QUANTUM ESPRESSO FOUNDATION

QUANTUMESPRESSO

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

Hands-on session – Day 1

QUANTUM ESPRESSO on HPC systems

Oscar Baseggio and Aurora Ponzi

obaseggi@sissa.it aponzi@sissa.it

MAX

SISSA

CINECA

ENCCS
EuroCC National Competence Centre Sweden

PAUL SCHERRER INSTITUT
PSI

MARVEL
NATIONAL CENTRE OF COMPETENCE IN RESEARCH

Swiss National Science Foundation

RECAP

Covered topics are:

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

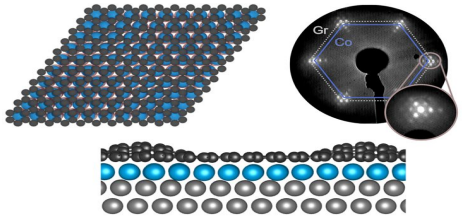
Exercise 1:** optimize CPU execution with npools

Exercise 2:** optimize CPU execution with ndiag

Exercise 3:** openMP acceleration

Exercise 4:** hands on the GPUs

PWscf FLOW



Input: molecular geometry

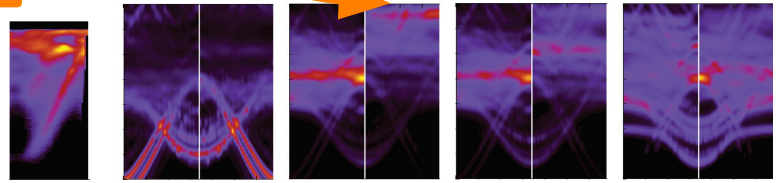
PWscf

KS Solvers
Davidson PPCG
ParO RMM-DIIS CG

Make KS potential
XClib cuFFT
cuBLAS

Diagonalization
cuSOLVER

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



Output: chem/phys properties

GETTING the MATERIAL

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

ssh USER@login.leonardo.cineca.it

Then clone the school repo and move there:

git clone https://github.com/ENCCS/max-coe-workshop.git
cd max-coe-workshop
cd Day-1
pwd

Check the result of pwd:

/leonardo_scratch/large/userexternal/USER/max-coe-workshop/Day-1

USEFUL LINKS and DOCUMENTATION

QUANTUM ESPRESSO (QE) website:
https://www.quantum-espresso.org/

Download page for code and documentation:
https://www.quantum-espresso.org/download-page/

QE users forum:
https://www.quantum-espresso.org/users-forum/

Mailing list:
users@lists.quantum-espresso.org

Exercise 0: first run with PW

INPUT FILE

https://www.quantum-espresso.org/Doc/INPUT_PW.html

```
 &control
    calculation='scf',
    tstress=.true.
 /
 &system
   Ibrav=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
```

MODULE LOAD and FIRST RUN

We only use **pw.x** for today hands-on. Go to exercise 0 (**ex0-atom** folder).
The executable of QEv7.2 for CPU is compiled in the directory:

/leonardo_work/EUHPC_TD02_030/builds/qe7.2/bin

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***.... ${ESPRESSO_DIR}/pw.x -in ${INDIR}/atom-pbe.in > atom-pbe.out**

Basic SLURM commands

Submit a job:
sbatch <filename>

Check the job status:
squeue -u <username>

Cancel a job from the queue:
scancel <jobid>

Submit our first job:

sbatch ex0-run.slurm

FIRST RUN

```
Program PWSCF v.7.2 starts on  4Mar2024 at 15:30:35
Git branch: heads/qe-7.2
Last git commit: b4c5e8deeb7d1863e7553a87b45f8d22c21926a5
Last git commit date: Mon Mar 27 16:12:09 2023 +0000
Last git commit subject: Merge branch 'merge_fix_hubbard' into 'develop'

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
506166 MiB available memory on the printing compute node when the environment starts
```

```
     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/
```

