# Quantum ESPRESSO

MAX DRIVING THE EXASCALE TRANSITION

## 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

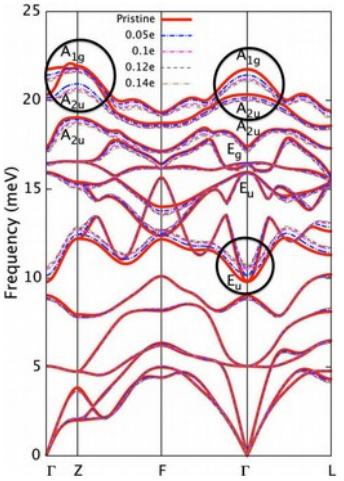
# Where is QuantumESPRESSO

- Documentation:
  - http://www.quantum-espresso.org/users-manual/
  - http://iopscience.iop.org/article/10.1088/1361-648X/aa8f79
  - http://dx.doi.org/10.1088/0953-8984/21/39/395502
- Support:
  - pw_forum@pwscf.org
  - Bugs, q-e-developers@qe-forge.org

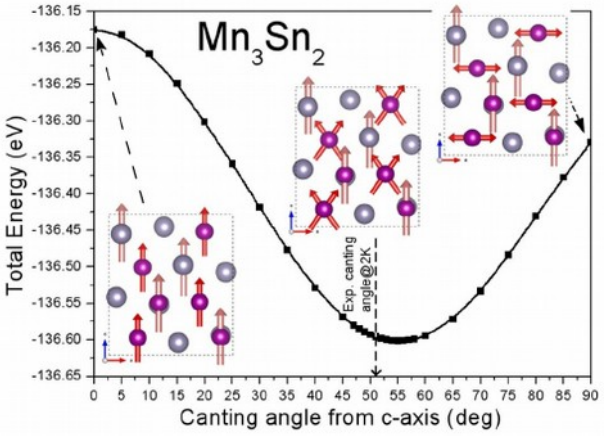
# What's inside QE

QE is actually a distribution of packages

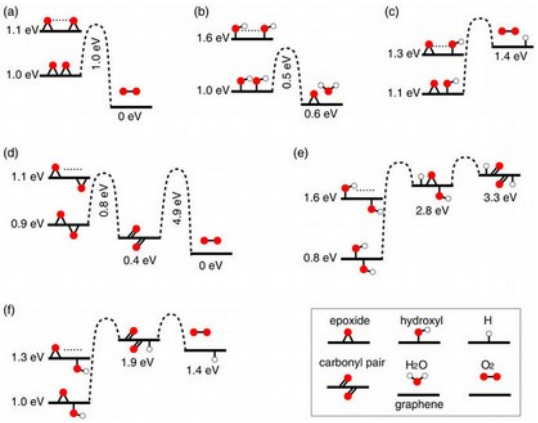
- PWscf
- CP
- PP
- PWGui
- PHonon
- NEB
- Atomic
- Pwcond
- Xspectra
- TDDFPT
- EPW




DOI:10.1038/srep08964



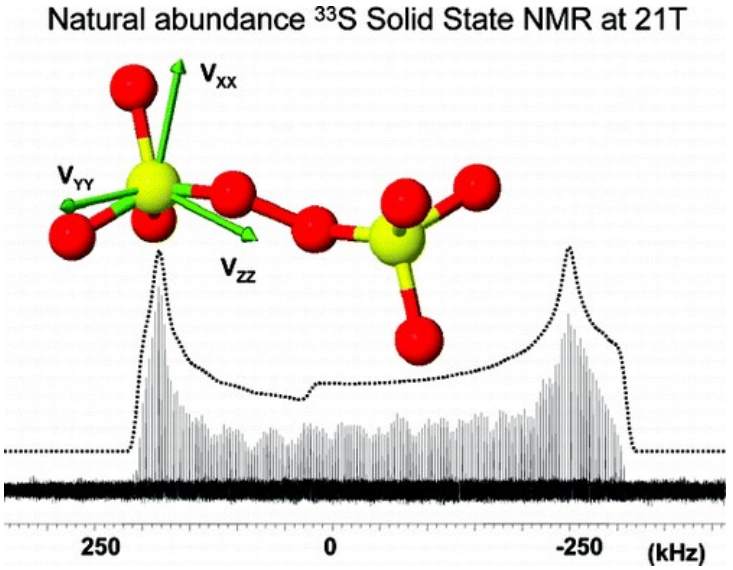

DOI: 10.1103/PhysRevB.88.054429



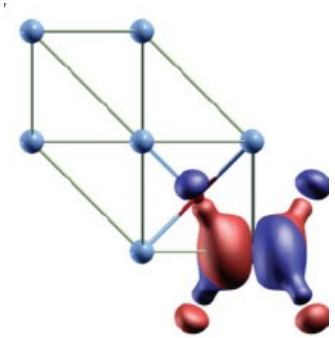

DOI:10.1038/srep02484

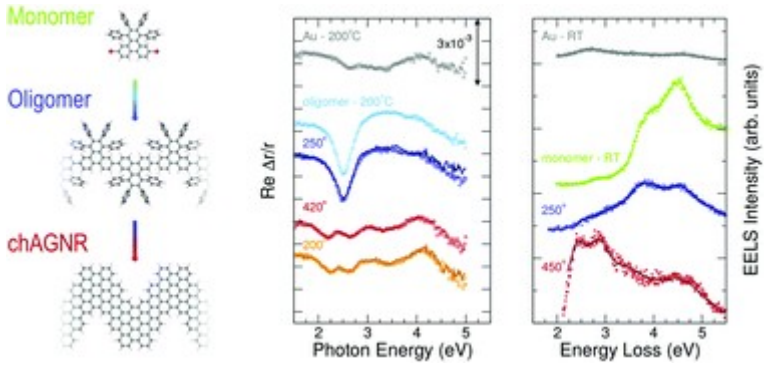
# Orbiting around QE

- 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

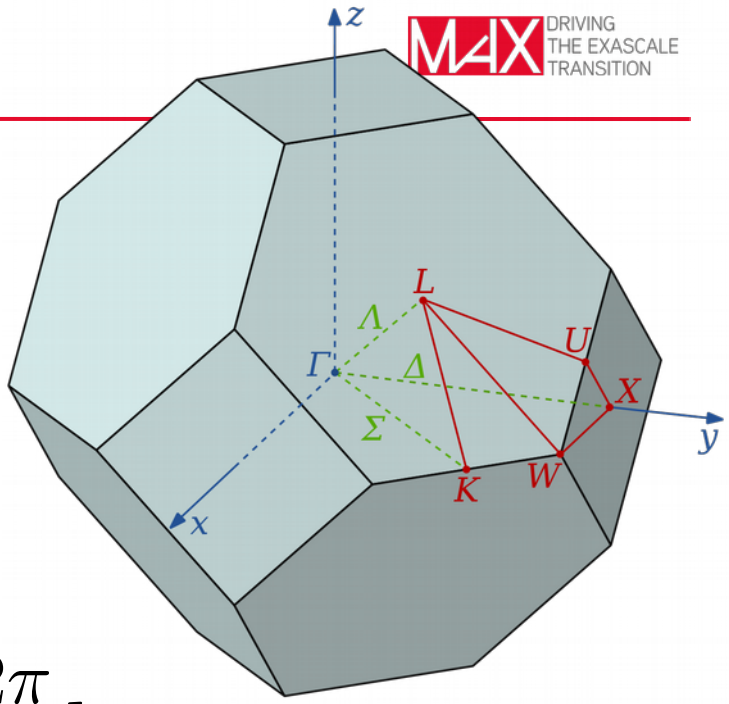
One of the equations you want to solve is

$$\rho(\boldsymbol{r}) = \sum_{i}^{\mathrm{occ}} |\phi_i(\boldsymbol{r})|^2$$

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

Khon Sham *orbitals* or *bands*

# Plane waves




## Orbitals in Fourier space

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

$$\mathbf{G}_m = m_1 \cdot \mathbf{a}^* + m_2 \cdot \mathbf{b}^* + m_3 \cdot \mathbf{c}^*$$

