Quantum ESPRESSO

Introduction to the code and parallelization schema

Pietro Bonfà, CINECA
What is QuantumESPRESSO

Quantum opEn-Source Package for Research in Electronic Structure, Simulation, and Optimization

An integrated software suite for first-principle simulations, using density-functional theory (DFT), a plane waves (PW) basis set and pseudopotentials.

It is an IOM-DEMOCRITOS initiative, in collaboration with several other institutions (ICTP, CINECA Bologna, EPF Lausanne, Princeton University, Paris VI, IJS Ljubljana,...)
Where is QuantumESPRESSO

- Development now on gitlab.com:
  - https://gitlab.com/QEF/q-e
  - Report issues
  - Fork the code

- Releases are on
  http://www.quantum-espresso.org
Where is QuantumESPRESSO

• Documentation:
  - http://dx.doi.org/10.1088/0953-8984/21/39/395502

• Support:
  - pw_forum@pwscf.org
  - Bugs, q-e-developers@qe-forge.org
What’s inside QE

QE is actually a distribution of packages

- PWscf
- CP
- PP
- PWGui
- PHonon
- NEB
- Atomic
- Pwcond
- Xspectra
- TDDFPT
- EPW
Orbiting around QE

- GIPAW
- GWL
- PLUMED
- WanT
- Wannier90
- Yambo
- Many others...

DOI: 10.1021/jp908206c
Prerequisites

- Be able to write an input file for pw.x
- Know the mean-field DFT approach
- Have minimal experience with parallel computing
- Followed the introduction to Marconi
Topics

- Concise recap on some concepts of PW based DFT codes.
- QE parallelization scheme.
- How to run a big input case, understand numbers, get the best out of QE, avoid pitfalls.
- Tools to understand what went wrong.
Parallelizing a PW based code

One of the equations you want to solve is

\[ \rho(r) = \sum_{i}^{occ} |\phi_i(r)|^2 \]

\[ \left[ -\frac{\hbar^2}{2m} \nabla^2 + V_{eff}(r) \right] \phi_i(r) = \epsilon_i \phi_i(r) \]

Khon Sham orbitals or bands
Plane waves

Orbitals in Fourier space

\[ \phi_i(r) = \int \phi_i(g) e^{i gr} d^3g \]

\[ G_m = m_1 \cdot a^* + m_2 \cdot b^* + m_3 \cdot c^* \]

\[ a^* = \frac{2\pi}{\Omega} b \times c \]

For a periodic system (Bloch’s Theorem)

\[ \phi_i(r) = u_{ik}(r) e^{i kr} \]

\[ \phi_{i,k}(r) = \sum_{G} c_{k,G} \frac{1}{\sqrt{\Omega}} e^{i(G+k)r} \]

Complete basis set!

\( k \) is in FBZ
Plane waves

Orbitals in Fourier space

$$\phi_i(r) = \int \phi_i(g)e^{igr}d^3g$$

$$G_m = m_1 \cdot a^* + m_2 \cdot b^* + m_3 \cdot c^*$$

$$a^* = \frac{2\pi}{\Omega} b \times c$$

For a periodic system (Bloch’s Theorem)

$$\phi_{i,k}(r) = \sum_{G} c_{k,G} \frac{1}{\sqrt{\Omega}} e^{i(G+k)r}$$

$$E_c = \frac{\hbar^2|G_{max} + k|^2}{2m_e}$$

$k$ is in FBZ
Plane waves

\[ E_{c} = \frac{\hbar^2 G_{\text{max}}^2}{2m_e} \]

\[ V_{\text{sphere}} = \frac{4}{3} \pi G_{\text{max}}^3 \]

\[ V_{PW} = \frac{(2\pi)^3}{\Omega} \]

\[ N_{G_{\text{max}}} \propto \Omega E_{c}^{3/2} \]