FIRST RUN

```
     vloc_psi              :      0.02s CPU      0.02s WALL (  43 calls)
     add_vuspsi            :      0.00s CPU      0.01s WALL (  43 calls)

     General routines
     calbec                :      0.00s CPU      0.01s WALL (  52 calls)
     fft                   :      0.04s CPU      0.08s WALL (  98 calls)
     ffts                  :      0.00s CPU      0.00s WALL (  16 calls)
     fftw                  :      0.01s CPU      0.01s WALL (         186 calls)
     interpolate           :      0.01s CPU      0.01s WALL (           8 calls)

     Parallel routines

     PWSCF                 :      0.70s CPU      1.20s WALL

   This run was terminated on:  15:30:36   4Mar2024

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

Exercise 1: parallelization with pools

PARALLELISM with POOLS

Kohn-Sham equations

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

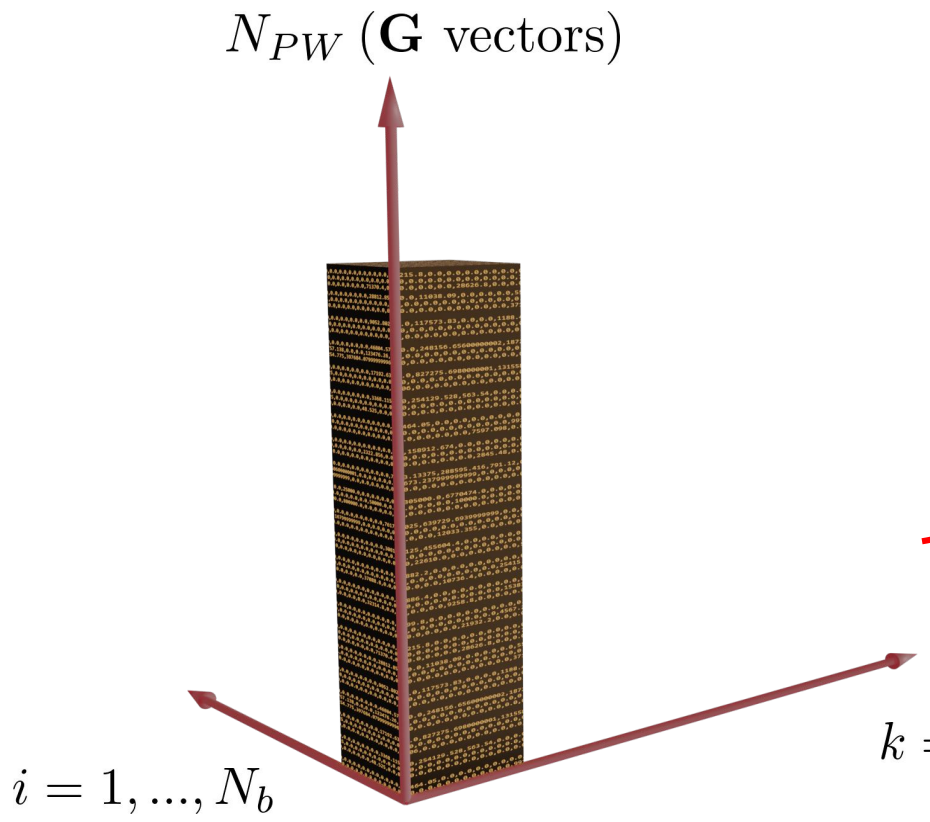
$$\psi_{ik}(\mathbf{r}) = \sum_{\mathbf{G}}^{N_{PW}} C_{\mathbf{G},ik} \frac{e^{i((\mathbf{G}+\mathbf{k})\cdot\mathbf{r})}}{\sqrt{\Omega}}, \quad \frac{\hbar^2}{2m}|\mathbf{k}+\mathbf{G}|^2 \leq E_{cut}$$

The code computes an unsymmetrized charge density

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

PARALLELISM with POOLS

Pool parallelism
What is happening?



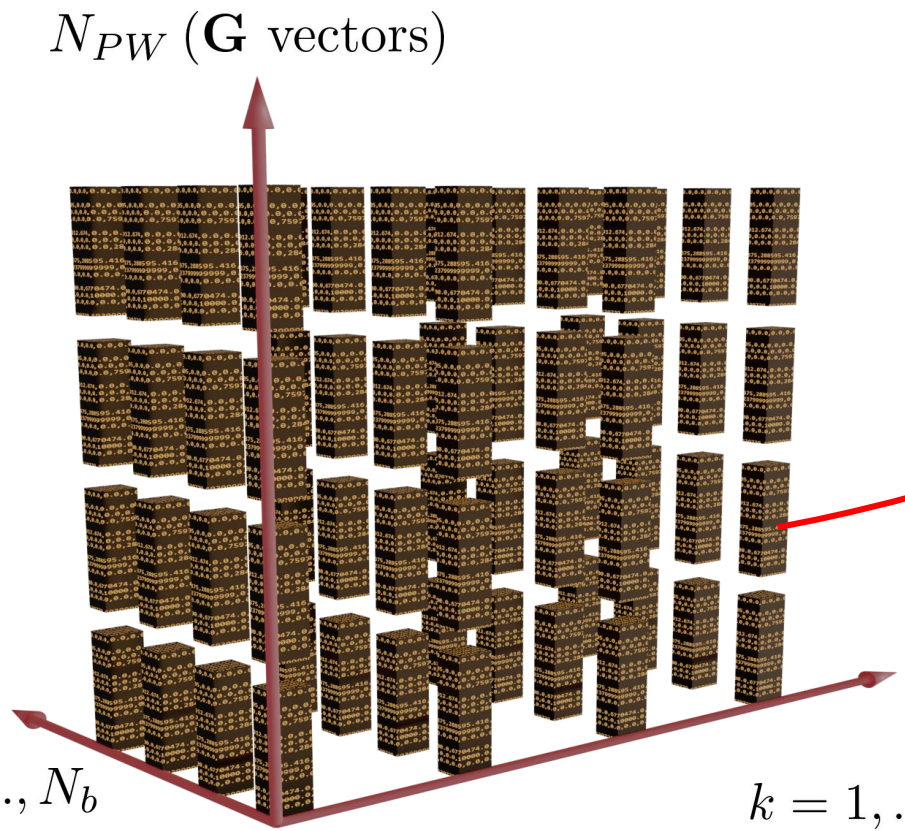
Remember the distribution of the wavefunction

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

Pool parallelism
What is happening?

Remember the distribution of the wavefunction

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

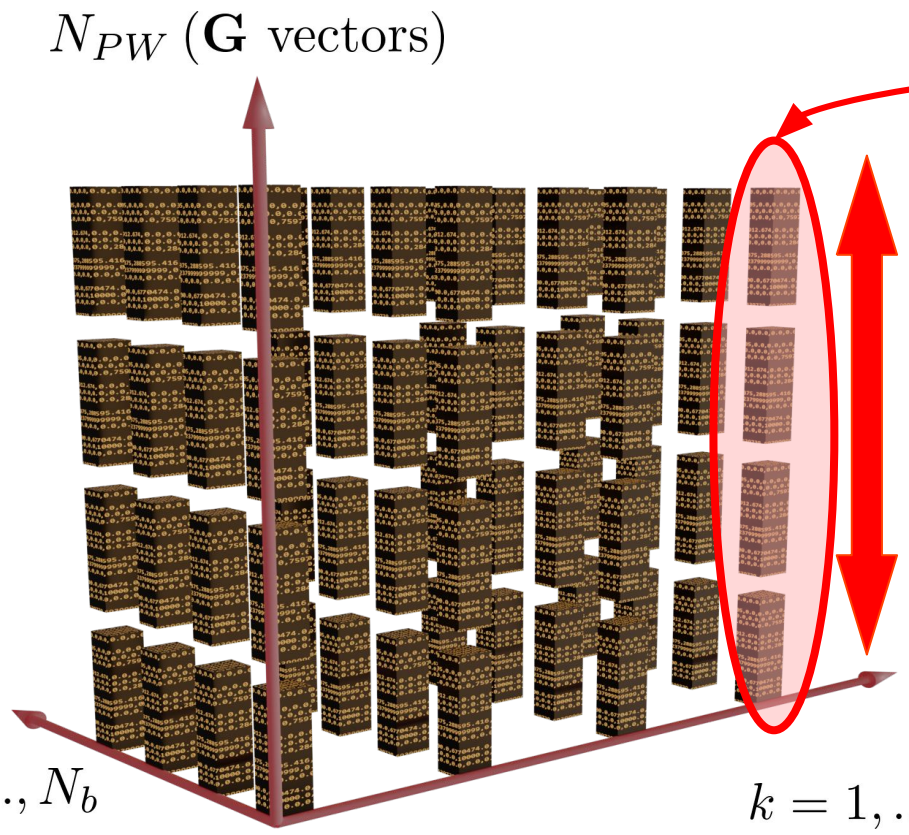


PARALLELISM with POOLS

Pool parallelism

What is happening?

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

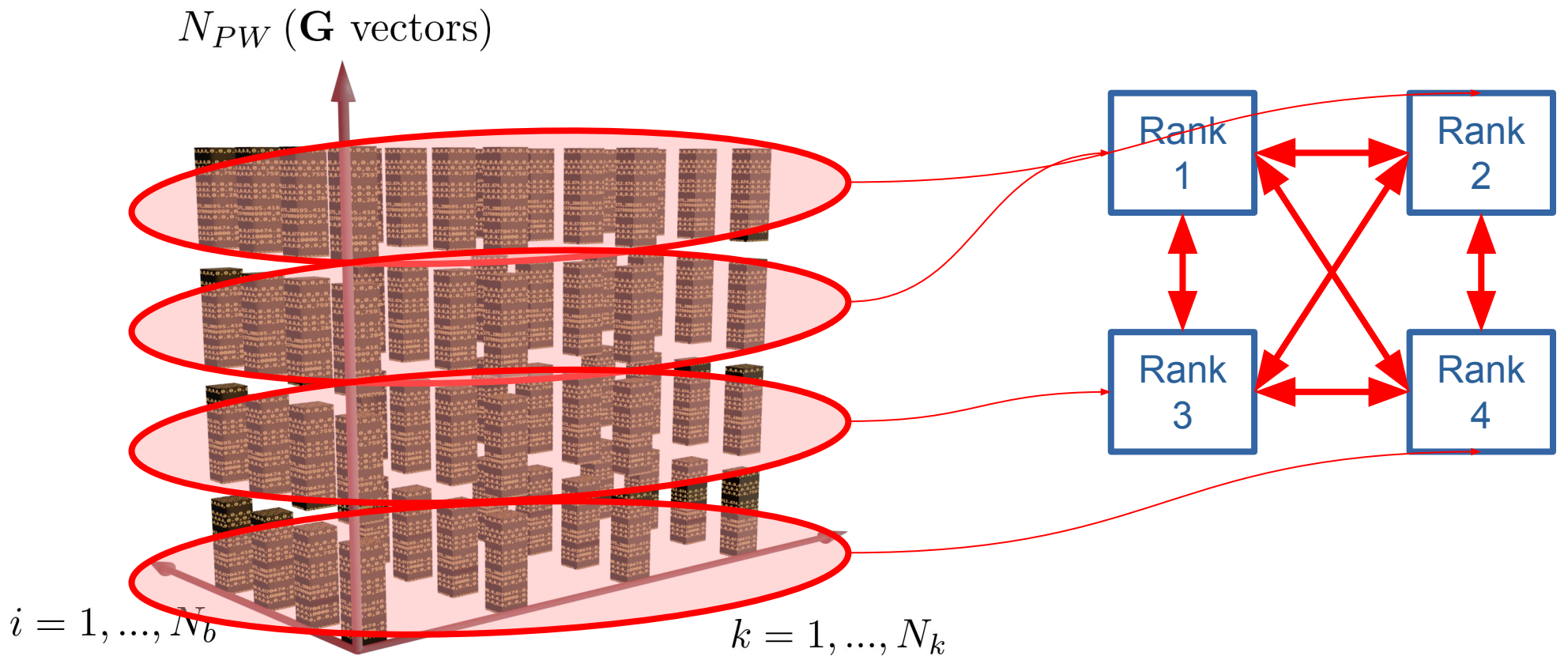


Since operators are usually applied to single orbitals, most of the communications (e.g. mp_sum) are usually done along the NPW dimension

PARALLELISM with POOLS

Pool parallelism
What is happening?

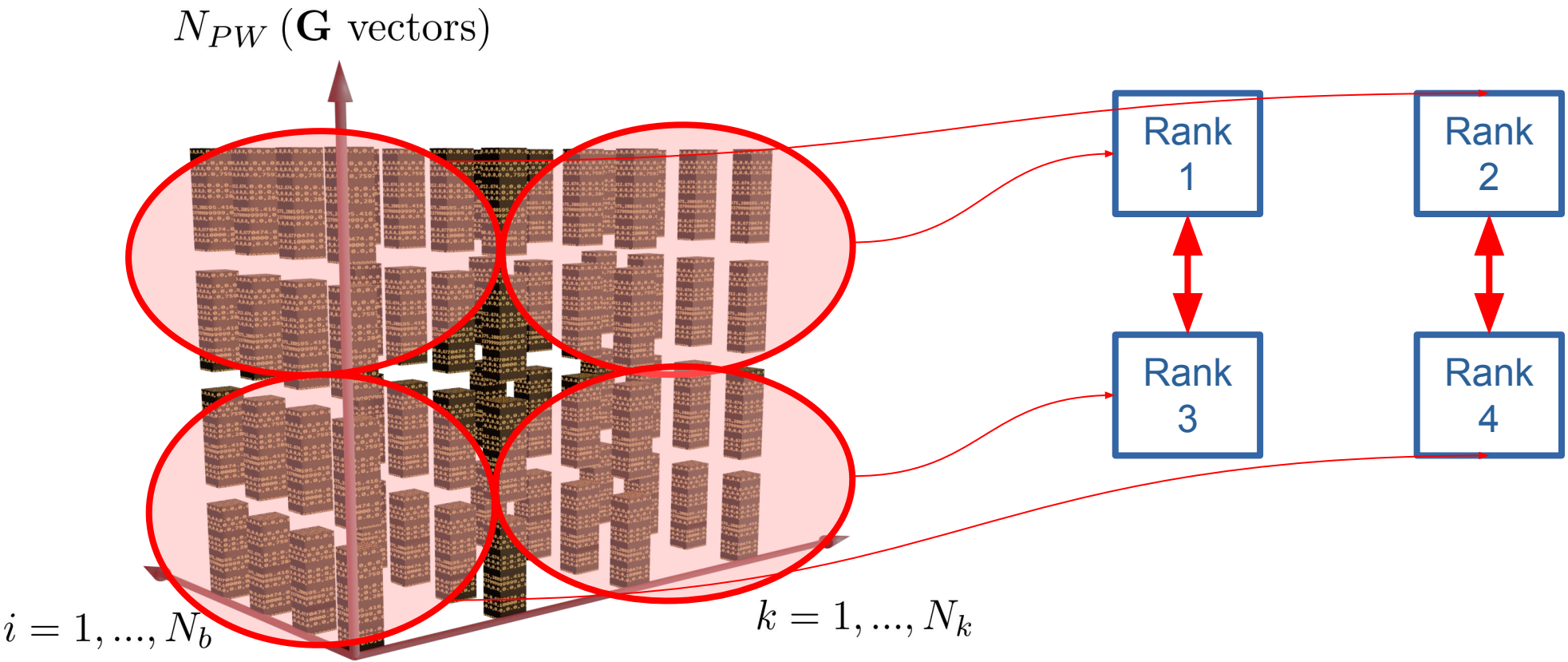
When we parallelize over PW, all processes need to communicate with each other



PARALLELISM with POOLS

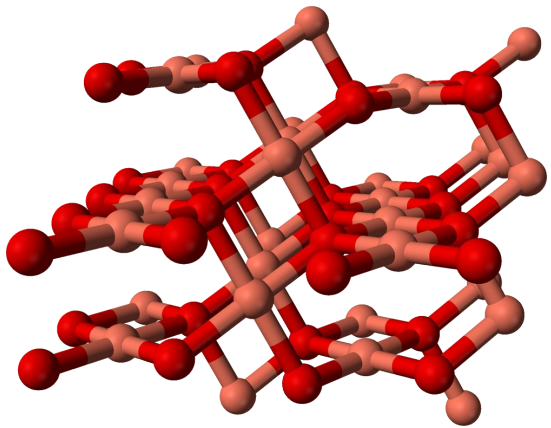
Pool parallelism
What is happening?

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=6
 /
.
.
.
K_POINTS automatic
2 2 2 0 0 0
```