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

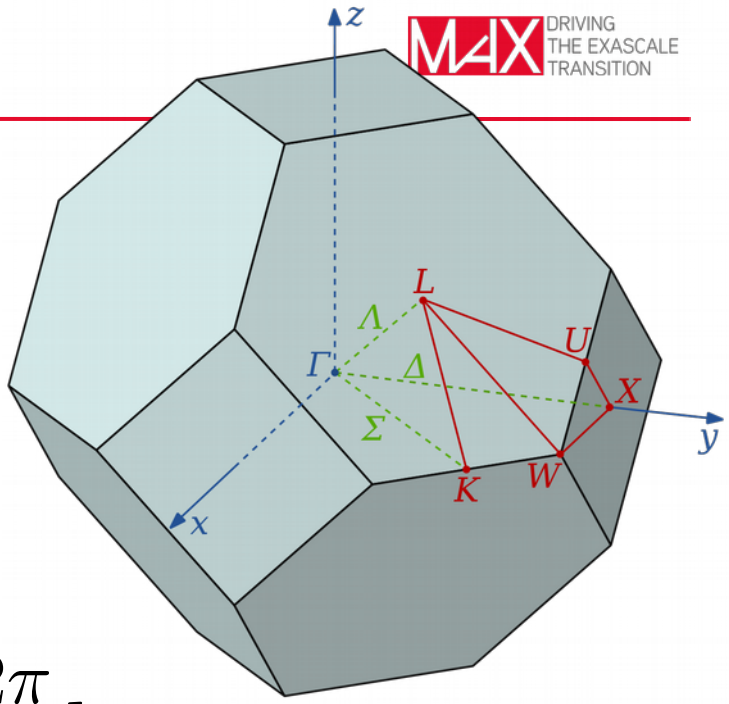
## For a periodic system (Bloch's Theorem)

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

## Complete basis set!

$\boldsymbol{k}$ is in FBZ

# Plane waves




## Orbitals in Fourier space

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

$$\mathbf{G}_m = m_1 \cdot \mathbf{a}^* + m_2 \cdot \mathbf{b}^* + m_3 \cdot \mathbf{c}^*$$

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

## For a periodic system (Bloch's Theorem)

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

$$E_c = \frac{\hbar^2 |\boldsymbol{G}_{max} + \boldsymbol{k}|^2}{2m_e}$$

$\boldsymbol{k}$ is in FBZ

# Plane waves

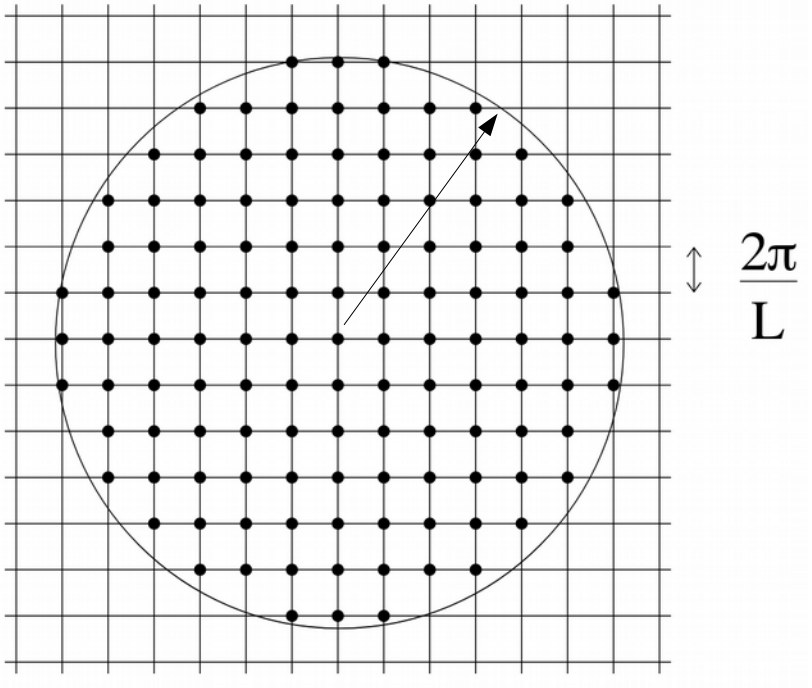
$$E_c = \frac{\hbar^2 G_{max}^2}{2m_e}$$

$$V_{sphere} = \frac{4}{3}\pi G_{max}^3$$

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

$$N_{G_{max}} \propto \Omega E_c^{3/2}$$

Ry: 0.0168, Ha: 0.0477



# Plane waves

- Hamiltonian matrix elements

$$\phi_{i,\boldsymbol{k}}(\boldsymbol{r}) = \sum_m c_{i,\boldsymbol{k},\boldsymbol{G}_m} \frac{1}{\sqrt{\Omega}} e^{i(\boldsymbol{G}_m+\boldsymbol{k})\boldsymbol{r}} = \sum_m c_{i,\boldsymbol{k},\boldsymbol{G}_m} |\boldsymbol{G}_m + \boldsymbol{k}\rangle$$

$$\langle \boldsymbol{G}' + \boldsymbol{k}'|V_{eff}(\boldsymbol{r})|\boldsymbol{G}+\boldsymbol{k}\rangle = \langle \boldsymbol{G}' + \boldsymbol{k}'| \sum_m V_{eff}(\boldsymbol{G}_m) e^{i\boldsymbol{G}_m\boldsymbol{r}} |\boldsymbol{G}+\boldsymbol{k}\rangle = \sum_m V_{eff}(\boldsymbol{G}_m)\delta_{\boldsymbol{k}'-\boldsymbol{k}}$$

$$\sum_{\boldsymbol{G}} \langle \boldsymbol{G}' + \boldsymbol{k}|H_{eff}|\boldsymbol{G}+\boldsymbol{k}\rangle c_{i,\boldsymbol{G}+\boldsymbol{k}} = \epsilon_i c_{i,\boldsymbol{G}'+\boldsymbol{k}}$$

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

m is an index on G vectors

# Computation and data partition

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

$$\phi_{\boldsymbol{i},\boldsymbol{k}}(\boldsymbol{r}) = \sum_{\boldsymbol{G}} c_{i,\boldsymbol{G}}(\boldsymbol{k}) e^{i(\boldsymbol{G}+\boldsymbol{k})\boldsymbol{r}}$$

# Wave-function & density

$$\rho(\boldsymbol{r}) = \sum_{i} \sum_{\boldsymbol{k}} w_{\boldsymbol{k}} f_{i\boldsymbol{k}} |\phi_{i\boldsymbol{k}}(r)|^2$$

$$\rho(\boldsymbol{r}) \propto \sum_{i} \sum_{\boldsymbol{k}} w_{\boldsymbol{k}} f_{i\boldsymbol{k}} \sum_{G_{max}} \sum_{G'_{max}} c_{i\boldsymbol{k}}(\boldsymbol{G'})^* c_{i\boldsymbol{k}}(\boldsymbol{G}) e^{i(\boldsymbol{G}-\boldsymbol{G'})\boldsymbol{r}}$$

$$\rho(\boldsymbol{r}) \propto \sum_{\boldsymbol{G} \leq 2G_{max}} \rho(\boldsymbol{G}) e^{i\boldsymbol{G}\boldsymbol{r}}$$

