do) that runs on processors of the same motherboard.

7 Troubleshooting

Almost all problems in Quantum ESPRESSO arise from incorrect input data and result in error stops. Error messages should be self-explanatory, but unfortunately this is not always true. If the code issues a warning messages and continues, pay attention to it but do not assume that something is necessarily wrong in your calculation: most warning messages signal harmless problems.

Typical pw.x and/or ph.x (mis-)behavior:

7.1 pw.x says 'error while loading shared libraries' or 'cannot open shared object file' and does not start

Possible reasons:

- If you are running on the same machines on which the code was compiled, this is a library configuration problem. The solution is machine-dependent. On Linux, find the path to the missing libraries; then either add it to file /etc/ld.so.conf and run ldconfig (must be done as root), or add it to variable LD_LIBRARY_PATH and export it. Another possibility is to load non-shared version of libraries (ending with .a) instead of shared ones (ending with .so).

- If you are *not* running on the same machines on which the code was compiled: you need either to have the same shared libraries installed on both machines, or to load statically all libraries (using appropriate `configure` or loader options). The same applies to Beowulf-style parallel machines: the needed shared libraries must be present on all PCs.

7.2 errors in examples with parallel execution

If you get error messages in the example scripts – i.e. not errors in the codes – on a parallel machine, such as e.g.:
"run example: -n: command not found"
you may have forgotten the "" in the definitions of PARA_PREFIX and PARA_POSTFIX.

7.3 pw.x prints the first few lines and then nothing happens (parallel execution)

If the code looks like it is not reading from input, maybe it isn't: the MPI libraries need to be properly configured to accept input redirection. See section "Running on parallel machines", or inquire with your local computer wizard (if any).

7.4 pw.x stops with error while reading data

There is an error in the input data, typically:

- a misspelled namelist variable,
- an empty input file.

Unfortunately with most compilers the code just reports "Error while reading XXX namelist" and no further useful information. Here are some other more subtle sources of trouble:

- Out-of-bound indices in dimensioned variables read in the namelist
- Input data files containing $\hat{\text{M}}$ (Control-M) characters at the end of lines, or non-ASCII characters (e.g. non-ASCII quotation marks, that at a first glance may look the same as the ASCII character). Typically, this happens with files coming from Windows or produced with "smart" editors.

Both may cause the code to crash with rather mysterious error messages. If none of the above applies and the code stops at the first namelist ("control") and you are running in parallel: your MPI libraries might not be properly configured to allow input redirection, see the previous item above this one.

7.5 pw.x mumbles something like 'cannot recover' or 'error reading recover file'

You are trying to restart from a previous job that either produced corrupted files, or did not do what you think it did. No luck: you have to restart from scratch.

7.6 pw.x stops with 'inconsistent DFT' error

As a rule, the flavor of DFT used in the calculation should be the same as the one used in the generation of PPs, and all PPs should be generated using the same flavor of DFT. This is actually enforced: the type of DFT is read from PP files and it is checked that the same DFT is read from all PPs. If this does not hold, the code stops with the above error message.

If you really want to use PPs generated with different DFT, or to perform a calculation with a DFT that differs from what used in PP generation, change the appropriate field in the PP file(s), at your own risk.

7.7 pw.x stops with error in cdiaghg or rdiaghg

Possible reasons for such behavior are not always clear, but they typically fall into one of the following cases:

- serious error in data, such as bad atomic positions or bad crystal structure/supercell;
- a bad PP, typically with a ghost, but also a US-PP with non-positive charge density, leading to a violation of positiveness of the S matrix appearing in the US-PP formalism;
- a failure of the algorithm performing subspace diagonalization. The LAPACK algorithms used by cdiaghg/rdiaghg are very robust and extensively tested. Still, it may seldom happen that such algorithms fail. Try to use conjugate-gradient diagonalization (diagonalization='cg'), a slower but very robust algorithm, and see what happens.

- buggy libraries. Machine-optimized mathematical libraries are very fast but sometimes not so robust from a numerical point of view. Suspicious behavior: you get an error that is not reproducible on other architectures or that disappears if the calculation is repeated with even minimal changes in parameters. Known cases: HP-Compaq alphas with cxml libraries, Mac OS-X with system blas/lapack. Try to use compiled BLAS and LAPACK (or better, ATLAS) instead of machine-optimized libraries.

7.8 pw.x crashes with no error message at all

This happens quite often in parallel execution, or under a batch queue, or if you are writing the output to a file. When the program crashes, part of the output, including the error message, may be lost, or hidden into error files where nobody looks into. It is the fault of the operating system, not of the code. Try to run interactively and to write to the screen. If this doesn't help, move to next point.

7.9 pw.x crashes with 'segmentation fault' or similarly obscure messages

Possible reasons:

- too much RAM memory requested, or too much stack memory requested (see next item).
- if you are using highly optimized mathematical libraries, verify that they are designed for your hardware. In particular, for Intel compiler and MKL libraries, verify that you loaded the correct set of CPU-specific MKL libraries.
- buggy compiler. If you are using Portland or Intel compilers on Linux PCs or clusters, see the Installation section.

