

Day-5: hands-on session

Pietro Bonfà¹, Carlo Cavazzoni², Pietro Delugas³, Milan Jocić⁴, Drejc Kopač⁵

1. DSMFI, University of Parma, CNR S3-Nano Modena, Italy
2. CINECA, Casalecchio di Reno, Italy
3. SISSA, Trieste, Italy
4. Institute of Physics, Belgrade, Serbia
5. National Institute of Chemistry, Slovenia

What is pw.x telling us...

```
Program PWSCF v.6.4.1 starts on 7Sep2019 at 9:45: 3
```

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), running on 24 processors

MPI processes distributed on 1 nodes

R & G space division: proc/nbgrp/npool/nimage = 24

What is pw.x telling us...

```
Program PWSCF v0.6.4.5.1e star
```

This program is part of the Quantum Espresso distribution.
for quantum simulation of solid state systems.
D.P. Giannozzi et al.,
D.P. Giannozzi et al.,
URL <http://www.quantum-espresso.org>
in publications or present
<http://www.quantum-espresso.org>
Parallel version (MPI), run
MPI processes distributed
in R & G space division: prc

```
Subspace diagonalization in iterative solution of the eigenvalue problem:  
one sub-group per band group will be used  
custom distributed-memory algorithm (size of sub-group: 4* 4 procs)
```

```
Message from routine setup:  
DEPRECATED: symmetry with ibrav=0, use correct ibrav instead
```

Parallelization info

sticks:	dense	smooth	PW	G-vecs:	dense	smooth	PW
Min	1199	640	159		383982	149548	18695
Max	1202	642	162		384004	149562	18714
Sum	28829	15389	3855		9215799	3589319	448895

```
Title:  
DyOtBuClTHF_100K.cif
```

```
bravais-lattice index      =          0  
lattice parameter (alat)   =     25.6474 a.u.  
unit-cell volume           = 37134.3792 (a.u.)^3  
number of atoms/cell       =         608  
number of atomic types     =          6  
number of electrons         =    1512.00  
number of Kohn-Sham states=        756  
kinetic-energy cutoff       =    80.0000 Ry  
charge density cutoff      = 600.0000 Ry  
convergence threshold      = 1.0E-09  
mixing beta                 =    0.5000  
number of iterations used  =          8 plain      mixing  
Exchange-correlation        = SLA PW PBE PBE ( 1 4 3 4 0 0 )  
nstep                      =          400
```

How is time spent by pw.x

```
init_run      :    42.99s CPU    46.16s WALL (      1 calls)
electrons     : 60819.95s CPU  63107.94s WALL (    83 calls)
update_pot    : 1461.58s CPU   1522.64s WALL (    82 calls)
forces        : 17437.52s CPU  17714.01s WALL (    82 calls)

Called by init_run:
wfcinit       :    28.44s CPU    29.07s WALL (      1 calls)
potinit       :     0.79s CPU     2.21s WALL (      1 calls)
hinit0        :     8.50s CPU     8.60s WALL (      1 calls)

Called by electrons:
c_bands       : 37126.13s CPU  37854.42s WALL (   889 calls)
sum_band      :  9663.72s CPU  10448.81s WALL (   889 calls)
v_of_rho      :   501.54s CPU   536.25s WALL (   890 calls)
newd          :  2620.20s CPU  3367.58s WALL (   890 calls)
mix_rho       :   116.23s CPU   122.01s WALL (   889 calls)

Called by c_bands:
init_us_2     :   296.76s CPU   297.22s WALL ( 1779 calls)
regterg       : 36350.79s CPU  36971.49s WALL (   889 calls)

Called by sum_band:
sum_band:bec :     6.01s CPU     6.08s WALL (   889 calls)
addusdens    : 3042.08s CPU  3745.04s WALL (   889 calls)

Called by *egterg:
h_psi         : 24521.86s CPU  24722.48s WALL ( 3704 calls)
s_psi         :  3235.74s CPU  3235.97s WALL ( 3704 calls)
g_psi         :   40.31s CPU   40.48s WALL ( 2814 calls)
rdiaghg      : 2592.35s CPU  2678.62s WALL ( 3540 calls)

Called by h_psi:
h_psi:pot    : 24426.82s CPU  24627.23s WALL ( 3704 calls)
h_psi:calbec : 3349.63s CPU  3389.39s WALL ( 3704 calls)
vloc_psi     : 17839.75s CPU  17998.35s WALL ( 3704 calls)
add_vuspsi   : 3237.38s CPU  3239.45s WALL ( 3704 calls)
```

How is time spent by pw.x

```
init_run      :    42.99s CPU    46.16s WALL (  1 call)
electrons     : 60819.95s CPU  63107.94s WALL ( 889 calls)
update_pot    : 1461.58s CPU   1522.64s WALL (  1 call)
forces        : 17437.52s CPU  17714.01s WALL (  1 call)

Called by init_run:
wfcinit       :    28.44s CPU    29.07s WALL (  1 call)
potinit       :     0.79s CPU     2.21s WALL (  1 call)
hinit0        :     8.50s CPU    8.60s WALL (  1 call)

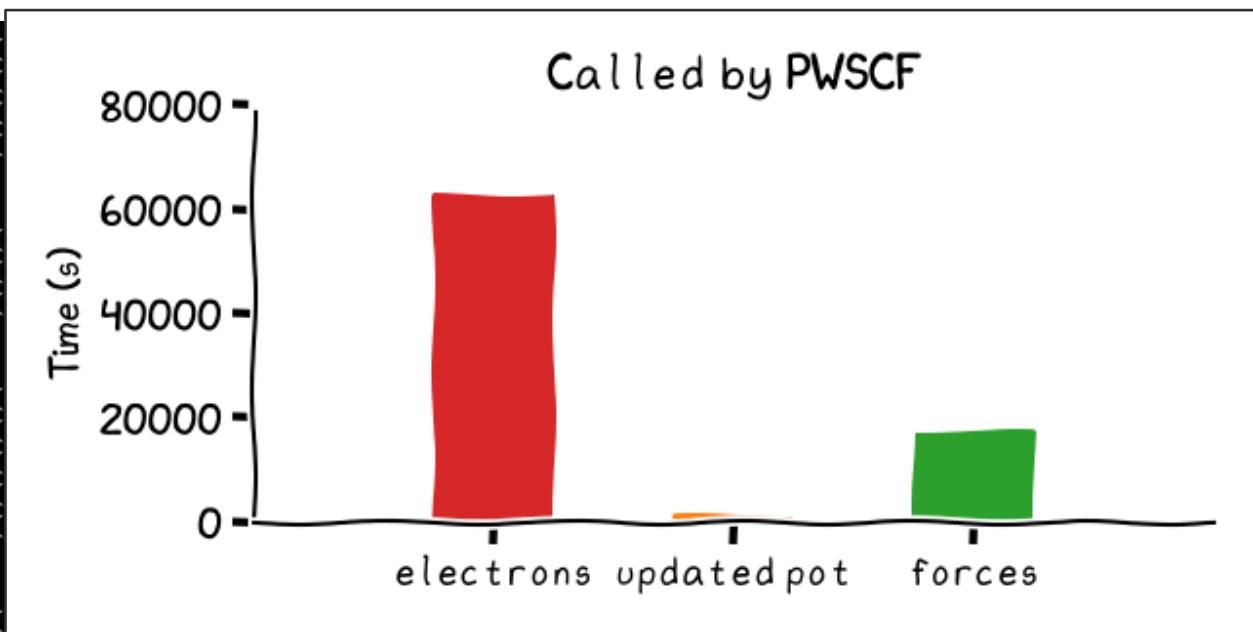
Called by electrons:
c_bands       : 37126.13s CPU  37854.42s WALL (  1 call)
sum_band      :  9663.72s CPU  10448.81s WALL (  1 call)
v_of_rho      :   501.54s CPU   536.25s WALL (  1 call)
newd          :  2620.20s CPU  3367.58s WALL (  1 call)
mix_rho       :   116.23s CPU   122.01s WALL (  1 call)

Called by c_bands:
init_us_2     :   296.76s CPU   297.22s WALL (  1 call)
regterg       : 36350.79s CPU 36971.49s WALL ( 889 calls)

Called by sum_band:
sum_band:bec :     6.01s CPU     6.08s WALL ( 889 calls)
addusdens    : 3042.08s CPU  3745.04s WALL ( 889 calls)

Called by *egterg:
h_psi         : 24521.86s CPU 24722.48s WALL ( 3704 calls)
s_psi         :  3235.74s CPU  3235.97s WALL ( 3704 calls)
g_psi         :   40.31s CPU   40.48s WALL ( 2814 calls)
rdiaghg      : 2592.35s CPU  2678.62s WALL ( 3540 calls)

Called by h_psi:
h_psi:pot    : 24426.82s CPU 24627.23s WALL ( 3704 calls)
h_psi:calbec :  3349.63s CPU  3389.39s WALL ( 3704 calls)
vloc_psi     : 17839.75s CPU 17998.35s WALL ( 3704 calls)
add_vuspsi   :  3237.38s CPU  3239.45s WALL ( 3704 calls)
```



How is time spent by pw.x

```
init_run      :    42.99s CPU    46.16s WALL (    1 calls)
electrons     : 60819.95s CPU  63107.94s WALL (3704 calls)
update_pot    : 1461.58s CPU   1522.64s WALL (    1 calls)
forces        : 17437.52s CPU  17714.01s WALL (3704 calls)

Called by init_run:
wfcinit       :    28.44s CPU    29.07
potinit       :     0.79s CPU    2.21
hinit0        :     8.50s CPU    8.60

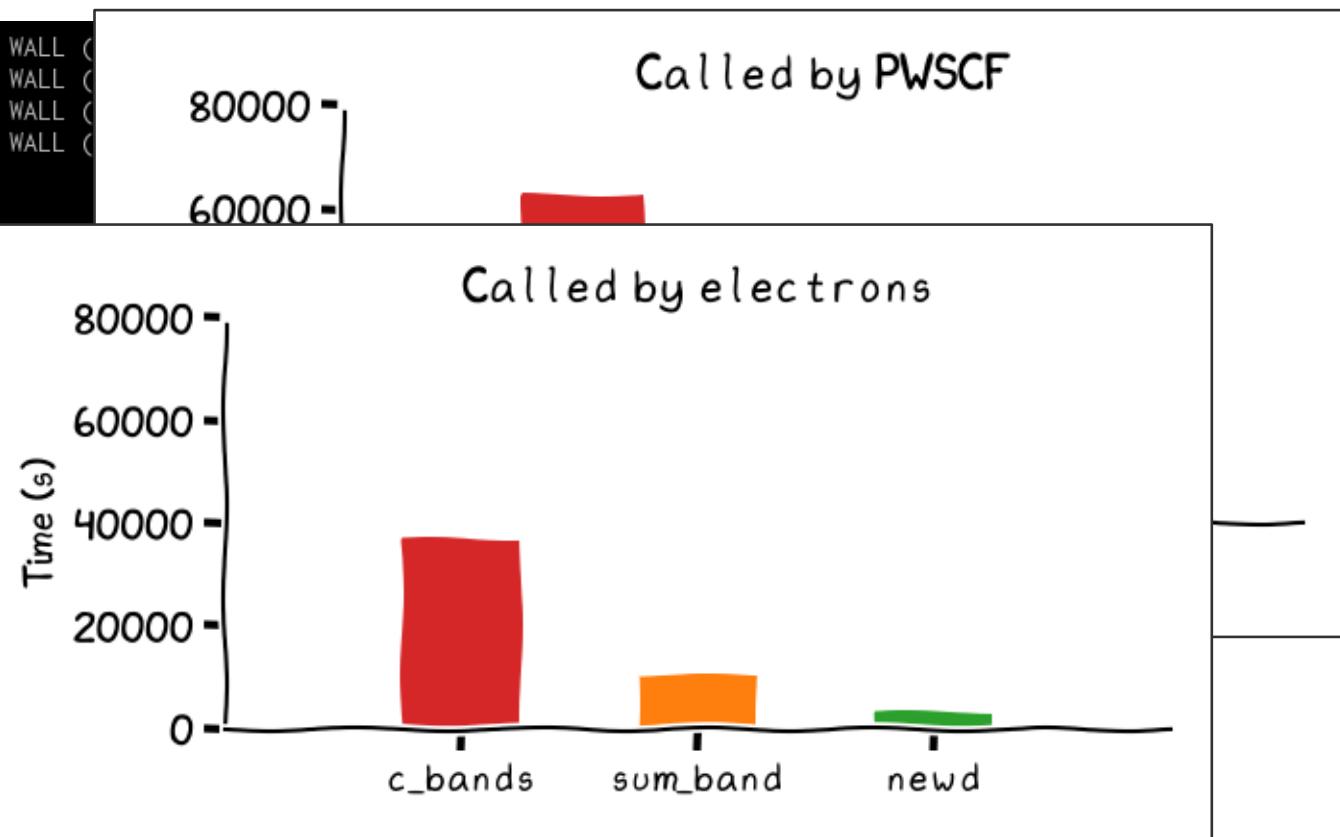
Called by electrons:
c_bands       : 37126.13s CPU  37854.42
sum_band      :  9663.72s CPU  10448.81
v_of_rho      :   501.54s CPU   536.25
newd          :  2620.20s CPU  3367.58
mix_rho       :   116.23s CPU  122.01

Called by c_bands:
init_us_2     :   296.76s CPU   297.22
regterg       : 36350.79s CPU 36971.49

Called by sum_band:
sum_band:bec :     6.01s CPU    6.08
addusdens    : 3042.08s CPU  3745.04

Called by *egterg:
h_psi         : 24521.86s CPU 24722.48s WALL ( 3704 calls)
s_psi         :  3235.74s CPU  3235.97s WALL ( 3704 calls)
g_psi         :   40.31s CPU   40.48s WALL ( 2814 calls)
rdiaghg      : 2592.35s CPU  2678.62s WALL ( 3540 calls)

Called by h_psi:
h_psi:pot    : 24426.82s CPU 24627.23s WALL ( 3704 calls)
h_psi:calbec :  3349.63s CPU  3389.39s WALL ( 3704 calls)
vloc_psi     : 17839.75s CPU 17998.35s WALL ( 3704 calls)
add_vuspsi   :  3237.38s CPU  3239.45s WALL ( 3704 calls)
```