# Solving the Hamiltonian problem

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

Diagonalize → $\{\phi_i, \epsilon_i\}, i = 1 \ldots N_G$

What is actually needed are $\sim N_e$ orbitals

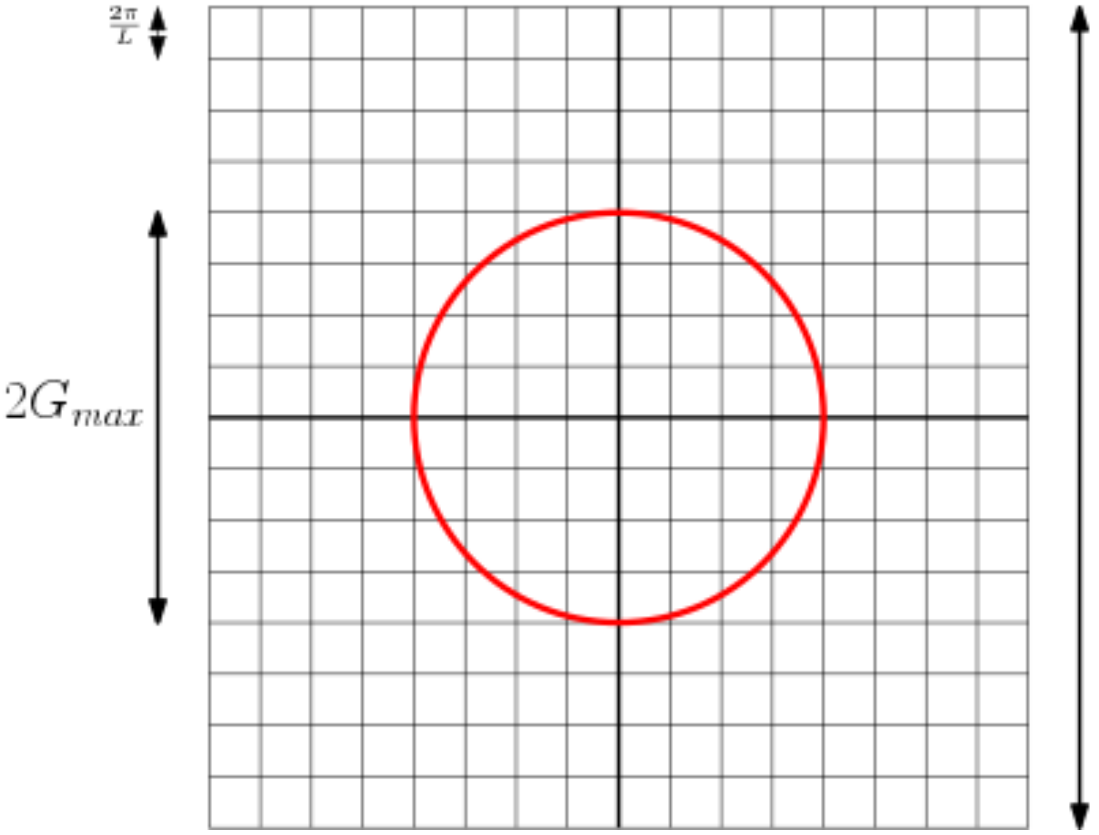
Guess $|\phi_j\rangle\ j = 1 \ldots N_b$ prepare $H(\rho)$

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

$$h_{\phi_i \phi_j} = \langle \phi_i | H | \phi_j \rangle$$

$$|\tilde{\phi}_k\rangle = \sum_j v_{jk} |\phi_j\rangle \longleftarrow \sum_j h_{ij} v_{jk} = \tilde{\epsilon}_k \sum_j \boldsymbol{v}_{jk}$$

# Real and reciprocal space

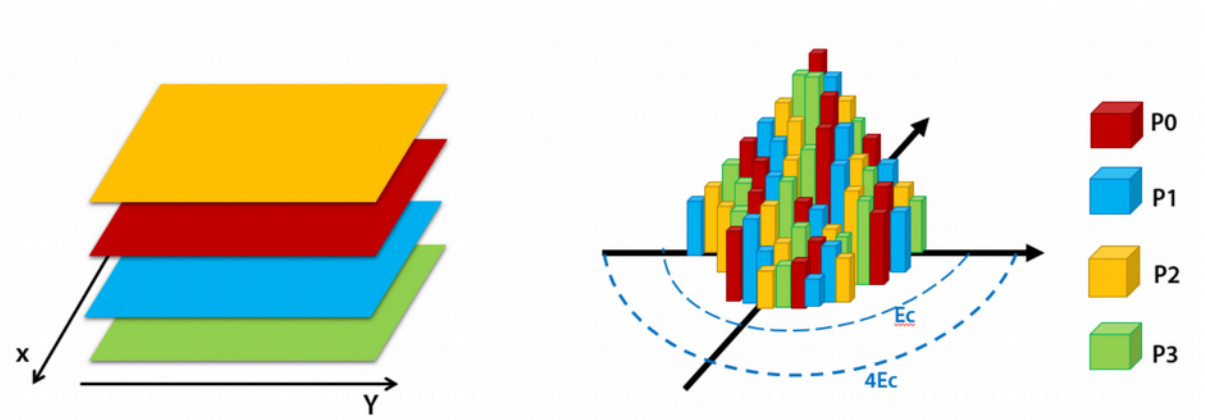


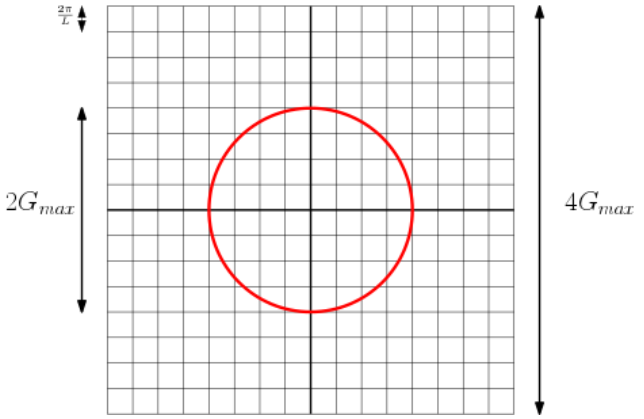

$$c_{i\boldsymbol{k}}(\boldsymbol{r}) = \sum_{\boldsymbol{G}} c_{i\boldsymbol{k}}(\boldsymbol{G}) e^{i\boldsymbol{G}\boldsymbol{r}}$$

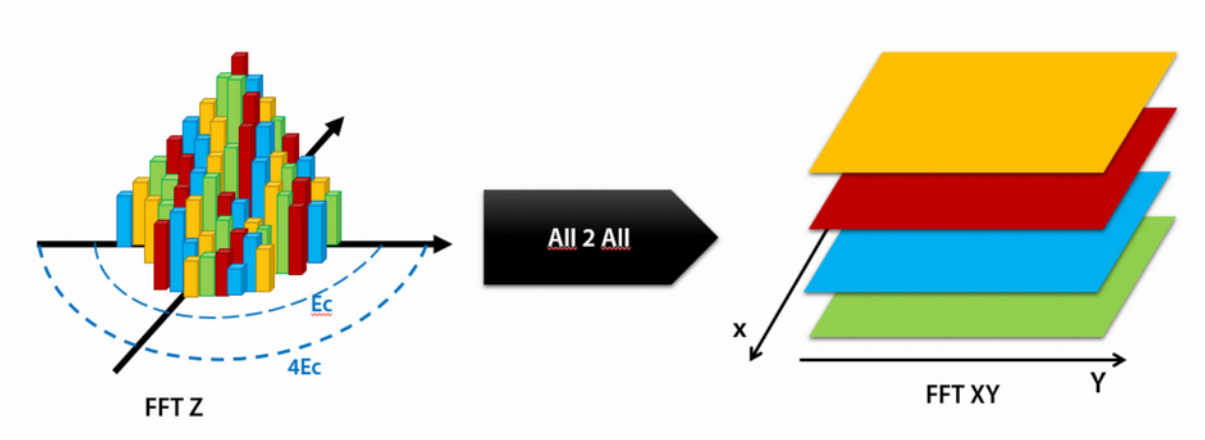
FFT

$$c_{i\boldsymbol{k}}(\boldsymbol{G}) = \frac{1}{N} \sum_{\boldsymbol{r}} c_{i\boldsymbol{k}}(\boldsymbol{r}) e^{-i\boldsymbol{G}\boldsymbol{r}}$$