7.10 pw.x crashes with 'error in davcio'

'davcio' is the routine that performs most of the I/O operations (read from disk and write to disk) in pw.x ; 'error in davcio 'means a failure of an I/O operation.

- If the error is reproducible and happens at the beginning of a calculation: check if you have read/write permission to the scratch directory

specified in variable 'outdir'. Also: check if there is enough free space available on the disk you are writing to, and check your disk quota (if any)

- If the error is irreproducible: your might have flaky disks; if you are writing via the network using nfs (which you shouldn't do anyway), your network connection might be not so stable, or your nfs implementation is unable to work under heavy load

- if it happens while restarting from a previous calculation: you might be restarting from the wrong place, or from wrong data, or the files might be corrupted.

7.11 pw.x works for simple systems, but not for large systems or whenever more RAM is needed

Possible solutions:

- increase the amount of RAM you are authorized to use (which may be much smaller than the available RAM). Ask your system administrator if you don't know what to do.

- reduce nbnd to the strict minimum, or reduce the cutoffs, or the cell size

- use conjugate-gradient (diagonalization='cg': slow but very robust): it requires less memory than the default Davidson algorithm.

- in parallel execution, use more processors, or use the same number of processors with less pools. Remember that parallelization with respect to k-points (pools) does not distribute memory: parallelization with respect to R- (and G-) space does.

- IBM only (32-bit machines): if you need more than 256 MB you must specify it at link time (option -bmaxdata).

- buggy or weird-behaving compiler. Some versions of the Portland and Intel compilers on Linux PCs or clusters have this problem. For Intel ifort 8.1 and later, the problem seems to be due to the allocation of large automatic arrays that exceeds the available stack. Increasing the stack size (with commands limits or ulimit) may (or may not) solve the problem. Versions > 3.2 try to avoid this problem by removing the stack size limit at startup. See:

http://www.democritos.it/pipermail/pw_forum/2007-September/007176.html,
http://www.democritos.it/pipermail/pw_forum/2007-September/007179.html

7.12 pw.x crashes in parallel execution with an obscure message related to MPI errors

Random crashes due to MPI errors have often been reported, typically in Linux PC clusters. We cannot rule out the possibility that bugs in QUANTUM ESPRESSO cause such behavior, but we are quite confident that the most likely explanation is a hardware problem (defective RAM for instance) or a software bug (in MPI libraries, compiler, operating system).

Debugging a parallel code may be difficult, but you should at least verify if your problem is reproducible on different architectures/software configurations/input data sets, and if there is some particular condition that activates the bug. If this doesn't seem to happen, the odds are that the problem is not in QUANTUM ESPRESSO. You may still report your problem, but consider that reports like "it crashes with...(obscure MPI error)" contain 0 bits of information and are likely to get 0 bits of answers.

Concerning MPI libraries in particular, useful information can be found in Axel's web site:
http://www.theochem.rub.de/ axel.kohlmeyer/cpmd-linux.html, and in the following message by Javier Antonio Montoya:
http://www.democritos.it/pipermail/pw_forum/2008-April/008818.html

7.13 pw.x runs but nothing happens

Possible reasons:

- in parallel execution, the code died on just one processor. Unpredictable behavior may follow.
- in serial execution, the code encountered a floating-point error and goes on producing NaNs (Not a Number) forever unless exception handling is on (and usually it isn't). In both cases, look for one of the reasons given above.
- maybe your calculation will take more time than you expect.

7.14 pw.x yields weird results

Possible solutions:

- if this happens after a change in the code or in compilation or preprocessing options, try 'make clean', recompile. The 'make' command should take care of all dependencies, but do not rely too heavily on it. If the problem persists, 'make clean', recompile with reduced optimization level.
- maybe your input data are weird.

7.15 pw.x stops with error message "the system is metallic, specify occupations"

You did not specify state occupations, but you need to, since your system appears to have an odd number of electrons. The variable controlling how metallicity is treated is occupations in namelist &SYSTEM. The default, occupations='fixed', occupies the lowest nelec/2 states and works only for insulators with a gap. In all other cases, use 'smearing' or 'tetrahedra'. See file INPUT_PW for more details.

7.16 pw.x stops with "internal error: cannot braket Ef " in efermig

Possible reasons:

- serious error in data, such as bad number of electrons, insufficient number of bands, absurd value of broadening;
- the Fermi energy is found by bisection assuming that the integrated DOS N(E) is an increasing function of the energy. This is not guaranteed for Methfessel-Paxton smearing of order 1 and can give problems when very few k-points are used. Use some other smearing function: simple Gaussian broadening or, better, Marzari-Vanderbilt 'cold smearing'.

7.17 pw.x yields 'internal error: cannot braket Ef' message in efermit, then stops because 'charge is incorrect'

There is either a serious error in data (bad number of electrons, insufficient number of bands), or too few tetrahedra (i.e. k-points). The tetrahedron method may become unstable in the latter case, especially if the bands are very narrow. Remember that tetrahedra should be used only in conjunction with uniform k-point grids.