How is time spent by pw.x

```
init_run      :    42.99s CPU    46.16s WALL (  1 call)
electrons     : 60819.95s CPU  63107.94s WALL ( 3704 calls)
update_pot    : 1461.58s CPU   1522.64s WALL (  1 call)
forces        : 17437.52s CPU  17714.01s WALL (  1 call)
```

Called by init_run:

```
wfcinit       :    28.44s CPU    29.07s WALL (  1 call)
potinit       :     0.79s CPU     2.21s WALL (  1 call)
hinit0        :     8.50s CPU    8.60s WALL (  1 call)
```

Called by electrons:

```
c_bands       : 37126.13s CPU  37854.42s WALL (  1 call)
sum_band      :  9663.72s CPU  10448.81s WALL (  1 call)
v_of_rho      :   501.54s CPU   536.25s WALL (  1 call)
newd          : 2533.33s CPU  2537.50s WALL (  1 call)
mix_rho       :
```

Called by c_bands:

```
init_us_2     :
regterg      :    36.00s CPU   36.00s WALL (  1 call)
```

Called by sum_band

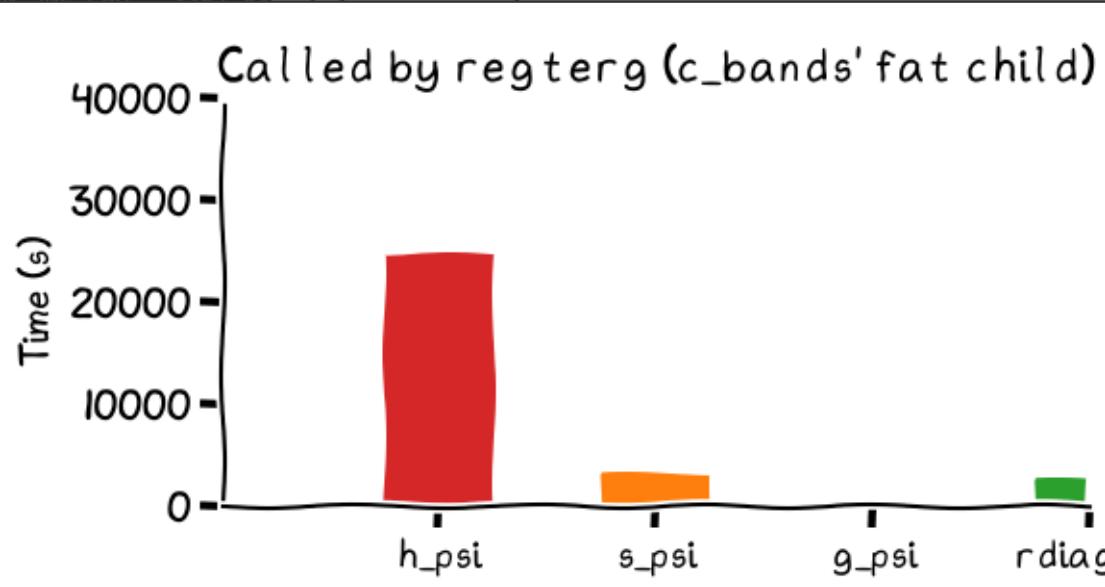
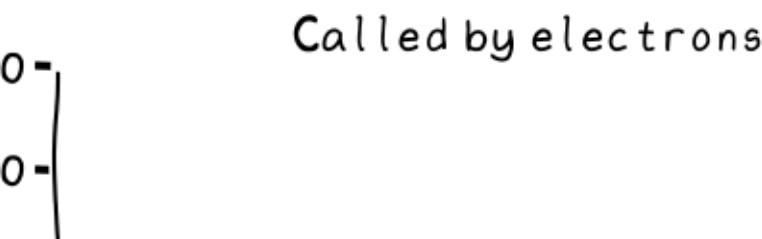
```
sum_band:bec  :
addusdens   :    3.00s CPU   3.00s WALL (  1 call)
```

Called by *egterg:

```
h_psi         :    24.00s CPU   24.00s WALL (  1 call)
s_psi         :     3.00s CPU   3.00s WALL (  1 call)
g_psi         :
rdiaghg     :     2.00s CPU   2.00s WALL (  1 call)
```

Called by h_psi:

```
h_psi:pot    :    24.00s CPU   24.00s WALL (  1 call)
h_psi:calbec: 3.00s CPU   3.00s WALL (  1 call)
vloc_psi     : 17839.75s CPU  17998.35s WALL ( 3704 calls)
add_vuspsi   : 3237.38s CPU  3239.45s WALL ( 3704 calls)
```



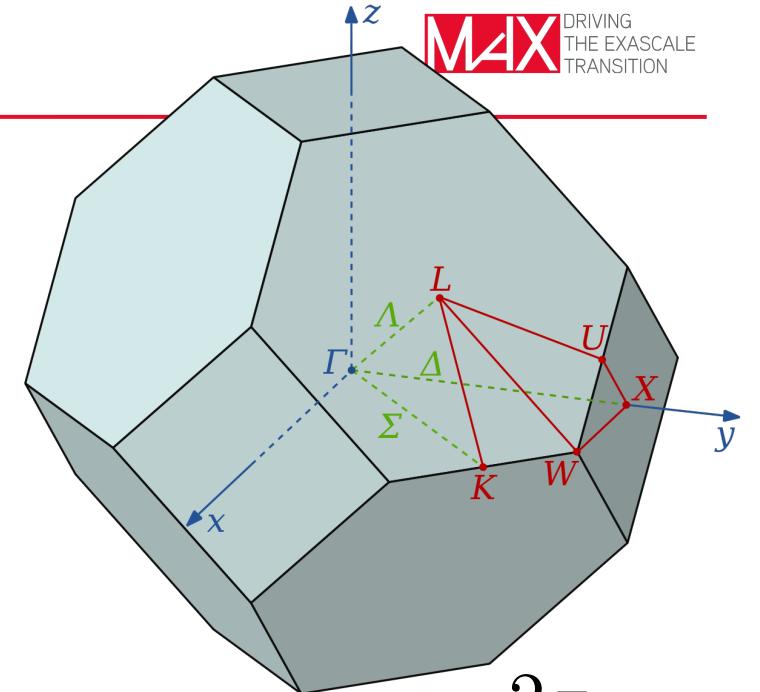
(real|complex) eigen iterative generalized
(real|complex) diagonalization H generalized

Plane waves

Orbitals in Fourier space

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

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



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

For a periodic system (Bloch's Theorem)

$$\phi_i(\mathbf{r}) = u_{ik}(\mathbf{r}) e^{i\mathbf{k}\mathbf{r}}$$

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

Complete basis set!

\mathbf{k} is in FBZ

Plane waves

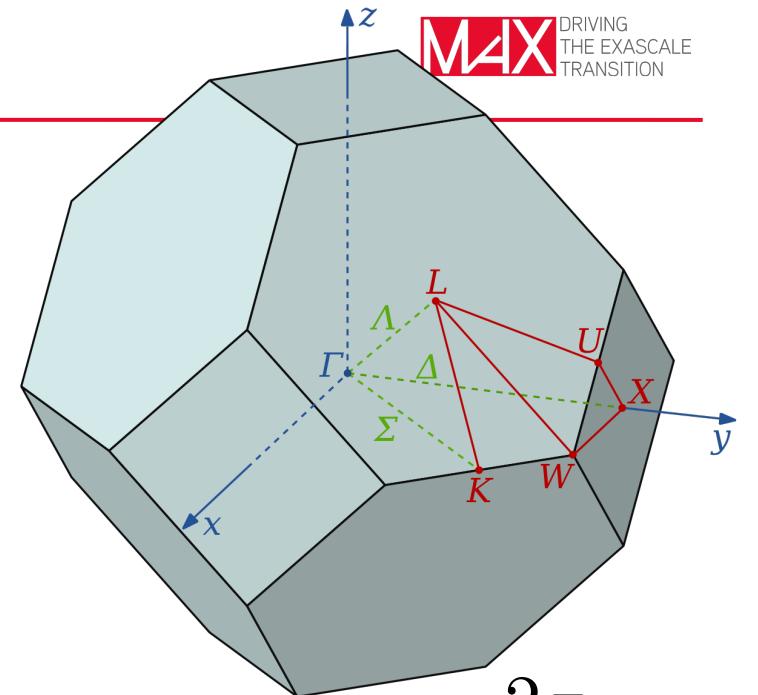
Orbitals in Fourier space

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

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

For a periodic system (Bloch's Theorem)

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



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

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

\mathbf{k} is in FBZ

Plane waves

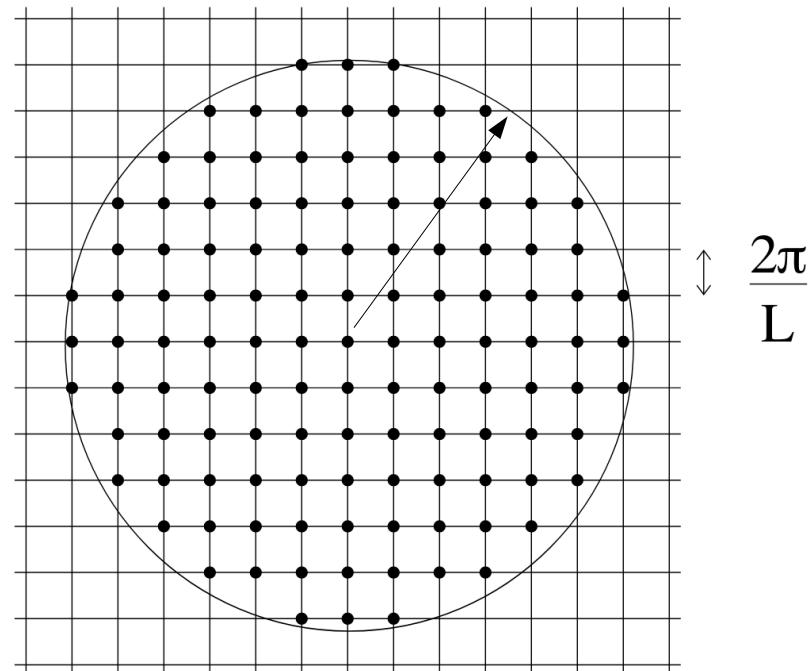
$$E_c = \frac{\hbar^2 G_{max}^2}{2m_e}$$

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

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

Cell volume

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



Computation and data partition

MAX DRIVING
THE EXASCALE
TRANSITION

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

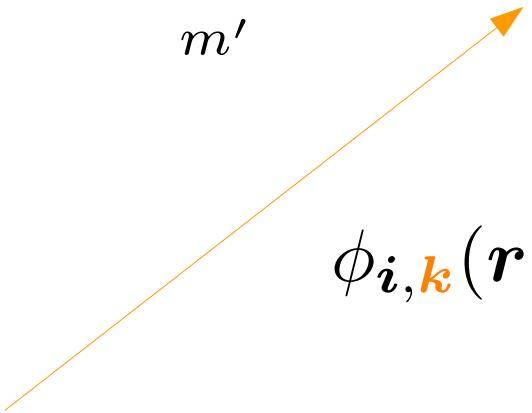
$$\phi_{i,\mathbf{k}}(\mathbf{r}) \propto \sum_{\mathbf{G}} c_{i,\mathbf{G}}(\mathbf{k}) e^{i(\mathbf{G}+\mathbf{k})\mathbf{r}}$$

How to split tasks and data among many workers
(not necessarily close to each others)

Computation and data partition

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

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



Multiple \mathbf{k} points:
pools of processors

Computation and data partition

MAX DRIVING
THE EXASCALE
TRANSITION

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

$$\phi_{i,\mathbf{k}}(\mathbf{r}) \propto \sum_{\mathbf{G}} c_{i,\mathbf{G}}(\mathbf{k}) e^{i(\mathbf{G} + \mathbf{k}) \cdot \mathbf{r}}$$

Multiple k points:
pools of processors

G domain decomposition: band group

Computation and data partition

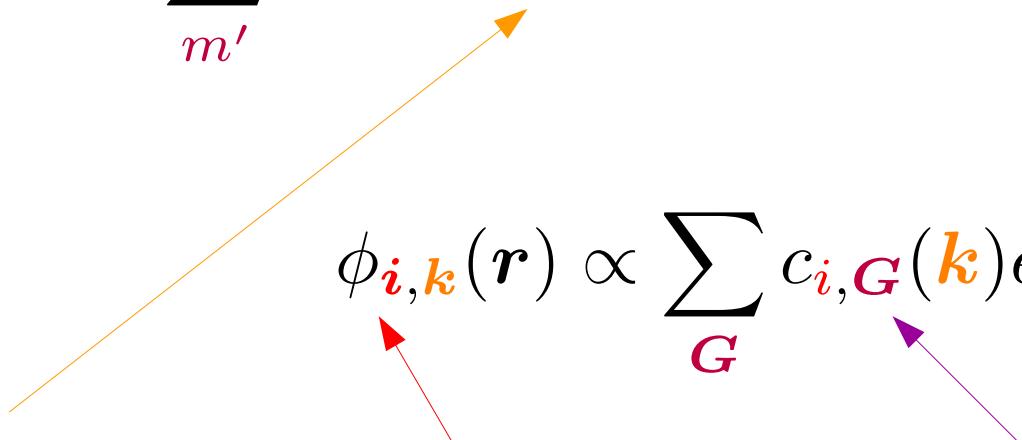
$$\sum_{m'} H_{m,m'}(\mathbf{k}) c_{i,m'}(\mathbf{k}) = \epsilon_i(\mathbf{k}) c_{i,m}(\mathbf{k})$$