# Real and reciprocal space




Eric Pascolo, master thesis

F(Gx, Gy, Gz)

F(Gx, Gy, Rz)

1D FFT

fft_scatter

F(Gx, Gy, Rz)

F(Rx, Ry, Rz)

2D FFT

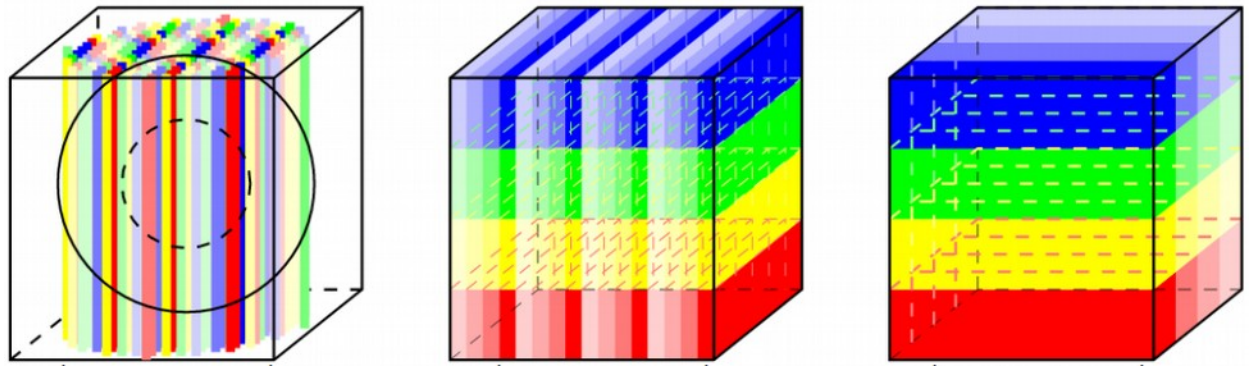
# Experiment

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



F(Gx, Gy, **Gz**) → 1D FFT → F(Gx, Gy, **Rz**)

fft_scatter

F(Gx, **Gy**, Rz) → 1D FFT → F(Gx, **Ry**, Rz)

fft_scatter

F(**Gx**, Ry, Rz) → 1D FFT → F(**Rx**, Ry, Rz)

# Task groups

- vloc_psi.f90

```
  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)
        !
!$omp parallel do
        DO j = 1, dffts%nnr
           psic (j) = psic (j) * v(j)
        ENDDO
!$omp end parallel do
        !
        CALL fwfft ('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, fftx_ntgrp(dffts)
      !
      tg_psic = (0.d0, 0.d0)
      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 fwfft ('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
```

More than one band at a time

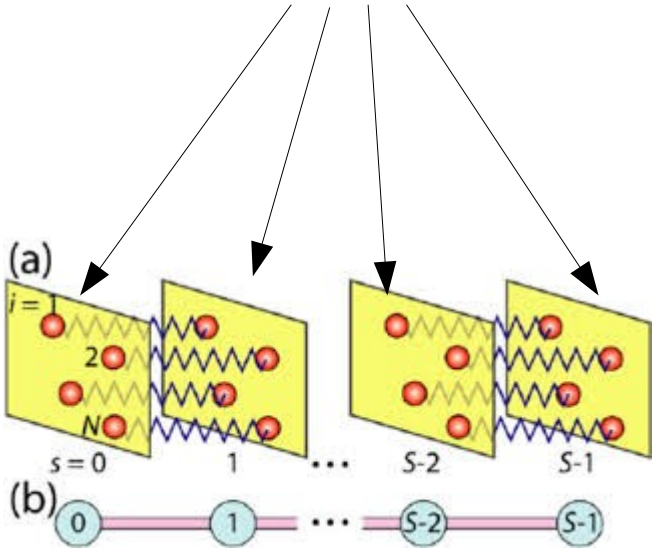
```
      DO idx = 1, fftx_ntgrp(dffts)

         IF( idx + ibnd - 1 <= m ) THEN
!$omp parallel do
            DO j = 1, n
               tg_psic(nls (igk_k(j,current_k))+ioff) =  psi(j,idx+ibnd-1)
            ENDDO
!$omp end parallel do
         ENDIF

         ioff = ioff + right_nnr
      ENDDO
```

More info at: http://qe-forge.org/pipermail/q-e-developers/2017-November/001782.html

# Other options for parallelism

## Nudged Elastic Band

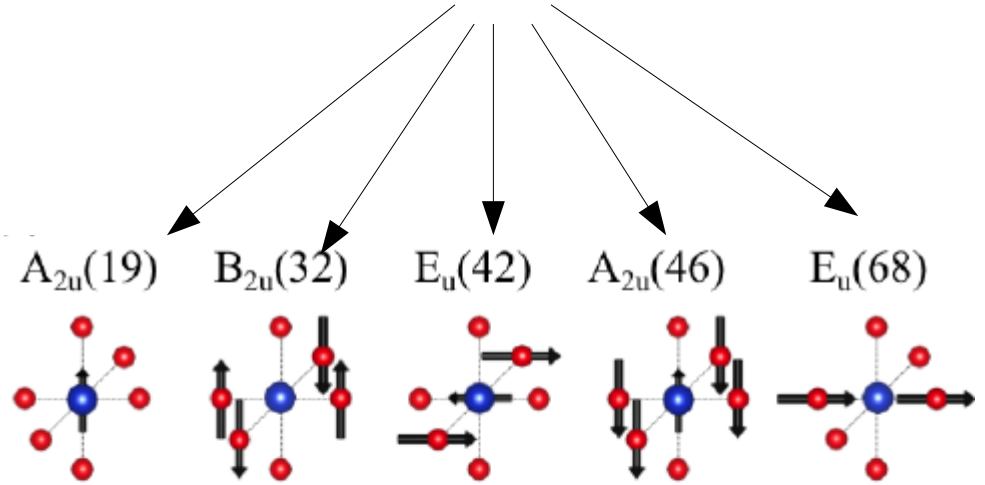
Images




DOI: 10.1016/j.cpc.2007.09.011

## PHonon (linear response)

Irreducible modes




DOI: 10.1103/PhysRevB.93.174419

# Other options for parallelism

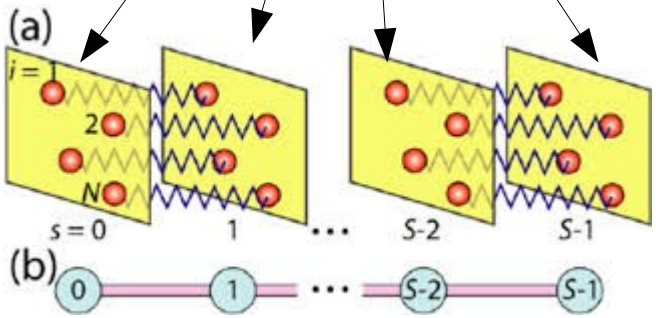
## Nudged Elastic Band

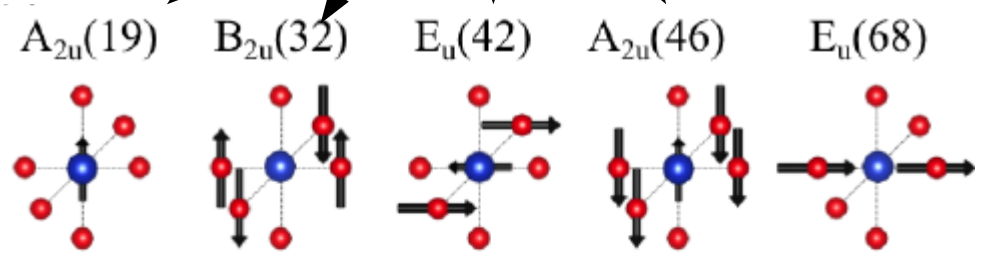
## PHonon (linear response)