Exercise 1

JOB SCRIPT

```
#!/bin/bash
#SBATCH --job-name=USERjob
#SBATCH --nodes 1
#SBATCH --exclusive
#SBATCH --time=00:20:00
#SBATCH --partition=boost_usr_prod
#SBATCH --qos=boost_qos_dbg
#SBATCH --ntasks-per-node=32
#SBATCH --cpus-per-task=1
#SBATCH --output=sysout.out
#SBATCH --error=syserr.err
#SBATCH --account=EUHPC_TD02_030
# # SBATCH --mail-user=YOUR_EMAIL - if you want

source ../environment-cpu.sh
export ESPRESSO_DIR=/leonardo_work/EUHPC_TD02_030/builds/qe7.2/bin

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

export OMP_NUM_THREADS=1

for ip in ...
do
mpirun -np 32 ${ESPRESSO_DIR}/pw.x -npool "$ip" -ndiag 1 -i ${INDIR}/pw.CuO.scf.in > pw_CuO_${ip}pools.out
done
```

SUBMIT JOB

Try to predict which the best value for **npool** will be and verify it by performing a series 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 4 8
do
```

2. Submit the job file:

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

3. Check the total WALL time at the end of the output file and plot it as a function of *npool*.
Which seems to be the best *npool* value? Why?

TIME PLOT

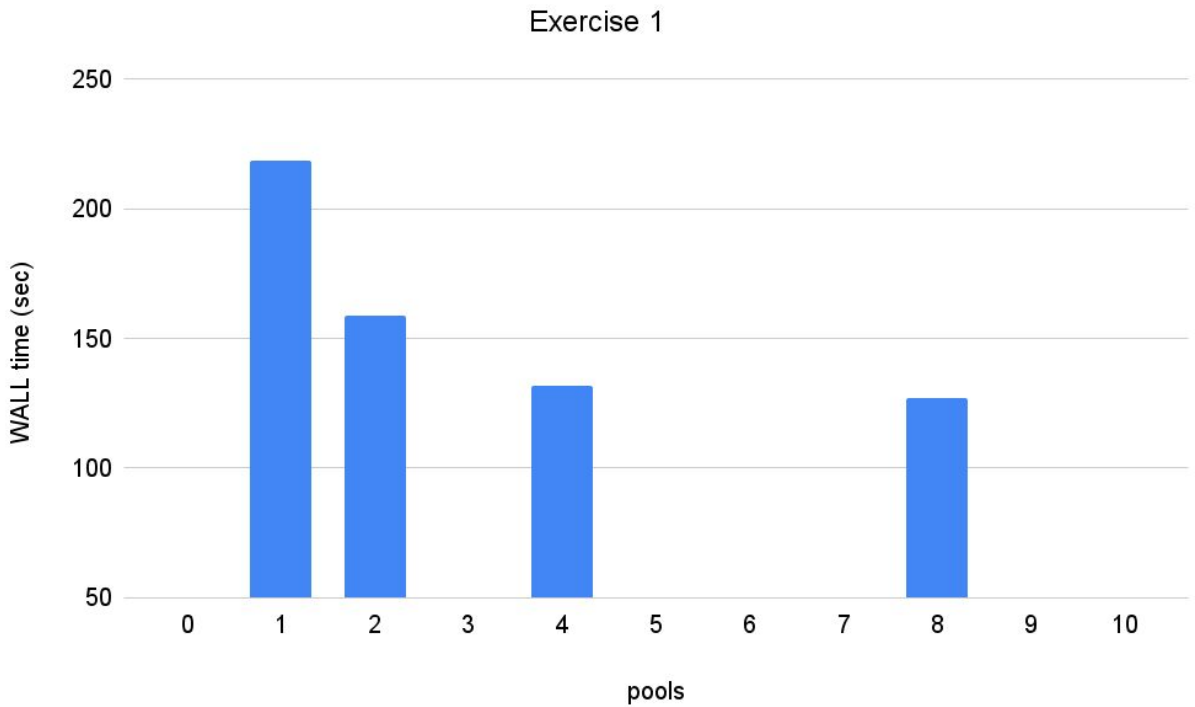
For each output file collect the “WALL time” at end of the file:

PWSCF : 3m31.00s CPU 3m36.57s 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.

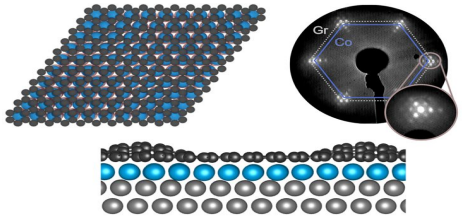
Pool parallelism

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



Exercise 2: parallel diagonalization

PWscf FLOW



Input: molecular geometry

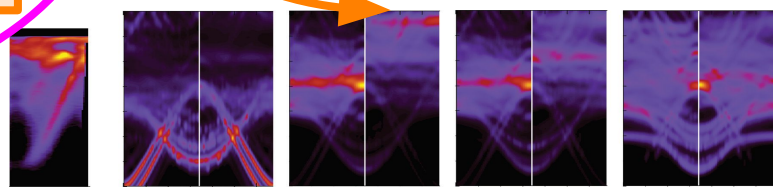
PWscf

KS Solvers
Davidson PPCG
ParO RMM-DIIS CG

Make KS potential
XClib cuFFT
cuBLAS

Diagonalization
cuSOLVER

