

---

Efficient materials modelling on HPC  
with QUANTUM ESPRESSO, Yambo and BigDFT

---

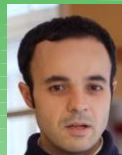
Hands-on session - Day 1

# QUANTUM ESPRESSO on hpc systems

Fabrizio Ferrari Ruffino, Pietro Delugas and  
Ivan Carnimeo (main author of the slides)



[ferrariruffino@iom.cnr.it](mailto:ferrariruffino@iom.cnr.it)



[pdelugas@sissa.it](mailto:pdelugas@sissa.it)



[icarnimeo@sissa.it](mailto:icarnimeo@sissa.it)

Covered topics are:

- \* optimisation of CPU-only runs,
- \* basic description of GPU acceleration,
- \* how to efficiently run calculations on GPU-accelerated architectures.

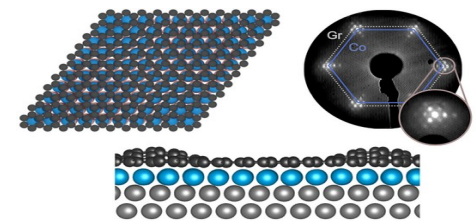
Exercise 1:\*\* optimize CPU execution with npools

Exercise 2:\*\* optimize CPU execution with ndiag

Exercise 3:\*\* openMP acceleration

Exercise 4:\*\* hands on the GPUs

Some preliminary notions  
PW SCF FLOW



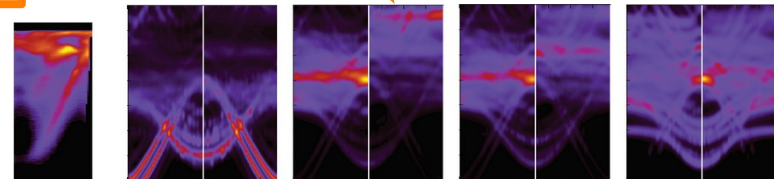
Input: molecular geometry

PWscf

**KS Solvers**  
Davidson    PPCG  
ParO    RMM-DIIS    CG

**Make KS potential**  
XClib    cuFFT  
cuBLAS

**Diagonalization**  
cuSOLVER



Output: chem/phys properties

this is achieved using a **modular organization** of the code which allows great flexibility and ease to use external and internal numerical libraries

# Day 1 hands on GETTING THE MATERIAL

Open a shell on your Virtual Machine or on your laptop and connect to the HPC cluster:

```
ssh USER@loggingpu.vega.izum.si
```

Then clone the school repo and move there:

```
git clone https://gitlab.com/QEF/materials-for-max-coe-enccs-workshop-2022.git  
cd materials-for-max-coe-enccs-workshop-2022  
cd Day-1  
pwd
```

Check the result of pwd:

```
/ceph/hpc/home/USER/materials-for-max-coe-enccs-workshop-2022/Day-1
```

# Some preliminary notions

## COMPILERS AND LIBRARIES

Three things to select when installing QE (available already on our cluster):

### 1) The compiler

gfortran

ifort

nvfortran  
(ex pgi)

flang  
(ARM)

### 2) The linear algebra libraries

(QE internal)

openblas

mkl  
(Intel, AMD CPU)

### 3) The FFT libraries

(QE internal)

fftw3

mkl  
(Intel, AMD CPU)

Exercise 0: first run with PW

# Exercise 0 - first run

## INPUT FILE

```
&control
  calculation='scf',
  tstress=.true.
/
&system
  lbrav=1,           ! Bravais lattice index
  CellDm(1)=10.0,   ! dimension of the cell
  nat=1,            ! 1 atom
  ntyp=1,           ! 1 type of atom
  nbnd=6,           ! 6 bands
  ecutwfc=25.0,     ! cutoff energy for w.f.
  Ecutrho=200.0,    ! " " density
  occupations='from_input',
/
&electrons
  mixing_beta=0.25,
  conv_thr=1.0e-8
/
ATOMIC_SPECIES
O 15.99994 o_pbe_v1.2.uspp.F.UPF ! pseudo file
ATOMIC_POSITIONS alat
O 0.000000000 0.000000000 0.000000000
K_POINTS {gamma}
OCCUPATIONS ! occupations of individual states
2.0 1.3333333333 1.3333333333 1.3333333333 0.0 0.0
```

# Some preliminary notions

## MODULE LOAD AND FIRST RUN

We only use **pw.x** for today hands-on. Go to exercise 0 (**ex0** folder).  
The build-in module of QE is loaded in the batch file (**ex0-run.slurm**):

```
module load QuantumESPRESSO/7.1-foss-2022a
```

Check that the module works by submitting a quick serial test.  
Fill the batch file for a serial run:

```
export OMP_NUM_THREADS=...number of threads...  
mpirun -np ....number of MPI.... pw.x -in atom-pbe.in > atom-pbe.out
```

Submit:

```
sbatch ./ex0-run.slurm
```

```
squeue -u ...YOUR USERNAME...
```



# Some preliminary notions

## FIRST RUN

Program PWSCF v.7.1 starts on 11Nov2022 at 12:36:46

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);

"P. Giannozzi et al., J. Chem. Phys. 152 154105 (2020);

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 1 processor cores

Number of MPI processes: 1

Threads/MPI process: 1

MPI processes distributed on 1 nodes

2443 MiB available memory on the printing compute node when the environment starts

# Some preliminary notions

## FIRST RUN

End of self-consistent calculation

k = 0.0000 0.0000 0.0000 ( 1052 PWs) bands (ev):

-23.3342 -8.2738 -8.2738 -8.2738 -0.5593 4.3552

highest occupied, lowest unoccupied level (ev): -8.2738 -0.5593

! total energy = -31.73128866 Ry  
estimated scf accuracy < 4.1E-09 Ry

The total energy is the sum of the following terms:

one-electron contribution = -32.14523474 Ry

hartree contribution = 17.37480263 Ry

xc contribution = -6.74658554 Ry

ewald contribution = -10.21427100 Ry

convergence has been achieved in 7 iterations

Computing stress (Cartesian axis) and pressure

negative rho (up, down): 3.988E-05 0.000E+00

total stress (Ry/bohr**3)			(kbar)	P=	-39.12
-0.00026593	0.00000000	0.00000000	-39.12	0.00	0.00
0.00000000	-0.00026593	0.00000000	0.00	-39.12	0.00
0.00000000	0.00000000	-0.00026593	0.00	0.00	-39.12

Writing all to output data dir ./pwscf.save/

# Some preliminary notions

## FIRST RUN

```
vloc_psi   :    0.20s CPU    0.21s WALL (   43 calls)
add_vuspsi :    0.00s CPU    0.00s WALL (   43 calls)
```

### General routines

```
calbec     :    0.00s CPU    0.00s WALL (   52 calls)
fft        :    0.22s CPU    0.34s WALL (   98 calls)
ffts       :    0.21s CPU    0.21s WALL (   16 calls)
fftw       :    0.19s CPU    0.20s WALL (  186 calls)
interpolate :    0.21s CPU    0.21s WALL (    8 calls)
```

### Parallel routines

```
PWSCF     :    2.13s CPU    4.16s WALL
```

This run was terminated on: 12:36:50 11Nov2022

```
=====
JOB DONE.
=====
```

## Exercise 1: parallelization with pools

## PARALLELISM WITH POOLS

Kohn-Sham equations

$$\hat{H}^{KS} \psi_{ik}(\mathbf{r}) = \varepsilon_{ik} \psi_{ik}(\mathbf{r})$$

$$\psi_{ik}(\mathbf{r}) = \sum_{\mathbf{G}}^{N_{PW}} C_{\mathbf{G},ik} \frac{e^{i((\mathbf{G}+\mathbf{k})\cdot\mathbf{r})}}{\sqrt{\Omega}}, \quad \frac{\hbar^2}{2m} |\mathbf{k} + \mathbf{G}|^2 \leq E_{cut}$$

The code computes an unsymmetrized charge density

$$\tilde{n}(\mathbf{r}) = \sum_{\mathbf{k} \in IBZ} \sum_v w_{\mathbf{k}} |\psi_{\mathbf{k},v}(\mathbf{r})|^2$$

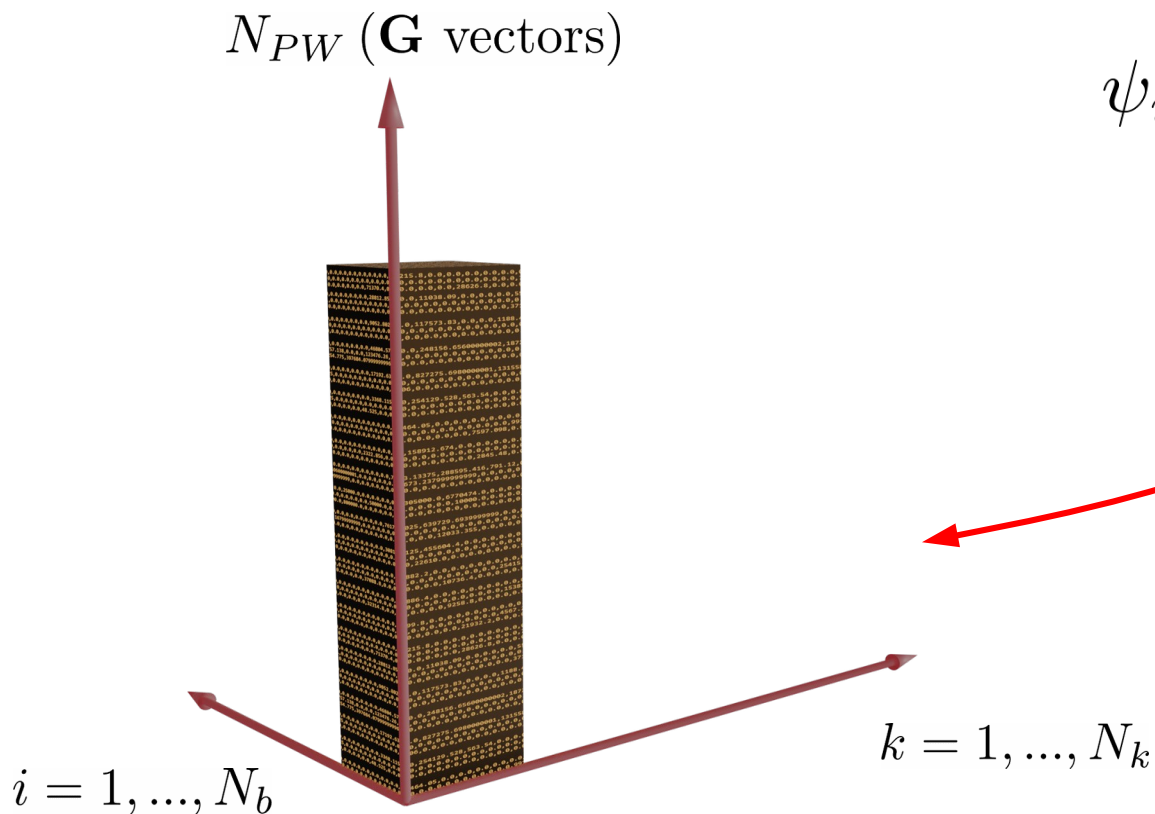
## PARALLELISM WITH POOLS

## Pool parallelism

What is happening?

