QUANTUM ESPRESSO FOUNDATION

QUANTUMESPRESSO

MaX School on Advanced Materials and Molecular Modelling
with QUANTUM ESPRESSO

QE-2021: Hands-on session – Day-9

(Hands-on: QE on HPC and GPU systems)

Ivan Carnimeo, Pietro Bonfa'
Paolo Pegolo, Mandana Safari, Riccardo Bertossa

MAX

ICTP The Abdus Salam International Centre for Theoretical Physics

CINECA

上海大学 SHANGHAI UNIVERSITY

Jožef Stefan Institute Ljubljana, Slovenia

cecam

SISSA

Covered topics are:

* compilation of Quantum ESPRESSO for CPU and CPU architectures;
* optimisation of CPU-only runs,
* basic description of GPU acceleration,
* how to efficiently run calculations on GPU-accelerated architectures.

Exercise 1:** preparing QE (CPU version)

Exercise 2:** optimize CPU execution

Exercise 3:** (very) basic concepts about GPUs

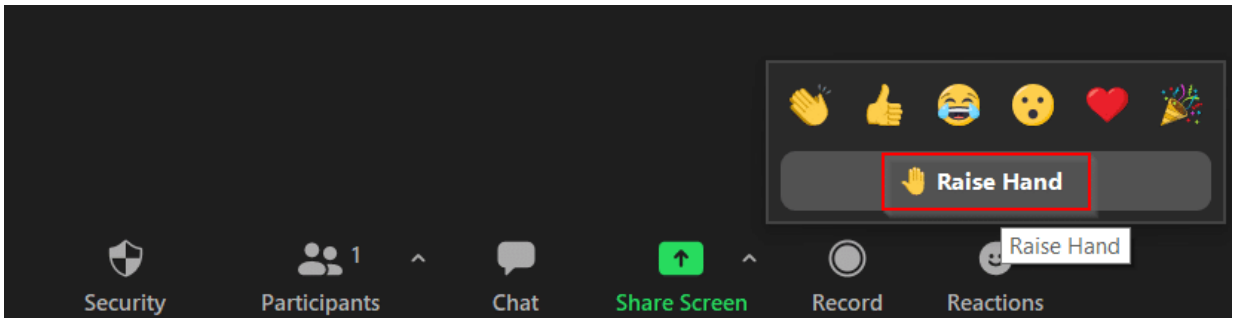
Exercise 4:** preparing QE (GPU version)

Exercise 5:** running with GPUs

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

ssh USER@login01-ext.m100.cineca.it

...when you are ready Raise Hand on Zoom

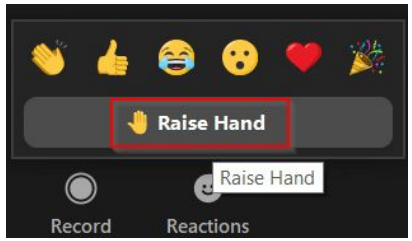


Then copy the repo to $CINECA_SCRATCH and move there:

```
cd materials-for-max-qe2021-online-school/
git pull
cp -r   Day-9/  $CINECA_SCRATCH
cd $CINECA_SCRATCH
cd Day-9/
pwd
```

Check that the result of pwd is

/m100_scratch/usertrain/USER/Day-9



Exercise 1: preparing QE (CPU version)

Exercise 1: preparing QE (CPU version)

Three things to keep in mind when installing QE:

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 1: preparing QE (CPU version)

We will first prepare an HPC ready installation of QE. This exercise will show how to compile QE and check for relevant libraries in the context of standard and accelerated systems.

Download the last release, extract it and rename it with the commands below:

cd exercise1.CPU-setup/

wget https://gitlab.com/QEF/q-e/-/archive/qe-6.7MaX-Release/q-e-qe-6.7MaX-Release.tar.bz2

tar xjf q-e-qe-6.7MaX-Release.tar.bz2

mv q-e-qe-6.7MaX-Release qe-cpu

cd qe-cpu

Exercise 1: preparing QE (CPU version)

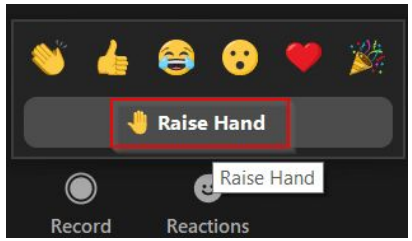
For the CPU version we will use hpc-sdk, SpectrumMPI, FFTW , which are a good combination for the OpenPower machines of Marconi100.

module purge
module load hpc-sdk/2020--binary spectrum_mpi/10.3.1--binary fftw/3.3.8--spectrum_mpi--10.3.1--binary

Configure QE with the following option, that will select nvfortran compilers from the hpc-sdk package and SpectrumMPI

./configure MPIF90=mpipgifort

Check that relevant libraries have been detected:
BLAS_LIBS=-lblas
LAPACK_LIBS=-L/cineca/prod/opt/compilers/hpc-sdk/2020/binary/Linux_ppc64le/2020/profilers/Nsight_Systems/host-linux-ppc64le -llapack -lblas
FFT_LIBS= -lfftw3



Exercise 1: preparing QE (CPU version)

We will only use pw.x for this hands-on. Let's compile it with the command

make -j4 pw

Now enjoy an espresso while you wait 3 minutes or so...

Check that your installation works by running in parallel a quick random test from the test-suite

mpirun -np 2 PW/src/pw.x -inp test-suite/pw_dft/dft1.in

You will find an error on pseudopotentials, but it is fine because it means that the installation works.

Exercise 2: optimize CPU execution

Job script

```
#!/bin/bash
#SBATCH --nodes=1              # number of nodes
#SBATCH --ntasks-per-node=16   # number of MPI per node
#SBATCH --cpus-per-task=4      # number of HW threads per task
#SBATCH --mem=230000MB
#SBATCH --time 00:30:00        # format: HH:MM:SS
#SBATCH --reservation=s_tra_qe
#SBATCH -A tra21_qe
#SBATCH -p m100_usr_prod
#SBATCH -J qeschool

module load    hpc-sdk/2020--binary    spectrum_mpi/10.3.1--binary   fftw/3.3.8--spectrum_mpi--10.3.1--binary

export QE_ROOT=/m100_scratch/usertrain/a08trd1f/Day-9/exercise1.CPU-setup/qe-cpu/

export PW=$QE_ROOT/bin/pw.x

# This sets OpenMP parallelism, in this case we do a pure MPI
export OMP_NUM_THREADS=1

# Run pw.x with default options for npool and ndiag
mpirun  ${PW} -npool 1 -ndiag 1 -inp pw.CuO.scf.in | tee pw.CuO.scf.npool01.ndiag01.log
```