Ry: 0.0168, Ha: 0.0477
Plane waves

- Hamiltonian matrix elements

\[ \phi_{i,k}(r) = \sum_m c_{i,k,G_m} \frac{1}{\sqrt{\Omega}} e^{i(G_m+k)r} = \sum_m c_{i,k,G_m} |G_m + k\rangle \]

\[ \langle G' + k'|V_{\text{eff}}(r)|G + k\rangle = \langle G' + k'| \sum_m V_{\text{eff}}(G_m)e^{iG_m r}|G + k\rangle = \sum_m V_{\text{eff}}(G_m)\delta_{k' - k} \]

\[ \sum_G \langle G' + k|H_{\text{eff}}|G + k\rangle c_{i,G+k} = \epsilon_i c_{i,G'+k} \]

\[ \sum_{m'} H_{m,m'}(k)c_{i,m'}(k) = \epsilon_i(k)c_{i,m}(k) \]

m is an index on G vectors
Computation and data partition

$$\sum_{m'} H_{m,m'}(k)c_{i,m'}(k) = \epsilon_i(k)c_{i,m}(k)$$

$$\phi_{i,k}(r) = \sum_G c_{i,G}(k)e^{i(G+k)r}$$
Wave-function & density

\[ \rho(r) = \sum_i \sum_k w_k f_{ik} |\phi_{ik}(r)|^2 \]

\[ \rho(r) \propto \sum_i \sum_k w_k f_{ik} \sum_{G_{max}} \sum_{G'_{max}} c_{ik}(G')^* c_{ik}(G) e^{i(G-G')r} \]

\[ \rho(r) \propto \sum_{G \leq 2G_{max}} \rho(G) e^{iGr} \]
Solving the Hamiltonian problem

\[ \sum_{m'} H_{m,m'}(k)c_{i,m'}(k) = \epsilon_i(k)c_{i,m}(k) \]

Diagonalize \( \{\phi_i, \epsilon_i\}, i = 1 \ldots N_G \)

What is actually needed are \( \sim N_e \) orbitals

Guess \( |\phi_j\rangle \) \( j = 1 \ldots N_b \) prepare \( H(\rho) \)

\[ |R(\phi_j)\rangle = (H - \tilde{\epsilon})|\phi_j\rangle \]

\[ |\tilde{\phi}_k\rangle = \sum_j v_{jk}|\phi_j\rangle \]

\[ \sum_j h_{ij}v_{jk} = \tilde{\epsilon}_k \sum_j v_{jk} \]
Real and reciprocal space

\[ c_{ik}(r) = \sum_G c_{ik}(G) e^{iGr} \]

\[ c_{ik}(G) = \frac{1}{N} \sum_r c_{ik}(r) e^{-iGr} \]

FFT
Real and reciprocal space

Eric Pascolo, master thesis

MATERIAL SCIENCE CODES ON INNOVATIVE HPC ARCHITECTURES: TARGETING EXASCALE
Experiment

loading....
Distribute FFT?

• In real space, distributed as slices
• In reciprocal space, distributed as sticks
• How to scale above nz planes?
  – Distribute the other dimension(s)
  – Group bands and compute multiple FFT
Multiple 1D FFTs

1D FFT
F(Gx, Gy, Gz)

1D FFT
fft_scatter
F(Gx, Gy, Rz)

1D FFT
fft_scatter
F(Gx, Ry, Rz)

1D FFT
F(Rx, Ry, Rz)
Task groups

• vloc_psi.f90

```fortran
IF( use_tg ) THEN
ELSE
   DO ibnd = 1, m
      !
      psic(:) = (0.d0, 0.d0)
      psic(nls(igk_k(1:n, current_k))) = psi(1:n, ibnd)
      !
      CALL invfft ('Wave', psic, dffts)
   !
   !$omp parallel do
   DO j = 1, dffts%nrr
      psic(j) = psic(j) * v(j)
   ENDDO
   !$omp end parallel do
   CALL fwwft ('Wave', psic, dffts)
   !
   ! addition to the total product
   !
   !$omp parallel do
   DO j = 1, n
      hpsi(j, ibnd) = hpsi(j, ibnd) + psic(nls(igk_k(j, current_k)))
   ENDDO
   !$omp end parallel do
   ENDDO
ENDIF
```