7.18 pw.x yields 'internal error: cannot braket Ef' message in efermit but does not stop

This may happen under special circumstances when you are calculating the band structure for selected high-symmetry lines. The message signals that occupations and Fermi energy are not correct (but eigenvalues and eigenvectors are). Remove occupations='tetrahedra' in the input data to get rid of the message.

7.19 the FFT grids in pw.x are machine-dependent

Yes, they are! The code automatically chooses the smallest grid that is compatible with the specified cutoff in the specified cell, and is an allowed value for the FFT library used. Most FFT libraries are implemented, or perform well, only with dimensions that factors into products of small numers (2, 3, 5 typically, sometimes 7 and 11). Different FFT libraries follow different rules and thus different dimensions can result for the same system on different machines (or even on the same machine, with a different FFT). See function allowed in Modules/fft_scalar.f90.

As a consequence, the energy may be slightly different on different machines. The only piece that explicitly depends on the grid parameters is the XC part of the energy that is computed numerically on the grid. The differences should be small, though, especially for LDA calculations.

Manually setting the FFT grids to a desired value is possible, but slightly tricky, using input variables nr1, nr2, nr3 and nr1s, nr2s, nr3s. The code will still increase them if not acceptable. Automatic FFT grid dimensions are slightly overestimated, so one may try *very carefully* to reduce them a little bit. The code will stop if too small values are required, it will waste CPU time and memory for too large values.

Note that in parallel execution, it is very convenient to have FFT grid dimensions along "z" that are a multiple of the number of processors.

7.20 pw.x does not find all the symmetries you expected

pw.x determines first the symmetry operations (rotations) of the Bravais lattice; then checks which of these are symmetry operations of the system (including if needed fractional translations). This is done by rotating (and translating if needed) the atoms in the unit cell and verifying if the rotated unit cell coincides with the original one.

Assuming that your coordinates are correct (please carefully check!), you may not find all the symmetries you expect because:

- the number of significant figures in the atomic positions is not large enough. In file PW/eqvect.f90, the variable accep is used to decide whether a rotation is a symmetry operation. Its current value (10^{-5}) is quite strict: a rotated atom must coincide with another atom to 5 significant digits. You may change the value of accep and recompile.

- they are not acceptable symmetry operations of the Bravais lattice. This is the case for C_{60}, for instance: the I_h icosahedral group of C_{60} contains 5-fold rotations that are incompatible with translation symmetry.

- the system is rotated with respect to symmetry axis. For instance: a C_{60} molecule in the fcc lattice will have 24 symmetry operations (T_h group) only if the double bond is aligned along one of the crystal axis; if C_{60} is rotated in some arbitrary way, pw.x may not find any symmetry, apart from inversion.

- they contain a fractional translation that is incompatible with the FFT grid (see next paragraph). Note that if you change cutoff or unit cell volume, the automatically computed FFT grid changes, and this may explain changes in symmetry (and in the number of k-points as a consequence) for no apparent good reason (only if you have fractional translations in the system, though).

- a fractional translation, without rotation, is a symmetry operation of the system. This means that the cell is actually a supercell. In this case, all symmetry operations containing fractional translations are disabled. The reason is that in this rather exotic case there is no simple way to select those symmetry operations forming a true group, in the mathematical sense of the term.

7.21 warning: 'symmetry operation # N not allowed'

This is not an error. If a symmetry operation contains a fractional translation that is incompatible with the FFT grid, it is discarded in order to prevent problems with symmetrization. Typical fractional translations are 1/2 or 1/3 of a lattice vector. If the FFT grid dimension along that direction is not divisible respectively by 2 or by 3, the symmetry operation will not transform the FFT grid into itself.

7.22 I do not get the same results in different machines!

If the difference is small, do not panic. It is quite normal for iterative methods to reach convergence through different paths as soon as anything changes. In particular, between serial and parallel execution there are operations that are not performed in the same order. As the numerical accuracy of computer numbers is finite, this can yield slightly different results.

It is also normal that the total energy converges to a better accuracy than its terms, since only the sum is variational, i.e. has a minimum in correspondence to ground-state charge density. Thus if the convergence threshold is for instance 10^{-8}, you get 8-digit accuracy on the total energy, but one or two less on other terms (e.g. XC and Hartree energy). It this is a problem for you, reduce the convergence threshold for instance to 10^{-10} or 10^{-12}. The differences should go away (but it will probably take a few more iterations to converge).

7.23 the CPU time is time-dependent!

Yes it is! On most machines and on most operating systems, depending on machine load, on communication load (for parallel machines), on various other factors (including maybe the phase of the moon), reported CPU times may vary quite a lot for the same job. Also note that what is printed is supposed to be the CPU time per process, but with some compilers it is actually the wall time.

7.24 'warning : N eigenvectors not converged ...'

This is a warning message that can be safely ignored if it is not present in the last steps of self-consistency. If it is still present in the last steps of self-consistency, and if the number of unconverged eigenvector is a significant part of the total, it may signal serious trouble in self-consistency (see next point) or something badly wrong in input data.

7.25 'warning : negative or imaginary charge...', or '...core charge ...', or 'npt with rhoup<0...' or 'rho dw<0...'