Images

Irreducible modes

Abstract layer for ensemble-like parallelization

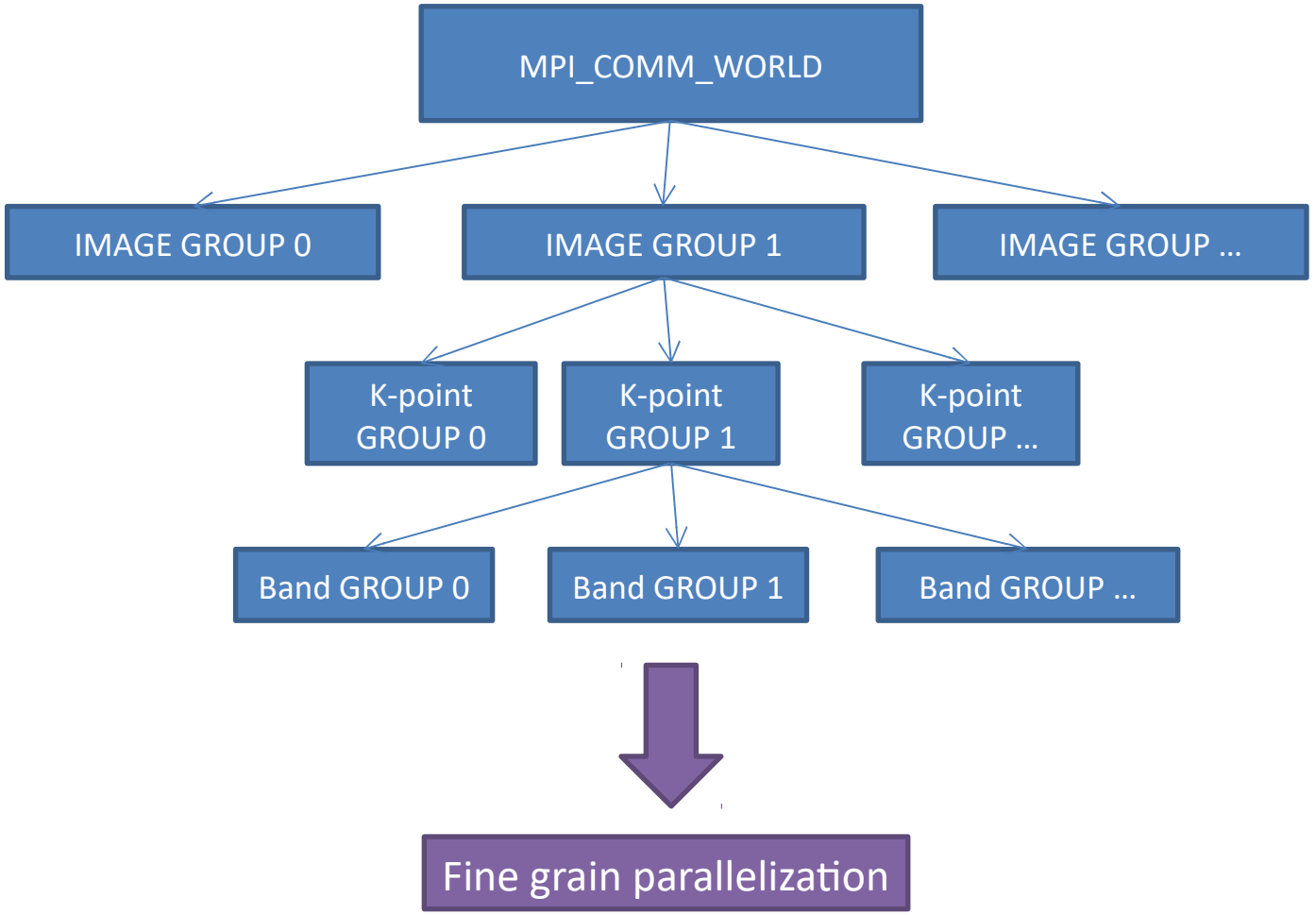


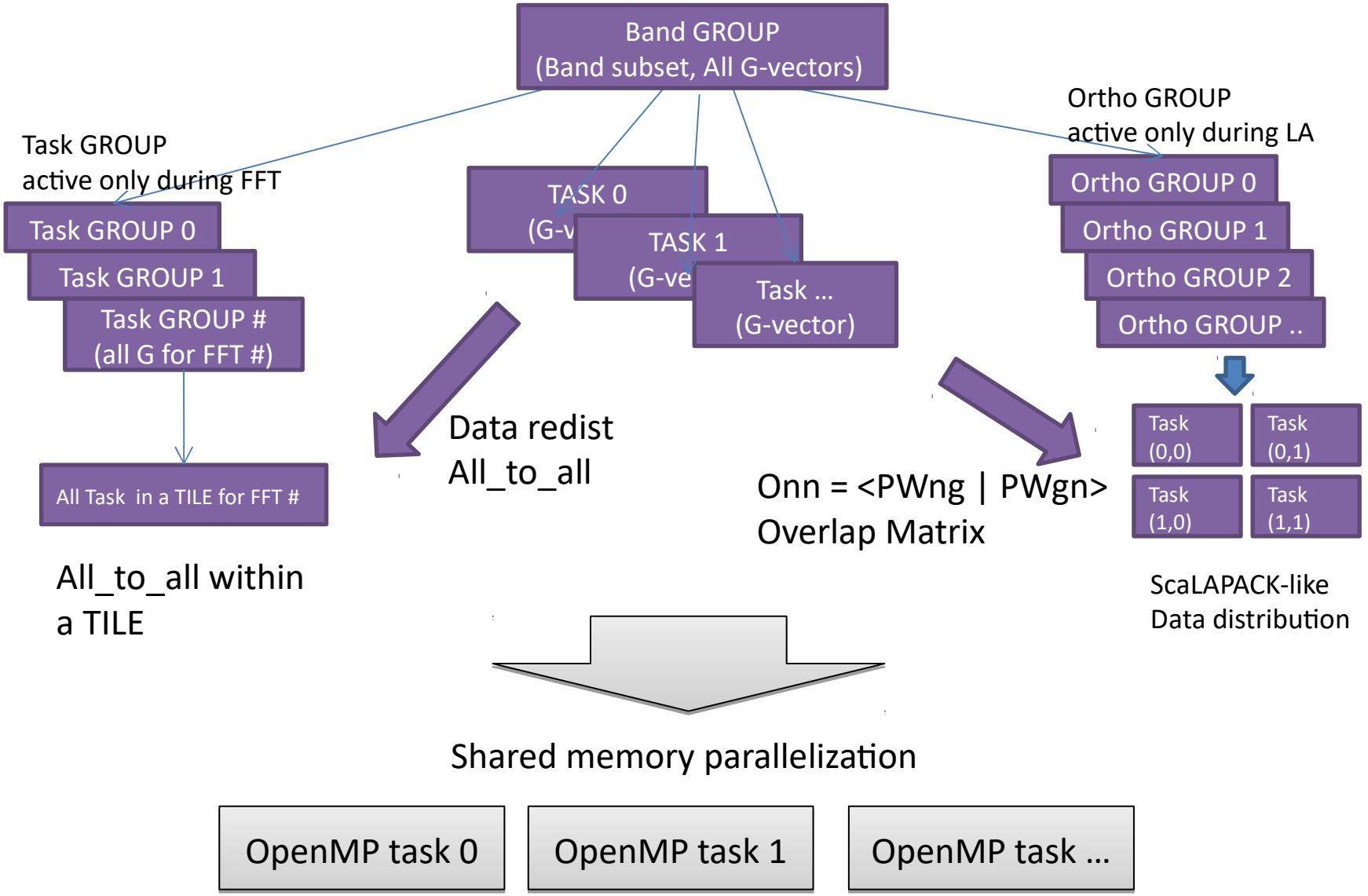

DOI: 10.1016/j.cpc.2007.09.011

DOI: 10.1103/PhysRevB.93.174419

# QE parallelization layout



# QE parallelization layout



# 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

```
mpirun pw.x –npool X –inp pw.in > pw.out
```

Output:

```
     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

```
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

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

Output

```
Subspace diagonalization (size of sub-
group: sqrt(Y)*sqrt(Y) procs)
```