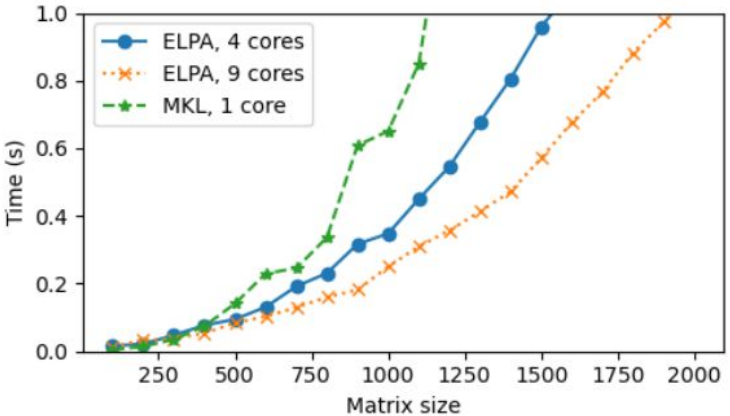
this is achieved using a **modular organization** of the code which allows great flexibility and ease to use external and internal numerical libraries

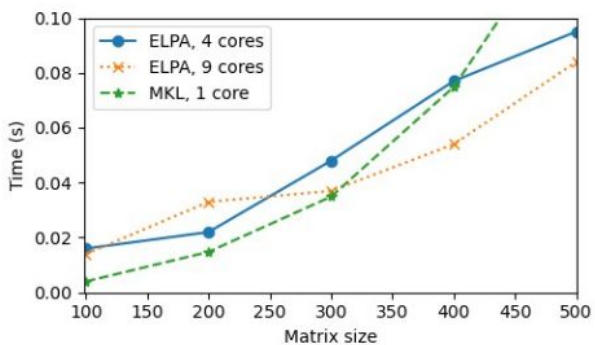


Output: chem/phys properties

PARALLEL DIAGONALIZATION

Trend of Parallel Diagonalization





- Results for all eigen-states.
- Additional communications in parallel Davidson.

by PIETRO BONFA' from MaX School 2021(ICTP meeting)

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 setting **-ndiag** 4

mpirun -np 32 ${ESPRESSO_DIR}/pw.x -npool 4 -ndiag "$id" -i ${INDIR}/pw.CuO.scf.in > pw_CuO_${id}diag.out

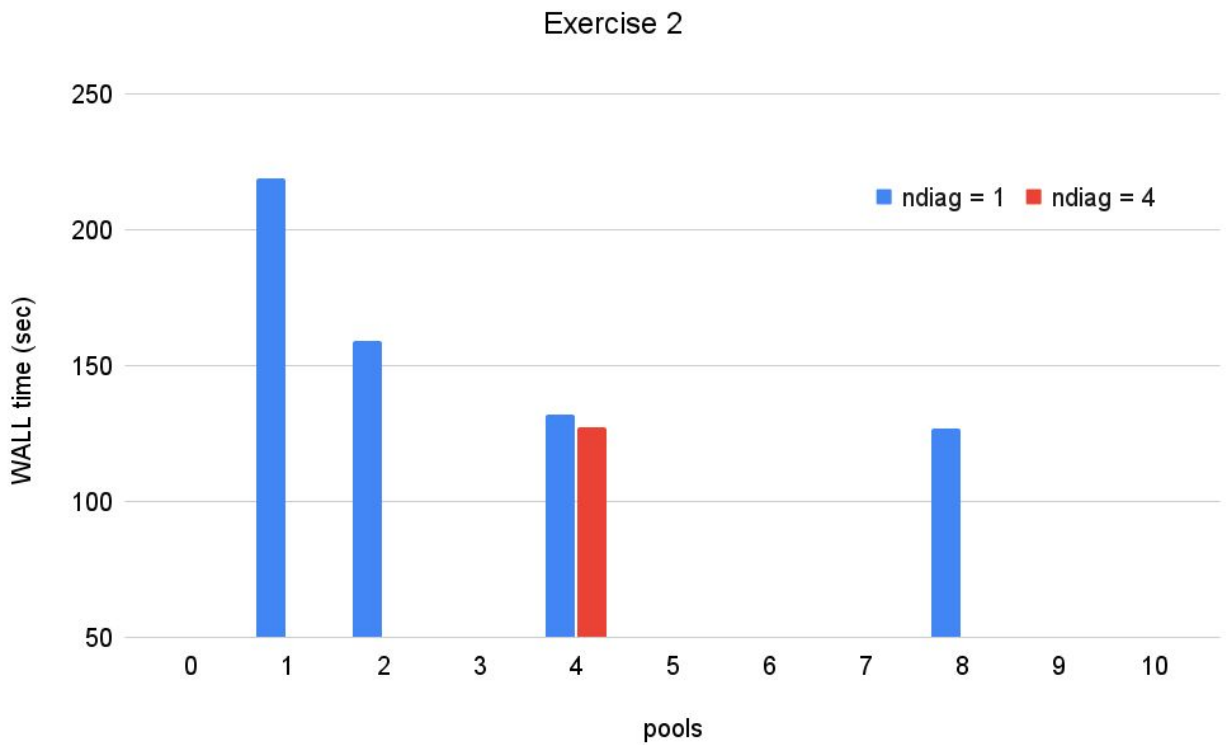
Inspect the beginning of the output file and look for this message

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;

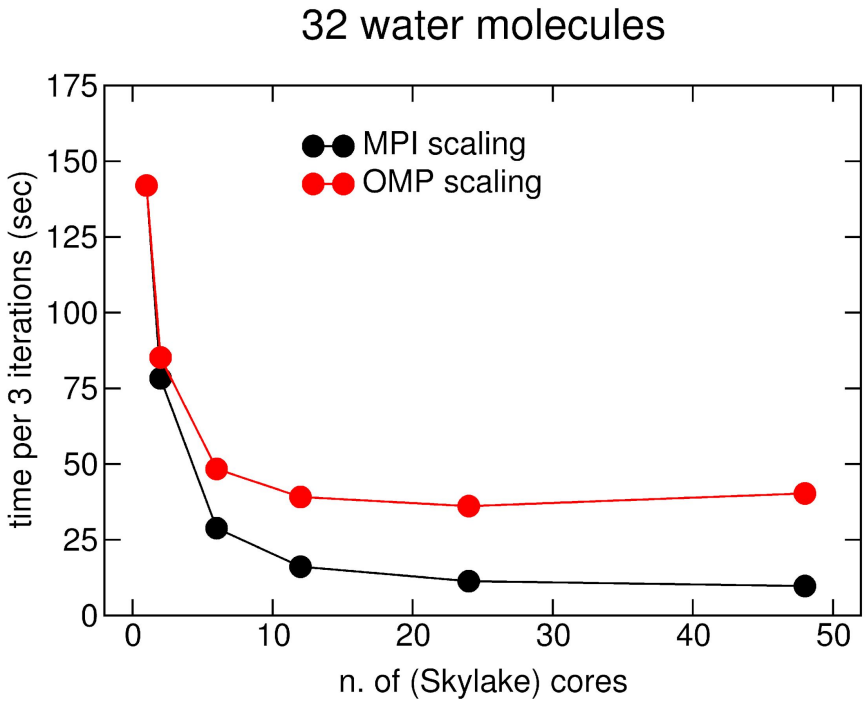
2) the **eigenvalue problem** is too small in this case to fully take advantage of parallel diagonalization;

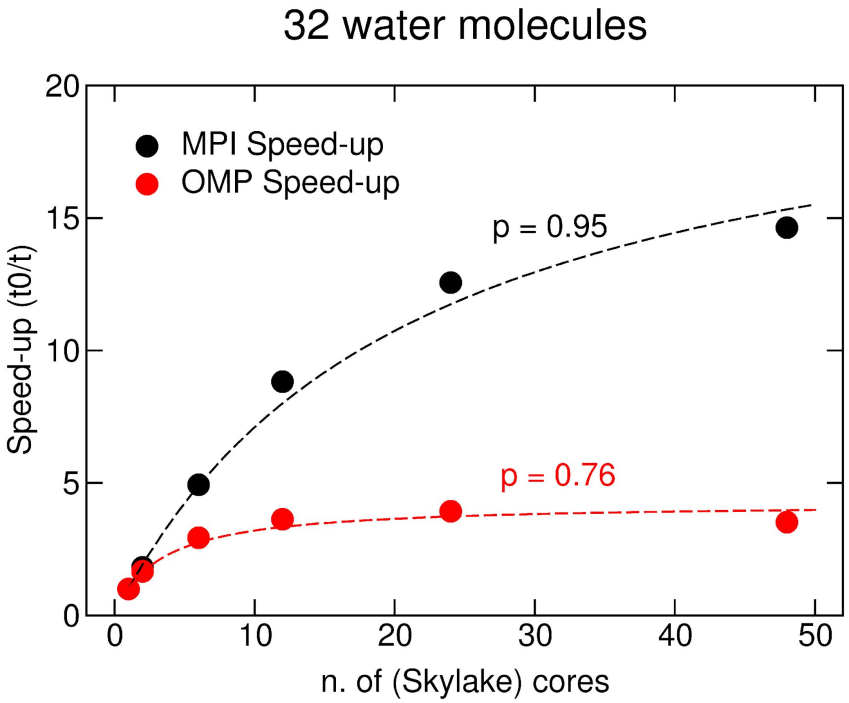
3) **other libraries**, e.g. Scalapack or ELPA, usually provide better performance in parallel diagonalization.

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

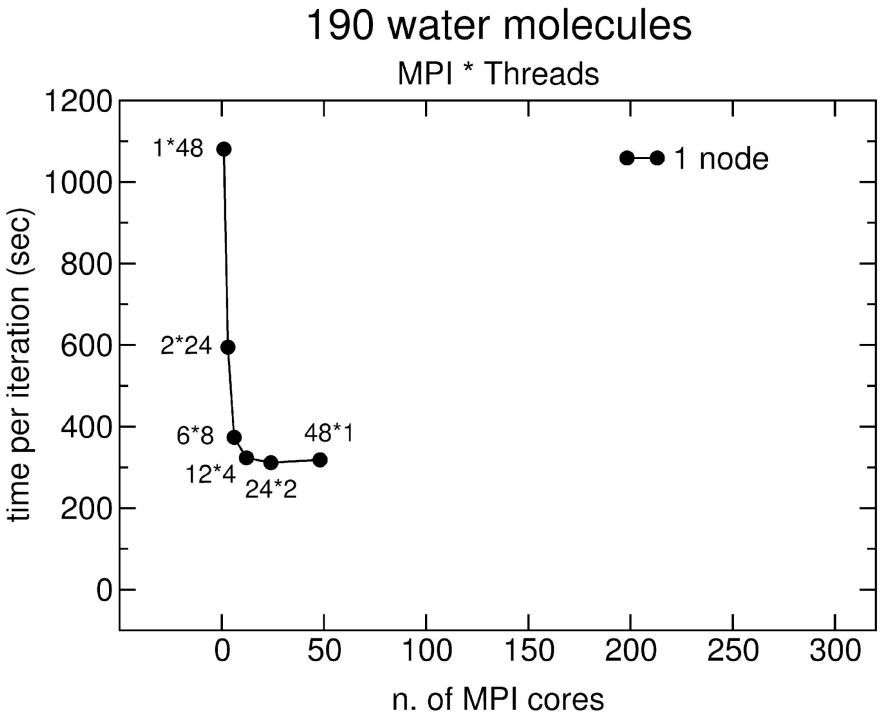




Scaling (Amdahl Law) for QUANTUM ESPRESSO code for both MPI and OpenMP portions of the code.

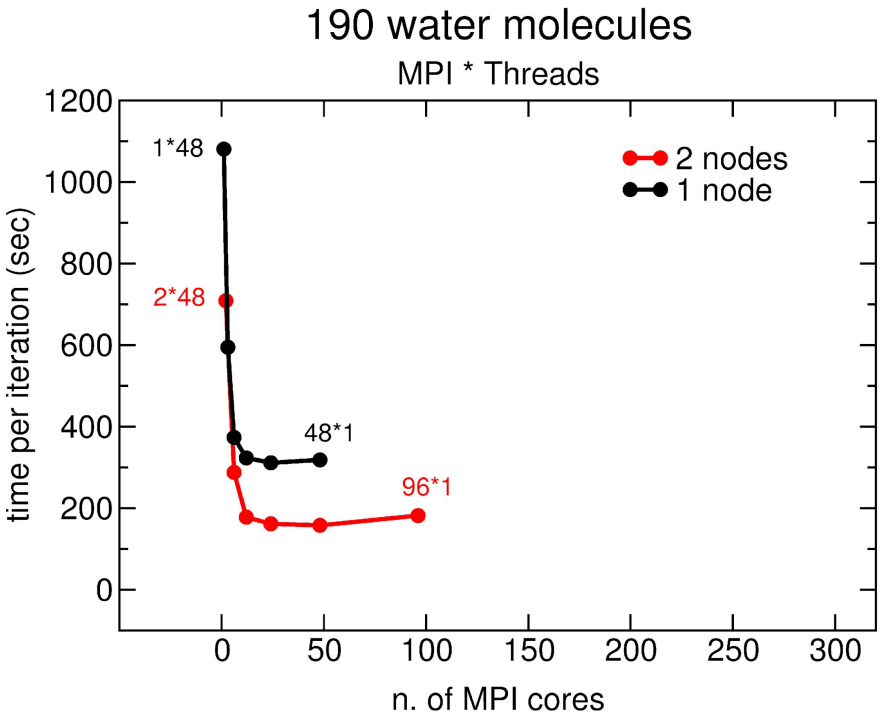
OMP parallelization is usually less efficient than MPI for QE, but involves less communications.

COMBINING with OPENMP



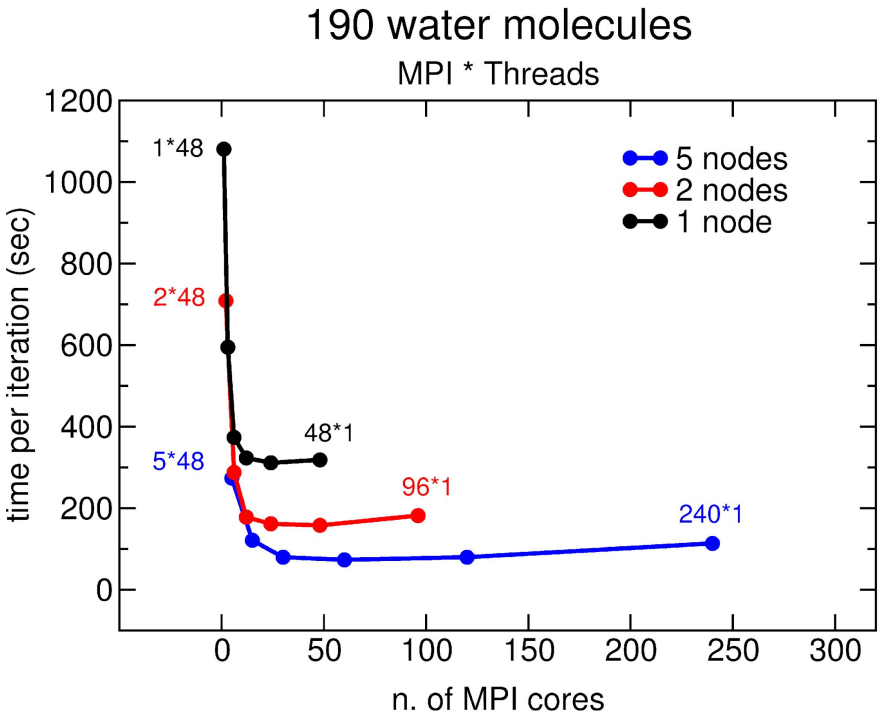
MPI and OMP threads can be combined to better exploit computational resources. OMP parallelization is usually less efficient than MPI for QE, but involves less communications.