These are warning messages that can be safely ignored unless the negative or imaginary charge is sizable, let us say of the order of 0.1. If it is, something seriously wrong is going on. Otherwise, the origin of the negative charge

is the following. When one transforms a positive function in real space to Fourier space and truncates at some finite cutoff, the positive function is no longer guaranteed to be positive when transformed back to real space. This happens only with core corrections and with ultrasoft pseudopotentials. In some cases it may be a source of trouble (see next point) but it is usually solved by increasing the cutoff for the charge density.

7.26 self-consistency is slow or does not converge

See the corresponding FAQ item

7.27 structural optimization is slow or does not converge

Typical structural optimizations, based on the BFGS algorithm, converge to the default thresholds (etot_conv_thr and forc_conv_thr) in 15-25 BFGS steps (depending on the starting configuration). This may not happen when your system is characterized by "floppy" low-energy modes, that make very difficult (and of little use anyway) to reach a well converged structure, no matter what. Other possible reasons for a problematic convergence are listed below.

Close to convergence the self-consistency error in forces may become large with respect to the value of forces. The resulting mismatch between forces and energies may confuse the line minimization algorithm, which assumes consistency between the two. The code reduces the starting self-consistency threshold conv thr when approaching the minimum energy configuration, up to a factor defined by upscale. Reducing conv_thr (or increasing upscale) yields a smoother structural optimization, but if conv_thr becomes too small, electronic self-consistency may not converge. You may also increase variables etot_conv_thr and forc_conv_thr that determine the threshold for convergence (the default values are quite strict).

A limitation to the accuracy of forces comes from the absence of perfect translational invariance. If we had only the Hartree potential, our PW calculation would be translationally invariant to machine precision. The presence of an exchange-correlation potential introduces Fourier components in the potential that are not in our basis set. This loss of precision (more serious for gradient-corrected functionals) translates into a slight but detectable loss of translational invariance (the energy changes if all atoms are displaced by the same quantity, not commensurate with the FFT grid). This sets a limit to the accuracy of forces. The situation improves somewhat by increasing the ecutrho cutoff.

7.28 pw.x stops during variable-cell optimization in checkallsym with 'non orthogonal operation' error

Variable-cell optimization may occasionally break the starting symmetry of the cell. When this happens, the run is stopped because the number of k-points calculated for the starting configuration may no longer be suitable. Possible solutions:

- start with a nonsymmetric cell;
- use a symmetry-conserving algorithm: the Wentzcovitch algorithm (cell dynamics='damp-w') should not break the symmetry.

7.29 some codes in PP/ complain that they do not find some files

For Linux PC clusters in parallel execution: in at least some versions of MPICH, the current directory is set to the directory where the executable code resides, instead of being set to the directory where the code is executed. This MPICH weirdness may cause unexpected failures in some postprocessing codes that expect a data file in the current directory. Workaround: use symbolic links, or copy the executable to the current directory.

7.30 ph.x stops with 'error reading file'

The data file produced by pw.x is bad or incomplete or produced by an incompatible version of the code. In parallel execution: if you did not set wf collect=.true., the number of processors and pools for the phonon run should be the same as for the self-consistent run; all files must be visible to all processors.

7.31 ph.x mumbles something like 'cannot recover' or 'error reading recover file'

You have a bad restart file from a preceding failed execution. Remove all files recover* in outdir.

7.32 ph.x says 'occupation numbers probably wrong' and continues; or 'phonon + tetrahedra not implemented' and stops

You have a metallic or spin-polarized system but occupations are not set to 'smearing'. Note that the correct way to calculate occupancies must be specified in the input data of the non-selfconsistent calculation, if the phonon code reads data from it. The non-selfconsistent calculation will not use this information but the phonon code will.

7.33 ph.x does not yield acoustic modes with $\omega = 0$ at $q = 0$

This may not be an error: the Acoustic Sum Rule (ASR) is never exactly verified, because the system is never exactly translationally invariant as it should be. The calculated frequency of the acoustic mode is typically less than 10 cm^{-1}, but in some cases it may be much higher, up to 100 cm^{-1}. The ultimate test is to diagonalize the dynamical matrix with program dynmat.x, imposing the ASR. If you obtain an acoustic mode with a much smaller ω (let us say $< 1cm^{-1}$) with all other modes virtually unchanged, you can trust your results.

"The problem is [...] in the fact that the exchange and correlation energy is computed in real space on a discrete grid and hence the total energy is invariant (...) only for translation in the FFT grid. Increasing the charge density cutoff increases the grid density thus making the integral more exact thus reducing the problem, unfortunately rather slowly...This problem is usually more severe for GGA than with LDA because the GGA functionals have functional forms that vary more strongly with the position; particularly so for isolated molecules or system with significant portions of "vacuum" because in the exponential tail of the charge density a) the finite cutoff (hence there is an effect due to cutoff) induces oscillations in rho and b) the reduced gradient is diverging."(info by Stefano de Gironcoli, June 2008)

7.34 ph.x yields really lousy phonons, with bad or negative frequencies or wrong symmetries or gross ASR violations

Possible reasons

- if this happens only for acoustic modes at $q = 0$ that should have $\omega = 0$: Acoustic Sum Rule violation, see the item before this one

- wrong data file read.
- wrong atomic masses given in input will yield wrong frequencies (but the content of file fildyn should be valid, since the force constants, not the dynamical matrix, are written to file).
- convergence threshold for either SCF (conv_thr) or phonon calculation (tr2_ph) too large: try to reduce them.
- maybe your system does have negative or strange phonon frequencies, with the approximations you used. A negative frequency signals a mechanical instability of the chosen structure. Check that the structure is reasonable, and check the following parameters:
  - The cutoff for wavefunctions, ecutwfc
  - For US PP: the cutoff for the charge density, ecutrho
  - The k-point grid, especially for metallic systems!