Remember the distribution of the wavefunction

$$\psi_{ik}(\mathbf{r}) = \sum_{\mathbf{G}}^{N_{PW}} C_{\mathbf{G},ik} \frac{e^{i((\mathbf{G}+\mathbf{k})\cdot\mathbf{r})}}{\sqrt{\Omega}}$$



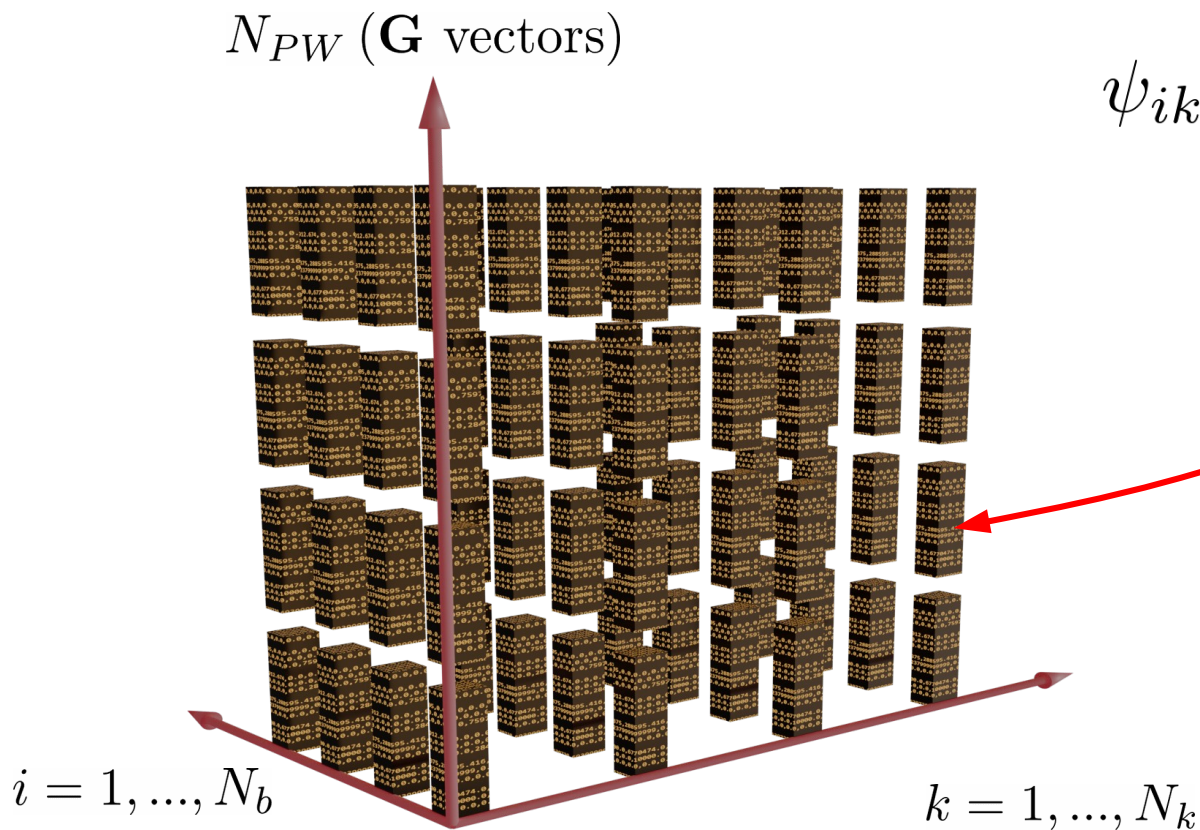
## PARALLELISM WITH POOLS

## Pool parallelism

What is happening?

Remember the distribution of the wavefunction

$$\psi_{ik}(\mathbf{r}) = \sum_{\mathbf{G}}^{N_{PW}} C_{\mathbf{G},ik} \frac{e^{i((\mathbf{G}+\mathbf{k})\cdot\mathbf{r})}}{\sqrt{\Omega}}$$

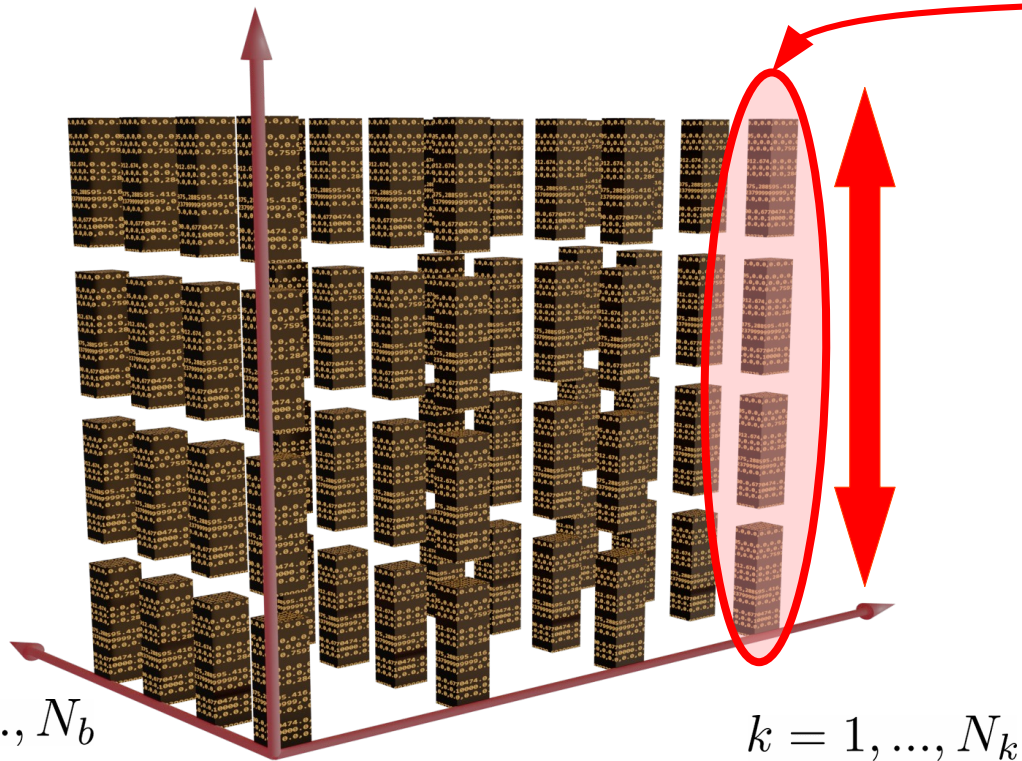


## PARALLELISM WITH POOLS

## Pool parallelism

What is happening?

$$\hat{H}^K \delta \psi_{ik}(\mathbf{r}) = \varepsilon_{ik} \psi_{ik}(\mathbf{r})$$

 $N_{PW}$  (**G** vectors)


Since operators are usually applied to single orbitals, most of the communications (e.g. mp\_sum) are usually done along the NPW dimension

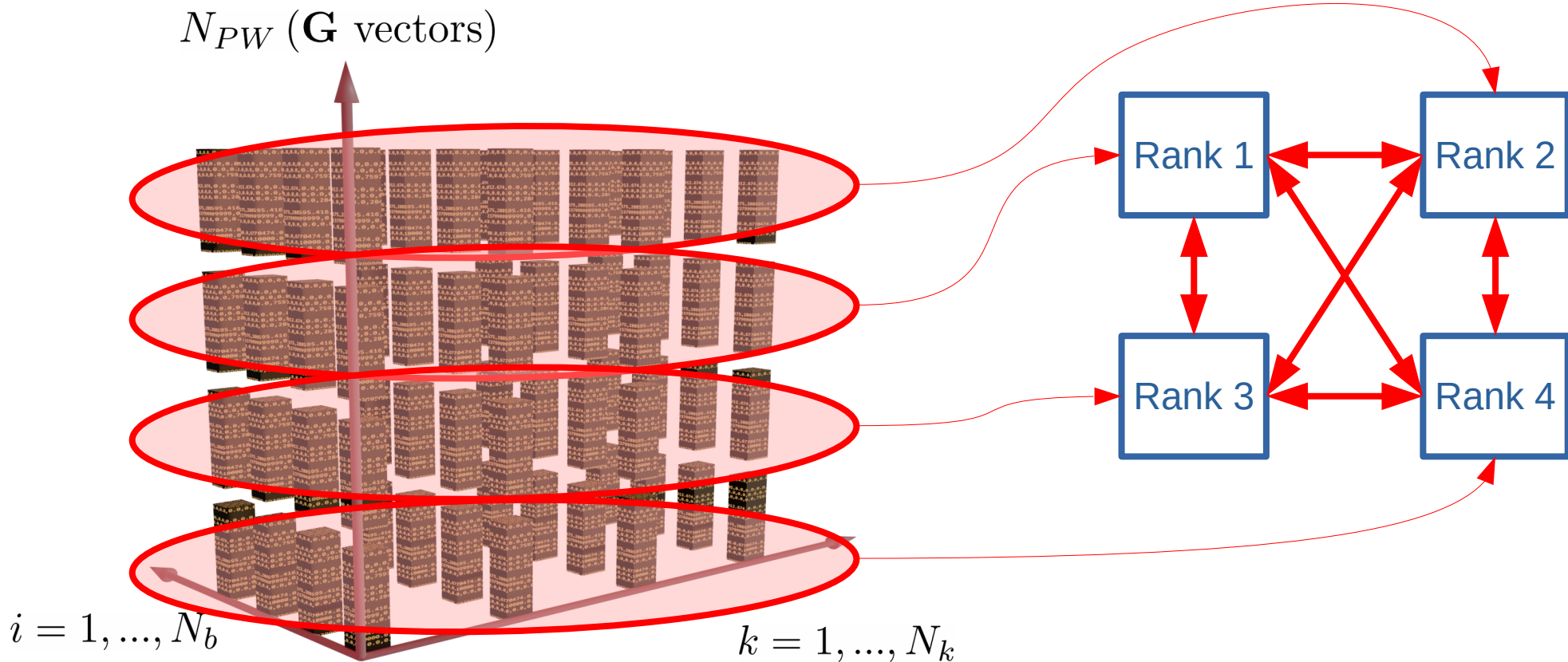


## PARALLELISM WITH POOLS

## Pool parallelism

What is happening?