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

Multiple k points:
pools of processors

Multiple KS states

G domain decomposition: band group



Solving the Hamiltonian problem

MAX DRIVING
THE EXASCALE
TRANSITION

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

Diagonalize  $\rightarrow \{\phi_i, \epsilon_i\}, i = 1 \dots N_G$

What is actually needed are $\sim \frac{1}{2} N_e$ orbitals

Guess $|\phi_j\rangle j = 1 \dots N_b$ prepare $H(\rho)$ $h_{\phi_i \phi_j} = \langle \phi_i | H | \phi_j \rangle$

Solving the Hamiltonian problem

MAX DRIVING
THE EXASCALE
TRANSITION

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

Diagonalize $\rightarrow \{\phi_i, \epsilon_i\}, i = 1 \dots N_G$

What is actually needed are $\sim \frac{1}{2} N_e$ orbitals

Guess $|\phi_j\rangle j = 1 \dots N_b$ prepare $H(\rho)$ $h_{\phi_i \phi_j} = \langle \phi_i | H | \phi_j \rangle$

$$h v_k = \tilde{\epsilon}_k v_k$$

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

Solving the eigenvalue problem

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

Diagonalize $\rightarrow \{\phi_i, \epsilon_i\}, i = 1 \dots N_G$

What is actually needed are $\sim \frac{1}{2} N_e$ orbitals

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

Now use $|\phi_j\rangle |\delta_j\rangle j = 1 \dots N_b$ to build

$$\delta_k = (H_{diag} - \epsilon_k S_{diag})^{-1} (H - \epsilon_k S) |\phi_k\rangle$$

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

Solving the eigenvalue problem

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

Diagonalize $\rightarrow \{\phi_i, \epsilon_i\}, i = 1 \dots N_G$

What is actually needed are $\sim \frac{1}{2} N_e$ orbitals

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

Now use $|\phi_j\rangle |\delta_j\rangle j = 1 \dots N_b$ to build

$$\delta_k = (H_{diag} - \epsilon_k S_{diag})^{-1} (H - \epsilon_k S) |\phi_k\rangle$$

Really small?

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

Wave-function & density

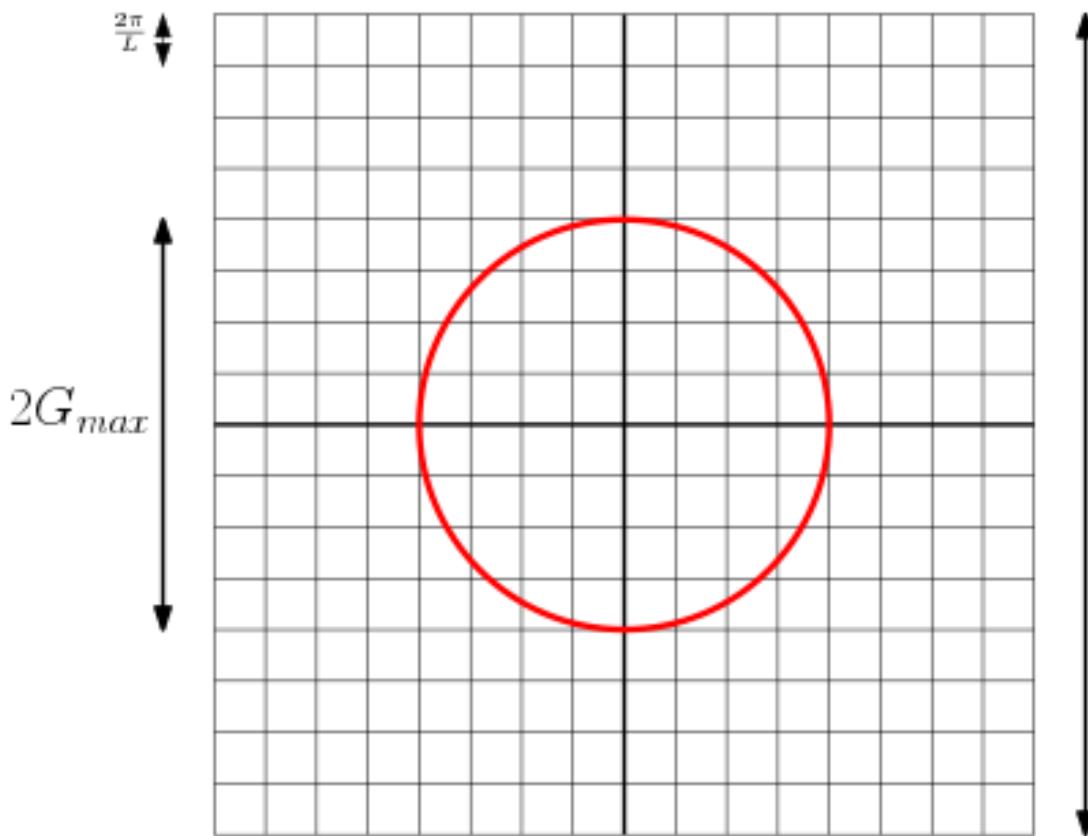
MAX DRIVING
THE EXASCALE
TRANSITION

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

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

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

Real and reciprocal space



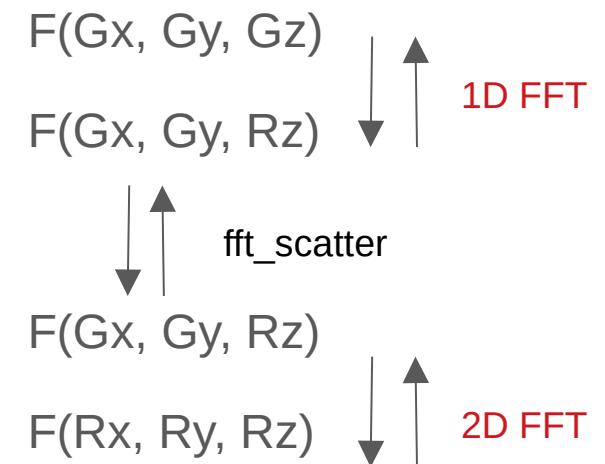
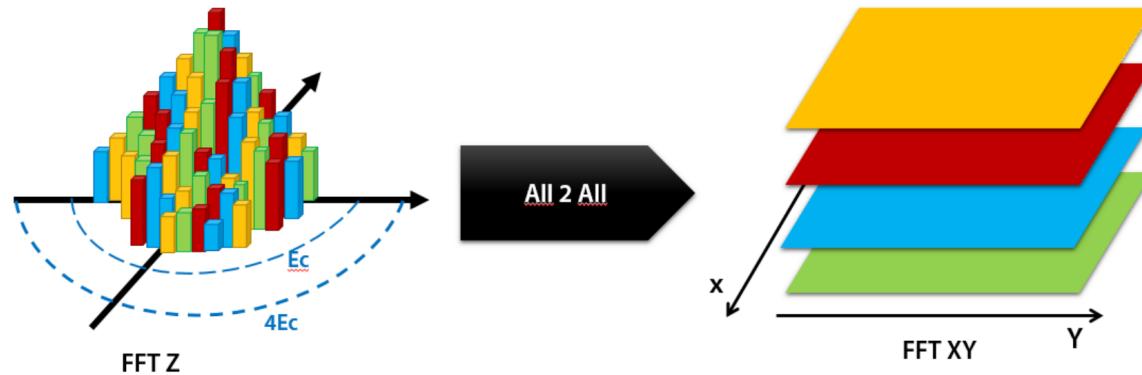
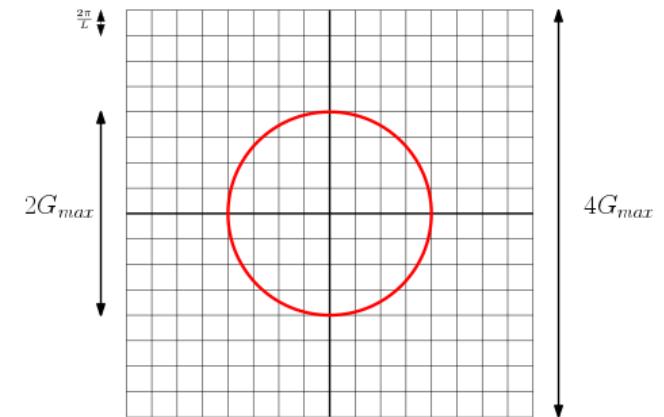
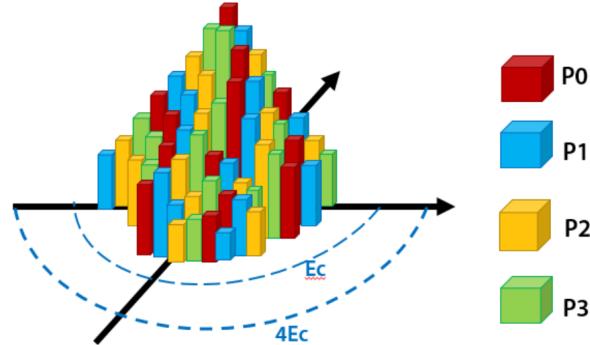
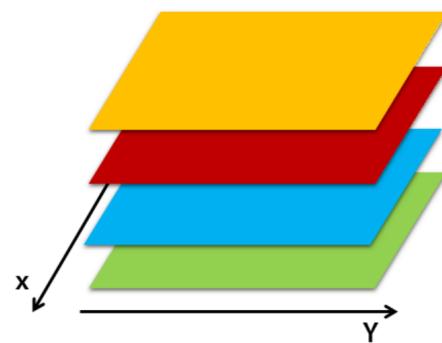
$$c_{ik}(\mathbf{r}) = \sum_{\mathbf{G}} c_{ik}(\mathbf{G}) e^{i\mathbf{Gr}}$$

FFT

$$c_{ik}(\mathbf{G}) = \frac{1}{N} \sum_{\mathbf{r}} c_{ik}(\mathbf{r}) e^{-i\mathbf{Gr}}$$

Real and reciprocal space

MAX DRIVING
THE EXASCALE
TRANSITION

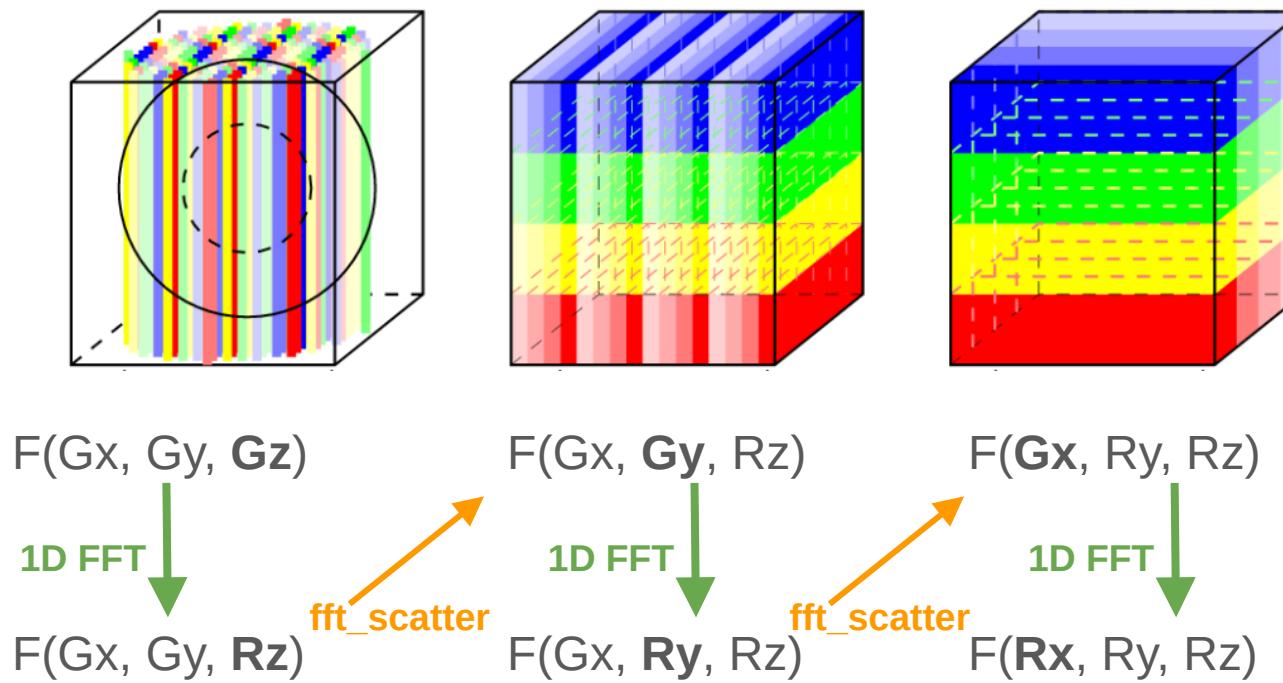


Eric Pascolo, master thesis

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)
 - Split processors and compute multiple FFT

Multiple 1D FFTs



Local potential

The local potential contribution is computed more efficiently in real space:

$$\psi_{ik}(\mathbf{G}) \xrightarrow{\text{FFT}} \psi_{ik}(\mathbf{r})$$

$$[v_{KS}\psi_{ik}](\mathbf{r}) = v_{KS}(\mathbf{r})\psi_{ik}(\mathbf{r})$$

$$[v_{KS}\psi_{ik}](\mathbf{r}) \xrightarrow{\text{FFT}} [v_{KS}\psi_{ik}](\mathbf{G})$$

Local potential

The local potential contribution is computed more efficiently in real space:

$$\psi_{ik}(\mathbf{G}) \xrightarrow{\text{FFT}} \psi_{ik}(\mathbf{r})$$

$$[v_{KS}\psi_{ik}](\mathbf{r}) = v_{KS}(\mathbf{r})\psi_{ik}(\mathbf{r})$$

$$[v_{KS}\psi_{ik}](\mathbf{r}) \xrightarrow{\text{FFT}} [v_{KS}\psi_{ik}](\mathbf{G})$$

Task groups

vloc_psi

```
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
!
```

correspondence between the fft mesh points and the array of g vectors.