Note that "negative" frequencies are actually imaginary: the negative sign flags eigenvalues of the dynamical matrix for which $\omega^2 < 0$.

7.35 'Wrong degeneracy' error in star_q

Verify the q-point for which you are calculating phonons. In order to check whether a symmetry operation belongs to the small group of q, the code compares q and the rotated q, with an acceptance tolerance of 10^{-5} (set in routine PW/eqvect.f90). You may run into trouble if your q-point differs from a high-symmetry point by an amount in that order of magnitude.

8 Frequently Asked Questions (FAQ)

8.1 Installation

Most installation problems have obvious origins and can be solved by reading error messages and acting accordingly. Sometimes the reason for a failure is less obvious. In such a case, you should look into the Installation Issues section of the User Guide, and into the pw_forum archive to see if a similar problem (with solution) is described. If you get really weird error messages during installation, look for them with your preferred Internet search engine (such as Google): very often you will find an explanation and a workaround.

"What Fortran compiler do I need to compile Q-E?" "'A:"' Any non-buggy, or not-too-buggy, fortran-95 compiler should work, with minimal or no changes to the code. `configure` may not be able to recognize your system, though.

"Why is `configure` saying that I have no fortran compiler?" "'A:"'Because you haven't one (really!); or maybe you have one, but it is not your execution path; or maybe it has been given an unusual name by your system manager. Install a compiler if you have none; if you have one, fix your execution path, or define an alias if it has a strange name.

"Why is `configure` saying that my fortran compiler doesn't work?" "'A:"' Because it doesn't work. Really! More exactly, `configure` has tried to compile a small test program and didn't succeed. Your compiler may not be properly installed. For Intel compiler on PC's: you may have forgotten to run the required initialization script for the compiler.

"`configure` says 'unsupported architecture', what should I do?" "'A:"' If compilation/linking still works, never mind, Otherwise, supply a suitable supported architecture to `configure`: see instructions in README.configure on what to do. Note that in most cases you may use `configure` to produce dependencies, then edit the file make.sys.

"Why doesn't `configure` recognize that I have a parallel machine?" "'A:"'You need a properly configured complete parallel environment. If any piece is missing, `configure` will revert to serial compilation. In particular:

- `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 most PC clusters (Beowulf), there is no parallel Fortran-95 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 logical places with logical names, but if they have strange names or strange locations, you will have to rename/move them, or to instruct `configure` to find them (see subsection "Libraries"). Note that 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 fail, `configure` will revert to serial compilation.

"Compilation fails with "internal error", what should I do?" "'A:"' Any message saying something like "internal compiler error" means that your compiler is buggy. If you paid real money for your compiler, complain with the software vendor. If not: sometimes reducing the optimization level will do the trick, sometimes rearranging the code solves the problem, but most often you will need to update your compiler to a less buggy version.

"Compilation fails at linking stage: "symbol ... not found"" "'A:"' If the missing symbols (i.e. routines that are called but not found) are in the code itself: most likely the fortran-to-C conventions used in file include/c_defs.h are not appropriate. Change them and retry.

If the missing symbols are in external libraries (Blas, Lapack, FFT, MPI libraries): there is a name mismatch between what the compiler expects and what the library provides. This case is described in detail in the User Guide.

If the missing symbols aren't found anywhere either in the code or in the libraries: they are system library symbols. i) If they are called by external libraries, you need to add a missing system library, or to use a different set of external libraries, compiled with the same compiler you are using. ii) If you are using no external libraries and still getting missing symbols, your compiler and compiler libraries are not correctly installed.

8.2 Pseudopotentials

"Can I mix Ultrasoft/Norm-Conserving/PAW pseudopotentials?" "'A:"' Yes, you can (if implemented, of course: a few calculations are not available with USPP, a few more are not for PAW). A small restrictions exists in cp.x, expecting atoms with USPP listed before those with NCPP, which in turn are expected before local PP's (if any). Otherwise you can mix and match, as long as the XC functional used in the generation of the PP is the same for all PP's. Note that it is the hardest atom that determines the cutoff.

"Where can I find pseudopotentials for atom X?" "'A:"' See the wiki page on [[Pseudopotentials]], follow those links. New contributions to

the PP table are appreciated. If X is one of the rare earths: please consider first if DFT is suitable for your system!

8.3 Input data

A large percentage of the problems reported to the mailing list are caused by incorrect input data. Before reporting a problem with strange crashes or strange results, PLEASE have a look at your structure with XCrysDen. XCrysDen can directly visualise the structure from both PWscf input data:

```
xcrysden --pwi "input-data-file"
```

and from PWscf output as well:

```
xcrysden --pwo "output-file".
```

Unlike most other visualizers, XCrysDen is periodicity-aware: you can easily visualize periodically repeated cells. You are advised to use always XCrysDen to check your input data!