Exercise 2: optimize CPU execution

Pool parallelism

First submit the job “as is”, with npool set to 1

mpirun ${PW} -npool 1 -ndiag 1 -inp pw.CuO.scf.in | tee pw.CuO.scf.npool01.ndiag01.log

with the command:

sbatch job.sh

Other useful commands:

squeue -u USER
scancel JOBID

Exercise 2: optimize CPU execution

Pool parallelism

Then open the job-script file (job.sh) and change the number of pools to be used -npool X, with X={2,4,8}. Don't forget to rename the output file as well.

```
mpirun  ${PW} -npool 2 -ndiag 1 -inp pw.CuO.scf.in | tee pw.CuO.scf.npool02.ndiag01.log
mpirun  ${PW} -npool 4 -ndiag 1 -inp pw.CuO.scf.in | tee pw.CuO.scf.npool04.ndiag01.log
mpirun  ${PW} -npool 8 -ndiag 1 -inp pw.CuO.scf.in | tee pw.CuO.scf.npool08.ndiag01.log
```

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

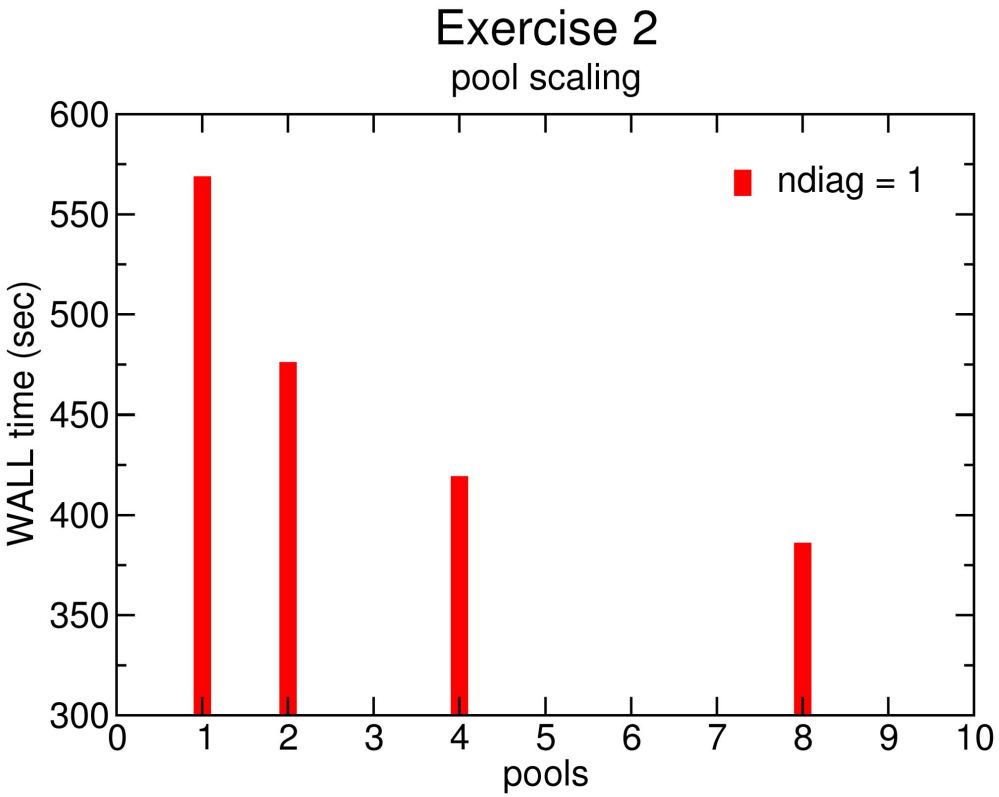
```
PWSCF      :  5m53.84s CPU   5m58.18s WALL
```

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.

Exercise 2: optimize CPU execution

Pool parallelism

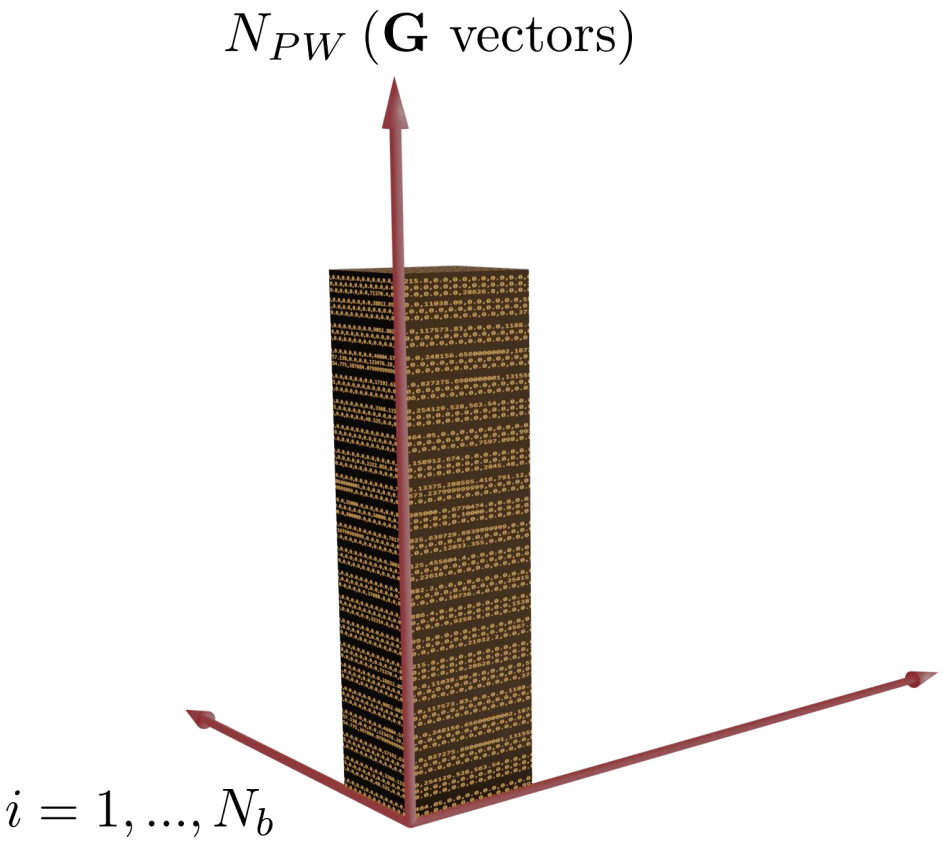
You should be able to produce a plot similar to this one:



Congrats! With the same computational resources, the time to solution is reduced by 1/3!