COMBINING with OPENMP



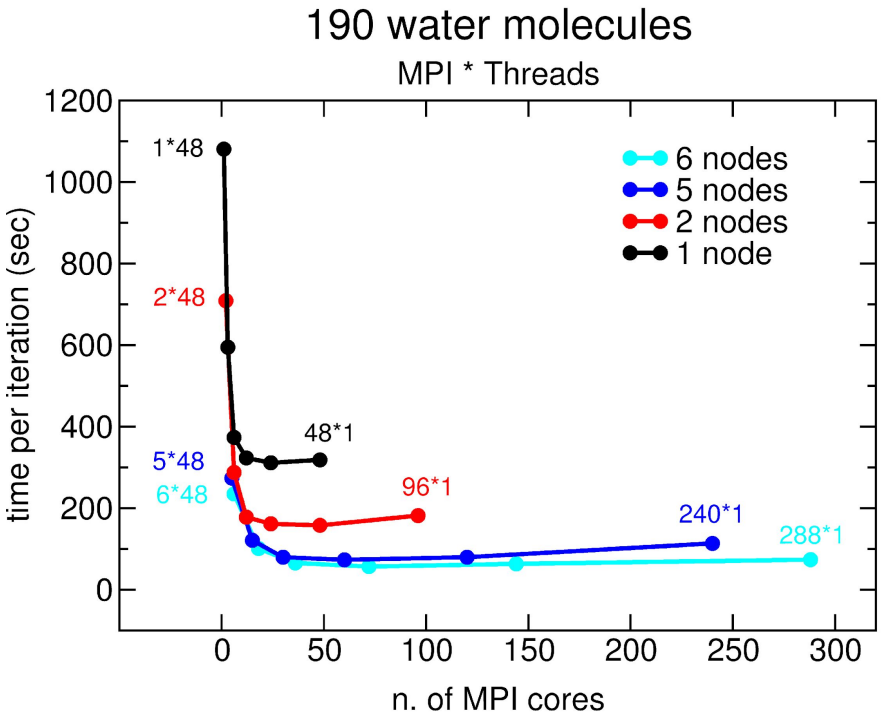
MPI and OMP threads can be combined to better exploit computational resources. OMP parallelization is usually less efficient than MPI for QE, but involves less communications.

COMBINING with OPENMP



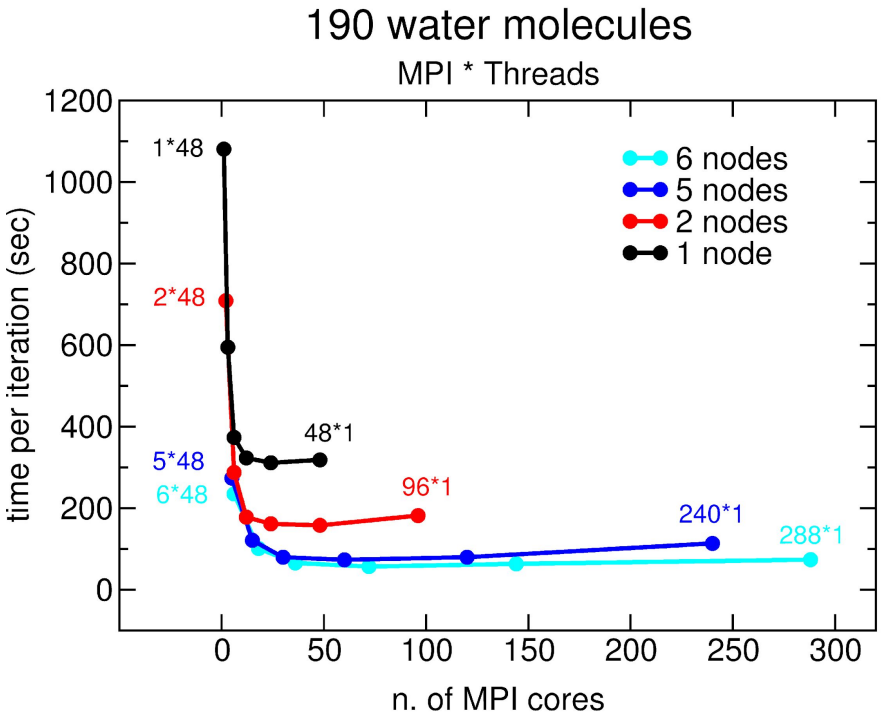
MPI and OMP threads can be combined to better exploit computational resources. OMP parallelization is usually less efficient than MPI for QE, but involves less communications.

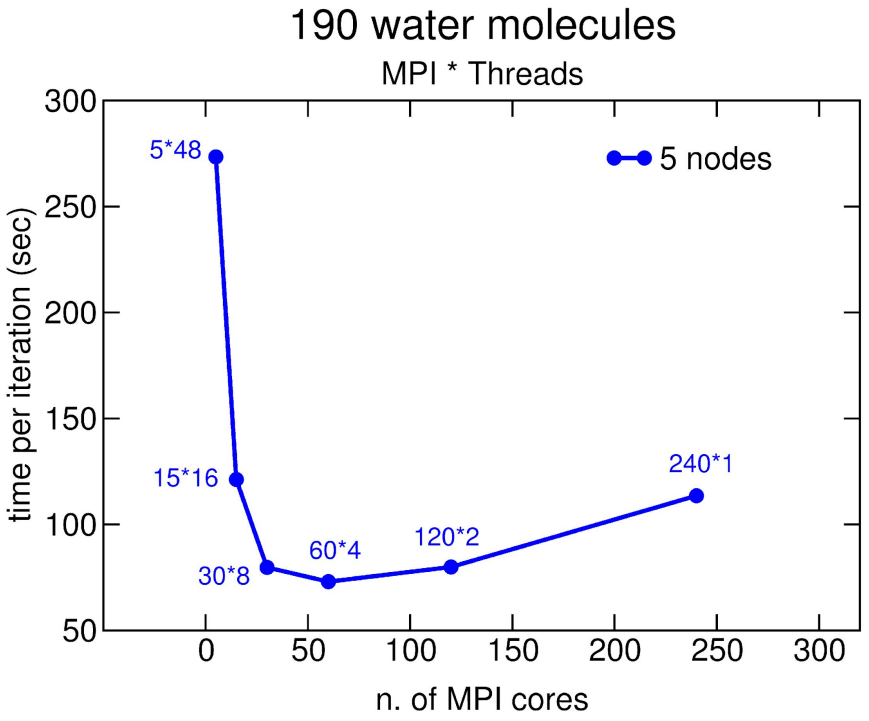
COMBINING with OPENMP



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.

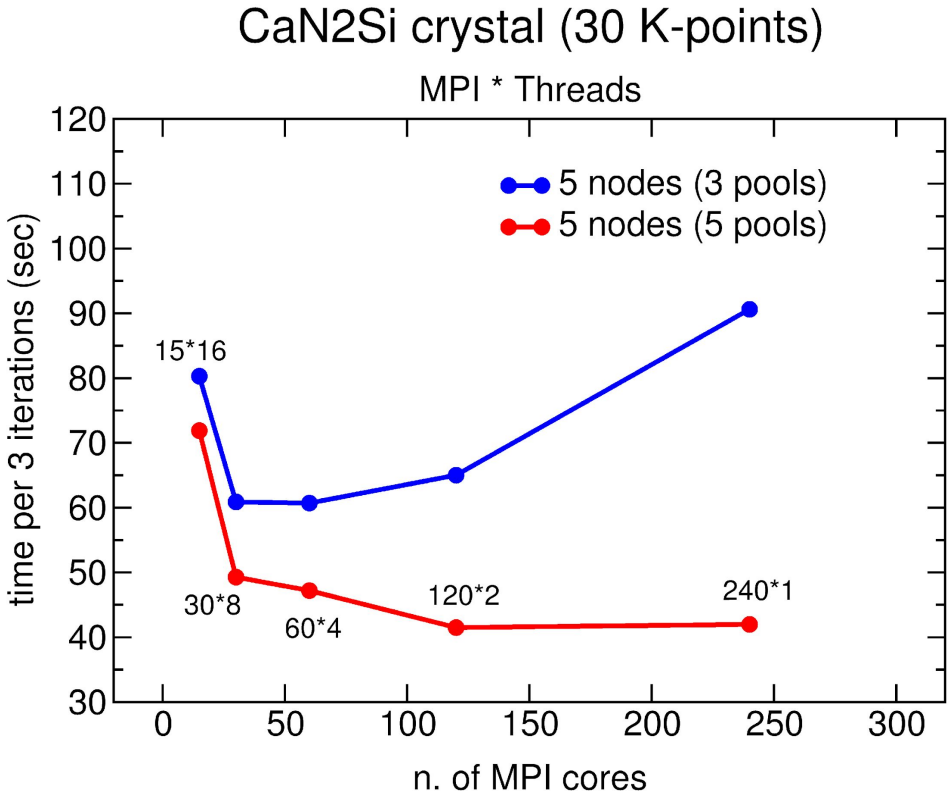
COMBINING with OPENMP





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

COMBINING with OPENMP



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

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 (https://wiki.u-gov.it/confluence/display/SCAIUS/UG3.2%3A+LEONARDO+UserGuide); 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***.
   What seems 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=boost_usr_prod
#SBATCH --qos=boost_qos_dbg
#SBATCH --ntasks-per-node= ...
#SBATCH --cpus-per-task= ...
#SBATCH --output=sysout.out
#SBATCH --error=syserr.err
#SBATCH --account=EUHPC_TD02_030
# # SBATCH --mail-user=YOUR_EMAIL - if you want

source ../environment-cpu.sh
export ESPRESSO_DIR=/leonardo_work/EUHPC_TD02_030/builds/qe7.2/bin

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

export OMP_NUM_THREADS= ...

mpirun -np ... ${ESPRESSO_DIR}/pw.x -npool 4 -i ${INDIR}/pw.CuO.scf.in > pw_run...mpix...omp.out
```