When we parallelize over PW, all processes need to communicate with each other

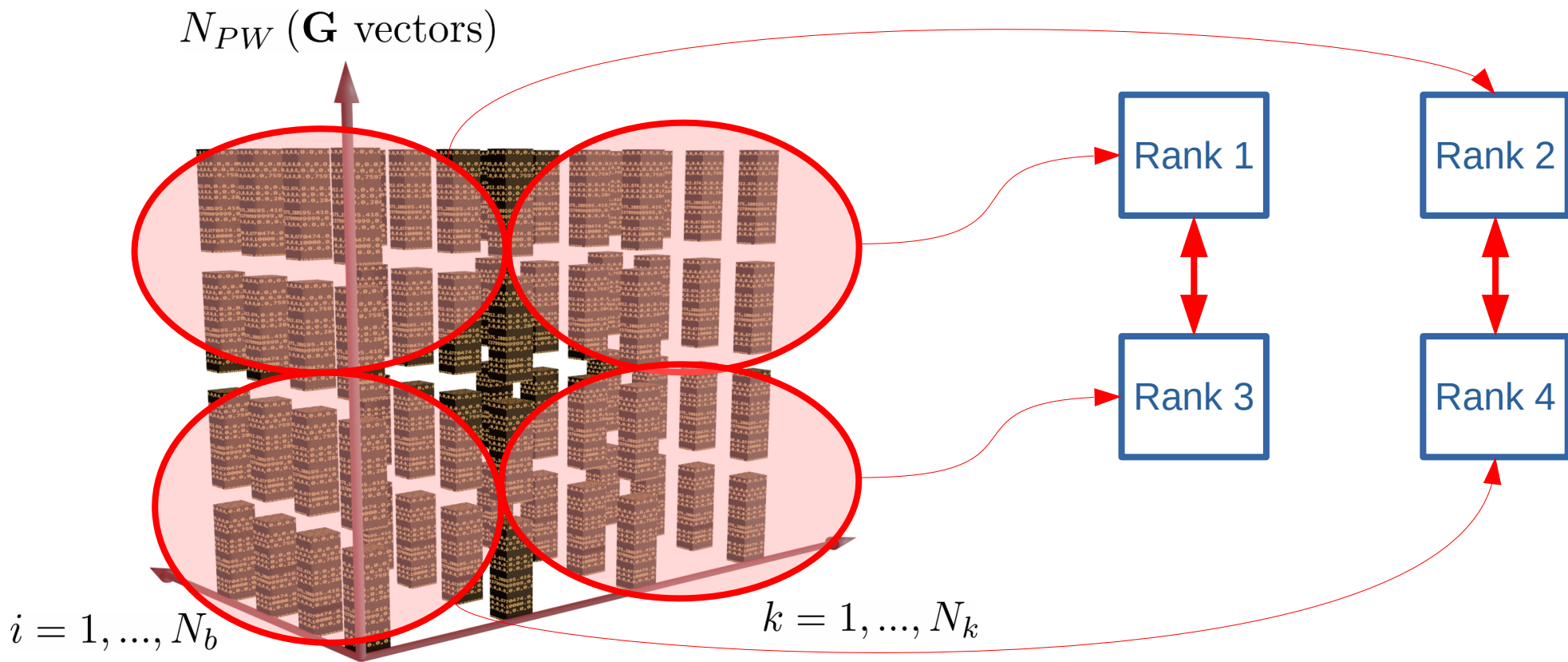


## PARALLELISM WITH POOLS

## Pool parallelism

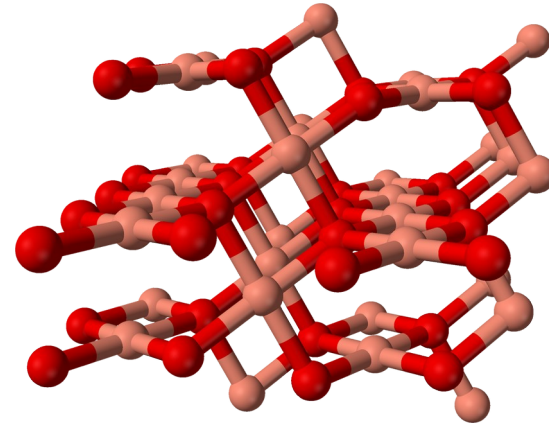
What is happening?

When we parallelize with pools, we strongly reduce communications among processes



## INPUT FILE - CuO

```
&control
  calculation='scf',
  outdir='./out'
/
&system
 ibrav = 0, nat=64, ntyp=2,
ecutwfc = 35,
ecutrho = 350,
smearing='mp',
occupations='smearing',
degauss=0.01,
nspin=2,
starting_magnetization(1)=0.0,
starting_magnetization(2)=0.5,
/
&electrons
  mixing_beta = 0.5,
  conv_thr = 1.0d-7,
  startingpot='atomic',
  startingwfc='atomic',
  electron_maxstep=4
/
&cell
  cell_dynamics = bfgs,
  press = 0
/
ATOMIC_SPECIES
O 1.0 o_pbe_v1.2.uspp.F.UPF
Cu 1.0 cu_pbe_v1.2.uspp.F.UPF
```



# Exercise 1

## JOB SCRIPT

```
#!/bin/bash
#SBATCH --job-name=USERjob
#SBATCH --nodes 1
#SBATCH --exclusive
#SBATCH --time=00:20:00
#SBATCH --partition=cpu
#SBATCH --ntasks-per-socket=2
#SBATCH --ntasks-per-node=16
#SBATCH --hint=nomultithread
#SBATCH --cpus-per-task=1
#SBATCH --output=sysout.out
#SBATCH --error=syserr.err
# # SBATCH --mail-user=YOUR_EMAIL - if you want

module purge
module load QuantumESPRESSO/7.1-foss-2022a

export EXDIR=${PWD}/..
export INDIR=${EXDIR}/inputs
export ESPRESSO_PSEUDO=${EXDIR}/../pseudo
export OMP_NUM_THREADS=1
mpiopt="-mca pml ucx -mca btl ^uct,tcp,openib,vader --map-by socket:PE=1 --rank-by core --report-bindings "

for ip in .....
do
mpirun $mpiopt -np 16 pw.x -npool "$ip" -ndiag 1 -i ${INDIR}/pw.CuO.scf.in > pw_CuO_${ip}pools.out
```

# Exercise 1

## SUBMIT JOB

Try to predict which will be the best value for **npool** and verify by performing a number of runs.

1. Open **ex1-pools.slurm** and customize the user-related SLURM options like job-name and mail-user (not essential);  
Replace the dots in the 'for' loop at the end of the file with a list of proper values for npool, e.g:

```
for ip in 1 2 3 4 5 6    -not necessarily right values here!  
do
```

2. Submit the job file:

```
sbatch ./ex1-pools.slurm
```

3. Check the total WALL time at the end of the output file and do a plot in function of npool.  
Which does it seem to be the best npool value? Why?

# Exercise 1

## TIME PLOT

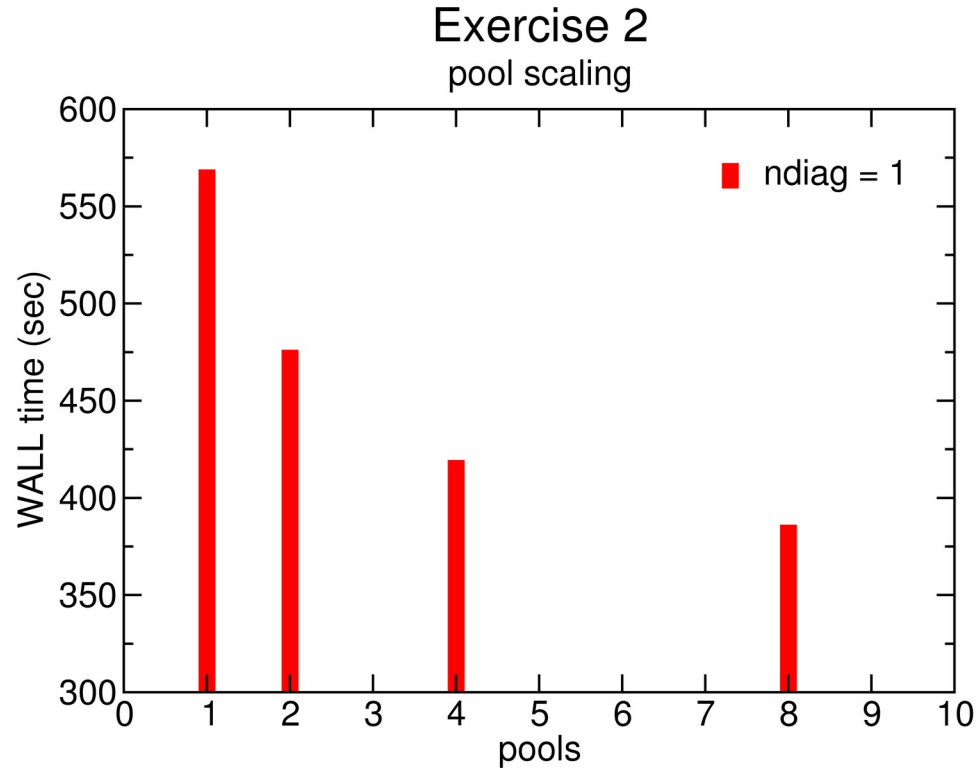
For each output file collect the “WALL time” at end of the file:

**PWSCF : 5m53.84s CPU 5m58.18s WALL**

NB: the CPU time is the amount of time spent by the CPU processing pw.x instructions, which is a considerable portion of the whole execution time, but neglects, for example, I/O. For this reason we use WALL time.

## Pool parallelism

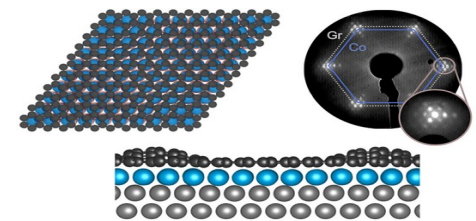
You should be able to produce a plot similar to this one (WALL time might differ):



## Exercise 2: parallel diagonalization



Some preliminary notions  
PW SCF FLOW



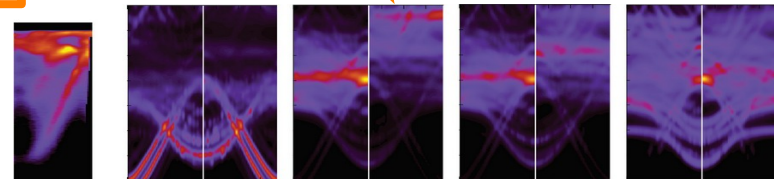
Input: molecular geometry

PWscf

**KS Solvers**  
Davidson    PPCG  
ParO    RMM-DIIS    CG

**Make KS potential**  
XClib    cuFFT  
cuBLAS

**Diagonalization**  
cuSOLVER

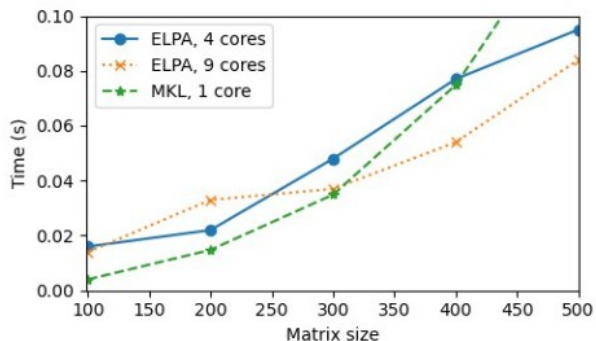
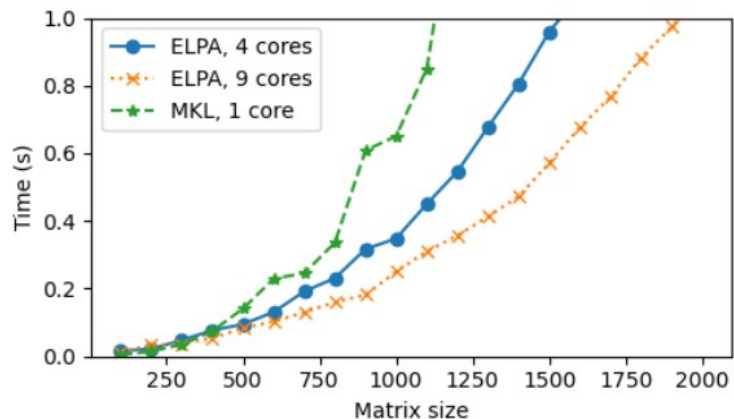


Output: chem/phys properties

this is achieved using a **modular organization** of the code which allows great flexibility and ease to use external and internal numerical libraries

## PARALLEL DIAGONALIZATION

## Trend of Parallel Diagonalization



- Results for all eigenstates.
- Additional communications in parallel Davidson.