Orbitals

Wave-function

Compute G-space to R-space

Compute R-space to G-space

Add to hpsi
Task groups

IF( use_tg ) THEN

CALL tg_get_nnr( dffts, right_nnr )

DO ibnd = 1, m, !tg_ntgrp(dffts)
    !
    call = (0.0d0, 0.0d0)
    ioff = 0
    DO idx = 1, fftx_ntgrp(dffts)
        !
        CALL invfft ('tgWave', tg_psic, dffts )
        !
        CALL tg_get_group_nr3( dffts, right_nr3 )
    !
    !$omp parallel do
    DO j = 1, dffts%nr1x*dffts%nr2x* right_nr3
        tg_psic (j) = tg_psic (j) * tg_v(j)
    ENDDO
    !$omp end parallel do
    !
    CALL fwwft ('tgWave', tg_psic, dffts )
    !
    ! addition to the total product
    ioff = 0
    !
    CALL tg_get_recip_inc( dffts, right_inc )
    !
    DO idx = 1, fftx_ntgrp(dffts)
        !
        ENDDO
ELSE

ENDDO

Other options for parallelism

Nudged Elastic Band

Images

DOI: 10.1016/j.cpc.2007.09.011

PHonon (linear response)

Irreducible modes

DOI: 10.1103/PhysRevB.93.174419
Other options for parallelism

Nudged Elastic Band

Images

DOI: 10.1016/j.cpc.2007.09.011

PHonon (linear response)

Irreducible modes

Abstract layer for ensemble-like parallelization

DOI: 10.1103/PhysRevB.93.174419
QE parallelization layout

```
MPI_COMM_WORLD
  IMAGE GROUP 0
  IMAGE GROUP 1
  IMAGE GROUP ...
    K-point GROUP 0
    K-point GROUP 1
    K-point GROUP ...
      Band GROUP 0
      Band GROUP 1
      Band GROUP ...

Fine grain parallelization
```
QE parallelization layout

Band GROUP
(Band subset, All G-vectors)

Task GROUP 0
(G-vector)

Task GROUP 1
(G-vector)

Task GROUP #
(all G for FFT #)

Task GROUP 0
(Task (0,0)

Task GROUP 1
(Task (0,1)

Task GROUP #
(Task (1,0)

Task GROUP 2
(Task (1,1)

Ortho GROUP
(Ortho GROUP 0

Ortho GROUP 1

Ortho GROUP 2

Ortho GROUP ..

All Task in a TILE for FFT #

All_to_all within a TILE

Data redist All_to_all

Onn = <PWng | PWgn>
Overlap Matrix

ScaLAPACK-like Data distribution

Shared memory parallelization

OpenMP task 0

OpenMP task 1

OpenMP task ...

MATERIAL SCIENCE CODES ON INNOVATIVE HPC ARCHITECTURES: TARGETING EXASCALE
Parallelization, in practice

Distribution of images

```
mpirun neb.x -nimage I -inp neb.in > neb.out
```

Output:
```
path-images division: nimage = I
```

Max value: total number of images in the simulation.

Constraints:
- Depend on code using this “abstract” parallelism

Tentative optimal value: `nimage = max possible value`
Parallelization, in practice

Distribution of k points

\[ \text{mpirun} \ \text{pw.x} \ -\text{npool} \ X \ -\text{inp} \ \text{pw.in} > \text{pw.out} \]

Output:

\[ \text{K-points division:} \ \ npool = X \]

Distribute k points among \( X \) pools of MPI procs.
Max value: \( n(k) \)

Constraints:
- at least 1 k point per MPI process
- Must be a divisor of the total number of processes

Tentative optimal value: \( npool = \max(n(k)) \)
Parallelization, in practice

Band group for Exact Exchange

```bash
mpirun pw.x -npool X -bgrp Y -inp pw.in > pw.out
```

Split KS states across the MPI procs. of the band group.

Constraints:
- ?