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

Max value: 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

```
     R & G space division:  proc/nbgrp/npool/nimage =    1152
     wavefunctions fft division:  Y-proc x Z-proc =      Y     96
     wavefunctions fft division:  task group distribution
                                  #TG    x Z-proc =      Y     96
```

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

Max value: n(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)
- Configue PW with scalapack, ELPA and OpenMP:

```
$ ./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)
```

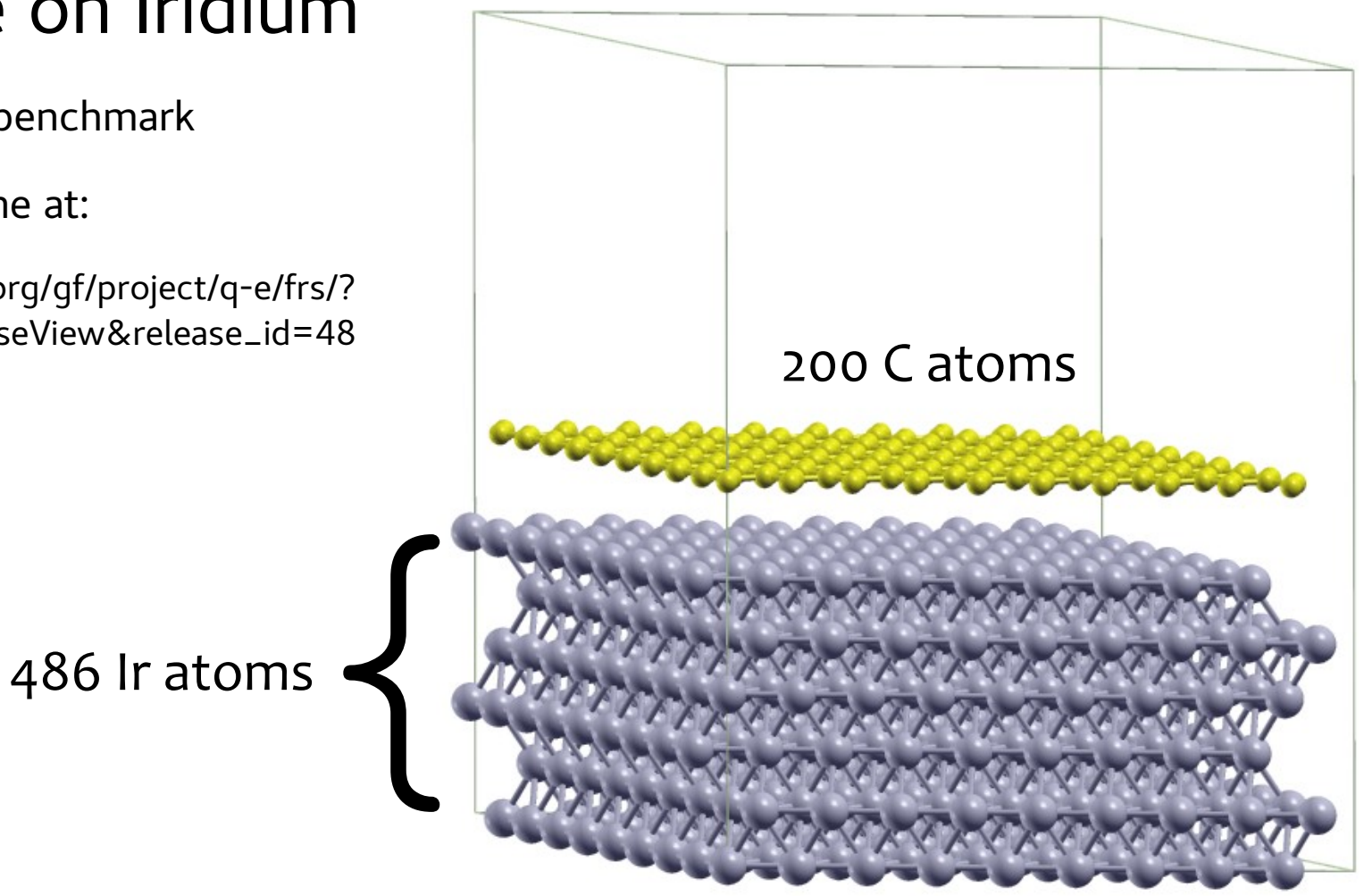
# A real case

## Grafene on Iridium

Standard QE benchmark

Available online at:

http://qe-forge.org/gf/project/q-e/frs/?action=FrsReleaseView&release_id=48



# Input

```
 &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?

# Naive run output

```
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
--------------------
sticks:   dense  smooth     PW     G-vecs:    dense   smooth      PW
Min         497     497    128                68583    68583    9018
Max         498     498    129                68592    68592    9025
Sum       17917   17917   4639              2469147  2469147  324807
```

# Memory

```
Dynamical RAM for                  wfc:     406.00 MB
Dynamical RAM for      wfc (w. buffer):    1217.99 MB
Dynamical RAM for            str. fact:       2.09 MB
Dynamical RAM for            local pot:       0.00 MB
Dynamical RAM for           nlocal pot:    1353.50 MB
Dynamical RAM for                 qrad:      13.00 MB
Dynamical RAM for           rho,v,vnew:      15.18 MB
Dynamical RAM for                rhoin:       5.06 MB
Dynamical RAM for             rho*nmix:      33.49 MB
Dynamical RAM for            G-vectors:       4.45 MB
Dynamical RAM for           h,s,v(r/c):    7056.75 MB
Dynamical RAM for           <psi|beta>:     490.12 MB
Dynamical RAM for                  psi:    1623.99 MB
Dynamical RAM for                 hpsi:    1623.99 MB
Dynamical RAM for                 spsi:    1623.99 MB
Dynamical RAM for       wfcinit/wfcrot:    1966.26 MB
Dynamical RAM for            addusdens:     704.31 MB
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

```
$ qsub -I -N GuiInteractive -A cin_staff -v "DISPLAY=$DISPLAY" -l
select=12:ncpus=36:mpiprocs=36:mem=100G -l walltime=0:25:00
```

# Happens more often than not

```
$ 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...

```
&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
 /
```

```
ocs=36:mem=118GB


ranularity=fine,1

-ntg 1 -inp pw.in > pw.out
```

RAM per process > **2.97 GB**

Estimated max dynamical RAM per process > **1.38 GB**

# 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
```