## PARALLEL DIAGONALIZATION

In this second exercise we want to speedup the code by solving the dense *eigenvalue problem* using more than one core.

Set **-npool** to 4 and activate parallel diagonalization by changing **-ndiag** 4

```
mpirun $mpiopt -np 16 pw.x -npool 4 -ndiag "$id" -i ${INDIR}/pw.CuO.scf.in > pw_CuO_${id}diag.out
```

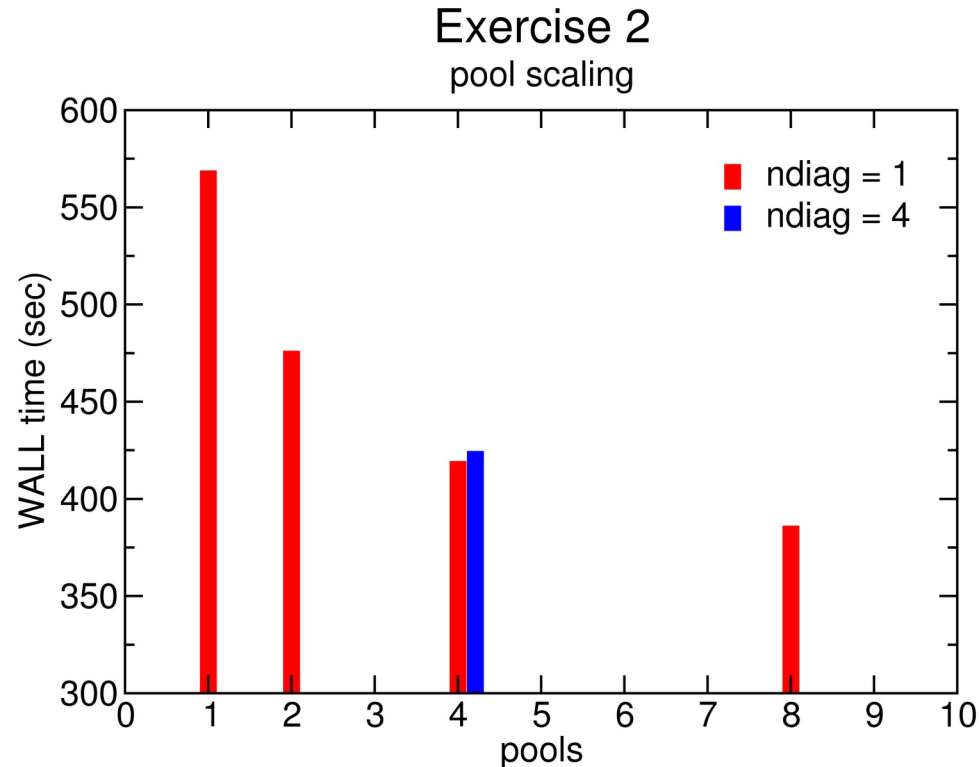
Inspect the beginning of the output file and look for this message

**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: 2\* 2  
procs)**

Check the time to solution.

## Parallel diagonalization

You should be able to produce a plot similar to this one:



# PARALLEL DIAGONALIZATION

Please consider that:

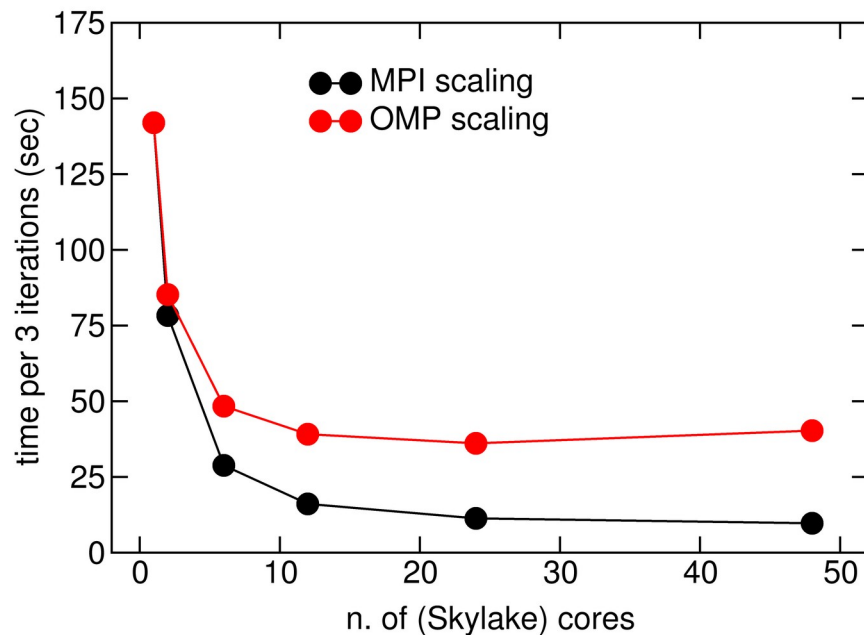
- 1) pool parallelism can be much more effective than this**, especially when the system size is larger and calculations are distributed among multiple nodes, since it can strongly reduce the slow inter-node communications;
- 2) the eigenvalue problem** is too small in this case to take fully advantage of parallel diagonalization;
- 3) other libraries**, e.g. Scalapack or ELPA, usually provide better performance in parallel diagonalization.

Please keep in mind that for larger systems, and using optimized libraries, the parallel diagonalization is a powerful option to strongly reduce the computational time to solution.

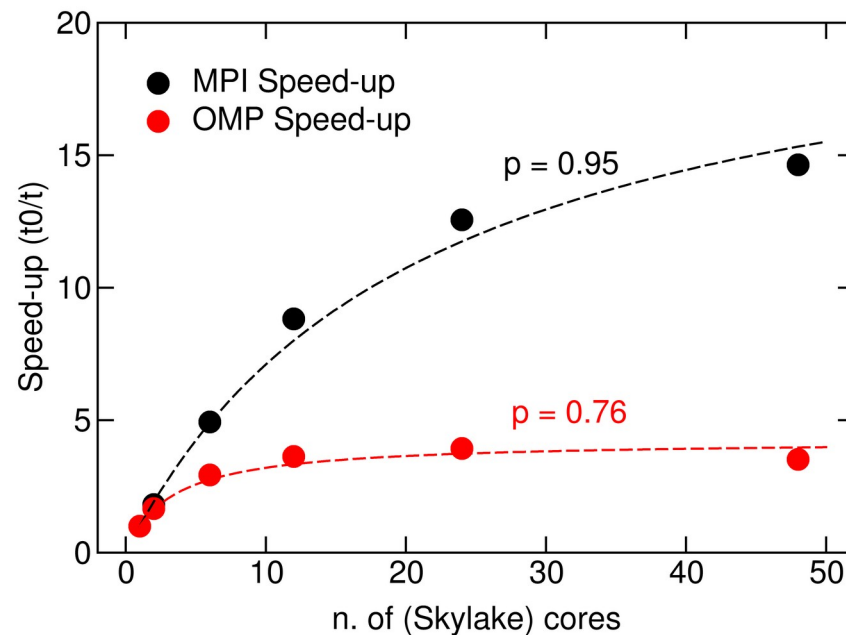
## Exercise 3: openMP parallelization

## COMBINING WITH OPENMP

32 water molecules



32 water molecules

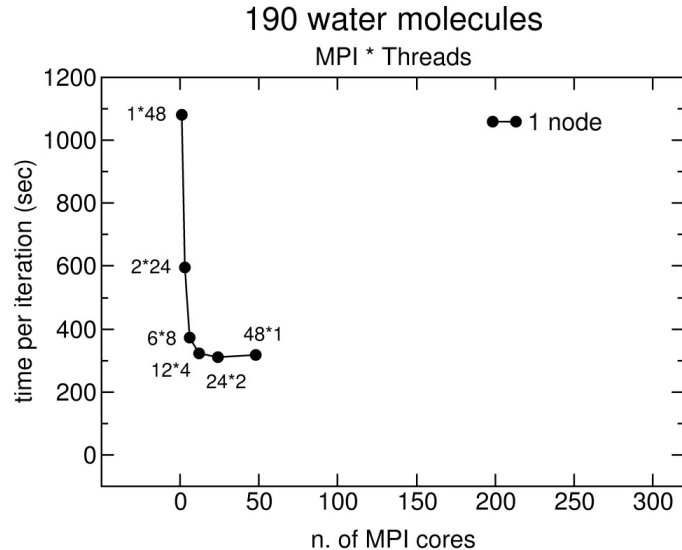


Scaling (Amdahl Law) for Quantum ESPRESSO code for both MPI and OpenMP portions of the code.

OMP parallelization is usually less efficient than MPI for QE, but involves less communications

# Exercise 3

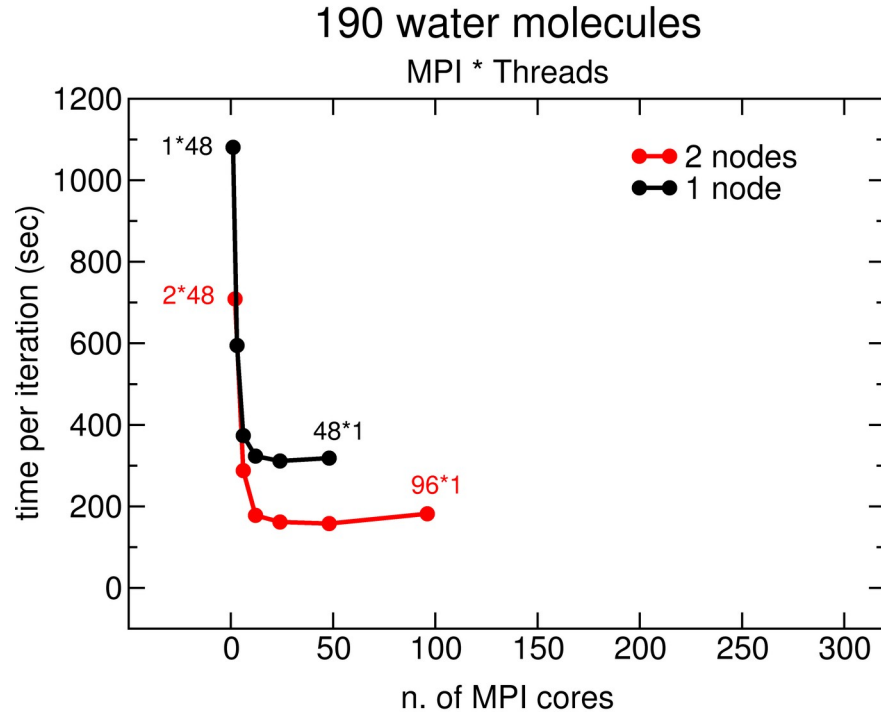
## COMBINING WITH OPENMP



MPI and OMP threads can be combined to better exploit computational resources. OMP parallelization is usually less efficient than MPI for QE, but involves less communications