Exercise 2: optimize CPU execution

Pool parallelism
What is happening?

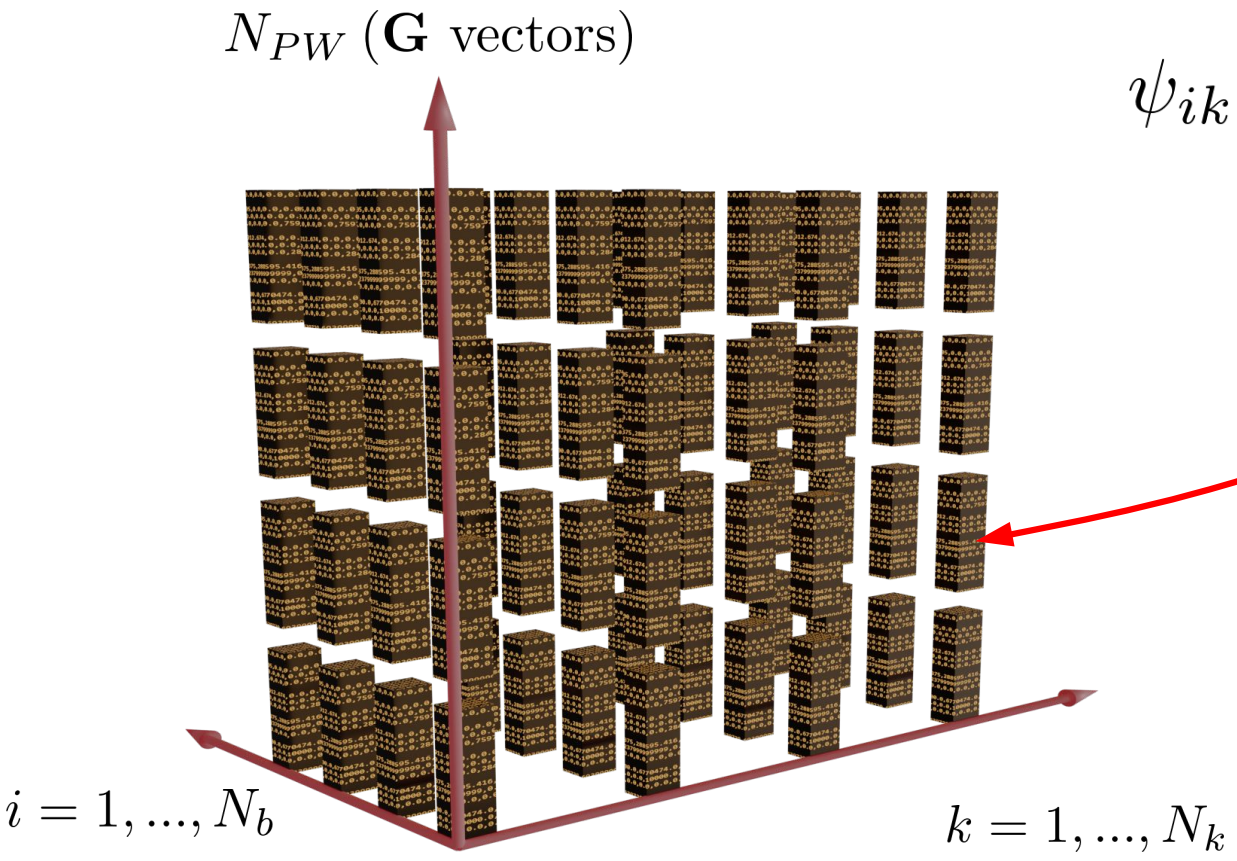


Remember the distribution of the wavefunction

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

Exercise 2: optimize CPU execution

Pool parallelism
What is happening?



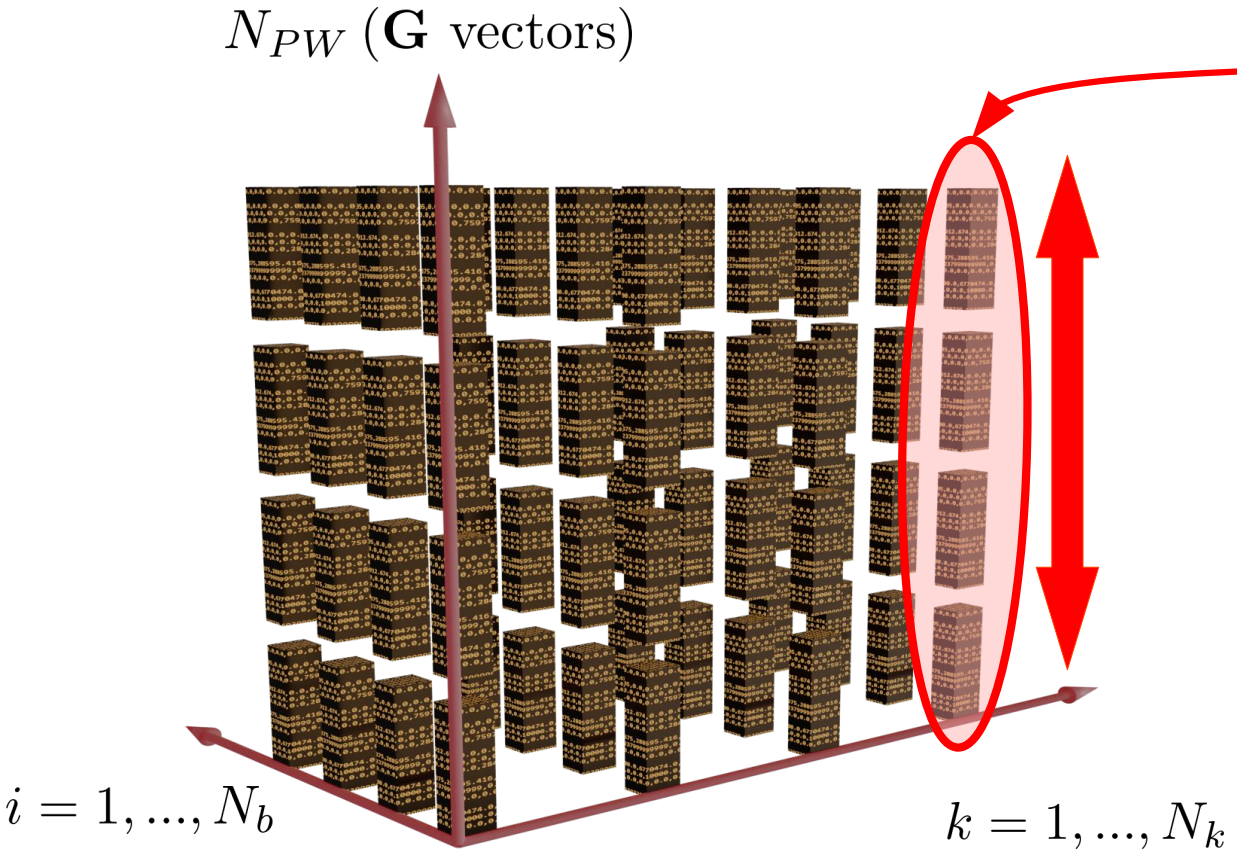
Remember the distribution of the wavefunction

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

Exercise 2: optimize CPU execution

Pool parallelism
What is happening?