The diagram illustrates the flow of data through the code. It starts with 'Orbitals' (represented by a green arrow pointing left), which is converted to a 'Wave-function' (blue arrow pointing right). This wave-function is then used in a 'Compute G-space to R-space' step (green arrow pointing right). After this, it undergoes a 'Compute R-space to G-space' step (blue arrow pointing left). Finally, the data is added to 'hpsi' (red arrow pointing right).

Task groups

```
IF( use_tg ) THEN
    CALL tg_get_nnr( dffts, right_nnr )

    DO ibnd = 1, m, fftx_ntgrp(dffts) ← More than one band at a time
        !
        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

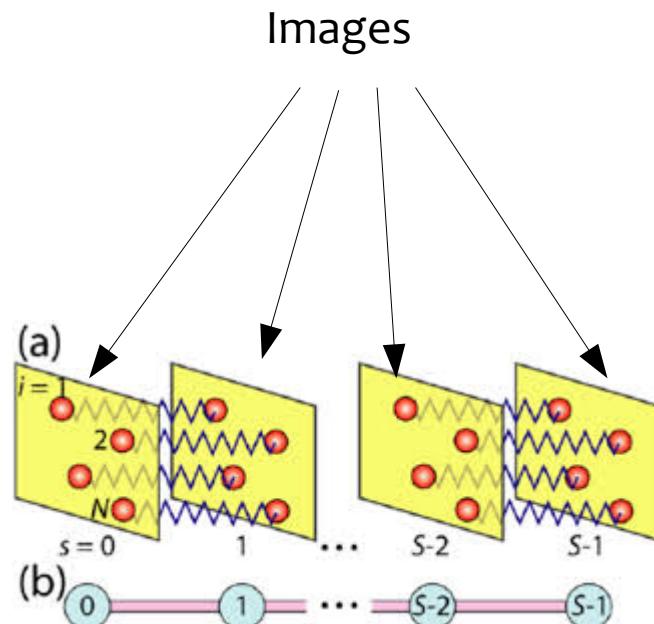
```

```
        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
    
```

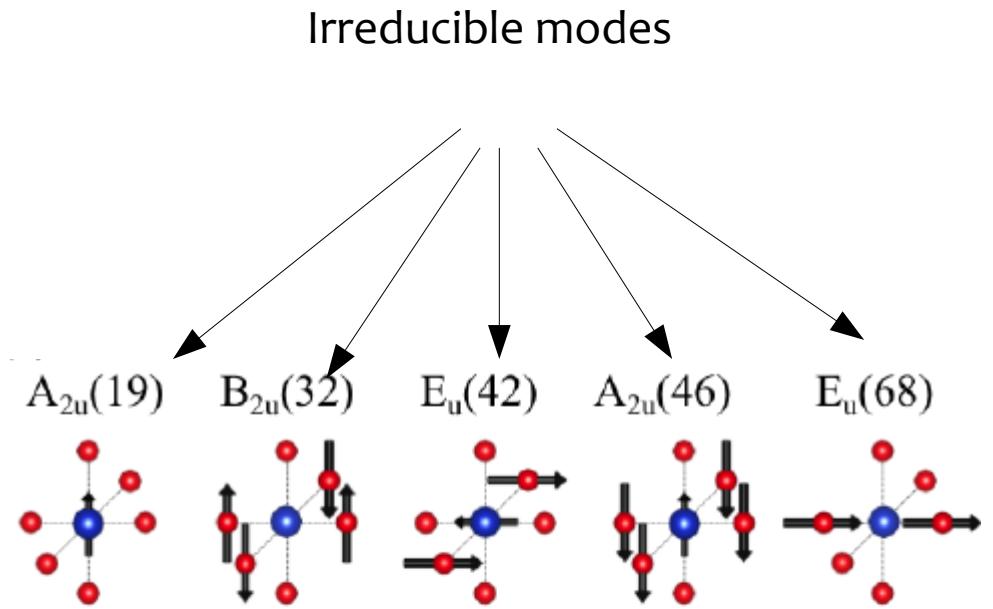
Other options for parallelism

MAX DRIVING
THE EXASCALE
TRANSITION

Nudged Elastic Band

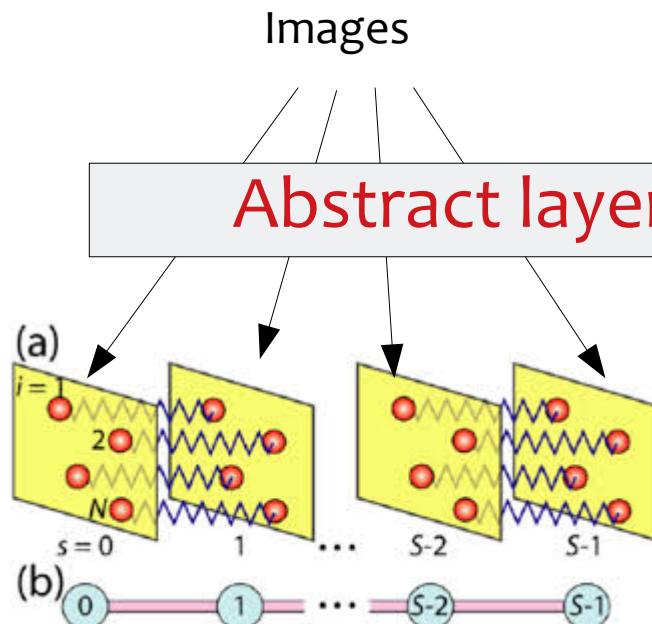


PHonon (linear response)



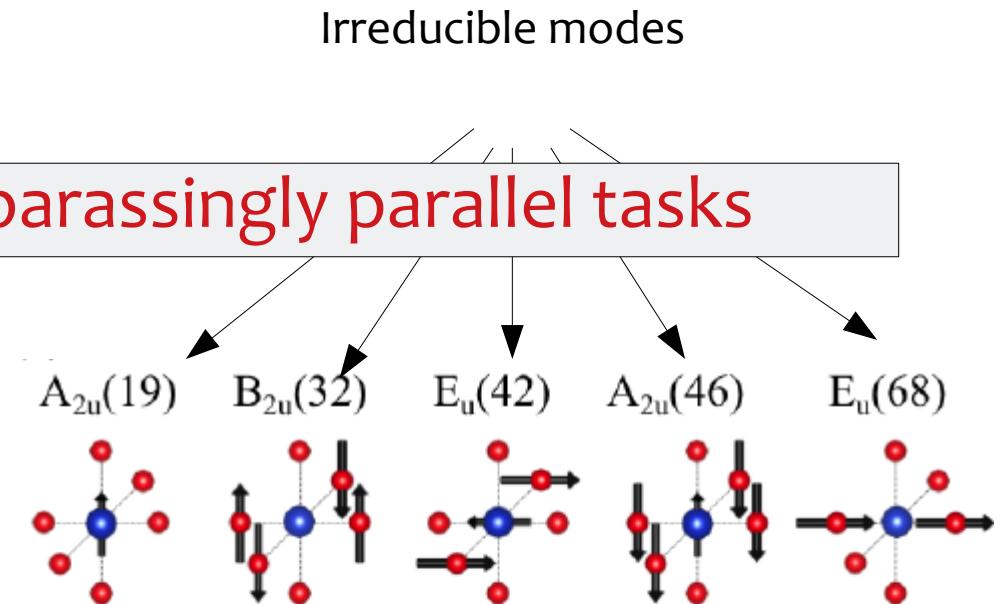
Other options for parallelism

Nudged Elastic Band



DOI: [10.1016/j.cpc.2007.09.011](https://doi.org/10.1016/j.cpc.2007.09.011)

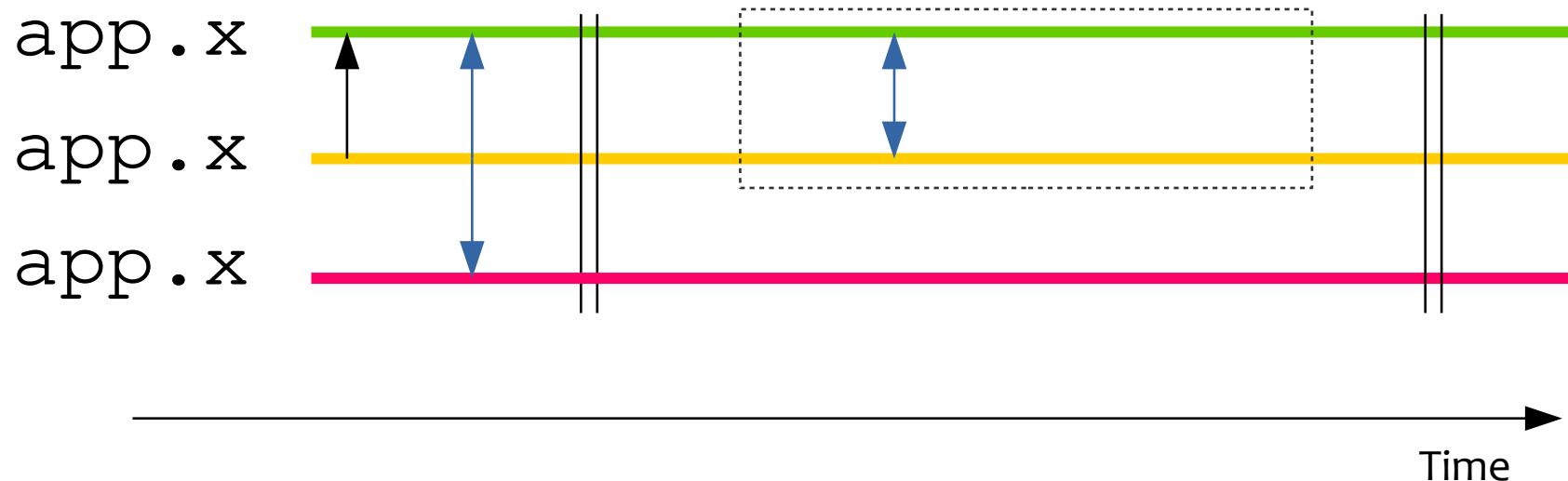
PHonon (linear response)



DOI: [10.1103/PhysRevB.93.174419](https://doi.org/10.1103/PhysRevB.93.174419)