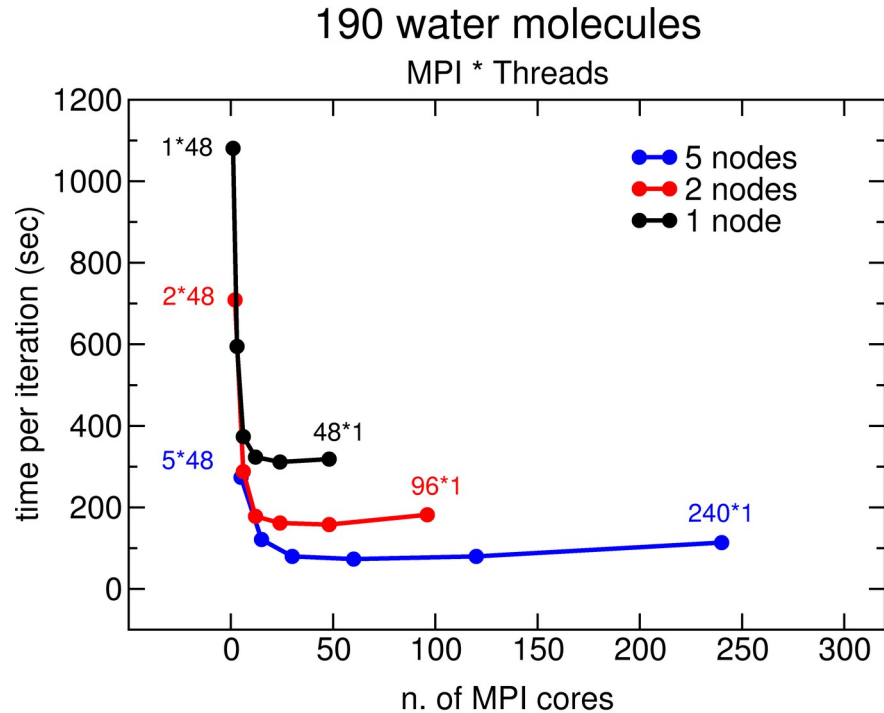


## COMBINING WITH OPENMP



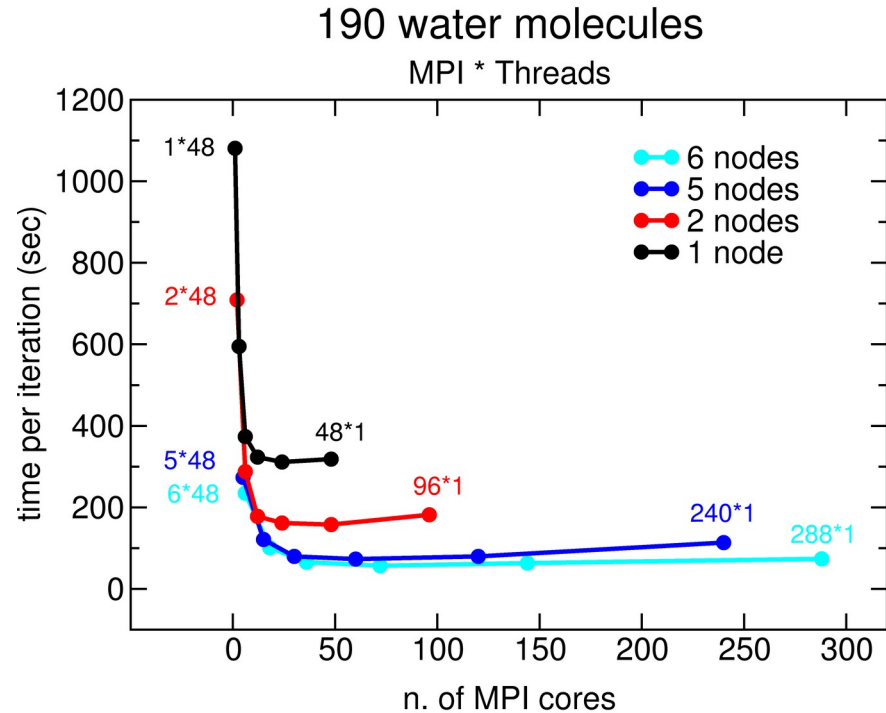
MPI and OMP threads can be combined to better exploit computational resources. OMP parallelization is usually less efficient than MPI for QE, but involves less communications

## COMBINING WITH OPENMP



MPI and OMP threads can be combined to better exploit computational resources. OMP parallelization is usually less efficient than MPI for QE, but involves less communications

## COMBINING WITH OPENMP

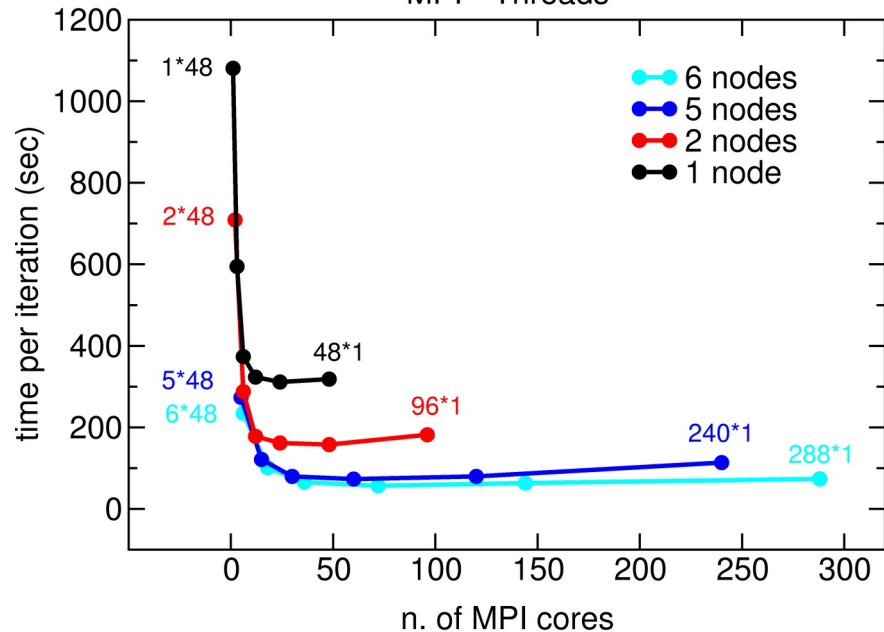


MPI and OMP threads can be combined to better exploit computational resources. OMP parallelization is usually less efficient than MPI for QE, but involves less communications

## COMBINING WITH OPENMP

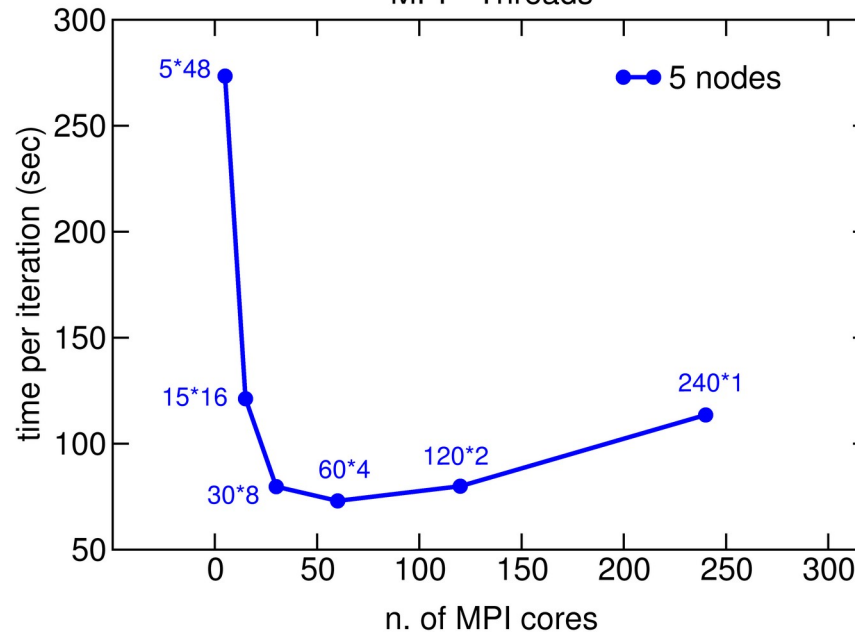
190 water molecules

MPI \* Threads



190 water molecules

MPI \* Threads

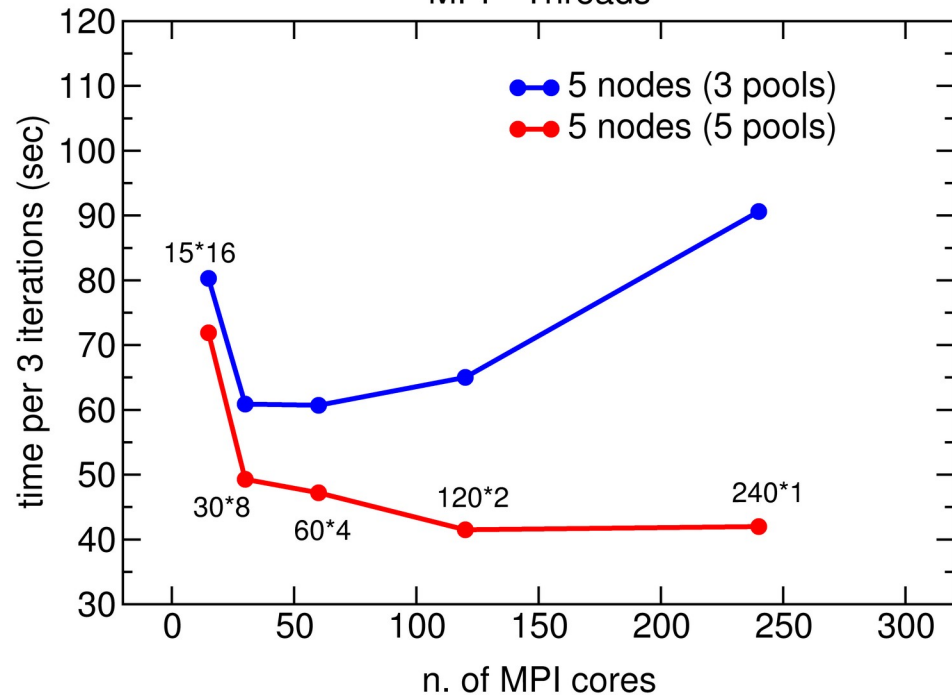


For large systems, OMP parallelization improves scaling because it allows to exploit many cores without burdening the calculation with communications

## COMBINING WITH OPENMP

CaN2Si crystal (30 K-points)

MPI \* Threads



A smart combination of MPIs, OMP Threads, and pools allows to achieve drastic reductions of computational burden

## Exercise 3

# INSTRUCTIONS

Find out how to best exploit the available CPU resources, by playing with the MPI related parameters (***number of tasks***, ***npools***) together with the ***number of threads***.

Use the batch file **ex3-omp.slurm** to submit your jobs (modify it at your convenience).

Hints on how to proceed:

1. Know the ***size of your node***, e.g. the amount of cores at your disposal;  
See how the time scaling of your jobs goes just by varying the number of tasks at first (keep just 1 thread each at first). Adapt the npool parameter at each run.
2. Now you can start to explore the ***OpenMP*** parallelization (you can focus to range **2:8 threads**).
3. Check the total WALL time at the end of the output files and do ***multiple plots*** in function of the number of ***MPI tasks*** and ***OpenMP threads***.  
Which does it seem to be the best configuration for this exercise?