X: 1 Y: 36 Z: 1

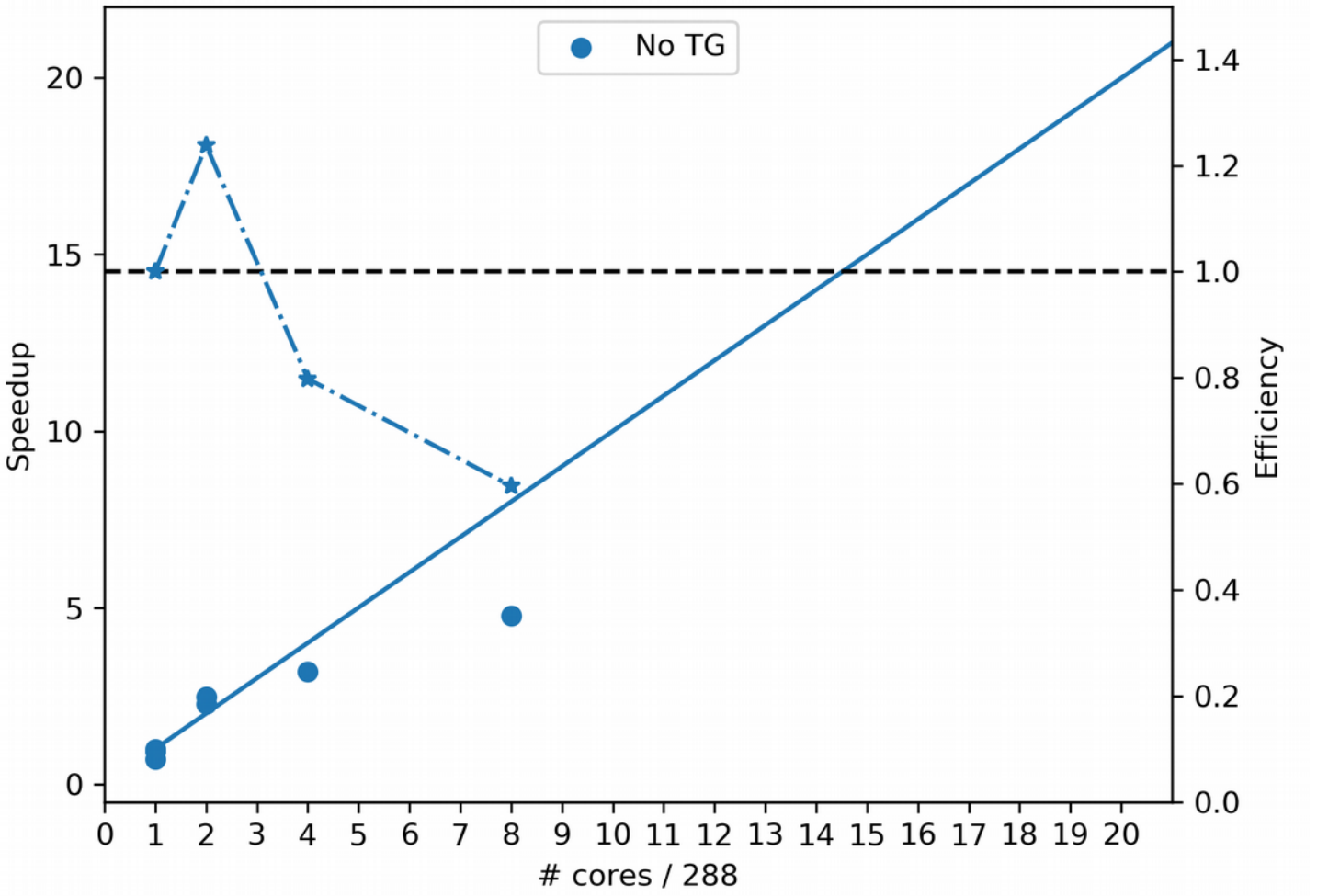
```
R & G space division:  proc/nbgrp/npool/nimage =     288

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

sticks:   dense  smooth     PW     G-vecs:    dense   smooth      PW
Min          62      62     16                 8568     8568    1120
Max          63      63     17                 8578     8578    1132
Sum       17917   17917   4639              2469147  2469147  324807

Dense  grid:  2469147 G-vectors     FFT dimensions: ( 180, 180, 216)
```

# Running on Marconi A1...

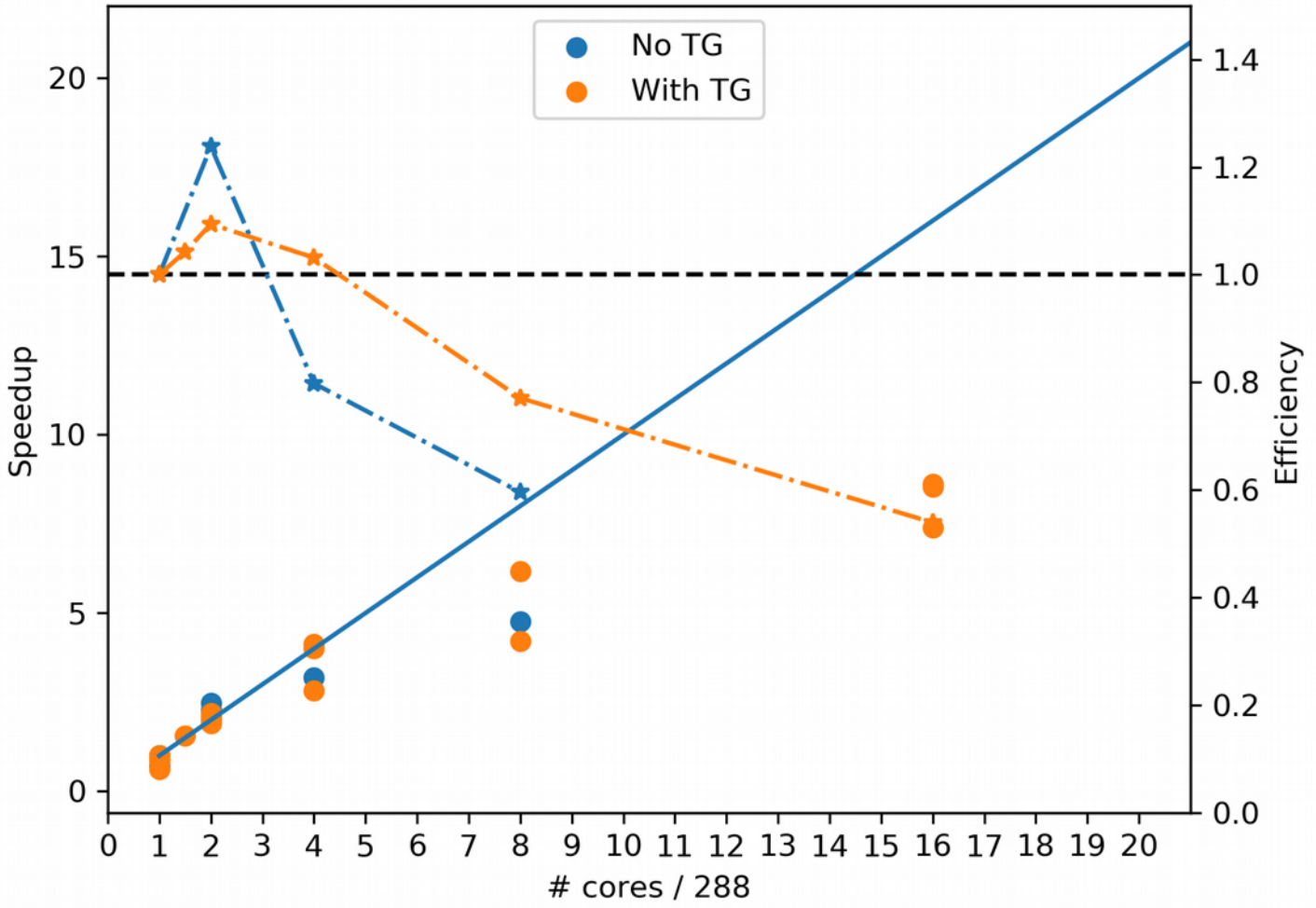
```
#!/bin/bash
#PBS -l select=8:ncpus=36:mpiprocs=36:mem=118GB
#PBS -l walltime=00:51:59

$ 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
```

| X | Y | Z | Time CPU | Time Wall |
|---|---|---|---|---|
| 1 | 36 | 1 | 2071.28s | 2108.70s |
| 1 | 144 | 1 | 1442.97s | 1470.78s |
| 1 | 256 | 1 | 1544.03s | 1576.22s |
| 1 | 144 | 2 | 1286.43s | 1312.28s |
| 1 | 144 | 4 | 1274.49s | 1299.47s |