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

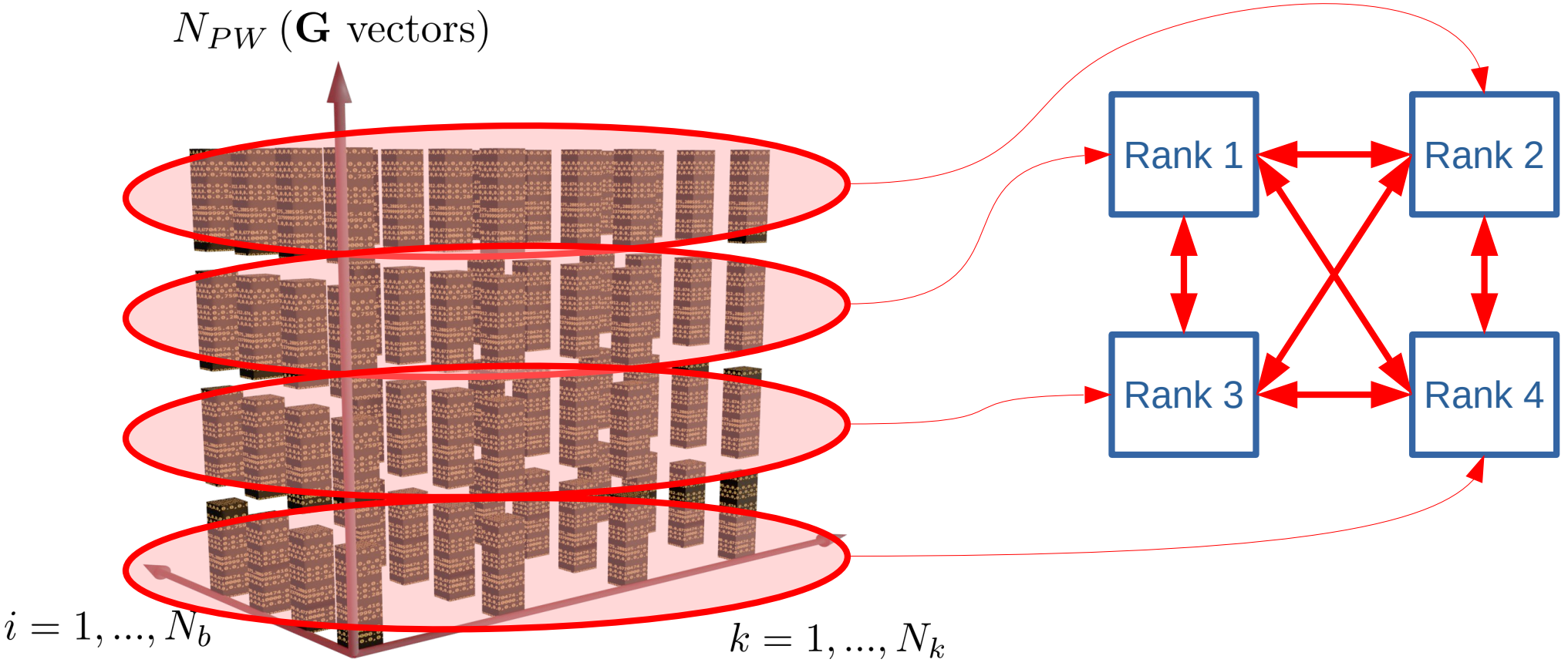


Since operators are usually applied to single orbitals, most of the communications (e.g. mp_sum) are usually done along the NPW dimension

Exercise 2: optimize CPU execution

Pool parallelism
What is happening?

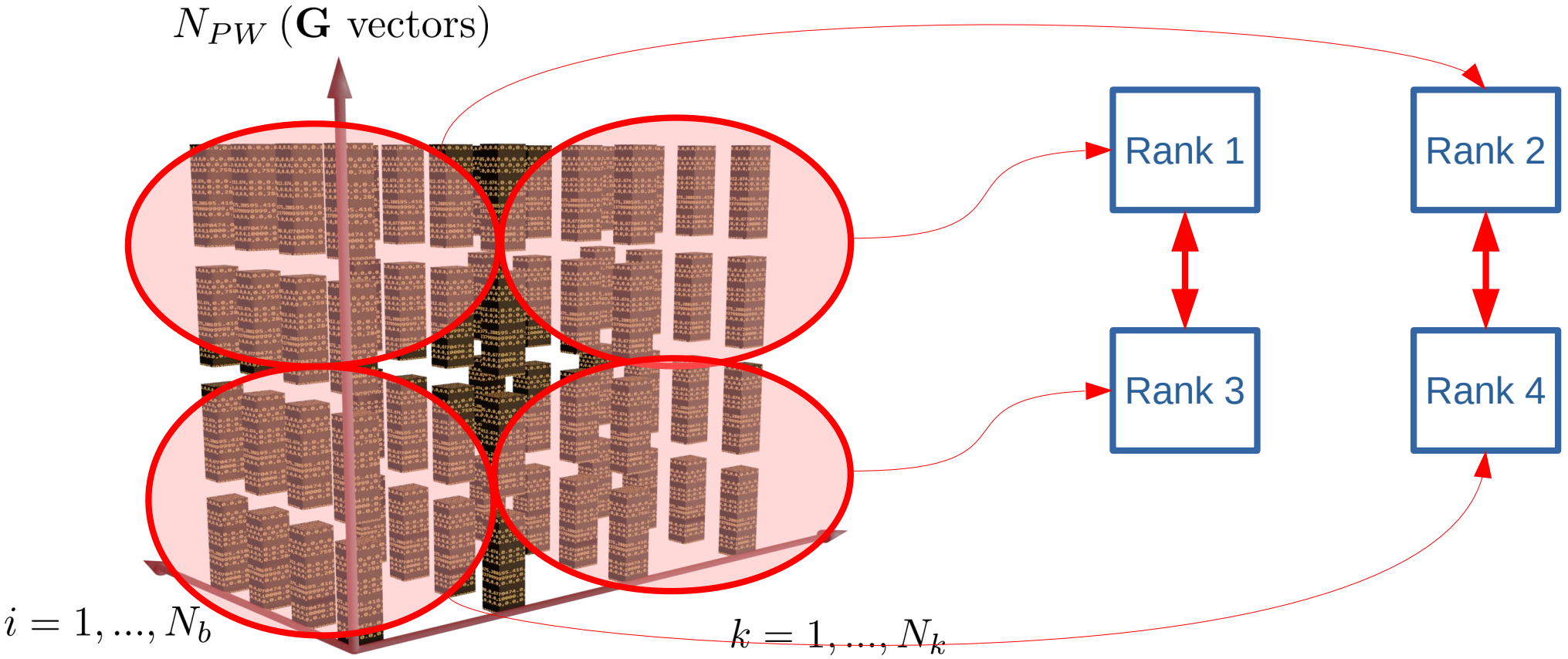
When we parallelize over PW, all processes need to communicate with each other