Message Passing Interface:

```
mpirun -np 3 app.x
```

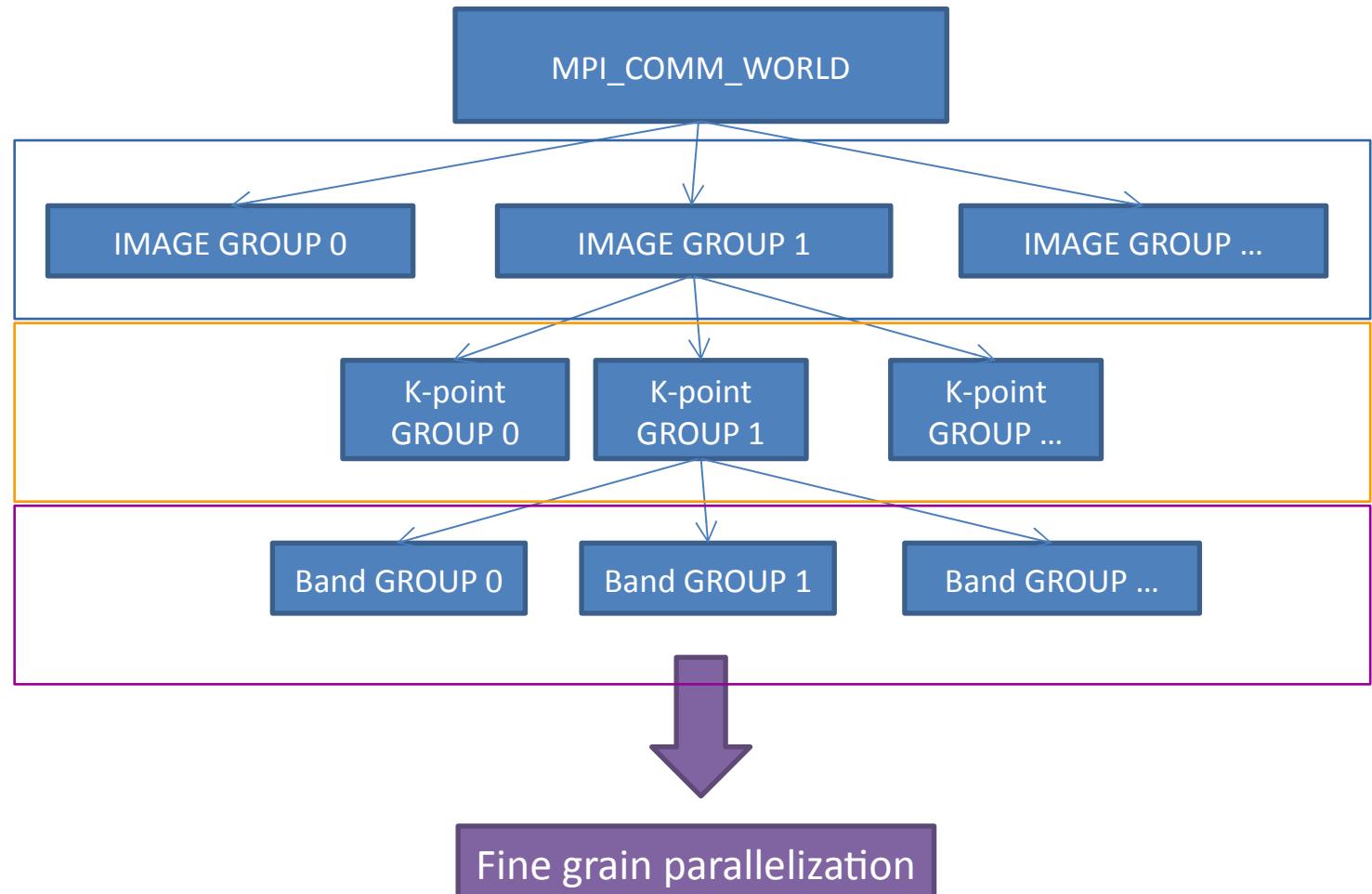


QE parallelization layout

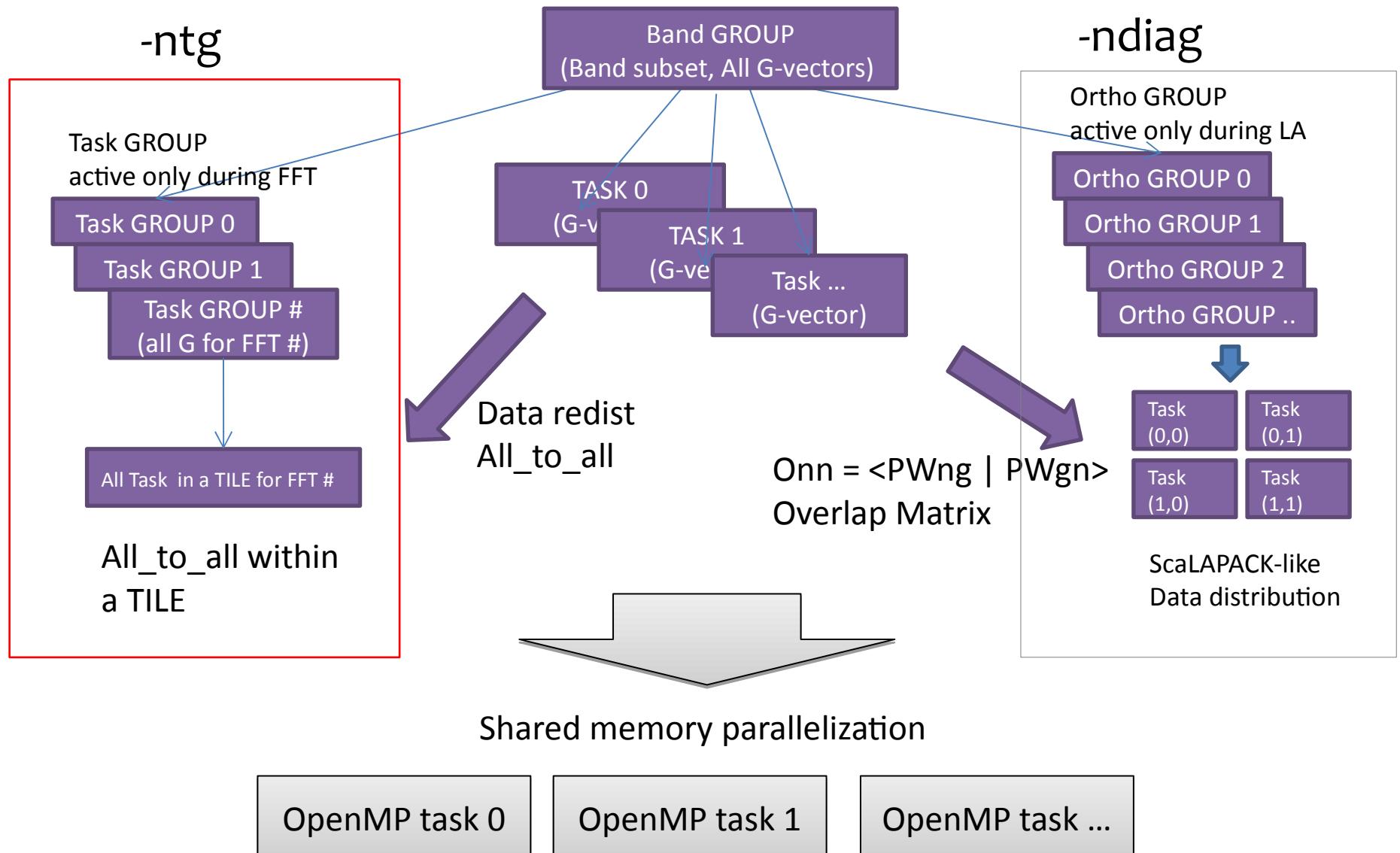
-nimage

-npool

what remains

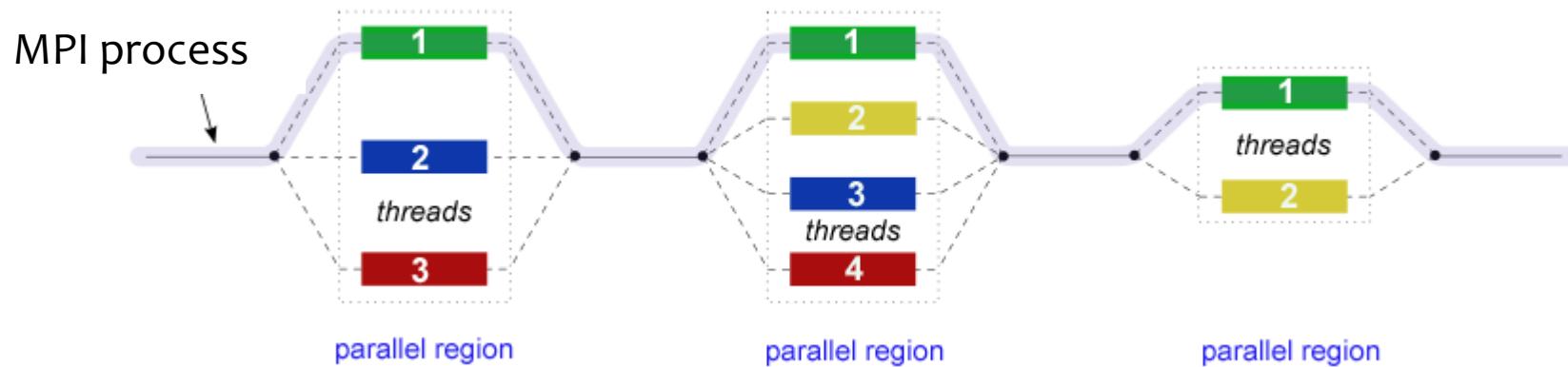


QE parallelization layout



What is OpenMP

Multi-platform shared memory multiprocessing programming.



Used in:

- * BLAS and LAPACK libraries (example, MKL)
- * FFT library
- * Explicit in QE, still mainly fork & join paradigm

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: $n_{\text{image}} = \text{max possible value}$

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



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
- Must be smaller than band-group size

Tentative optimal value:

- Use it 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 $ntg \times nFFT = nBGRP$; 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: use when $nBGRP \sim nr^3$ or larger, depends on the HPC system.

Parallelization, in practice

- OpenMP
 - In the code and **in the libraries.**
 - Only when MPI is saturated.
 - Generally no more than 8 threads.
 - Don't forget it!
- 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 (already done for you today)
- Load/use machine specific math libs. (eg. Intel® MKL)
- Configure QE 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=
```

Advanced compilation options

- Check make.inc
- Add optimization flags, eg.
 - KNL → AVX512MIC
 - SKL → AVX512CORE
- Possibly add profiling libraries
 - Gperftools → memory profiling
 - GPTL → accurate timing

Check flags

How make.inc should look like on an Intel based platform...

```
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

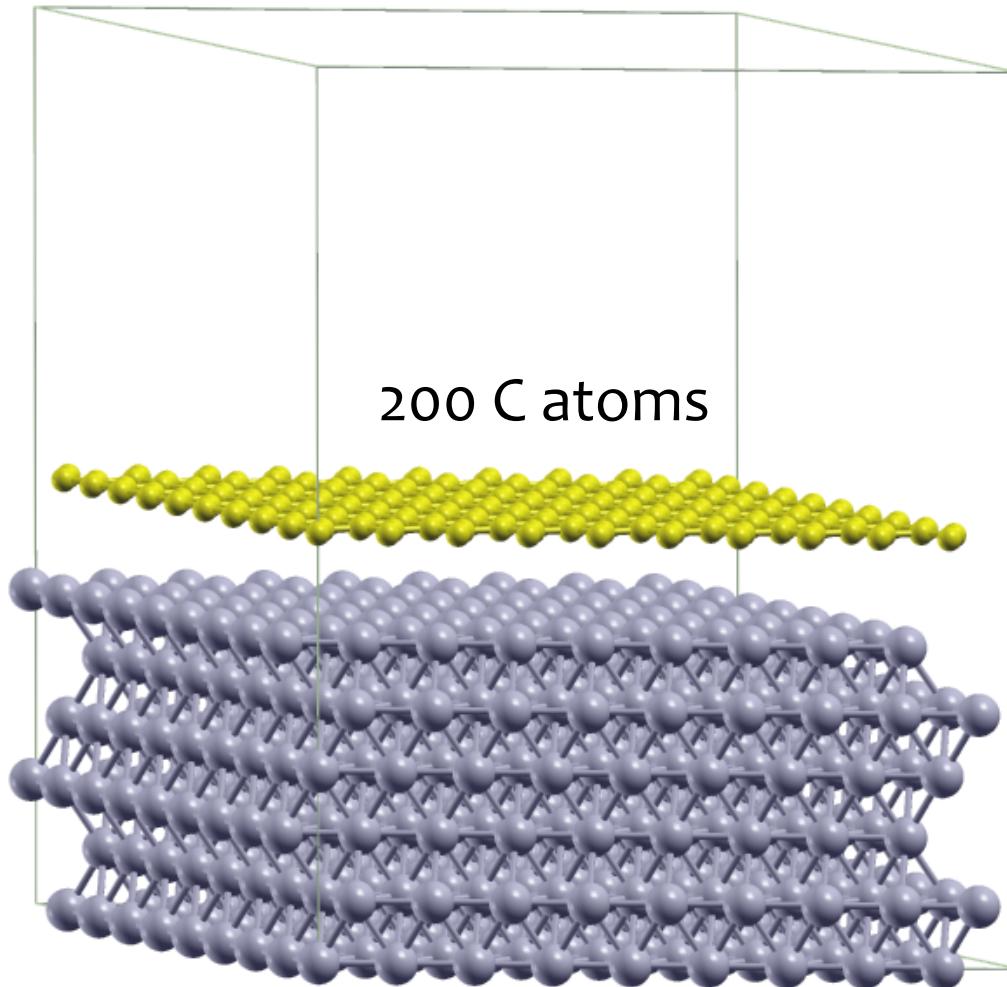
Grafene on Iridium

Standard QE benchmark

Available online at:

<https://gitlab.com/max-centre/benchmarks>

486 Ir atoms



Input

```
&control
  calculation = 'scf'
  prefix='GRIR'
  restart_mode='from_scratch'
  pseudo_dir='./',
  outdir='/path/to/scratch'
/
&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	qgrad:	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

MAX DRIVING
THE EXASCALE
TRANSITION

```
$ salloc --nodes=12 --ntasks-per-node=36
```

Happens more often than not

```
$ salloc --nodes=12 --ntasks-per-node=36
```

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

```
$ salloc --nodes=4 --ntasks-per-node=36 --ncpus-per-task=1 --  
mem=100G -t 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  
  
mpirun pw.x -npool X -ndiag Y -ntg Z -inp pw.in > pw.out
```

Running on CINECA's BDW...



```
#!/bin/bash
#SBATCH -N 4
#SBATCH --tasks-per-node=36
#SBATCH --partition=bdw_usr_prod
#SBATCH --time=23:55:00

$ 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 CINECA's BDW...

```
&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
/
```

```
  bandstructure=band
  smearing=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 CINECA's BDW...

```
&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'
  diag_david_ndim='2'
  electron_maxstep = 1
/
```

```
d
granularity=fine,1
-ntg 1 -inp pw.in > pw.out
```

RAM per process > **2.97 GB**

Running on CINECA's BDW...