"Where can I find the crystal structure/atomic positions of XYZ?" "'A:"' The following site contains a lot of crystal structures: http://cst-www.nrl.navy.mil/lattice .
"Since this seems to come up often, I'd like to point out that the American Mineralogist Crystal Structure Database (http://rruff.geo.arizona.edu/AMS/amcsd) is another excellent place to find structures, though you will have to use it in conjunction with Bilbao (http://www.cryst.ehu.es), and have some understanding of space groups and Wyckoff positions".

"Where can I find the Brillouin Zone/high-symmetry points/irreps for XYZ?" "'A:"' "You might find this web site useful: http://www.cryst.ehu.es/cryst/get_kvec.html" (info by Cyrille Barreteau, nov. 2007). Or else: in textbooks, such as e.g. "The mathematical theory of symmetry in solids", by Bradley and Cracknell.

"Where can I find Monkhorst-Pack grids of k-points?" "'A:"' Auxiliary code "kpoints.x" (in pwtools/, executable produced by "make tools") generates uniform grids of k-points that are equivalent to Monkhorst-Pack grids.

"How do I perform a calculation with spin-orbit interactions?" "'A:"' The following input variables are relevant for a spin-orbit calculation:

```
noncolin=.true./.false.
lspinorb=.true./.false.
starting_magnetization (one for each type of atoms)
```

To make a spin-orbit calculation noncolin must be true. If starting_magnetization is set to zero (or not given) the code makes a spin orbit calculation without spin magnetization (it assumes that time reversal symmetry holds and it does not calculate the magnetization). The states are still two component spinors but the total magnetization is zero.

If starting_magnetization is different from zero it makes a non collinear spin polarized calculation with spin orbit. The final spin magnetization might be zero or different from zero depending on the system.

Furthermore to make a spin-orbit calculation you must use fully relativistic pseudopotentials at least for the atoms in which you think that spin orbit is large. If all the pseudopotentials are scalar relativistic the calculation becomes equivalent to a noncolinear calculation without spin orbit. (Andrea Dal Corso, 2007-07-27)

"How can I choose parameters for variable-cell molecular dynamics?" "'A"': "A common mistake many new users make is to set the time step dt improperly to the same order of magnitude as for CP algorithm, or not setting dt at all. This will produce a "not evolving dynamics". Good values for the original RMW (RM Wentzcovitch) dynamics are dt= $50 \div 70$. The choice of the cell mass is a delicate matter. An off-optimal mass will make convergence slower. Too small masses, as well as too long time steps, can make the algorithm unstable. A good cell mass will make the oscillation times for internal degrees of freedom comparable to cell degrees of freedom in non-damped Variable-Cell MD. Test calculations are advisable before extensive calculation. I have tested the damping algorithm that I have developed and it has worked well so far. It allows for a much longer time step (dt=$100 \div 150$) than the RMW one and is much more stable with very small cell masses, which is useful when the cell shape, not the internal degrees of freedom, is far out of equilibrium. It also converges in a smaller number of steps than RMW." (Info from Cesar Da Silva: the new damping algorithm is the default since v. 3.1).

8.4 Parallel execution

Effective usage of parallelism requires some basic knowledge on how parallel machines work and how parallelism is implemented in QUANTUM ESPRESSO. If you have no experience and no clear ideas (or not idea at all), consider reading the section of the User Guide explaining some basic parallelism for QUANTUM ESPRESSO.

"Why is my parallel job running in such a lousy way?" "'A:"' A frequent reason for lousy parallel performances is a conflict between MPI parallelization (implemented in QUANTUM ESPRESSO) and the autoparallelizing feature of MKL libraries. Set the environment variable OPEN_MP_THREADS to 1. See the section dedicated to this problem in the User Guide.

8.5 Frequent errors during execution

"Why is the code saying "Wrong atomic coordinates"?" "'A:"' Because they are: two or more atoms in the list of atoms have overlapping, or anyway too close, positions. Can't you see why? look better (use XCrysDen: see above) and remember that the code checks periodic images as well.

"The code stops with an "error reading namelist xxxx"" "'A:"' Most likely there is a misspelled variable in namelist xxxx. If there isn't any (have you looked carefully? really?? REALLY???), beware control characters like DOS M̂ (control-M): they can confuse the namelist-reading code.

If this message concerns the first namelist to be read (usually "&control"), and if you are running in parallel: try "code -inp input_file" instead of "code< input_file". Some MPI libraries do not properly handle input redirection.

"The code stops with an "error in davcio"" "'A:"' Possible reasons: disk is full; outdir does not exist; outdir exists but is not a directory (i.e. it is a file) or it is not writable; you changed some parameter(s) in the input (like wf_collect, or the number of processors/pools) without doing a bit of cleanup in your temporary files.

"The code stops with a "wrong charge" error" "'A:"' In most cases: you are treating a metallic system as if it were insulating.