## Exercise 3

# JOB SCRIPT

```
#!/bin/bash
#SBATCH --job-name=USERjob
#SBATCH --nodes 1
#SBATCH --exclusive
#SBATCH --time=00:20:00
#SBATCH --partition=cpu
#SBATCH --ntasks-per-socket=2
#SBATCH --ntasks-per-node=16
#SBATCH --hint=nomultithread
#SBATCH --cpus-per-task=4
#SBATCH --output=sysout.out
#SBATCH --error=syserr.err
# # SBATCH --mail-user=YOUR_EMAIL - if you want
```

```
module purge
module load QuantumESPRESSO/7.1-foss-2022a
.....
```

```
mpiopt="-mca pml ucx -mca btl ^uct,tcp,openib,vader --map-by socket:PE=4 --rank-by core --report-bindings "
```

```
for nthr in ....
do
export OMP_NUM_THREADS="$nthr"
mpirun $mpiopt -np 16 pw.x -npool 8 -i ${INDIR}/pw.CuO.scf.in > pw_run16x${nthr}.out
done
```

## Exercise 4: QE on GPUs



## COMPILERS AND LIBRARIES

Three things to keep in mind when installing QE:

1) The compiler

nvfortran (ex-pgi)

2) The linear algebra libraries

cuBLAS

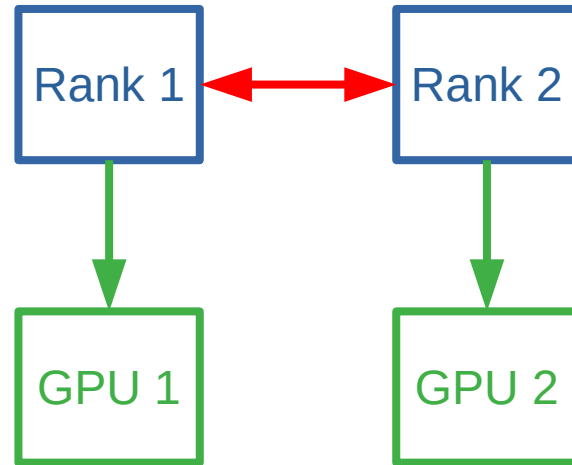
3) The FFT libraries

cuFFT

## Exercise 4

# MPI AND GPUS

When we use GPUs, each MPI process off-loads the calculation to one GPU:



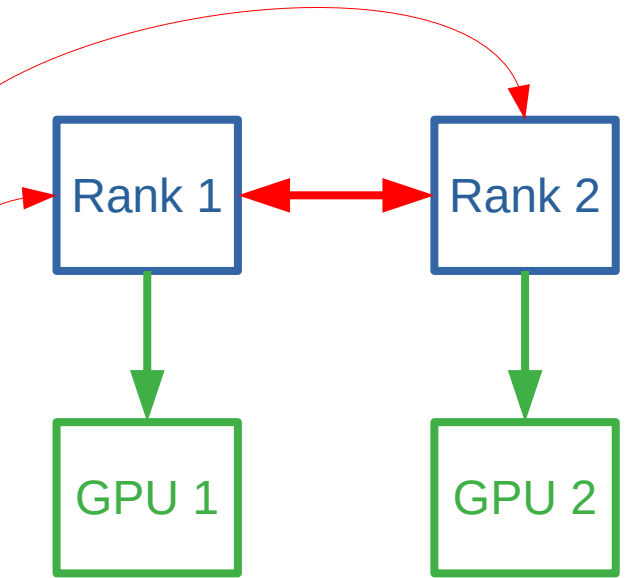
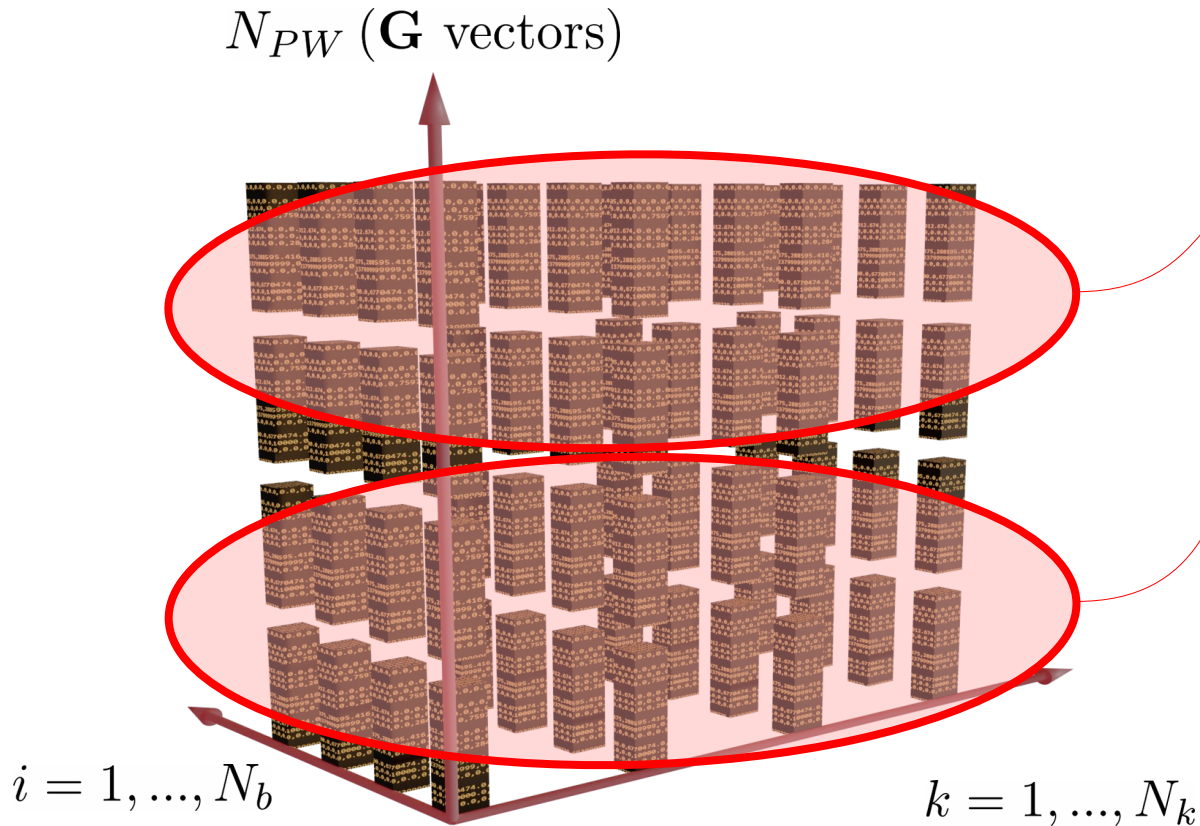
It is convenient to use one  
MPI per GPU

# Exercise 4 MPI AND GPUS

## GPU parallelism

What is happening?

When we use GPUs, each process off-loads the calculation to one GPU



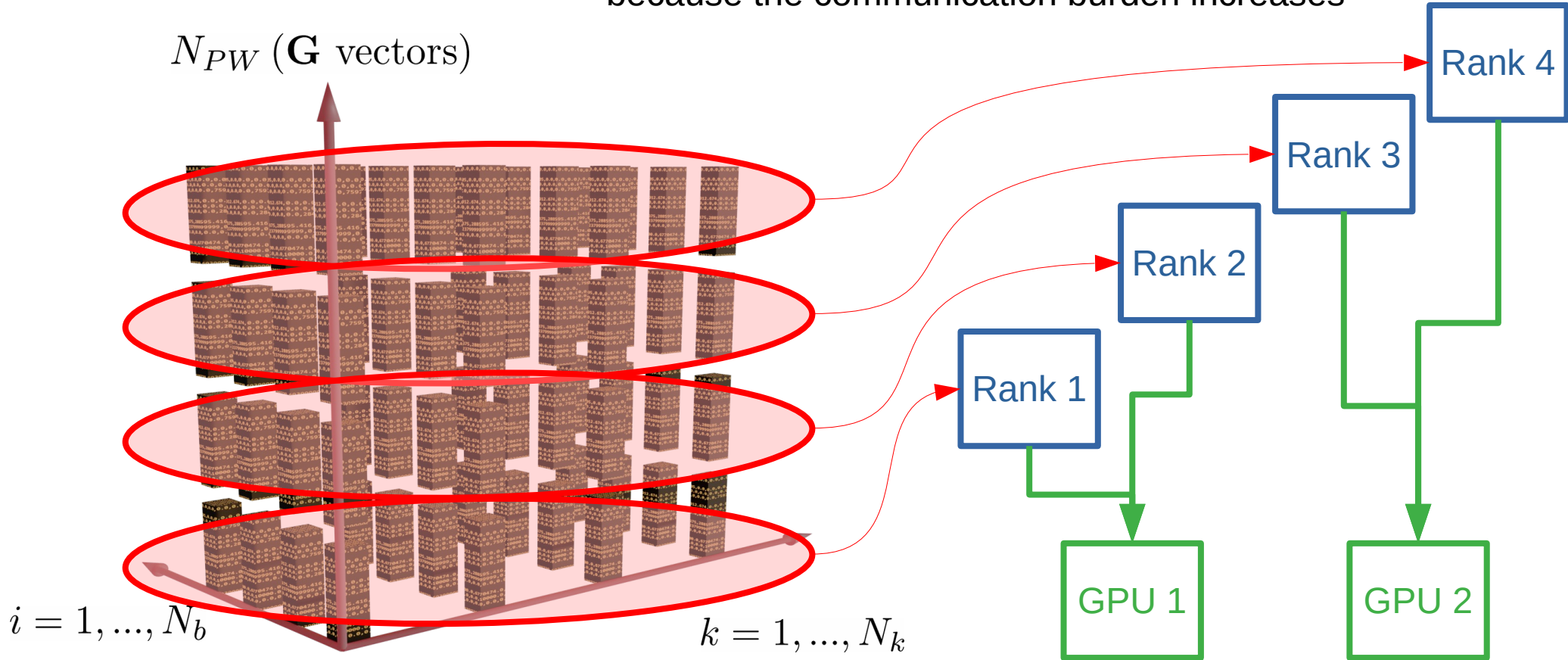
It is convenient to use one MPI per GPU

# Exercise 4 MPI AND GPUS

## GPU parallelism

What is happening?