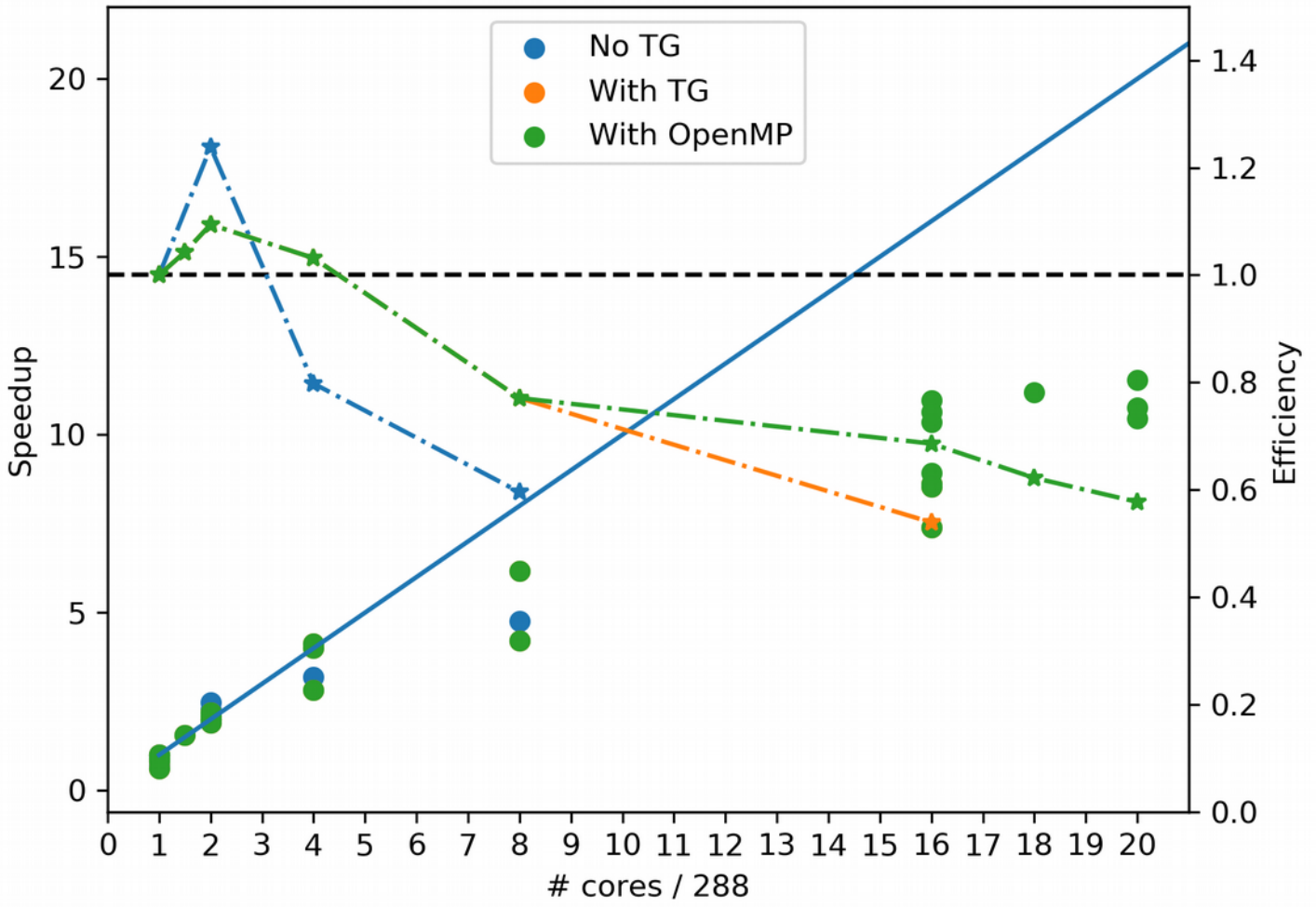
# Running on Marconi A1...



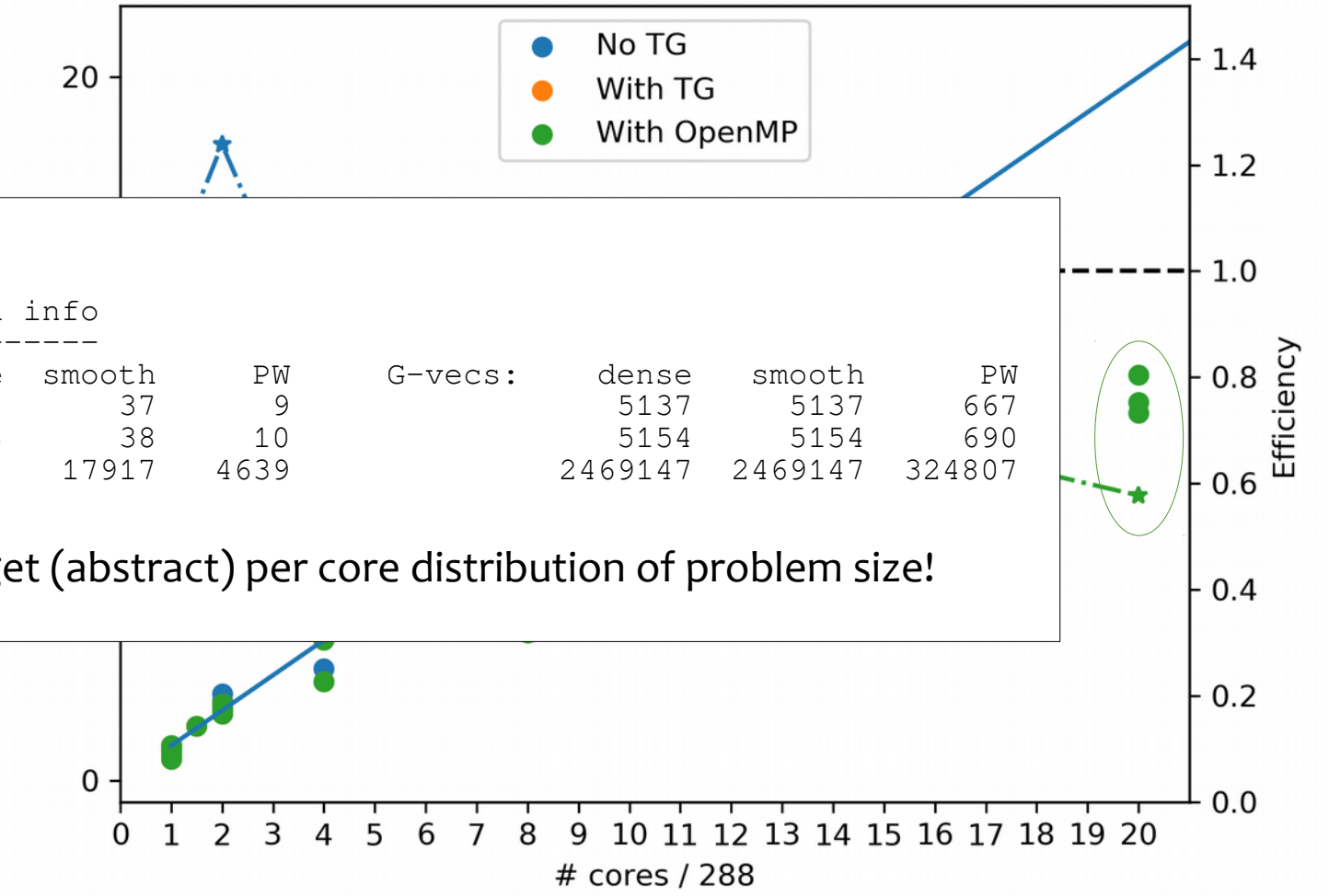
# Running on Marconi A1...



# 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


H2020
technology
platform
computing
strategic
HPC
exascale
MaX
materials
excellence
European
e-infrastructure
H2020-EINFRA-2015-1
Molinari
Modena
research
design
development
meeting
launch
new
project
news
data
Center
ETP4HPC
performance
centre-of-excellence
simulations
agenda
leadership
training
business
ecosystem
framework
education
computing
enabling
partners
support
call
Luisa
industry
transition
workable
PSI-K
discovery
modelling
researchers
strategy
High
user
driving
structure
projects
portal
open
activity
services
pilot
faster
research
code
1000x
supercomputing
cutting-edge
people
educational
navigation
material
challenge
funded
Update
welcome
changing
end-user
mission