8.6 Self Consistency

"What are the units for quantity XYZ?" "A:"' Unless otherwise specified, all PWscf input and output quantities are in atomic "Rydberg" units, i.e. energies in Ry, lengths in Bohr radii, etc.. Note that CP uses instead atomic "Hartree" units: energies in Ha, lengths in Bohr radii.

"Self-consistency is slow or does not converge at all" "'A:"' Bad input data will often result in bad scf convergence. Please check your structure first.

Assuming that your input data is sensible :

1. Verify if your system is metallic or is close to a metallic state, especially if you have few k-points. If the highest occupied and lowest unoccupied state(s) keep exchanging place during self-consistency, forget about reaching convergence. A typical sign of such behavior is that the self-consistency error goes down, down, down, than all of a sudden up again, and so on. Usually one can solve the problem by adding a few empty bands and a small broadening.

2. Reduce mixing beta from the default value (0.7) to $\sim 0.3 \div 0.1$ or smaller. Try the mixing mode value that is more appropriate for your problem. For slab geometries used in surface problems or for elongated cells, mixing mode='local-TF' should be the better choice, dampening "charge sloshing". You may also try to increase mixing ndim to more than 8 (default value). Beware: the larger mixing ndim, the larger the amount of memory you need.

3. Specific to US PP: the presence of negative charge density regions due to either the pseudization procedure of the augmentation part or to truncation at finite cutoff may give convergence problems. Raising the ecutrho cutoff for charge density will usually help, especially in gradient-corrected calculations.

"How is the charge density (the potential, etc.) stored? What position in real space corresponds to an array value?" "'A:"' The index of arrays used to store functions defined on 3D meshes is actually a shorthand for three indeces, following the FORTRAN convention ("leftmost index runs faster"). An example will explain this better. Suppose you have a 3D array of dimension (nr1,nr2,nr3), say psi(nr1,nr2,nr3). FORTRAN compilers store this array sequentially in the computer RAM in the following way:

```
        psi(1,1,1)
        psi(2,1,1)
        ...
        psi(nr1,1,1)
        psi(1,2,1)
        psi(2,2,1)
        ...
        psi(nr1,2,1)
        ...
        psi(nr1,nr2,1)
        psi(1,1,nr3)
etc
```

Let ind be the position of the (i,j,k) element in the above list: the relation between ind and (i,j,k) is:

```
ind = i + (j + 1) * nr1 + (k + 1) *  nr2 * nr1
```

This should clarify the relation between 1D and 3D indexing. In real space, the (i,j,k) point of the mesh is

$$r_{ijk} = \frac{i-1}{nr1}\tau_1 + \frac{j-1}{nr2}\tau_2 + \frac{k-1}{nr3}\tau_3$$

where the τ_i are the basis vectors of the Bravais lattice. The latter are stored row-wise in the "AT" array:
τ_1 = at(:, 1), tau_2 = at(:, 2), τ_3 = at(:, 3)
(info by Stefano Baroni)

"What is the difference between total and absolute magnetization?" "'A:"' The total magnetization is the integral of the magnetization in the cell:

$$M_T = \int (n_{up} - n_{down}) d^3r.$$

The absolute magnetization is the integral of the absolute value of the magnetization in the cell:

$$M_A = \int |n_{up} - n_{down}| d^3r.$$

In a simple ferromagnetic material they should be equal (except possibly for an overall sign)'. In simple antiferromagnets (like FeO, NiO) M_T is zero and M_A is twice the magnetization of each of the two atoms. (info by Stefano de Gironcoli)

"How can I calculate magnetic moments for each atom?" "'A:"' There is no 'right' way of defining the local magnetic moment around an atom in a multi-atom system. However an approximate way to define it is via the projected density of states on the atomic orbitals (code projwfc.x, see example08 for its use as a postprocessing tool). This code generate many files with the density of states projected on each atomic wavefunction of each atom and a BIG amount of data on the standard output, the last few lines of which contain the decomposition of Lowdin charges on angular momentum and spin component of each atom.

"What is the order of Y_{lm} components in projected DOS / projection of atomic wavefunctions?" "'A:"' "The order is, I think:

```
1   $P_{0,0}(t)$
2   $P_{1,0}(t)$
3   $P_{1,1}(t)cos\phi$
4   $P_{1,1}(t)sin\phi$
5   $P_{2,0}(t)$
6   $P_{2,1}(t)cos\phi$
7   $P_{2,1}(t)sin\phi$
8   $P_{2,2}(t)cos2\phi$
9   $P_{2,2}(t)sin2\phi$
```

and so on; $P_{l,m}$=Legendre Polynomials, $t = cos\theta = z/r$, $\phi = atan(y/x)$. No warranty. Anybody really interested in knowing "for sure" which spherical harmonic combination is which should look into routine ylmr2 in flib/ylmr2.f90".

"Why is the sum of partial Lowdin charges not equal to the total charge?" "Lowdin charges (as well as other conventional atomic charges) do not satisfy any sum rule. You can easily convince yourself that ths is the case because the atomic orbitals that are used to calculate them are arbitrary to some extent. If yu like, you can think that the missing charge is "delocalized" or "bonding" charge, but this would be another way of naming the conventional (to some extent) character of Lowdin charge." (Stefano Baroni, Sept. 2008).