Adding more MPIs usually will not improve performances, and might also be less efficient because the communication burden increases

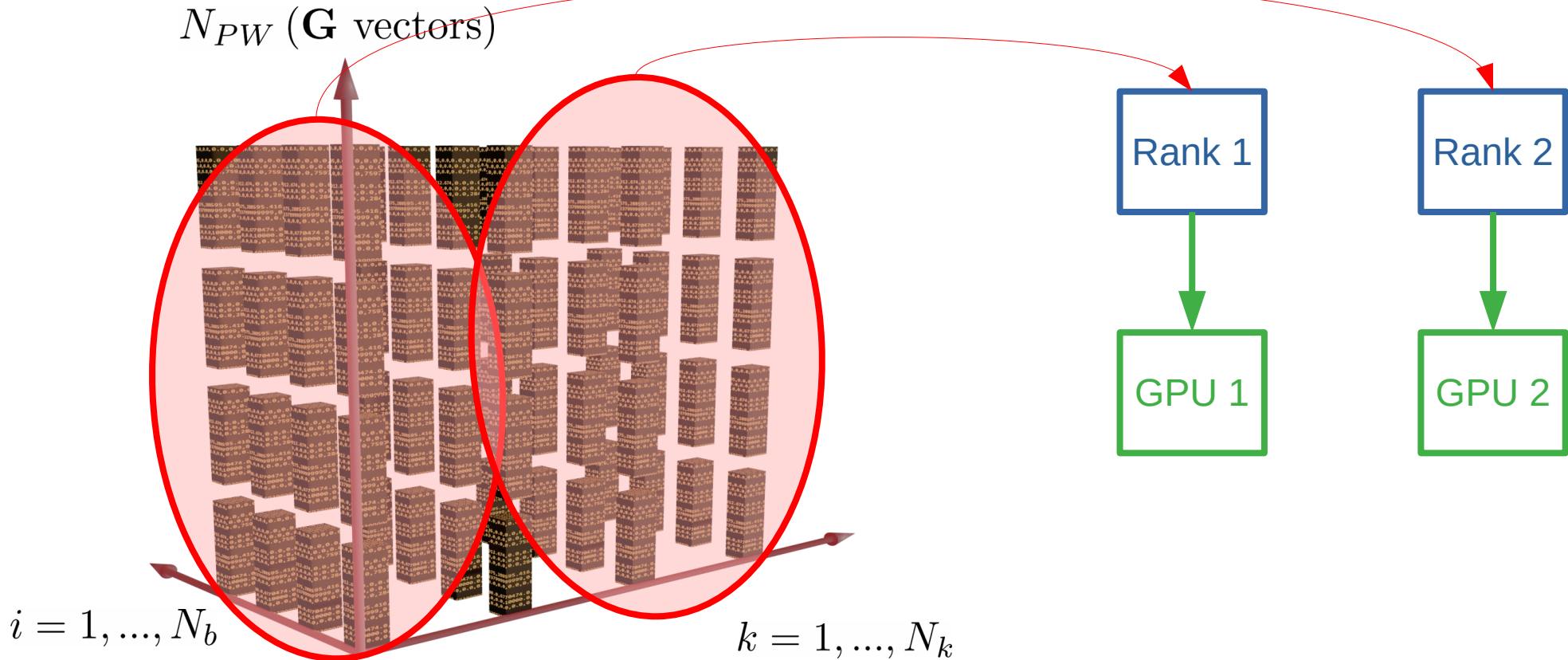


# Exercise 4 CPU AND GPUS

## GPU parallelism

What is happening?

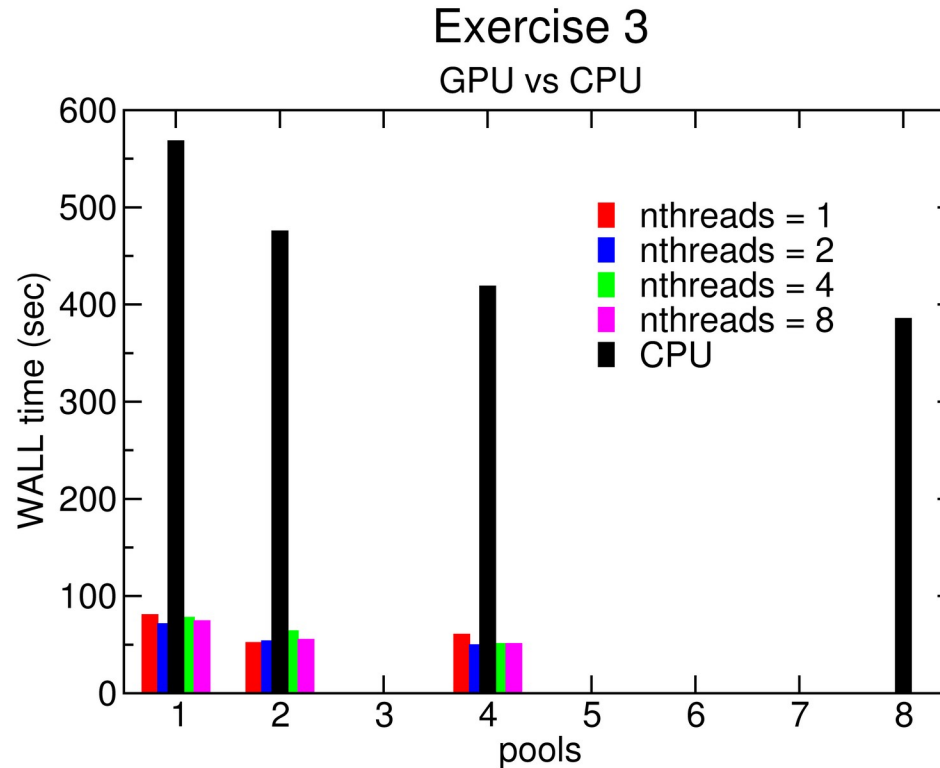
Again, using pools, will improve communications



## Exercise 4

# CPU AND GPUS

You should be able to produce a plot similar to this one:



Evaluate the ratio between the best time to solution of your CPU and GPU tests.

## MPI AND GPUS – JOB SCRIPT

```
#!/bin/bash
#SBATCH --job-name=USERjob
#SBATCH --nodes 1
#SBATCH --time=00:10:00
#SBATCH --partition=gpu
#SBATCH --ntasks-per-node=4
#SBATCH --ntasks-per-socket=2
#SBATCH --cpus-per-task=32
#SBATCH --gres=gpu:4
#SBATCH --hint=nomultithread
#SBATCH --output=sysout.out
#SBATCH --error=syserr.err
# # SBATCH --mail-user=YOUR_EMAIL - if you want
```

```
export WORK=/ceph/hpc/data/d2021-135-users
module purge
module use ${WORK}/modules
echo $WORK
module load QuantumESPRESSO/DEV-NVHPC-21.2
```

```
export EXDIR=${PWD}/..
export INDIR=${EXDIR}/inputs
export ESPRESSO_PSEUDO=${EXDIR}/../pseudo
export OMP_NUM_THREADS=1
```

```
mpiopt="-mca pml ucx -mca btl ^uct,tcp,openib,vader --map-by socket:PE=32 --rank-by core --report-bindings "
```

```
mpirun $mpiopt -np 4 pw.x -i ${INDIR}/pw.CnSnI3.in > pw.CnSnI3.gpu.out
```

## Exercise 4

# CPU vs GPUs

First launch the GPU job as is:

```
#SBATCH --ntasks-per-node=4    # number of MPI per node  
#SBATCH --ntasks-per-socket=2  # number of MPI per socket  
#SBATCH --cpus-per-task=32     # number of HW threads per tas
```

```
export OMP_NUM_THREADS=1
```

```
mpirun ... ${PW} -i ${INDIR}/pw.CnSnI3.in > pw.CnSnI3.gpu.out
```



## Exercise 4

# CPU vs GPUs

To get a (very) rough idea of the comparison CPU nodes vs GPU nodes you can run the same job on CPU and try to match the performance.

```
#SBATCH --ntasks-per-node=...    # number of MPI per node  
#SBATCH --ntasks-per-socket=...  # number of MPI per socket  
#SBATCH --cpus-per-task=...      # number of HW threads per tas
```

```
export OMP_NUM_THREADS=...
```

```
mpirun ${PW} -npool ... -ndiag ... -i ${INDIR}/pw.CnSnI3.in > pw.CnSnI3.gpu.out
```

## Final considerations

# MINIMAL RECIPE

- 1) Choose a number of **MPI tasks** depending on the:
  - z dimension of **FFT grid**;
  - number of **k-points**;
- 2) Use **GPUs** when present:
  - **1 GPU** per **MPI** task;
  - combine with **openMP** when possible.
- 3) Choose the number of **pools** depending on:
  - dimension of **FFT grid**;
  - number of **k-points**;
  - number of **MPI tasks**;
  - number of **nodes**.
- 4) Set the parallel **diagonalization**: - VERY **large systems** only (nbands>100).
- 5) Choose the number of **threads**:
  - **up to 4**: advantageous;
  - **4 to 8**: sometimes advantageous (but not too much);
  - >8: very rarely advantageous;
  - >>8: never advantageous.

# Final considerations CPU AND GPUS

## Size (Ta2O5):

el = 544

Nat = 96

Ecut = NC/130/520 Ry

NPW = 477k

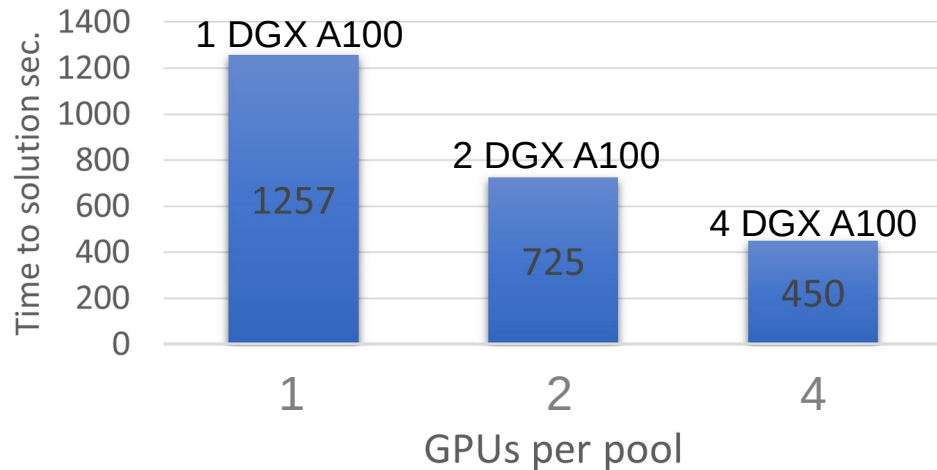
Nbnd = 326

Nks = 26

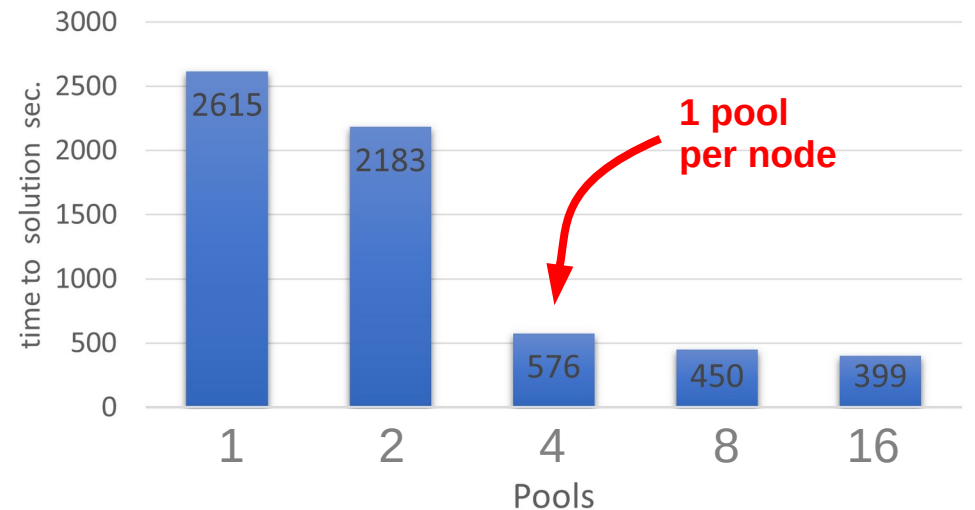
The larger the batch of data offloaded to the GPUs, the lesser the communications, the faster, as far as you have memory for them (memory on Ampere architecture up to 80GB).

At software level, versatile parallelization strategies based on **multiple data distribution schemes**, allow an optimal exploitation of the hardware architecture

### Plane wave scaling (8 pools)



### Pool parallelism (4 DGX A100 nodes)



Taken from L. Stuber (NVIDIA), I. Carnimeo (SISSA), P. Delugas (SISSA), F. Spiga (NVIDIA) unpublished benchmarks.  
All calculations on A100 kindly provided by NVIDIA Corp.