Tentative optimal value: ?
Parallelization, in practice

Parallel diagonalization

```bash
mpirun pw.x -npool X -ndiag Y -inp pw.in > pw.out
```

Output
Subspace diagonalization (size of subgroup: \( \sqrt{Y} \times \sqrt{Y} \) procs)

Distribute and parallelize matrix diagonalization and matrix-matrix multiplications needed in iterative diagonalization (pw.x) or orthonormalization(cp.x).

Max value: \( \frac{n(MPI)}{X} \)
Constraints:
• Must be square

Tentative optimal value:
• Must be > 1 for inputs with more than 100 KS
• depends on many architectural parameters
Parallelization, in practice

FFT task groups

```
mpirun pw.x -npool X -ntg Y -inp pw.in > pw.out
```

Output

<table>
<thead>
<tr>
<th>R &amp; G space division:</th>
<th>proc/nbgrp/npool/nimage = 1152</th>
</tr>
</thead>
<tbody>
<tr>
<td>wavefunctions fft division:</td>
<td>Y-proc x Z-proc = Y 96</td>
</tr>
<tr>
<td>wavefunctions fft division:</td>
<td>task group distribution</td>
</tr>
<tr>
<td>#TG x Z-proc =</td>
<td>Y 96</td>
</tr>
</tbody>
</table>

Each plane-wave group of processors is split into \( \text{ntg} \) task groups of nFFT processors, with \( \text{ntg} \times \text{nFFT} = \text{nBGRP} \); each task group takes care of the FFT over \( N_{KS}/\text{ntg} \) states.

Max value: \( n(\text{MPI})/X \)

Constraints:
- Must be divisor of nBGRP (for nyfft)

Tentative optimal value: >1 only when nBGRP>nr3, depends on the HPC system.
Parallelization, in practice

- OpenMP
  - In the code and **in the libraries**.
  - Only when MPI is saturated.
  - No more than 8 threads.
  - Use it when needed!

- Multithreading
  - Generally not useful (not memory bound)
Parallelization, recap

- **Pools:**
  - Very effective, small communication
  - A lot of memory duplication!
- **G vectors:**
  - Lower memory duplication
  - More communication
- **OpenMP:**
  - Practically no memory duplication
  - When MPI is saturated
- **Diagonalization method:**
  - Davidson: faster, more memory
  - CG: slower, less memory
 Compile QE

• Get & compile ELPA (version 2016 here)
• Load/use machine specific Math Kernels (eg. Intel® MKL)
• Configure PW with scalapack, ELPA and OpenMP:

```bash
$ ./configure --enable-openmp --with-scalapack=intel \
--with-elpa-include=/path/to/elpa-2016.11.001.pre/modules \
--with-elpa-lib=/path/to/elpa-2016.11.001.pre/.libs/libelpa_openmp.a
```

The following libraries have been found:
- BLAS_LIBS= -lmkl_intel_lp64  -lmkl_intel_thread -lmkl_core
- LAPACK_LIBS=
- SCALAPACK_LIBS=-lmkl_scalapack_lp64  -lmkl_blacs_intelmpi_lp64
- ELPA_LIBS=/marconi_scratch/userinternal/pbonfa01/school/pw-elpa-bdw/elpa-2016.11.001.pre/.libs/libelpa_openmp.a
- FFT_LIBS=
Compile QE

• Check make.inc

• Add optimization flags, eg.
  - KNL → AVX512MIC
  - SKL → AVX512CORE

• Possibly add profiling libraries
  - Gperftools → memory profiling
  - GPTL → accurate timing
Check flags

MANUAL_DFLAGS =
DFLAGS = -D__DFTI -D__MPI -D__SCALAPACK -D__ELPA_2016
FDFLAGS = $(DFLAGS) $(MANUAL_DFLAGS)

[...]