```
#!/bin/bash
#SBATCH -N 8
#SBATCH --tasks-per-node=36
#SBATCH -mem=115G

$ 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 CINECA's BDW...

```
#!/bin/bash
#SBATCH -N 8
#SBATCH --tasks-per-node=36
#SBATCH -mem=115G

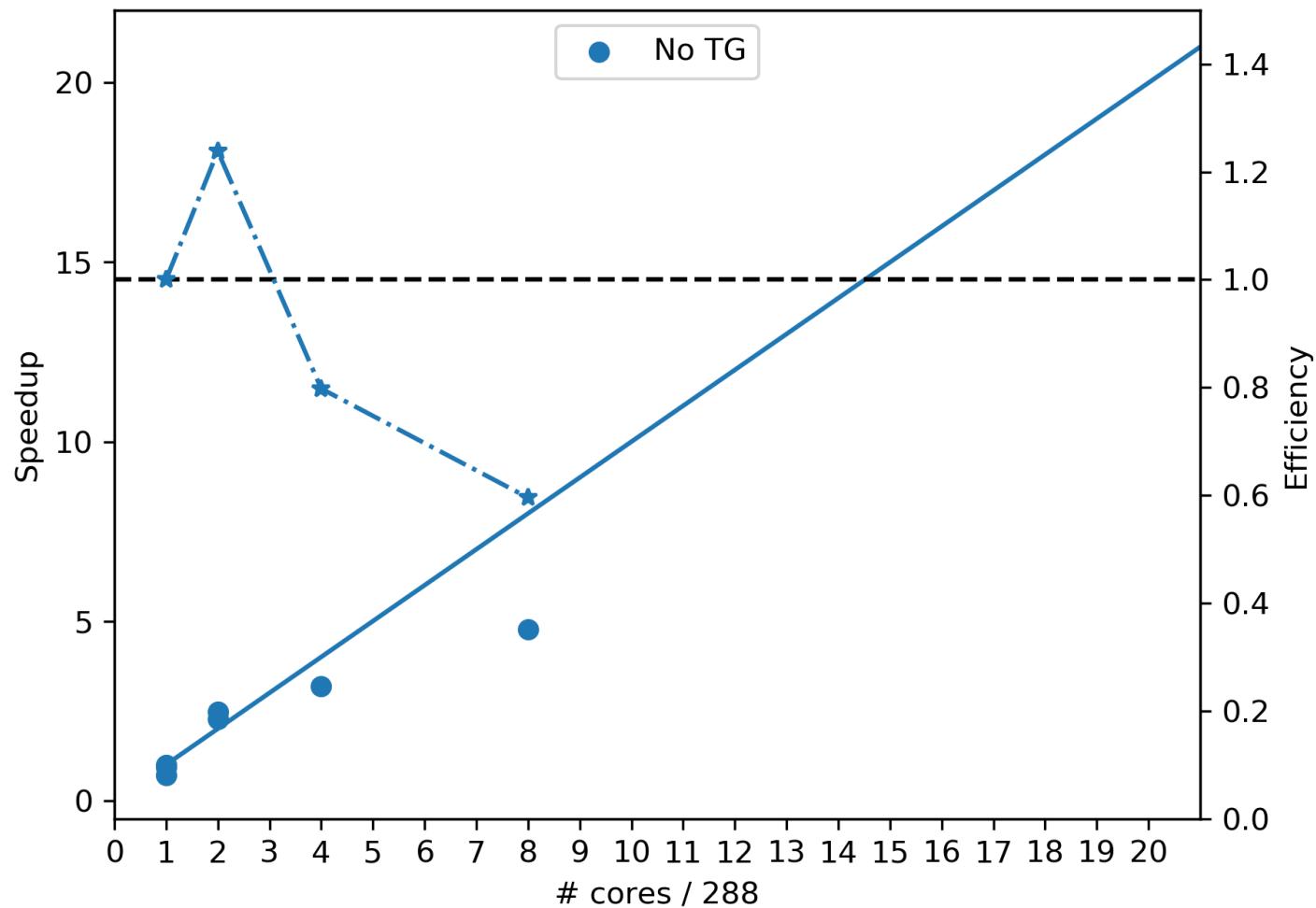
$ export OMP_NUM_THREADS=1
$ export MKL_NUM_THREADS=1
$ export KMP_AFFINITY=scatter,granularity=fine,1

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

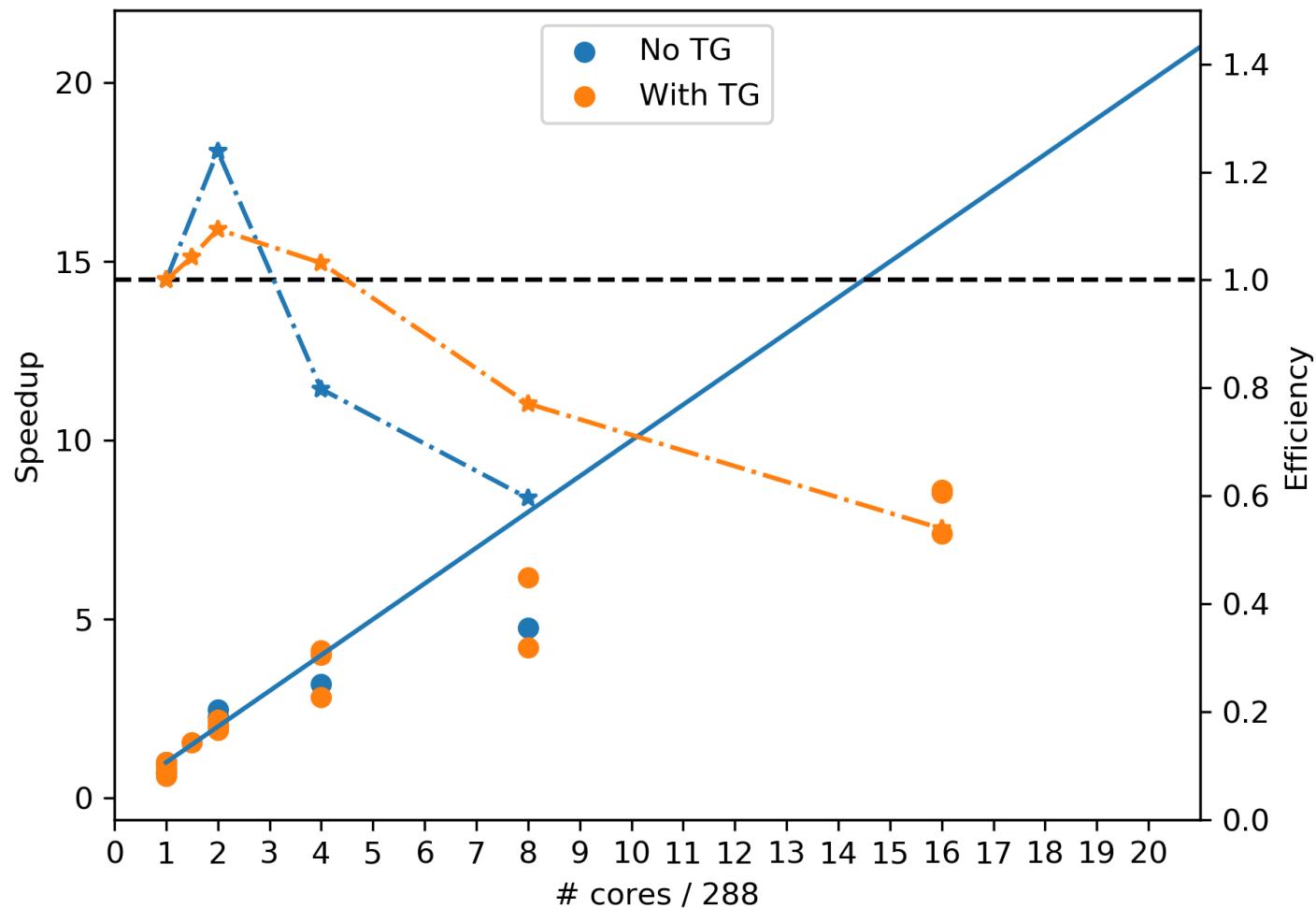
Running on CINECA's BDW...

MAX DRIVING
THE EXASCALE
TRANSITION



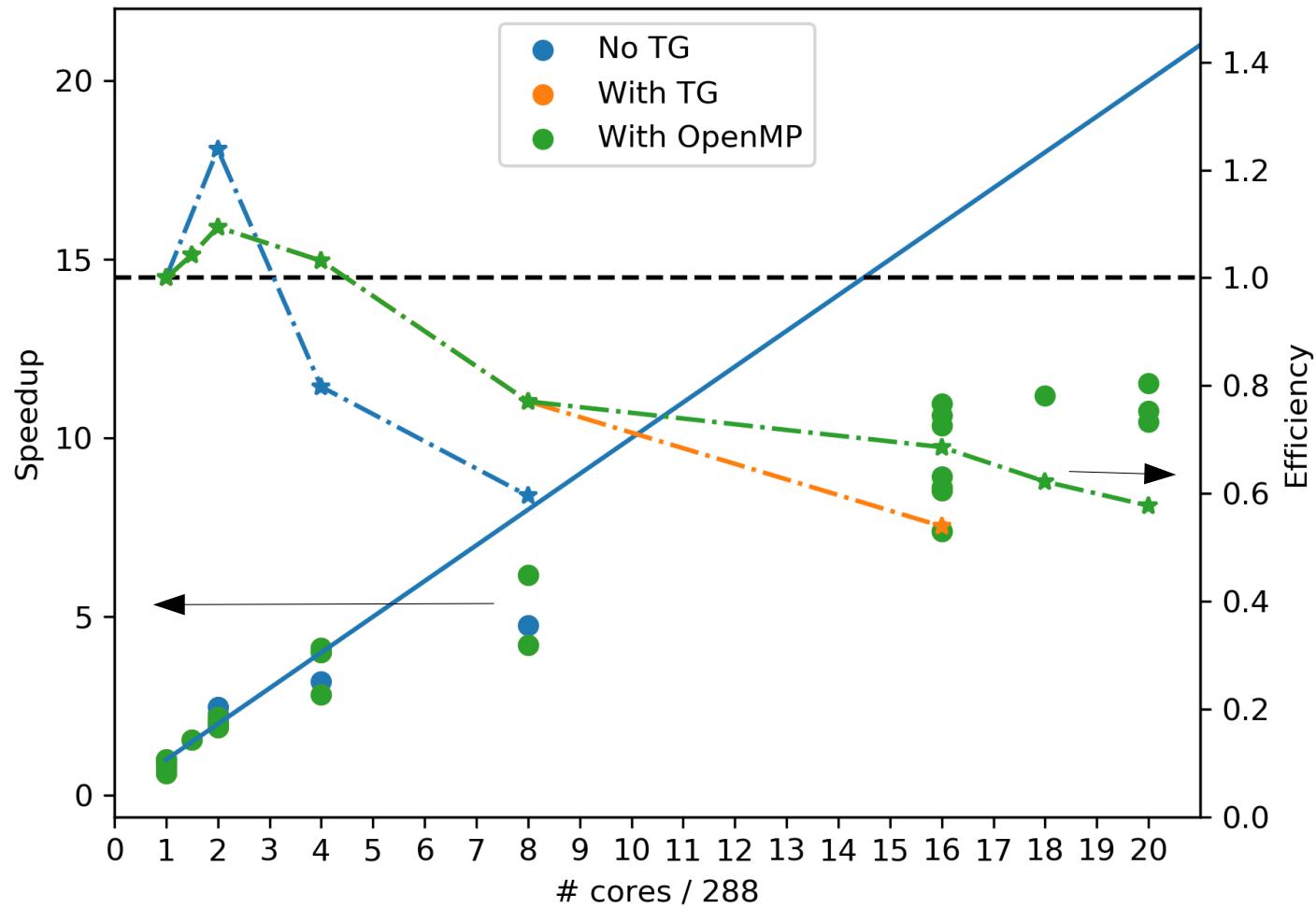
Running on CINECA's BDW...

MAX DRIVING
THE EXASCALE
TRANSITION



Running on CINECA's BDW...

MAX DRIVING
THE EXASCALE
TRANSITION



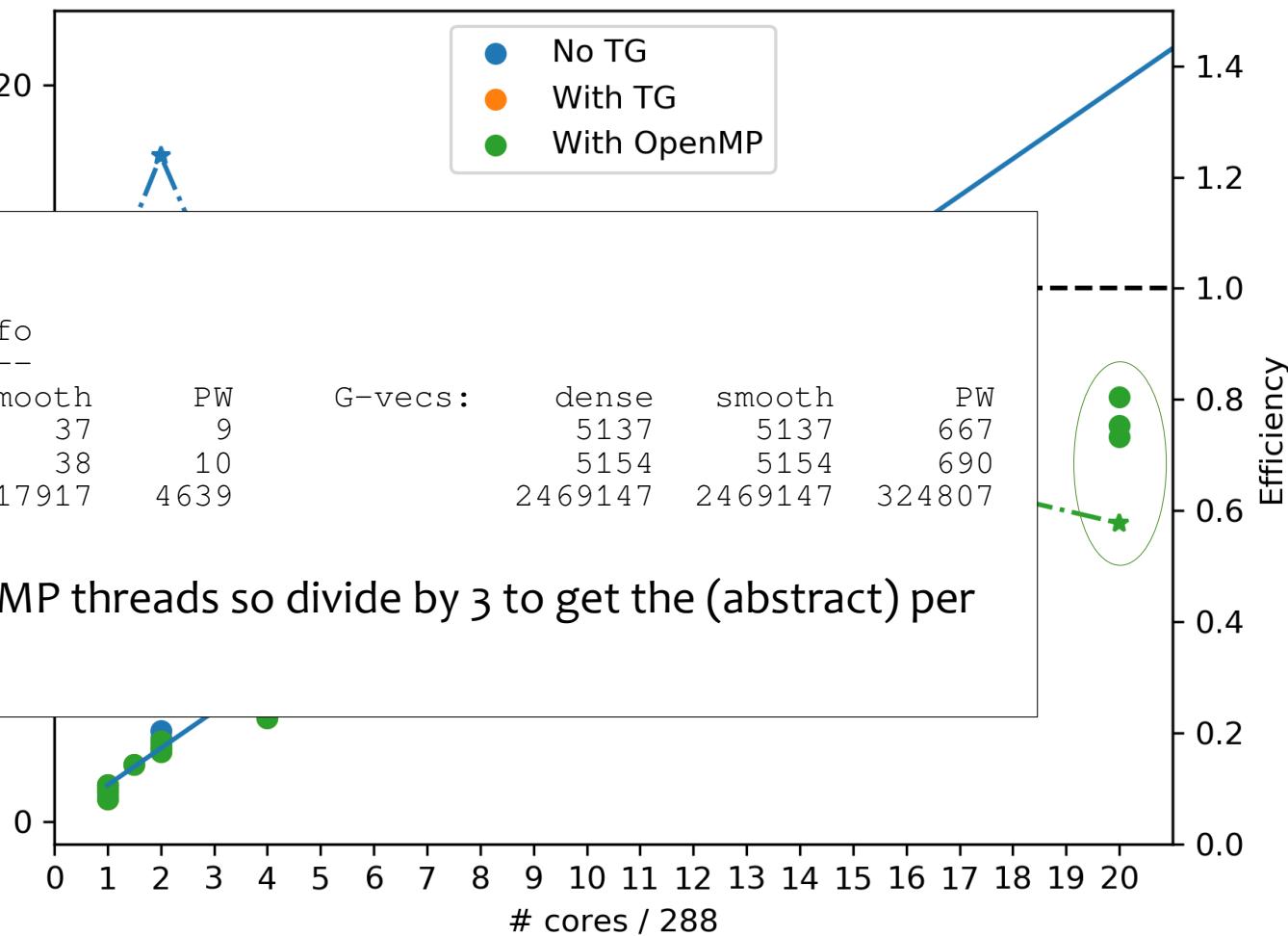
Running on CINECA's BDW...

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

I was using 3 OpenMP threads so divide by 3 to get the (abstract) per core problem size!



During the hands-on...

- You will be working in 16 teams.
- You'll have access to 16 nodes equipped with:
 - 2 x 8 cores Intel Haswell @ 2.60 GHz
 - 60 GB memory
 - 2 NVIDIA Tesla K40 (discussed later)
- You will be guided towards the setup of an optimized version of QE.
 - You are welcome to *try it at home*, we will help (with lower priority though)
- You will experiment with parallel options, but you will not see amazing speedups.
- Exercises are not just copy/paste, you will have to sort out some problems by yourself.

Slurm scheduler



Optimize resource allocation on clusters

Cheat sheet:

`sbatch job.sh` # Submit a job

`squeue -u $USER` # Check status

`scancel jobid` # Cancel a job

Slurm jobfile