Exercise 2: optimize CPU execution

Pool parallelism
What is happening?

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



Exercise 2: optimize CPU execution

Parallel diagonalization

In this second part 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 ${PW} -npool 4 -ndiag 4 -inp pw.CuO.scf.in | tee pw.CuO.scf.npool04.ndiag04.log

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

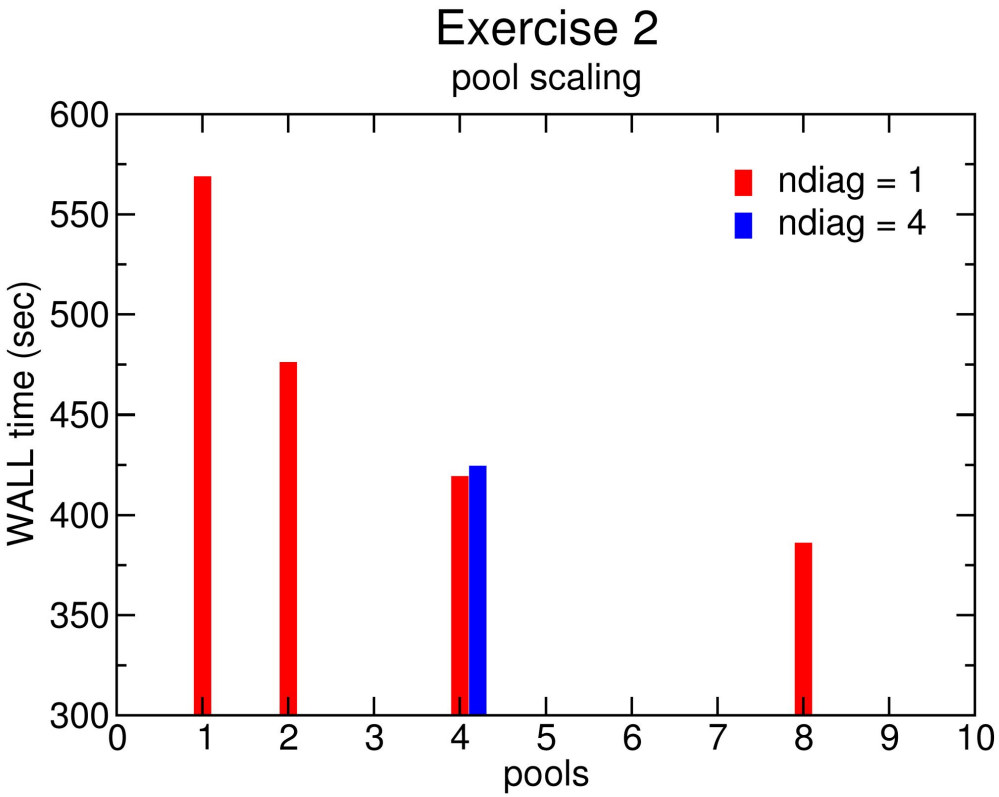
Subspace diagonalization in iterative solution of the eigenvalue problem:
one sub-group per band group will be used custom distributed-memory algorithm (size of sub-group: 2* 2 procs)

Check the time to solution

Exercise 2: optimize CPU execution

Parallel diagonalization

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



Exercise 2: optimize CPU execution

Please consider that:

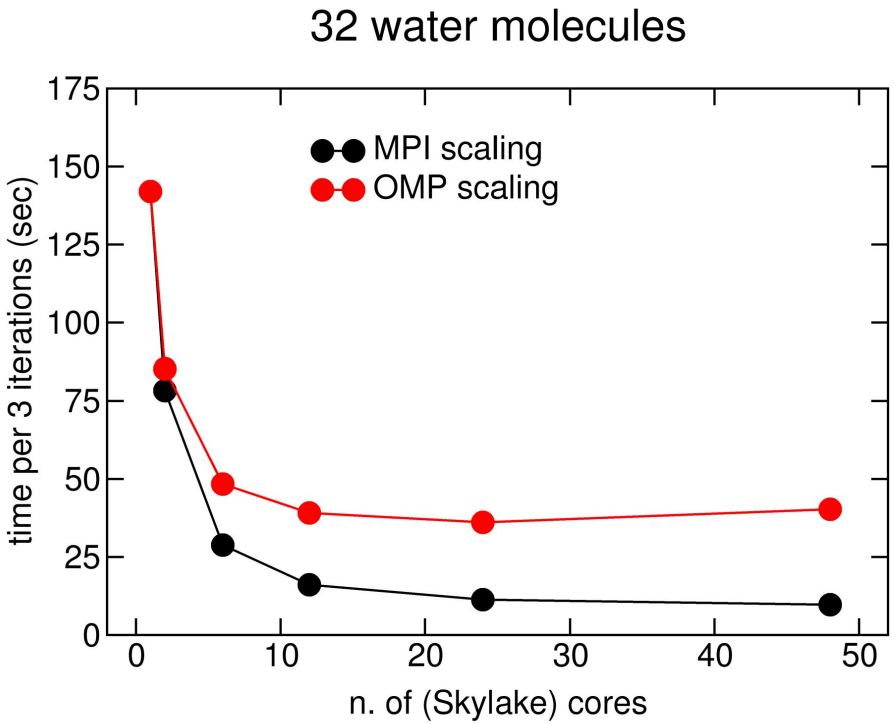
1) **pool parallelism can be much more effective than this**, especially when the system size is larger and calculations are distributed among multiple nodes, since it can strongly reduce the slow inter-node communications;