MPIF90 = mpiifort
F90 = ifort
CC = icc
F77 = ifort

# C preprocessor and preprocessing flags - for explicit preprocessing,
# if needed (see the compilation rules above)
# preprocessing flags must include DFLAGS and IFLAGS

CPP = cpp
CPPFLAGS = -P -traditional $(DFLAGS) $(IFLAGS)
A real case

Grafene on Iridium

Standard QE benchmark

Available online at:

http://qe-forge.org/gf/project/q-e/frs/?action=FrsReleaseView&release_id=48
&control
  calculation = 'scf'
  prefix='GRIR'
  restart_mode='from_scratch'
  pseudo_dir='/',
  outdir='/path/to/scratch',
  wf_collect=.true.
/
&system
  ibrav=  4
  celldm(1) = 46.5334237988185d0
  celldm(3) = 1.274596
  nat=686
  ntyp= 2,
  ecutwfc=30
  occupations = 'smearing'
  smearing='mv'
  degauss=0.025d0
  nspin = 2
  starting_magnetization(1) = +.00
  starting_magnetization(2) = +.00
/
&electrons
  conv_thr = 1.0d-5
  mixing_beta=0.3d0
  mixing_mode='local-TF'
  startingwfc='atomic'
  diagonalization='david'
  electron_maxstep = 1
/
ATOMIC_SPECIES
  C  12.010  C.pbe-paw_kj-x.UPF
  Ir 192.22  Ir.pbe-paw_kj.UPF
K_POINTS {automatic}
  2 2 2 0 0 0

How many k points?
How many states?
How many G vectors?
How would you run it?
Program PWSCF v.6.2 starts on 1Dec2017 at 9:56:26

This program is part of the open-source Quantum ESPRESSO suite for quantum simulation of materials; please cite
"P. Giannozzi et al., J. Phys.:Condens. Matter 21 395502 (2009);
"P. Giannozzi et al., J. Phys.:Condens. Matter 29 465901 (2017);
URL http://www.quantum-espresso.org",
in publications or presentations arising from this work. More details at http://www.quantum-espresso.org/quote

Parallel version (MPI & OpenMP), running on 144 processor cores
Number of MPI processes: 144
Threads/MPI process: 1

MPI processes distributed on 4 nodes
K-points division: npool = 4
R & G space division: proc/nbgrp/npool/nimage = 36
Reading input from grir686.in

Current dimensions of program PWSCF are:
Max number of different atomic species (ntypx) = 10
Max number of k-points (npk) = 40000
Max angular momentum in pseudopotentials (lmaxx) = 3
file Ir.pbe-paw_kj.UPF: wavefunction(s) 5D renormalized

Subspace diagonalization in iterative solution of the eigenvalue problem:
a serial algorithm will be used

Parallelization info

<table>
<thead>
<tr>
<th></th>
<th>dense</th>
<th>smooth</th>
<th>PW</th>
<th>G-vecs:</th>
<th>dense</th>
<th>smooth</th>
<th>PW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>497</td>
<td>497</td>
<td>128</td>
<td>68583</td>
<td>68583</td>
<td>9018</td>
<td></td>
</tr>
<tr>
<td>Max</td>
<td>498</td>
<td>498</td>
<td>129</td>
<td>68592</td>
<td>68592</td>
<td>9025</td>
<td></td>
</tr>
<tr>
<td>Sum</td>
<td>17917</td>
<td>17917</td>
<td>4639</td>
<td>2469147</td>
<td>2469147</td>
<td>324807</td>
<td></td>
</tr>
</tbody>
</table>
## Memory