# Final considerations CPU AND GPUs

## Size:

el = 4445

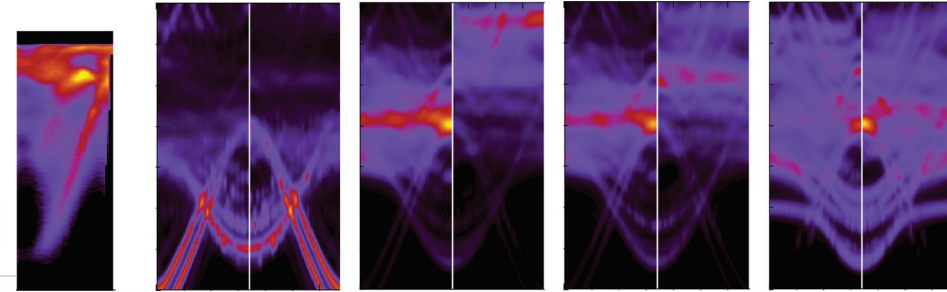
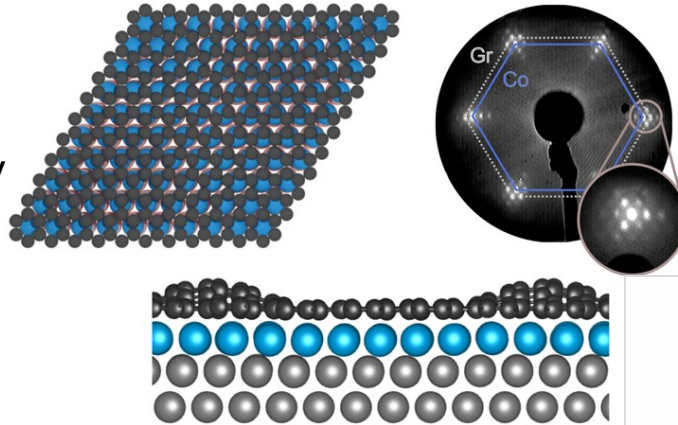
Nat = 605

Ecut = NC/75/300 Ry

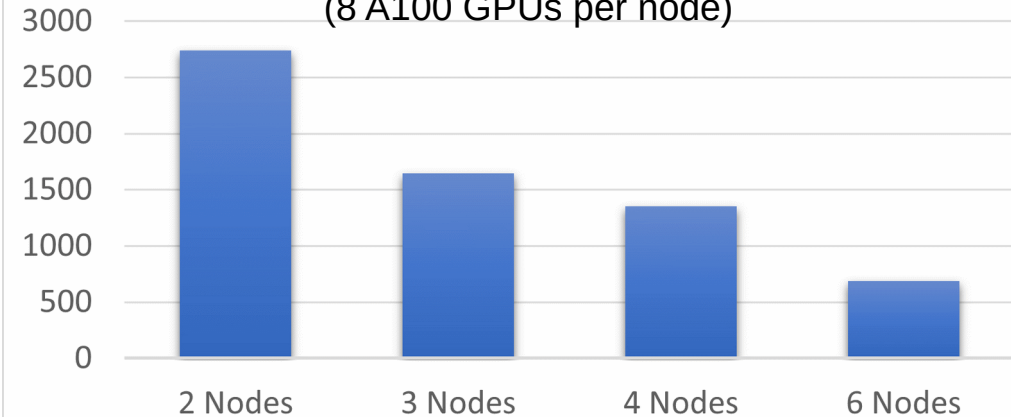
NPW = 994k

Nbnd = 2688

Nks = 12



GrColr benchmark  
( t.t.s. from restart )  
( 8 A100 GPUs per node )



Good scaling performances have been found for large systems, by combining plane waves and pools parallelization strategies

Appl. Phys. Lett. 118, 121602 (2021);  
Nano Lett. (2018) 18, 2268–2273

Taken from L. Stuber (NVIDIA), I. Carnimeo (SISSA), P. Delugas (SISSA), F. Spiga (NVIDIA) unpublished benchmarks.  
All calculations on A100 kindly provided by NVIDIA Corp.

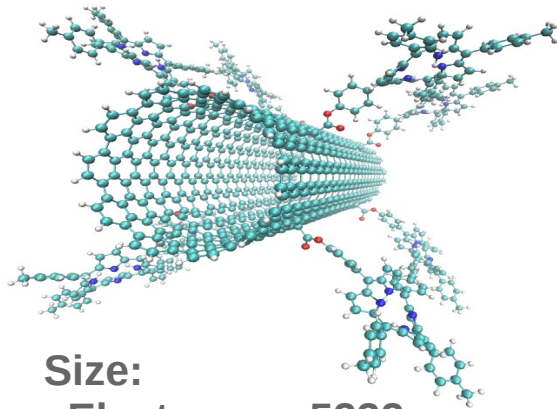
MPI =	16	48	32	96
NPOOLS =	4	6	4	12

# Final considerations CPU AND GPUs

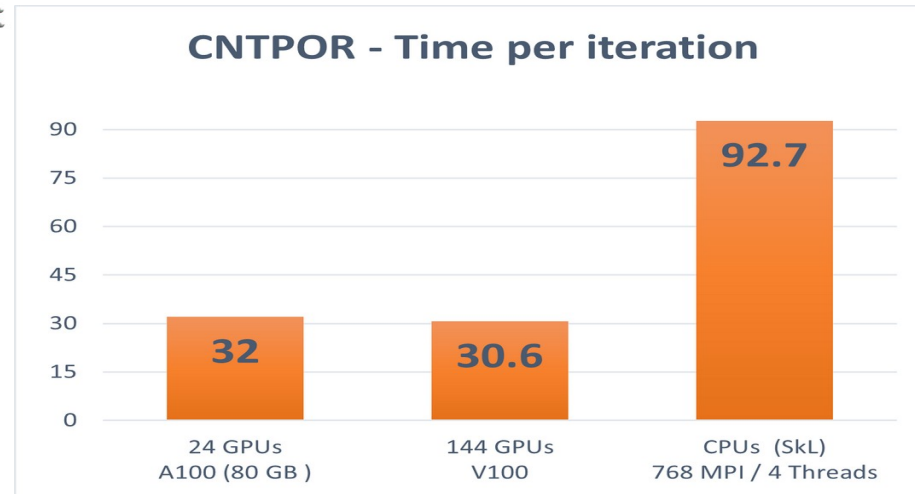
The QUANTUM ESPRESSO™ codes have been engineered to exploit **exascale** computational facilities, through an extensive porting to **hybrid CPU-GPU architectures**, using mixed (CUDA Fortran/OpenACC, OpenMP) offload schemes, in order to enhance **portability to hardware from different vendors**

GPU porting

	6.4	6.5a1	6.5a2	6.7	6.8	7.0	develop
Davidson							
Energy							
Magnetism							
USPP							
PAW							
Forces							
Stress							
KS Solvers							
DFT+U							
XC							
VdW(D3)							
CP							
Phonon							
turbo_eels							
turbo_lanczos							



Size:  
Electrons = 5232  
Atoms = 1532



# Final considerations CPU AND GPUs

## Size:

**el = 5232**

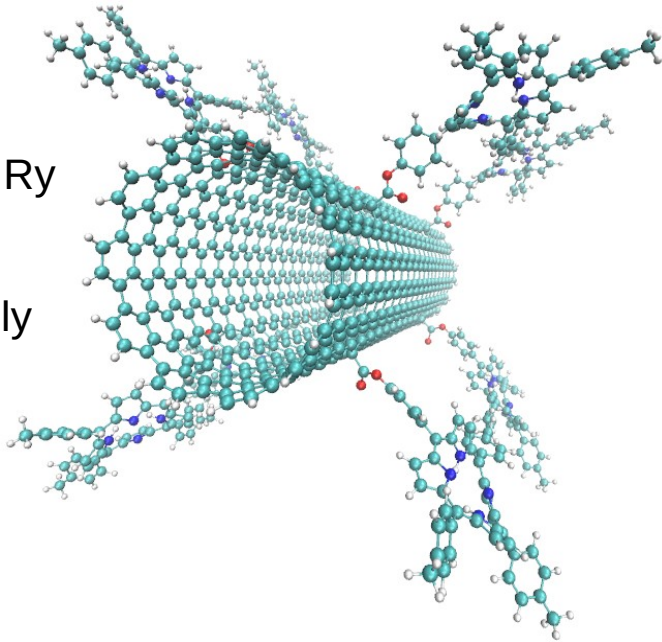
Nat = 1532

Ecut = US/25/200 Ry

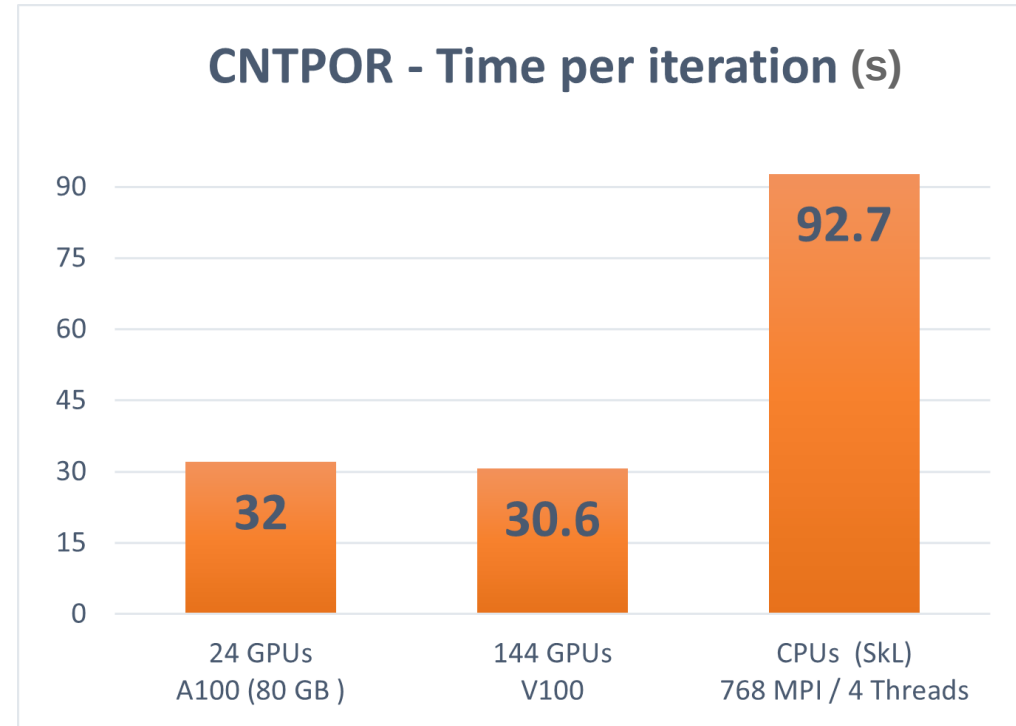
NPW = 27M

Nbnd = 2616

Nks = Gamma only



Benchmark tests on a functionalized Carbon Nanotube show that 24 GPUs of the Ampere A100 (80 GB) architecture perform as 144 Volta V100 (16 GB) and much better than 3072 SkyLake cores



Taken from L. Stuber (NVIDIA), I. Carnimeo (SISSA), P. Delugas (SISSA), F. Spiga (NVIDIA) unpublished benchmarks.  
All calculations on A100 kindly provided by NVIDIA Corp.

---

Efficient materials modelling on HPC  
with QUANTUM ESPRESSO, Yambo and BigDFT

---

Hands-on session - Day 1

Thank you,  
see you tomorrow