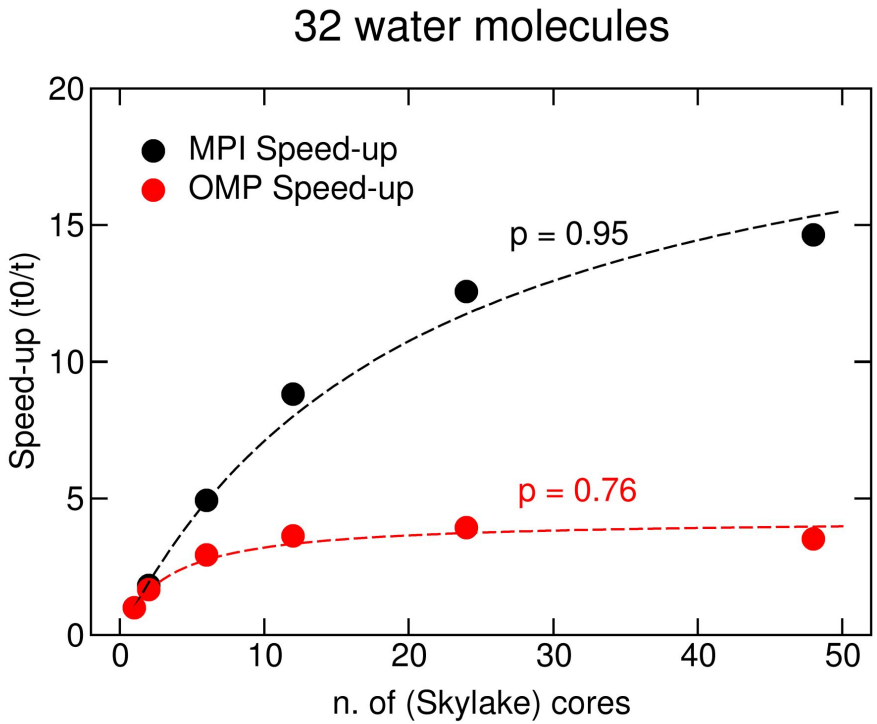
2) the **eigenvalue problem** is too small in this case to take fully advantage of parallel diagonalization;

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.

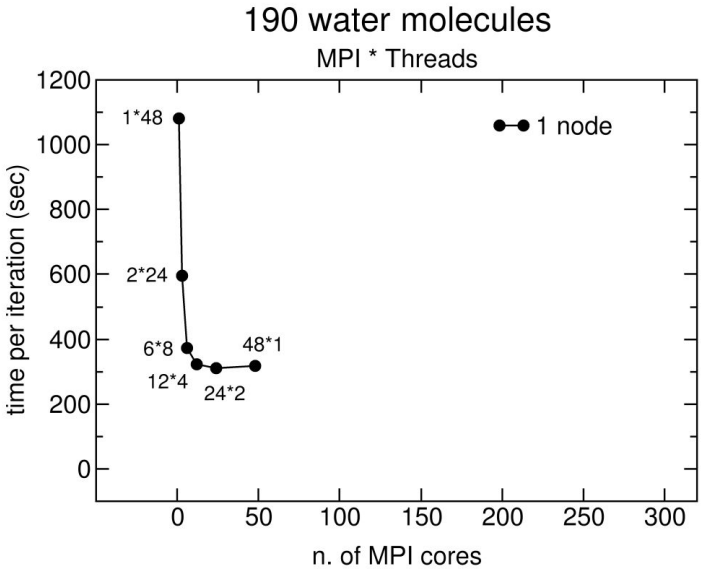
Exercise 2: more on CPU execution...





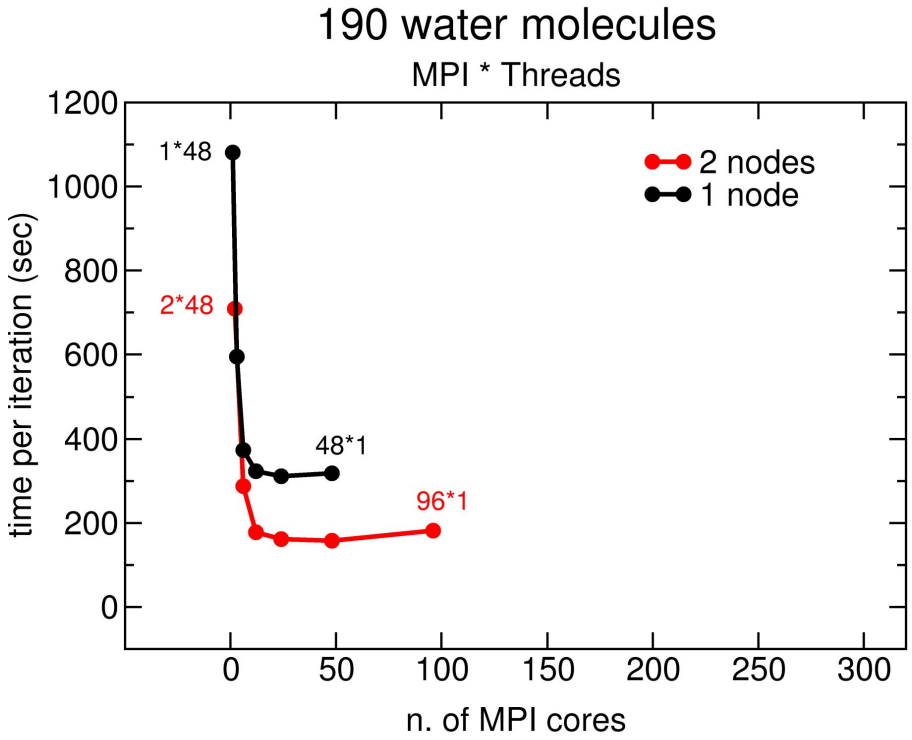
Scaling (Amdahl Law) for Quantum ESPRESSO code for both MPI and OpenMP portions of the code.
OMP parallelization is usually less efficient than MPI for QE, but involves less communications

Exercise 2: more on CPU execution...