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

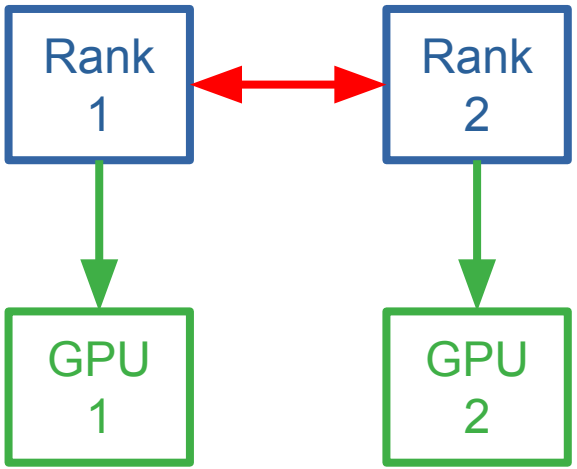
cuBLAS

3) The FFT libraries

cuFFT

MPI and GPUs

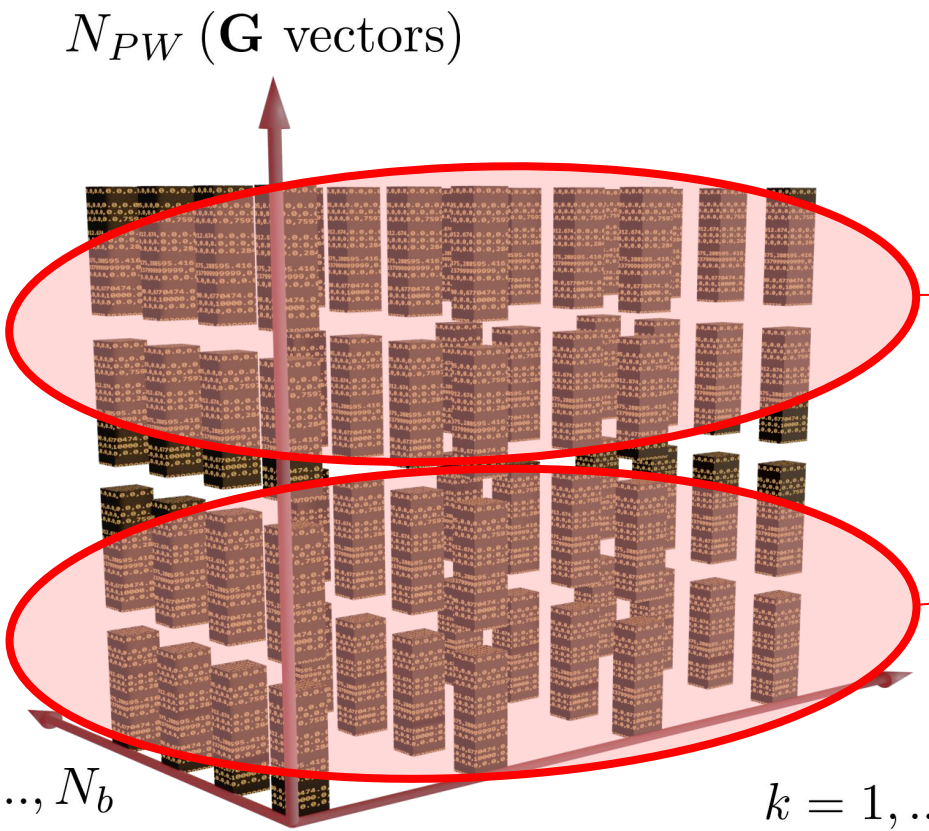
When we use GPUs, each MPI process off-loads the calculation to one GPU:

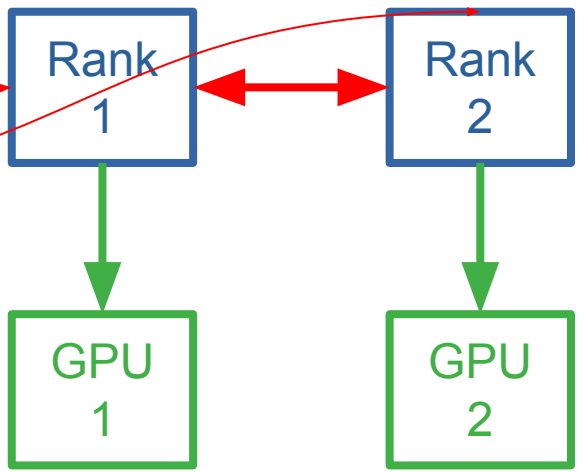


It is convenient to use one MPI per GPU

GPU parallelism
What is happening?

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

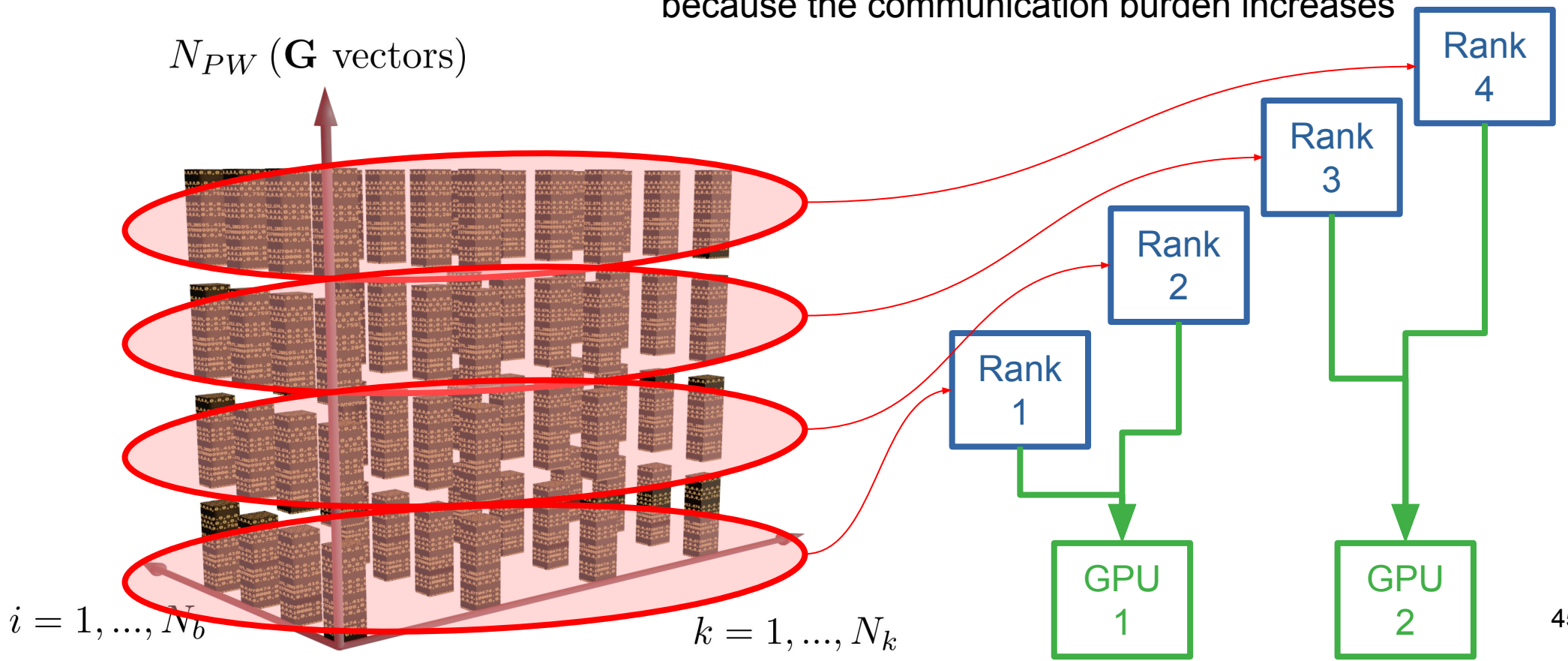




It is convenient to use one MPI per GPU

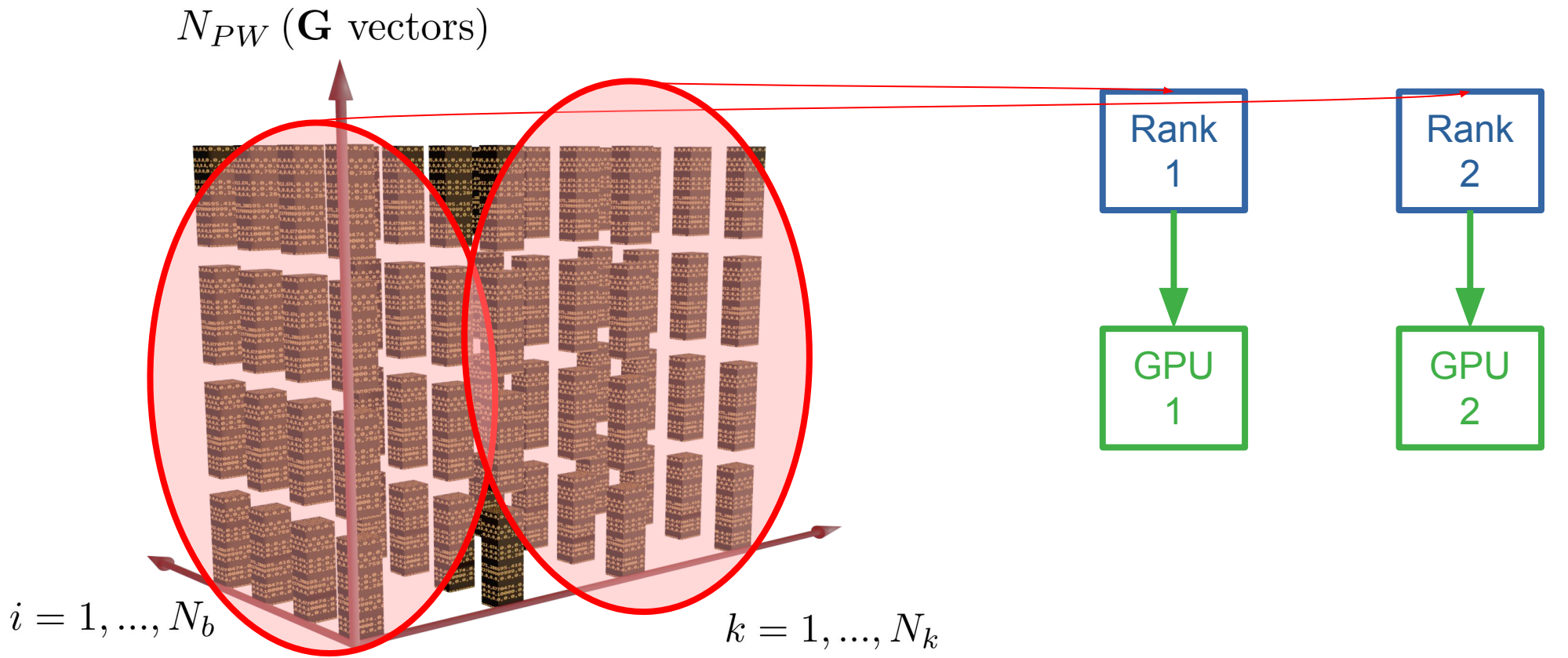
GPU parallelism
What is happening?

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



GPU parallelism
What is happening?