See also the definition of "spilling parameter": Sanchez-Portal et al., Sol. State Commun. 95, 685 (1995). The spilling parameter measures the ability of the basis provided by the pseudo-atomic wfc to represent the PW eigenstates, by measuring how much of the subspace of the Hamiltonian eigenstates falls outside the subspace spanned by the atomic basis.

"Why do I get a strange value of the Fermi energy?" "The value of the Fermi energy (as well as of any energy, for that matter) depends of the reference level. What you are referring to is probably the "Fermi energy referred to the vacuum level" (i.e. the work function). In order to obtain that, you need to know what the vacuum level is, which cannot be said from a bulk calculation only" (Stefano Baroni, Sept. 2008).

"Why I don't get zero pressure/stress at equilibrium?" It depends. If you make a calculation with fixed cell parameters, you will never get exactly zero pressure/stress, unless you use the cell that yields perfect equilibrium for your pseudopotentials, cutoffs, k-points, etc.. Such cell will anyway be slightly different from the experimental one. Note however that pressures/stresses in the order of a few KBar correspond to very small differences in terms of lattice parameters.

If you obtain the equilibrium cell from a variable-cell optimization, do not forget that the pressure/stress calculated with the modified kinetic energy functional (very useful for variable-cell calculations) slightly differ from those calculated without it.

"Why do I get "negative starting charge"?" Self-consistency requires an initial guess for the charge density in order to bootstrap the iterative algorithm. This first guess is usually built from a superposition of atomic charges, constructed from pseudopotential data.

More often than not, this charges are a slightly too hard to be expanded very accurately in plane waves, hence some aliasing error will be introduced. Especially if the unit cell is big and mostly empty, some local low negative charge density will be produced.

"This is NOT harmful at all, the negative charge density is handled properly by the code and will disappear during the self-consistent cycles", but if it is very high (let's say more than 0.001*number of electrons) it may be a symptom that your charge density cutoff is too low. (L. Paulatto - November 2008)

"How do I calculate the work function?" Work function = (average potential in the vacuum) - (Fermi Energy). The former is estimated in a supercell with the slab geometry, by looking at the average of the electrostatic potential (typically without the XC part). See the example in examples/WorkFct_example.

8.7 Phonons

Is there a simple way to determine the symmetry of a given phonon mode? "'A:"' A symmetry analyzer was added in v.3.2 by Andrea Dal Corso. The following info may still be of interest to somebody, though. ISOTROPY package: http://stokes.byu.edu/iso/isotropy.html.

"Please follow http://dx.doi.org/10.1016/0010-4655(94)00164-W and http://dx.doi.org/10.1016/0010-4655(74)90057-5. These are connected to some programs found in the Computer Physics Communications Program Library (http://www.cpc.cs.qub.ac.uk) which are described in the articles:
ACKJ v1.0 Normal coordinate analysis of crystals, J.Th.M. de Hosson.
ACMI v1.0 Group-theoretical analysis of lattice vibrations, T.G. Worlton, J.L. Warren. See erratum Comp. Phys. Commun. 4(1972)382.
ACMM v1.0 Improved version of group-theoretical analysis of lattice dynamics, J.L. Warren, T.G. Worlton." (Info from Pascal Thibaudeau)

I am not getting zero acoustic mode frequencies, why? "'A:"' "If you treat, e.g., a molecule, the first six frequencies should vanish. However, due to convergence (number of plane waves, size of the supercell, etc.) they often appear as imaginary or small real, even if all other frequencies are converged with respect to ecut and celldm.

If you have a bulk structure, then imaginary frequencies indicate a lattice instability. However, they can appear also as a result of a non-converged groundstate (Ecut, k-point grid, ...).

Recently I also found that the parameters tr2_ph for the phonons and conv_thr for the groundstate can affect the quality of the phonon calculation, especially the "vanishing" frequencies for molecules." (Info from Katalyn Gaal-Nagy)

Why do I get negative phonon frequencies? "'A:"' If these occur for acoustic frequencies at Gamma point, see above. If these occur for rotational modes in a molecule into a supercell: it is a fictitious effect of the finite supercell size. If these occur in other cases, it depends. It may be a problem of bad convergence (see above) or it may signal a real instability.

An example: large negative phonon frequencies in 1-dimensional chains. "It is because probably some of atoms are sitting on the saddle points of the energy surface. Since QE symmetrizes charge density to avoid small numerical oscillation, the system cannot break the symmetry with the help of numerical noise. Check your system's stability by displacing one or more atoms a little bit along the direction of eigen-vector which has negative frequency. The eigen-vector can be found in the output of dynamical matrices

of ph.x. One example here is: for 1d aluminum chain, the LO mode will be negative if you place two atoms at (0.0,0.0,0.0) and (0.0,0.0,0.5) of crystal coordinates. To break the symmetry enforced by QE code, change the second atom coordinate to (0.0,0.0,0.505). Relax the system. You will find the atom will get itself a comfortable place at (0.0,0.0,0.727), showing a typical dimerization effect." (info by Nicola Marzari).