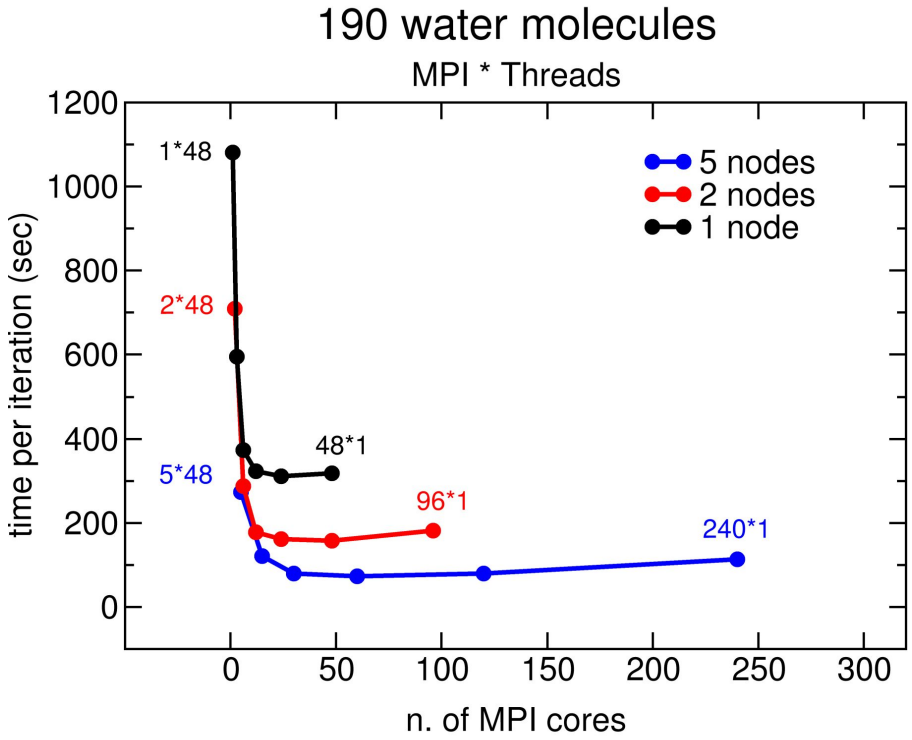
MPI and OMP threads can be combined to better exploit computational resources. OMP parallelization is usually less efficient than MPI for QE, but involves less communications

Exercise 2: more on CPU execution...



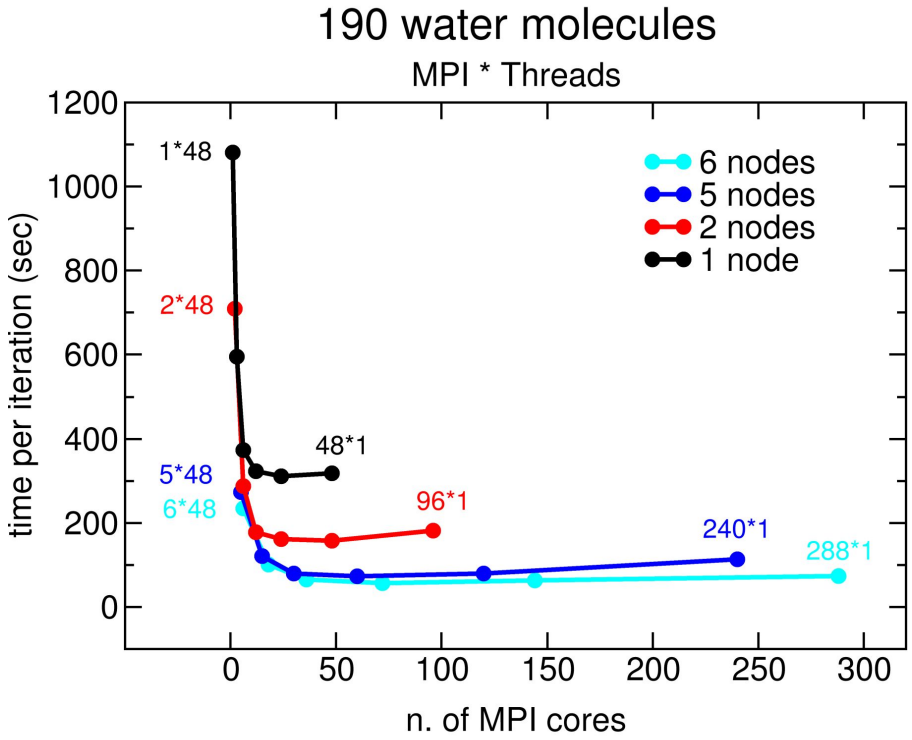
MPI and OMP threads can be combined to better exploit computational resources. OMP parallelization is usually less efficient than MPI for QE, but involves less communications

Exercise 2: more on CPU execution...



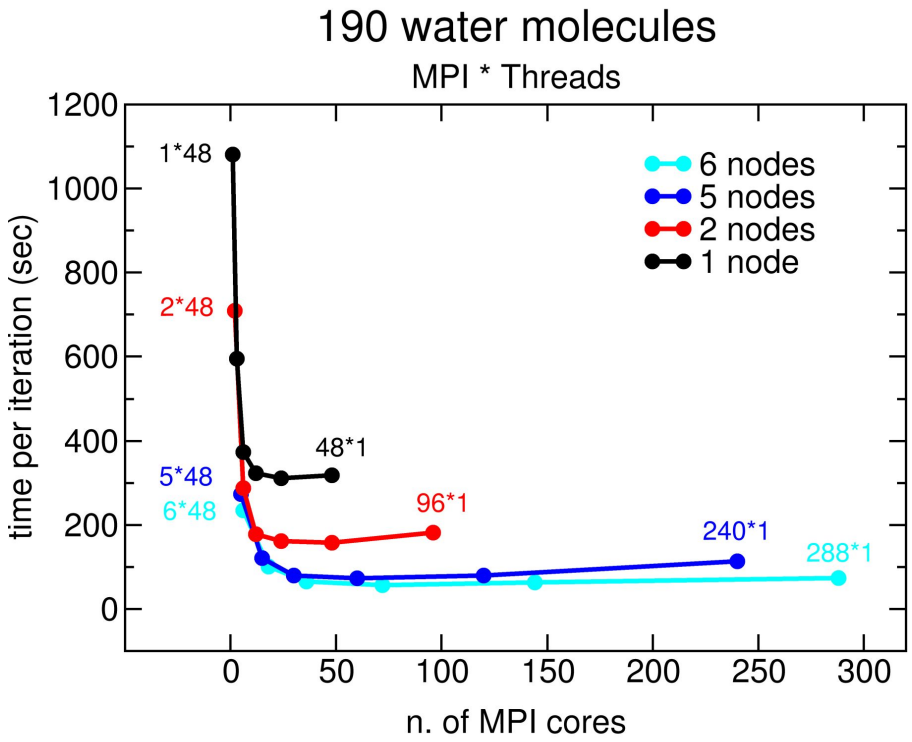
MPI and OMP threads can be combined to better exploit computational resources. OMP parallelization is usually less efficient than MPI for QE, but involves less communications