```
#!/bin/bash

#SBATCH --nodes=1
#SBATCH --ntasks=X           # Number of MPI tasks
#SBATCH --ntasks-per-node=X   # Number of MPI tasks per node
#SBATCH --cpus-per-task=Y     # Number of CPUs/per MPI process
#SBATCH --ntasks-per-core=1    # Used for hyperthreading
#SBATCH --reservation=qe2019
#SBATCH --gres=gpu:2          # make 2 GPUs visible

module load mpi/openmpi-x86_64 MKL/mkl_2019.4.243

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

srun --mpi=pmix /path/to/executable -inp file.in > file.out
```

Input/output

- A parallel filesystem is basically a parallel application.
- Different performance have different cost:
 - Home directory
 - Long term storage
 - *Scratch space*

Always set the `outdir` folder in input to a fast scratch space.
(in our case `/data0/QE/YOURUSERNAME`)

Your turn...



Open

<https://gitlab.com/QEF/material-for-ljubljana-qe-summer-school/tree/master/Day-5>

Open a terminal in the VM...

GPU accelerated QE

MAX DRIVING
THE EXASCALE
TRANSITION



QUANTUM ESPRESSO

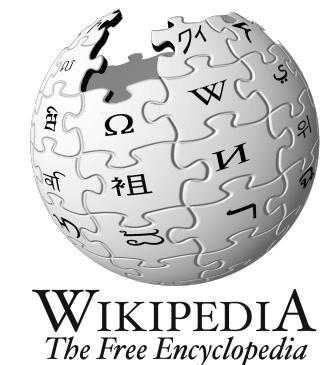


Collaboration and support from: J. Romero, M. Marić, M. Fatica, E. Phillips (NVIDIA)
F. Spiga (ARM), A. Chandran (FZJ), I. Girotto (ICTP), Y. Luo (ANL), T. Kurth (NERSC), B. Cook (NERSC),
P. Giannozzi (Univ. Udine), P. Delugas, S. De Gironcoli (SISSA).

Heterogenous Computing

MAX DRIVING
THE EXASCALE
TRANSITION

Gain performance or energy efficiency not just by adding the same type of processors, but by adding dissimilar coprocessors, usually incorporating specialized processing capabilities to handle particular tasks.



General Purpose Graphical Processing Units



The trick...

Difference between GPU and CPU hardware

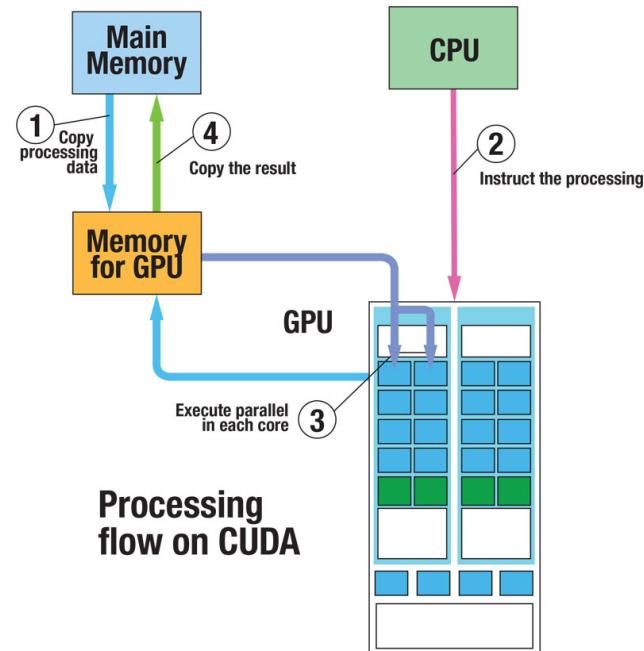


More transistors devoted to data processing
(but less optimized memory access and speculative execution)

GPU programming

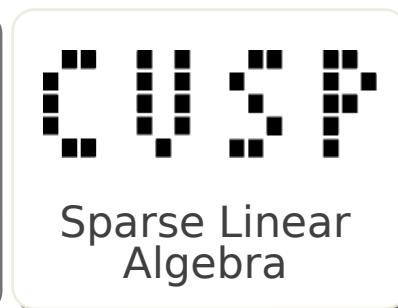
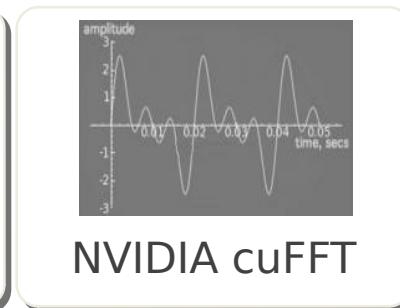
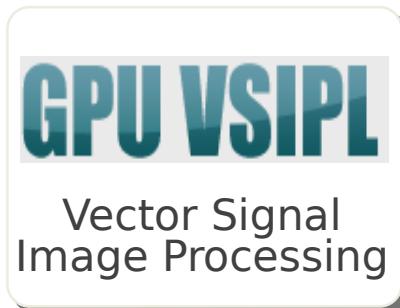
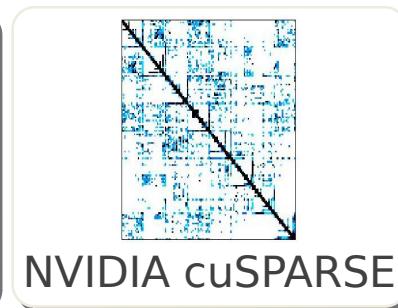
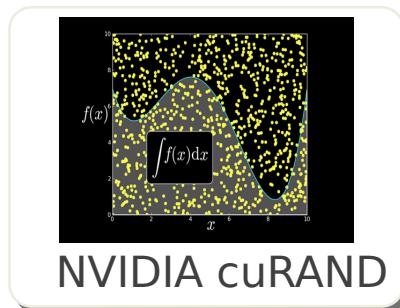
Typical code progression

- 1) Memory allocated on host and device
- 2) Data is transferred **from** the *Host to the Device*
- 3) Kernel is launched by the Host on the Device
- 4) Data is transferred **from** the *Device to the Host*.
- 5) Memory is deallocated.



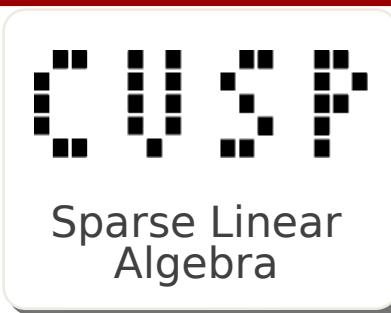
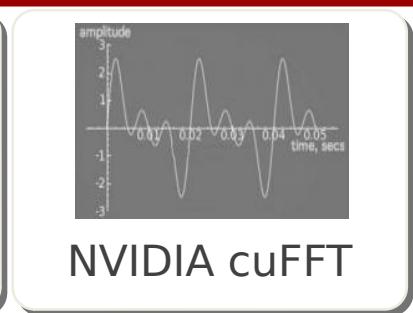
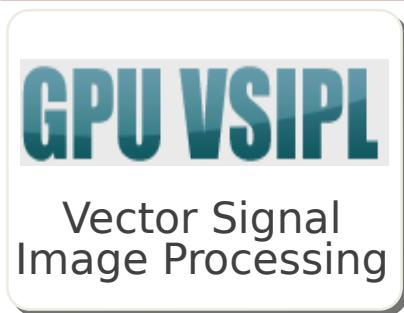
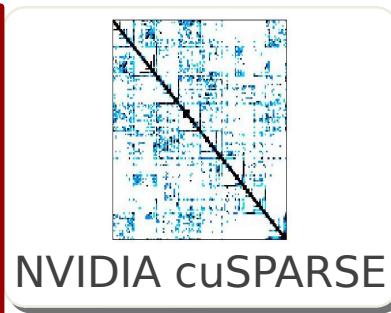
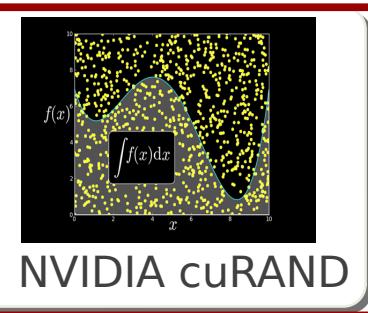
Libraries

MAX DRIVING
THE EXASCALE
TRANSITION



Libraries

MAX DRIVING
THE EXASCALE
TRANSITION



GPU Hardware details

MAX DRIVING
THE EXASCALE
TRANSITION

- NVIDIA Tesla K40

Released: 10/2013



One GPUs (GK110B) per device

12GB RAM per GPU

**15 Multiprocessors (MP),
192 CUDA Cores/MP = 2880 CUDA
Cores**

Running at 500-800 Mhz

PCI Express 3.0 Host Interface

GigaThread Engine



L2 Cache



Memory Controller

Memory Controller

Memory Controller

Memory Controller

Memory Controller

Memory Controller

K40 chipset

- **Core/DP Units:** SP and DP arithmetic logic units.
- **Special Functions Units (SFUs):** Execute transcendental instructions such as *sin*, *cosine*, *reciprocal*, and *square root*. Each SFU executes one instruction per thread, per clock.

Take home message:

GPUs have different SP/DP ratios, you may not get the speedup you expect.



Comparison of double precision performance for a single card and a single node:

K40 : ~ 1400 GFLOP/s

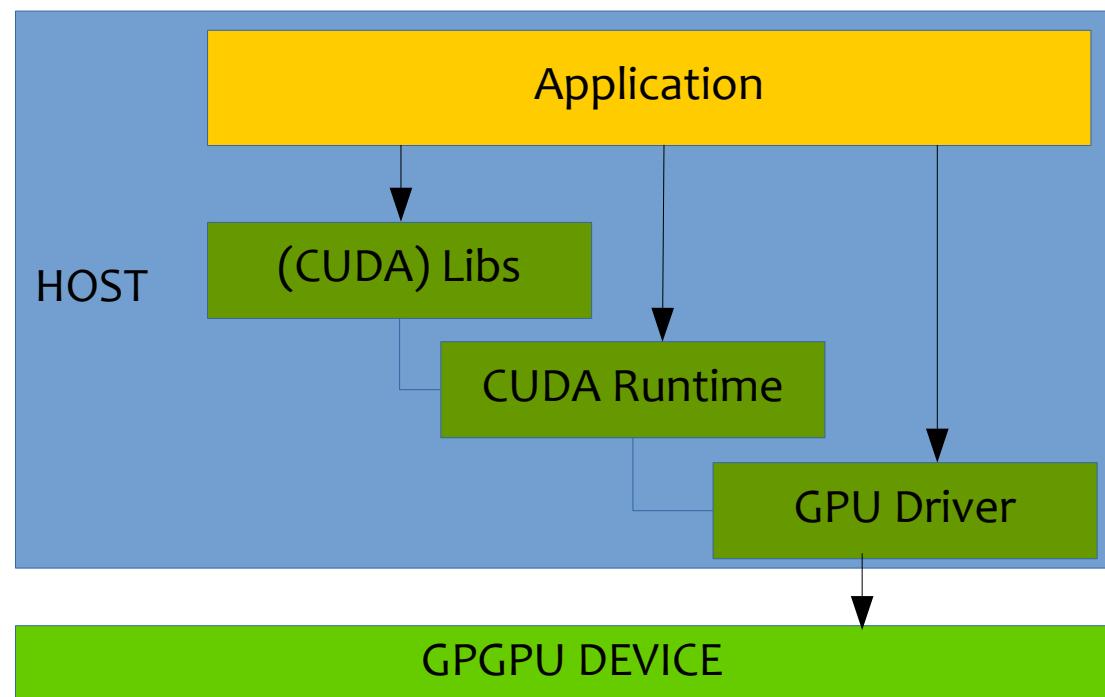
Intel Haswell: ~ 665 GFLOP/s



(<https://ark.intel.com/content/www/us/en/ark/products/83359/intel-xeon-processor-e5-2640-v3-20m-cache-2-60-ghz.html>)

CUDA Runtime

- In order to compile QE-GPU you'll need to know a bit about *CUDA* ...



CUDA Compute Capabilities

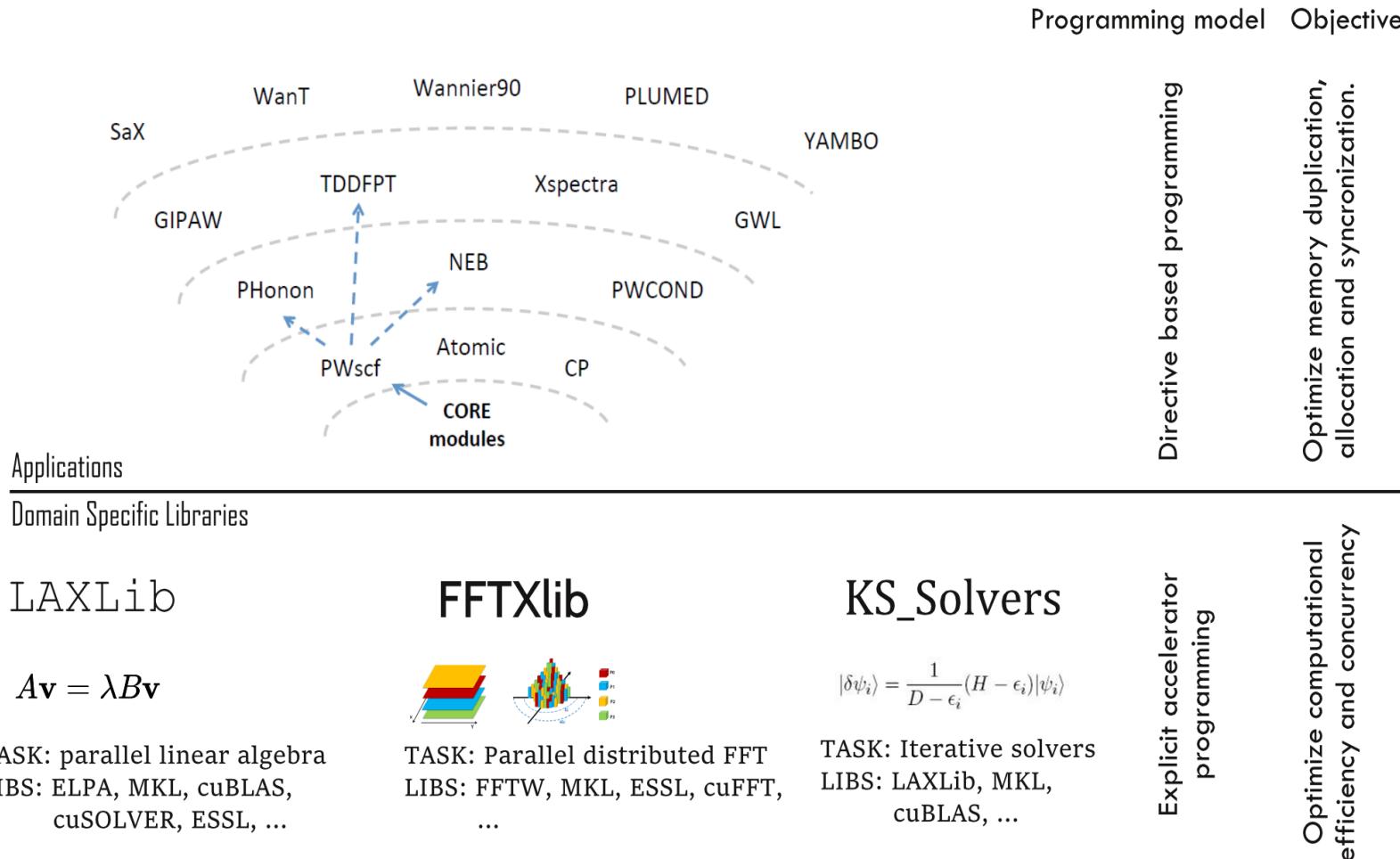
MAX DRIVING THE EXASCALE TRANSITION