<table>
<thead>
<tr>
<th>RAM Usage</th>
<th>Size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>wfc:</td>
<td>406.00</td>
</tr>
<tr>
<td>wfc (w. buffer):</td>
<td>1217.99</td>
</tr>
<tr>
<td>str. fact:</td>
<td>2.09</td>
</tr>
<tr>
<td>local pot:</td>
<td>0.00</td>
</tr>
<tr>
<td>nlocal pot:</td>
<td>1353.50</td>
</tr>
<tr>
<td>grad:</td>
<td>13.00</td>
</tr>
<tr>
<td>rho, v, vnew:</td>
<td>15.18</td>
</tr>
<tr>
<td>rhoin:</td>
<td>5.06</td>
</tr>
<tr>
<td>rho*nmix:</td>
<td>33.49</td>
</tr>
<tr>
<td>G-vectors:</td>
<td>4.45</td>
</tr>
<tr>
<td>h, s, v(r/c):</td>
<td>7056.75</td>
</tr>
<tr>
<td>&lt;psi</td>
<td>beta&gt;:</td>
</tr>
<tr>
<td>psi:</td>
<td>1623.99</td>
</tr>
<tr>
<td>hpsi:</td>
<td>1623.99</td>
</tr>
<tr>
<td>spsi:</td>
<td>1623.99</td>
</tr>
<tr>
<td>wfcinit/wfcrot:</td>
<td>1966.26</td>
</tr>
<tr>
<td>addusdens:</td>
<td>704.31</td>
</tr>
</tbody>
</table>

Estimated static dynamical RAM per process > 2.59 GB

Estimated max dynamical RAM per process > 14.72 GB

Estimated total dynamical RAM > 2120.01 GB

1) This is a lower estimate!
2) MPI itself requires memory
3) Rule of thumb: crash without error message.
Happens more often than not

```bash
$ qsub -I -N GuiInteractive -A cin_staff -v "DISPLAY=$DISPLAY" -l select=12:ncpus=36:mpiprocs=36:mem=100G -l walltime=0:25:00
```
Program PWSCF v.6.2 starts on 29Nov2017 at 15:22:59

This program is part of the open-source Quantum ESPRESSO suite for quantum simulation of materials; please cite
"P. Giannozzi et al., J. Phys.:Condens. Matter 21 395502 (2009);
"P. Giannozzi et al., J. Phys.:Condens. Matter 29 465901 (2017);
URL http://www.quantum-espresso.org",
in publications or presentations arising from this work. More details at http://www.quantum-espresso.org/quote

Parallel version (MPI & OpenMP), running on 15552 processor cores
Number of MPI processes: 432
Threads/MPI process: 36

MPI processes distributed on 12 nodes
K-points division: npool = 2
R & G space division: proc/nbgrp/npool/nimage = 216
Reading input from grir686.in

Current dimensions of program PWSCF are:
Max number of different atomic species (ntypx) = 10
Max number of k-points (npk) = 40000
Max angular momentum in pseudopotentials (lmaxx) = 3
Minimal jobscript

```
$ qsub -I -N GuiInteractive -A cin_staff -v "DISPLAY=${DISPLAY}" -l
  select=4:ncpus=36:mpiprocs=36:mem=100G -l walltime=0:25:00

Waiting for job to start...

$ export OMP_NUM_THREADS=1
$ export MKL_NUM_THREADS=1
$ export KMP_AFFINITY=scatter,granularity=fine,1
$
$ # For most OpenMP codes, type=scatter should provide the best
  performance, as it minimizes cache and memory bandwidth contention
  for all processor models.

mpirun pw.x -npool X -ndiag Y -ntg Z -inp pw.in > pw.out
```
# Running on Marconi A1...

```
#!/bin/bash
#PBS -l select=4:ncpus=36:mpiprocs=36:mem=118GB
#PBS -l walltime=04:51:59

$ export OMP_NUM_THREADS=1
$ export MKL_NUM_THREADS=1
$ export KMP_AFFINITY=scatter,granularity=fine,1

mpirun pw.x -npool 1 -ndiag 144 -ntg 1 -inp pw.in > pw.out
```