Again, using pools will improve communications



MPI and GPUs – JOB SCRIPT

```
#!/bin/bash
#SBATCH --job-name=USERjob
#SBATCH --nodes 1
#SBATCH --exclusive
#SBATCH --time=00:10:00
#SBATCH --partition=boost_usr_prod
#SBATCH --qos=boost_qos_dbg
#SBATCH --ntasks-per-node=4
#SBATCH --cpus-per-task=8
#SBATCH --gres=gpu:4
#SBATCH --output=sysout.out
#SBATCH --error=syserr.err
#SBATCH --account=EUHPC_TD02_030
# # SBATCH --mail-user=YOUR_EMAIL - if you want

module purge
module load profile/chem-phys
module load quantum-espresso

export EXDIR=${PWD}/..
export INDIR=${EXDIR}/inputs
export ESPRESSO_PSEUDO=${EXDIR}/../pseudo
export OMP_NUM_THREADS=8

for ip in 1 2 4 8
do
mpirun -np 4 pw.x -npool "$ip" -i ${INDIR}/pw.CnSnI3.in > pw.CnSnI3.gpu.${ip}pools.out
done
```

CPU vs GPUs

First, launch the GPU job as it is:

```
#SBATCH --ntasks-per-node = 4      # number of MPI per node
#SBATCH –cpus-per-task = 8         # number of HW threads per task
#SBATCH --gres = gpu:4

export OMP_NUM_THREADS = 8

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

CPU vs GPUs

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

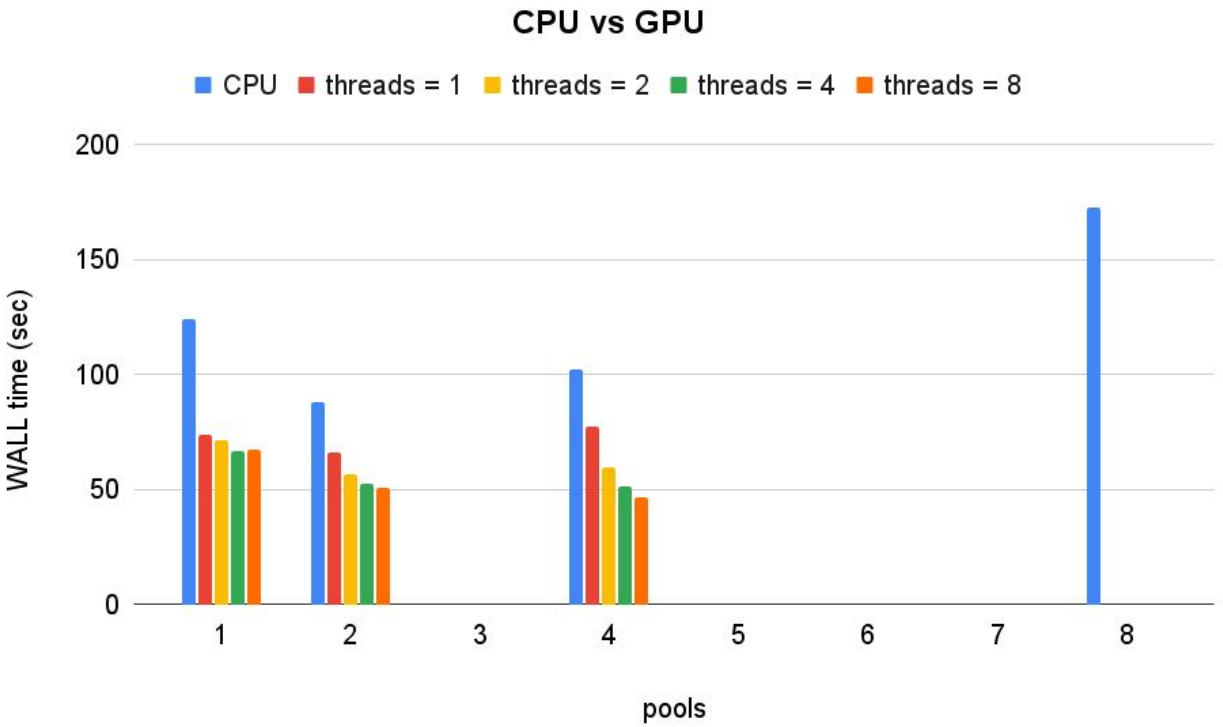
```
#SBATCH –ntasks-per-node=...      # number of MPI per node
#SBATCH –cpus-per-task=...        # number of HW threads per task

export OMP_NUM_THREADS=...

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

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.

MINIMAL RECIPE

1) Choose a number of ***MPI tasks*** depending on the system dimension (e.g. 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: - 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.

CPU and GPUs

Size (Ta2O5):

el = 544

Nat = 96

Ecut = NC/130/520 Ry

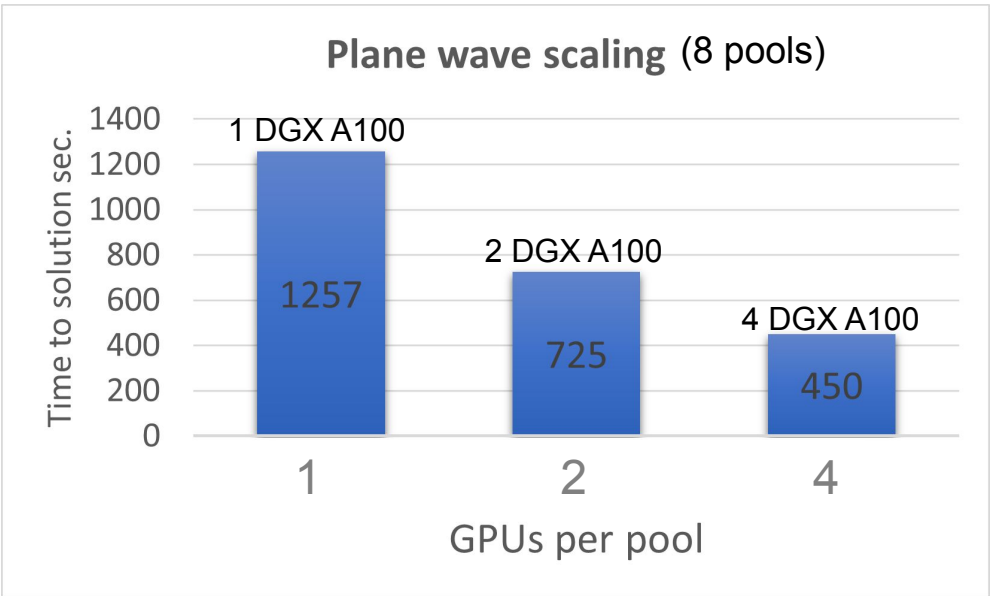
NPW = 477k

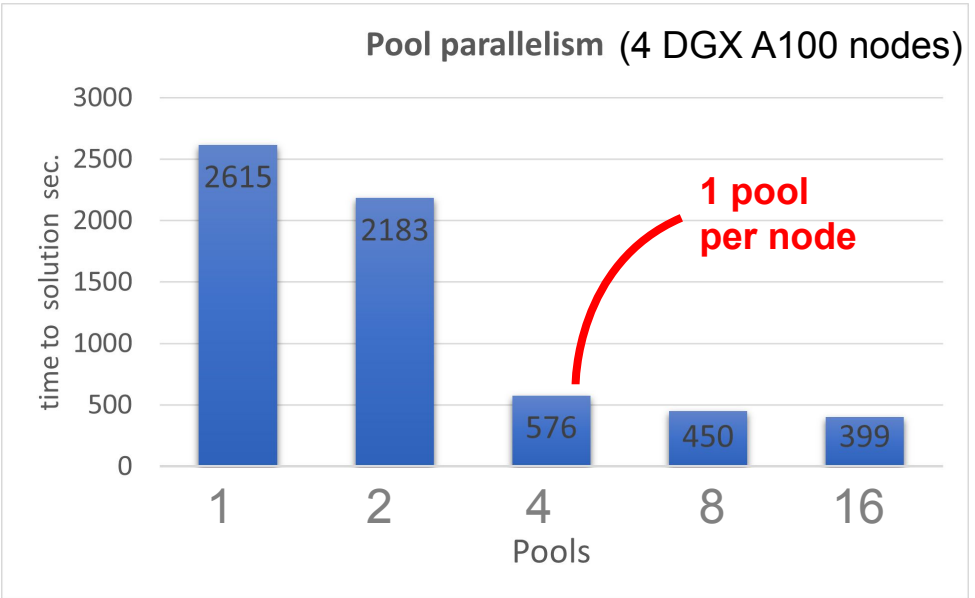
Nbnd = 326

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.

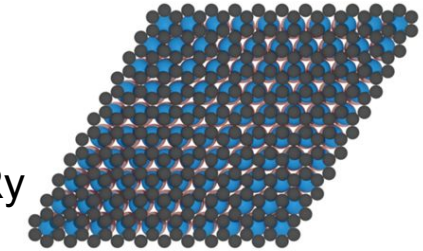


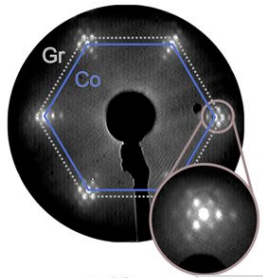


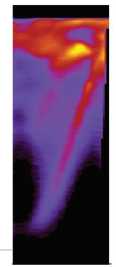
Taken from L. Stuber (NVIDIA), I. Carnimeo (SISSA), P. Delugas (SISSA), F. Spiga (NVIDIA) unpublished benchmarks.
All calculations on A100 kindly provided by NVIDIA Corp.