The *compute capabilities* codify the features and specifications of the target device.

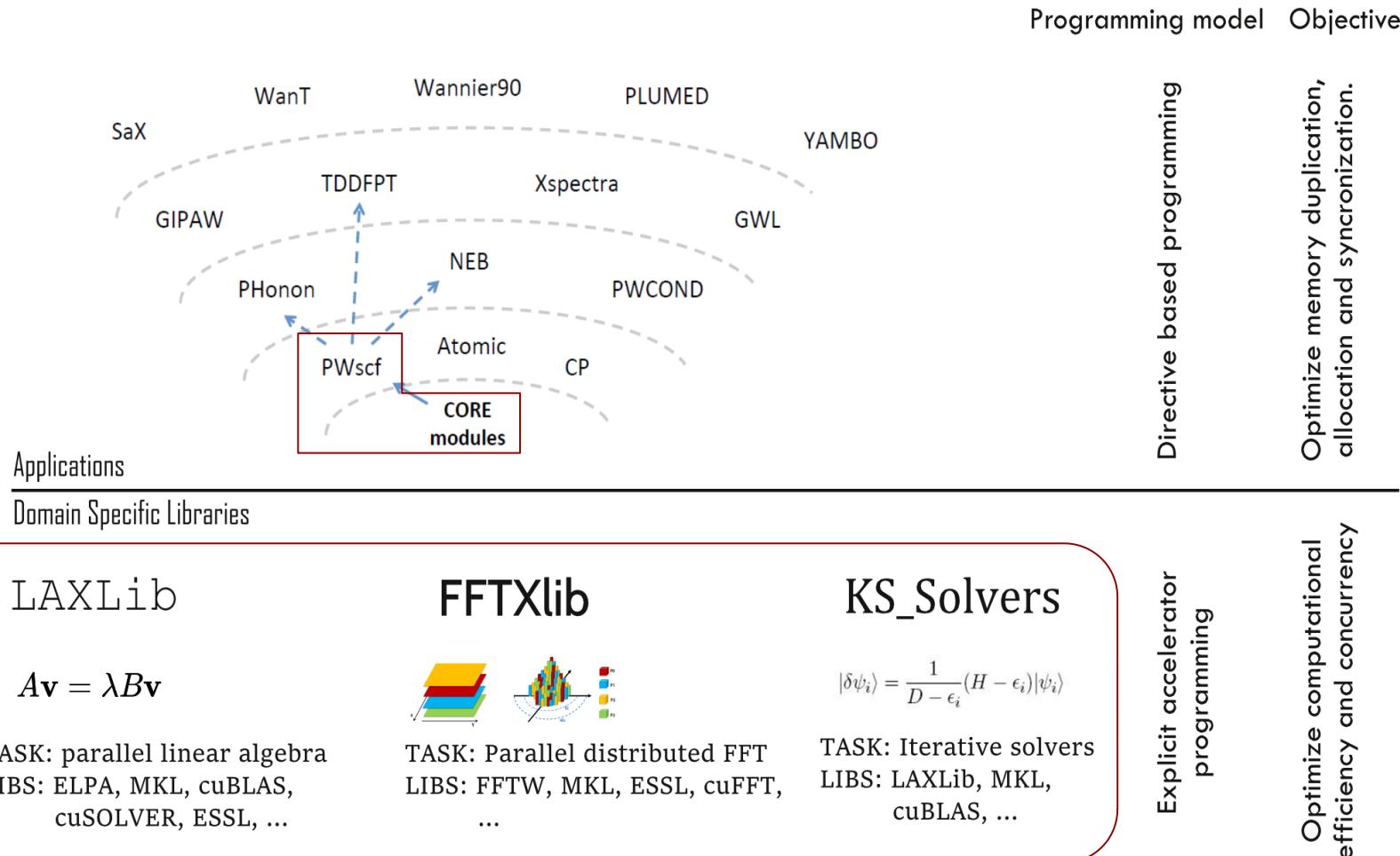
- Tesla K40: 3.5
- Tesla K80: 3.7
- Tesla P100: 6.0
- Tesla V100: 7.0

Feature Support	Compute Capability					
	3.0	3.2	3.5, 3.7, 5.0, 5.2	5.3	6.x	7.x
(Unlisted features are supported for all compute capabilities)						
Atomic addition operating on 32-bit floating point values in global and shared memory (atomicAdd())				Yes		
Atomic addition operating on 64-bit floating point values in global memory and shared memory (atomicAdd())				No		Yes
Warp vote and ballot functions (Warp Vote Functions)						
<code>_threadfence_system()</code> (Memory Fence Functions)						
<code>_syncthreads_count()</code> , <code>_syncthreads_and()</code> , <code>_syncthreads_or()</code> (Synchronization Functions)						
Surface functions (Surface Functions)						
3D grid of thread blocks						
Unified Memory Programming						
Funnel shift (see reference manual)	No			Yes		
Dynamic Parallelism	No			Yes		
Half-precision floating-point operations: addition, subtraction, multiplication, comparison, warp shuffle functions, conversion		No		Yes		
Tensor Core			No			Yes

GPU accelerated QE



GPU accelerated QE



Some details about pw.x

A number of different accelerated versions of pw.x have been released over the last years:

GPU version	Total Energy (K points)	Forces	Stress	Collinear Magnetism	Non-collinear magnetism	Gamma trick	US PP	PAW PP	DFT+U	All other functions
v5.4	A	W	W	B (?)	U	A	A	?	W (?)	W (?)
v6.1	A	A	A	A	U	W (*)	A	A (*)	U (?)	U (?)
v6.4	A	W	W	A	A	A	A	A (*)	W	W

Accelerated, **W**orking, **U**navailable, **B**roken

* Acceleration obtained from other parts of the code.

How fast?

- Strongly depends on your hardware, both Host and Device(s)
- On HPC systems ~3x speedup



9216 POWER9 22-core CPUs (2x node)
&
27,648 Nvidia Tesla V100 GPUs
(6xnode)

How fast?

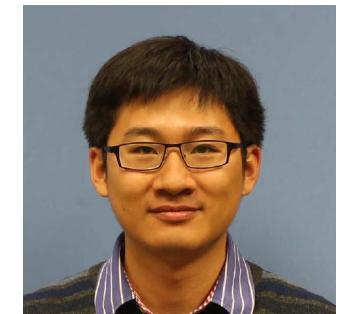
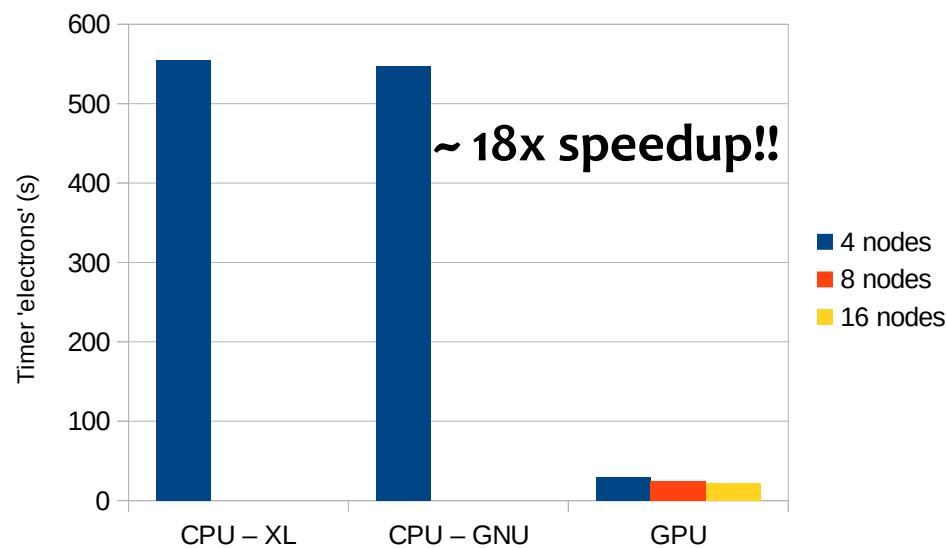
- Strongly depends on your hardware, both Host and Device(s)
- On HPC systems ~3x speedup

Top 10 positions of the 53nd TOP500 in June 2019^[21]

Rank	Rmax Rpeak (PFLOPS)	Name	Model	Processor	Interconnect	Vendor	Site country, year
1	148.600 200.795	Summit	IBM Power System AC922	POWER9, Tesla V100	InfiniBand EDR	IBM	Oak Ridge National Laboratory United States, 2018
2	94.640 125.712	Sierra	IBM Power System S922LC	POWER9, Tesla V100	InfiniBand EDR	IBM	Lawrence Livermore National Laboratory United States, 2018
3	93.015 125.436	Sunway TaihuLight	Sunway MPP	SW26010	Sunway ^[22]	NRCPC	National Supercomputing Center in Wuxi China, 2016 ^[22]

How fast?

- Strongly depends on your hardware, both Host and Device(s)
- On HPC systems ~3x speedup, but...



Courtesy of Dr. Ye Luo

	Components	
Processor	CPU	GPU
Type	POWER9	V100
Count	9,216 2 × 18 × 256	27,648 6 × 18 × 256
Peak FLOPS	9.96 PF	215.7 PF
Peak AI FLOPS		3.456 EF

Where to get the code



- Current development repository:

<https://gitlab.com/QEF/q-e-gpu>

- Last release: 6.4.1a1
- Compile requirements:
 - PGI compilers
<https://www.pgroup.com/products/community.htm>
 - CUDA Runtime and Libraries

How to compile the code

- Same as standard QE distribution, but three new options must be provided:

```
--with-cuda=/path/to/cuda/ or yes  
--with-cuda-cc=target-compute-capabilities  
--with-cuda-runtime=runtime-version
```

- OpenMP is **mandatory** (and no scalapack unless you know what you are doing)
- You can find detailed instructions on the wiki:

<https://gitlab.com/QEF/q-e-gpu/wikis>

How to run the code



The rules:

1 GPU \leftrightarrow 1 MPI

Fill the CPU with OpenMP threads

No task grouns (-ntg 1)

No parallel eigensolver (-ndiag 1)

K-point pools are great™, but device memory is limited.

How to run the code



You may get additional speedup by *breaking the rules*:

more GPUs \leftrightarrow 1 MPI

Don't fill the CPU with OpenMP threads

Thank you

MAX DRIVING THE EXASCALE TRANSITION