Estimated max dynamical RAM per process > 2.97 GB
Running on Marconi A1...

```
#!/bin/bash
#PBS -l select=4:ncpus=36:mpiprocs=36:mem=118GB
#PBS -l walltime=04:51:59
$ export OMP_NUM_THREADS=1
$ export MKL_NUM_THREADS=1
$ export KMP_AFFINITY=scatter,granularity=fine,1
mpirun pw.x -npool 1 -ndiag 144 -ntg 1 -inp pw.in > pw.out
```

```
&control
  calculation = 'scf'
  prefix='GRIR'
  max_seconds=20000 !42000
  restart_mode='from_scratch'
  pseudo_dir='./'
  outdir='./tmp',
  disk_io = 'high'
/
&system
  ibrav= 4
  celldm(1) = 46.5334237988185d0
  celldm(3) = 1.274596
  nat= 686
  ntyp= 2,
  ecutwfc=30
  !nbnd=3300
  occupations = 'smearing'
    smearing='mv'
    degauss=0.025d0
  nspin = 2
  starting_magnetization(1) = +.00
  starting_magnetization(2) = +.00
/
&electrons
  conv_thr = 1.0d-5
  mixing_beta=0.3d0
  mixing_mode='local-TF'
  startingwfc='atomic'
  diagonalization='cg'
  electron_maxstep = 1
/
```

Estimated max dynamical RAM per process > 1.38 GB

RAM per process > 2.97 GB
Running on Marconi A1...

```
#!/bin/bash
#PBS -l select=8:ncpus=36:mpipooces=36:mem=118GB
#PBS -l walltime=00:51:59

$ export OMP_NUM_THREADS=1
$ export MKL_NUM_THREADS=1
$ export KMP_AFFINITY=scatter,granularity=fine,1

mpirun pw.x -npool X -ndiag Y -ntg Z -inp pw.in > pw.out
```

**X: 1  Y: 36  Z: 1**

R & G space division:  proc/nbgrp/npool/nimage = **288**

Subspace diagonalization in iterative solution of the eigenvalue problem:
ELPA distributed-memory algorithm (size of sub-group:  6* 6 procs)

<table>
<thead>
<tr>
<th>sticks:</th>
<th>dense</th>
<th>smooth</th>
<th>PW</th>
<th>G-vecs:</th>
<th>dense</th>
<th>smooth</th>
<th>PW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>62</td>
<td>62</td>
<td>16</td>
<td>8568</td>
<td>8568</td>
<td>1120</td>
<td></td>
</tr>
<tr>
<td>Max</td>
<td>63</td>
<td>63</td>
<td>17</td>
<td>8578</td>
<td>8578</td>
<td>1132</td>
<td></td>
</tr>
<tr>
<td>Sum</td>
<td>17917</td>
<td>17917</td>
<td>4639</td>
<td>2469147</td>
<td>2469147</td>
<td>324807</td>
<td></td>
</tr>
</tbody>
</table>

Dense grid: 2469147 G-vectors

FFT dimensions: (180, 180, **216**)
Running on Marconi A1...

```
#!/bin/bash
#PBS -l select=8:ncpus=36:mpiprocs=36:mem=118GB
#PBS -l walltime=00:51:59

$ export OMP_NUM_THREADS=1
$ export MKL_NUM_THREADS=1
$ export KMP_AFFINITY=scatter,granularity=fine,1

mpirun pw.x -npool X -ndiag Y -ntg Z -inp pw.in > pw.out
```

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>Time CPU</th>
<th>Time Wall</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>36</td>
<td>1</td>
<td>2071.28s</td>
<td>2108.70s</td>
</tr>
<tr>
<td>1</td>
<td>144</td>
<td>1</td>
<td>1442.97s</td>
<td>1470.78s</td>
</tr>
<tr>
<td>1</td>
<td>256</td>
<td>1</td>
<td>1544.03s</td>
<td>1576.22s</td>
</tr>
<tr>
<td>1</td>
<td>144</td>
<td>2</td>
<td>1286.43s</td>
<td>1312.28s</td>
</tr>
<tr>
<td>1</td>
<td>144</td>
<td>4</td>
<td>1274.49s</td>
<td>1299.47s</td>
</tr>
</tbody>
</table>
Running on Marconi A1...
Running on Marconi A1...
Running on Marconi A1...

But...

Parallelization info
---------------------
sticks: dense smooth PW G-vecs: dense smooth PW
Min 37 37 9 5137 5137 667
Max 38 38 10 5154 5154 690
Sum 17917 17917 4639 2469147 2469147 324807

Divide by 3 to get (abstract) per core distribution of problem size!
Thank you