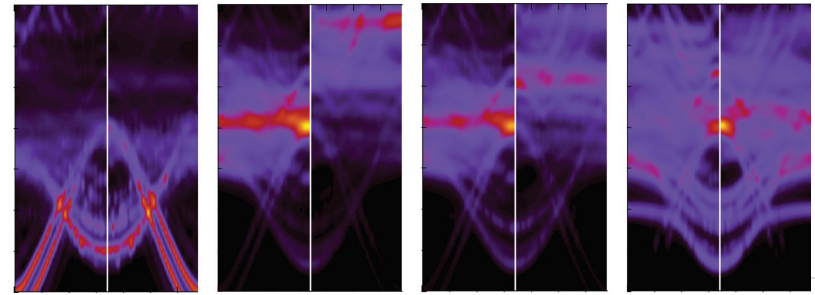
CPU and GPUs

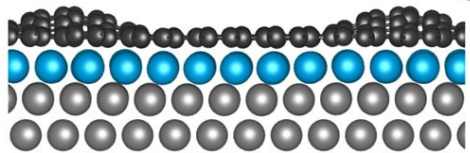
Size:
el = 4445
Nat = 605
Ecut = NC/75/300 Ry
NPW = 994k
Nbnd = 2688
Nks = 12



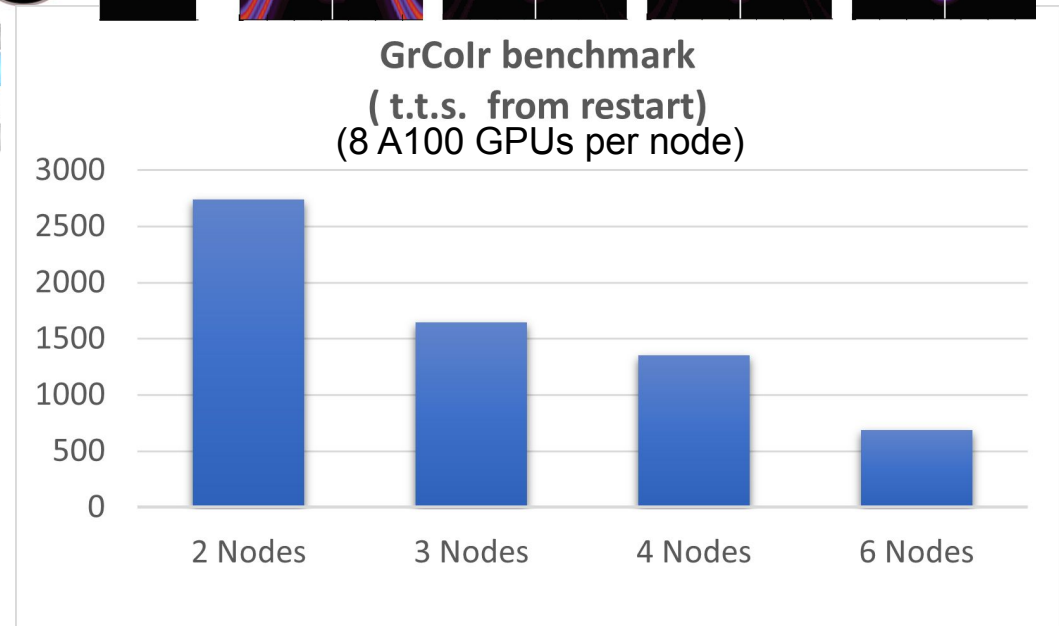








Good scaling performances have been found for large systems, by combining plane waves and pools parallelization strategies.



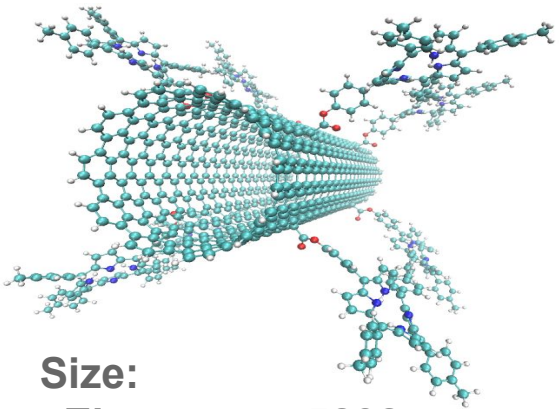
Appl. Phys. Lett. 118, 121602 (2021);
Nano Lett. (2018) 18, 2268−2273

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.

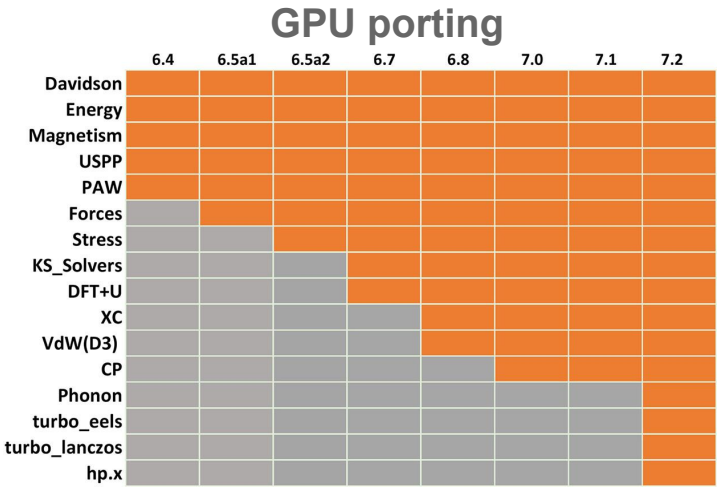
MPI =	16	48	32	96
NPOOLS =	4	6	4	12

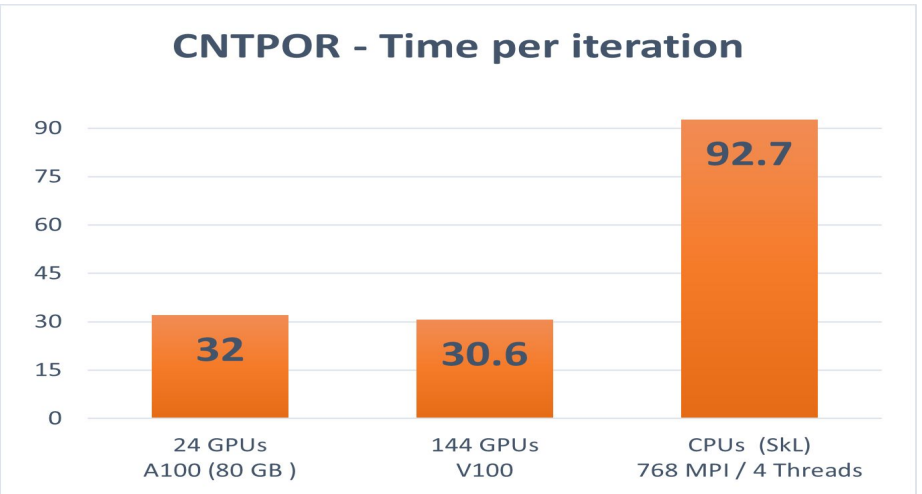
CPU and GPUs

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**



Size:
Electrons = 5232
Atoms = 1532





CPU and GPUs

Size:

el = 5232

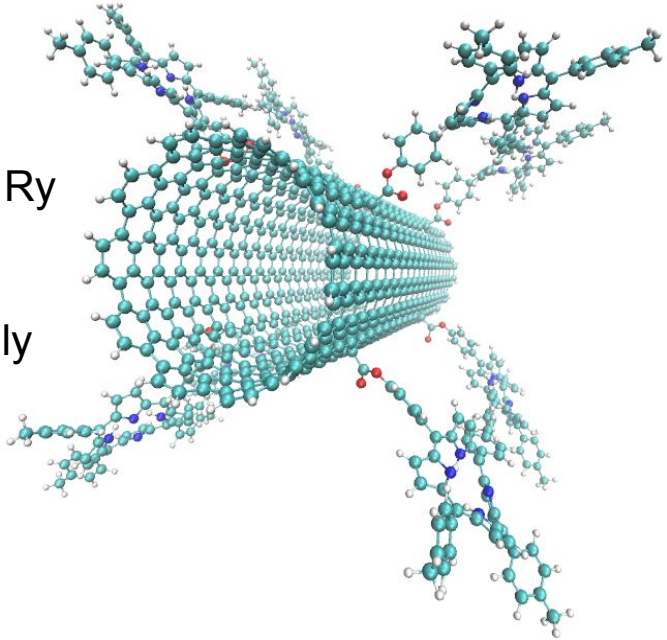
Nat = 1532

Ecut = US/25/200 Ry

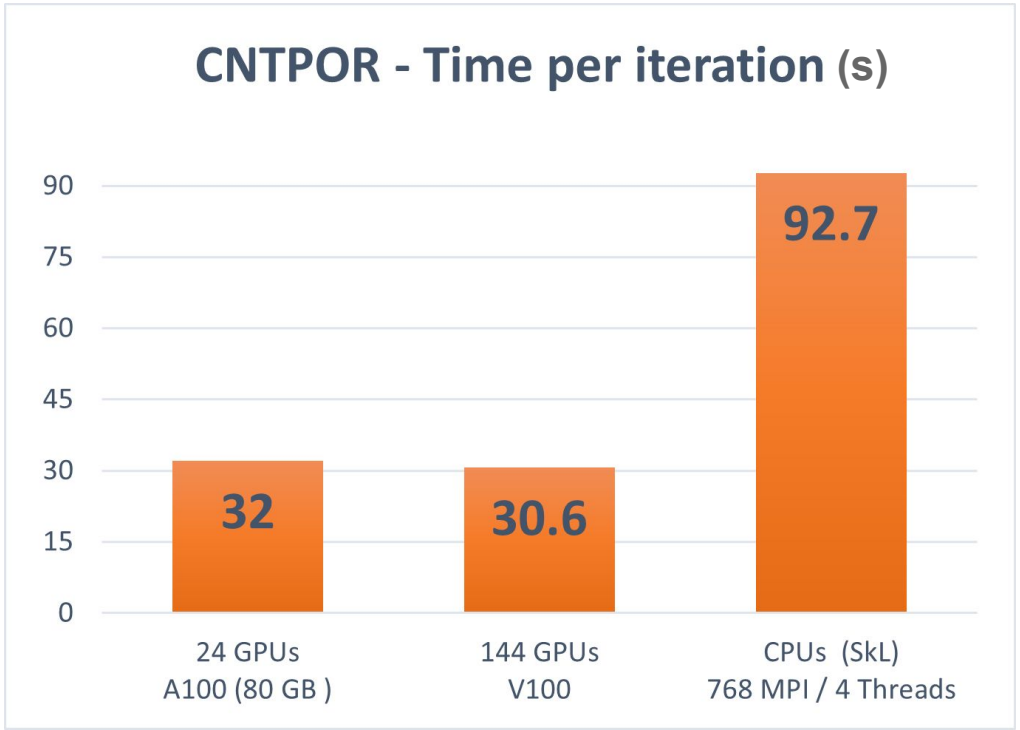
NPW = 27M

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.

Acknowledgement

Ivan Carnimeo

Fabrizio Ferrari Ruffino

Pietro Delugas

QUANTUM ESPRESSO FOUNDATION

QUANTUMESPRESSO

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

Hands-on session – Day 1

Thank you,
See you tomorrow

MAX

SISSA

CINECA

ENCCS
EuroCC National Competence Centre Sweden

PAUL SCHERRER INSTITUT
PSI

MARVEL
NATIONAL CENTRE OF COMPETENCE IN RESEARCH

Swiss National Science Foundation

COMPILERS AND LIBRARIES

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

1) The compilers

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)