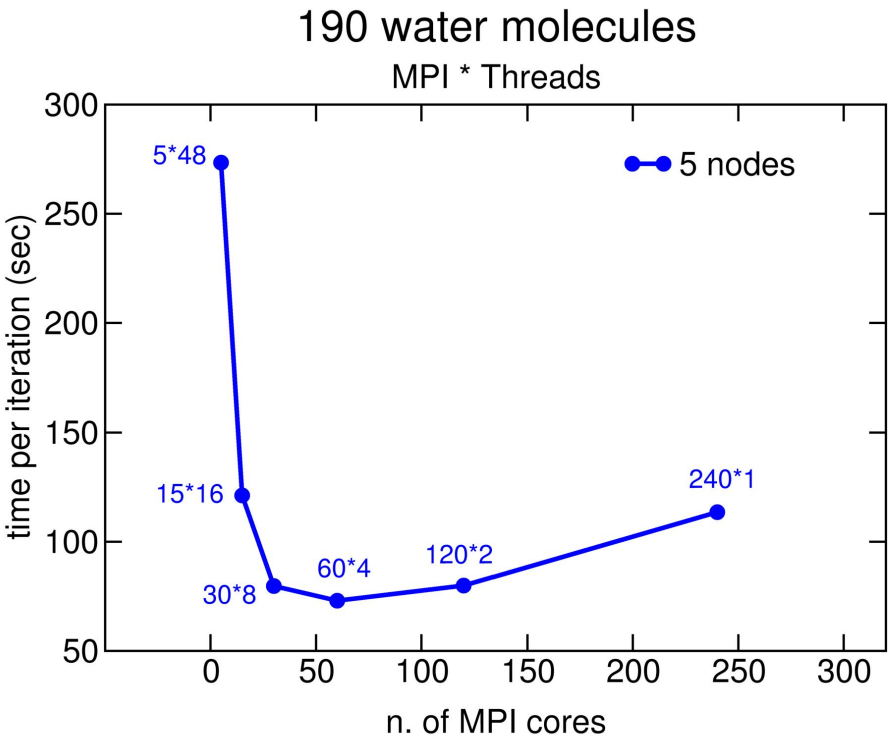
Exercise 2: more on CPU execution...



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

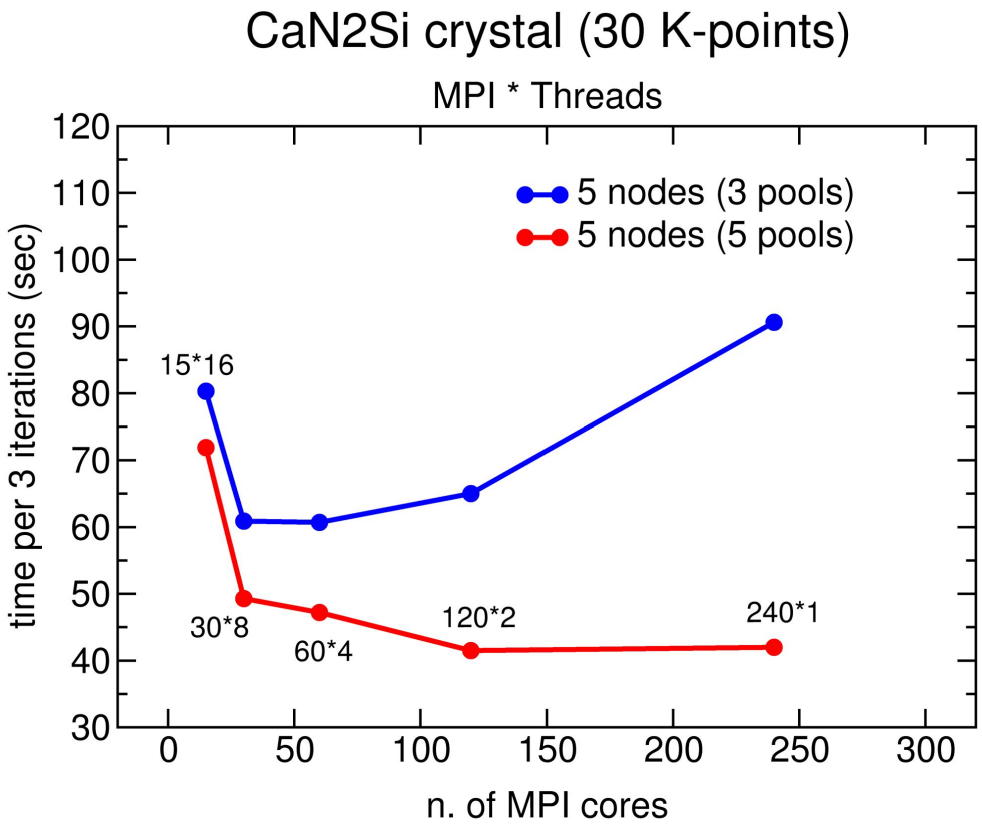
Exercise 2: more on CPU execution...





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

Exercise 2: more on CPU execution...



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

Exercise 3: (very) basic concepts about GPUs

Three things to keep in mind when installing QE:

1) The compiler

nvfortran

2) The linear algebra libraries

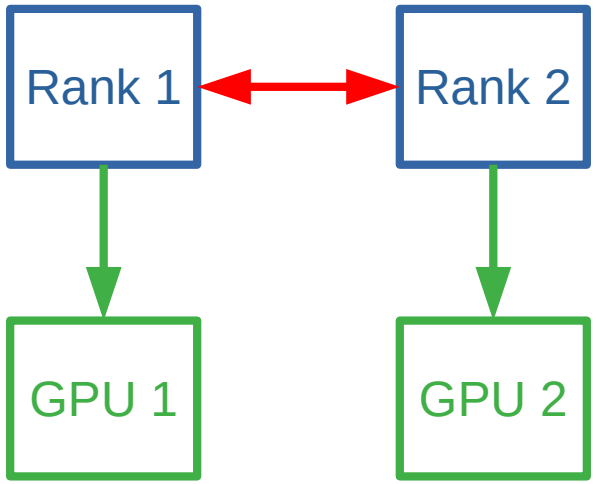
cuBLAS

3) The FFT libraries

cuFFT

Exercise 5: running with GPUs

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

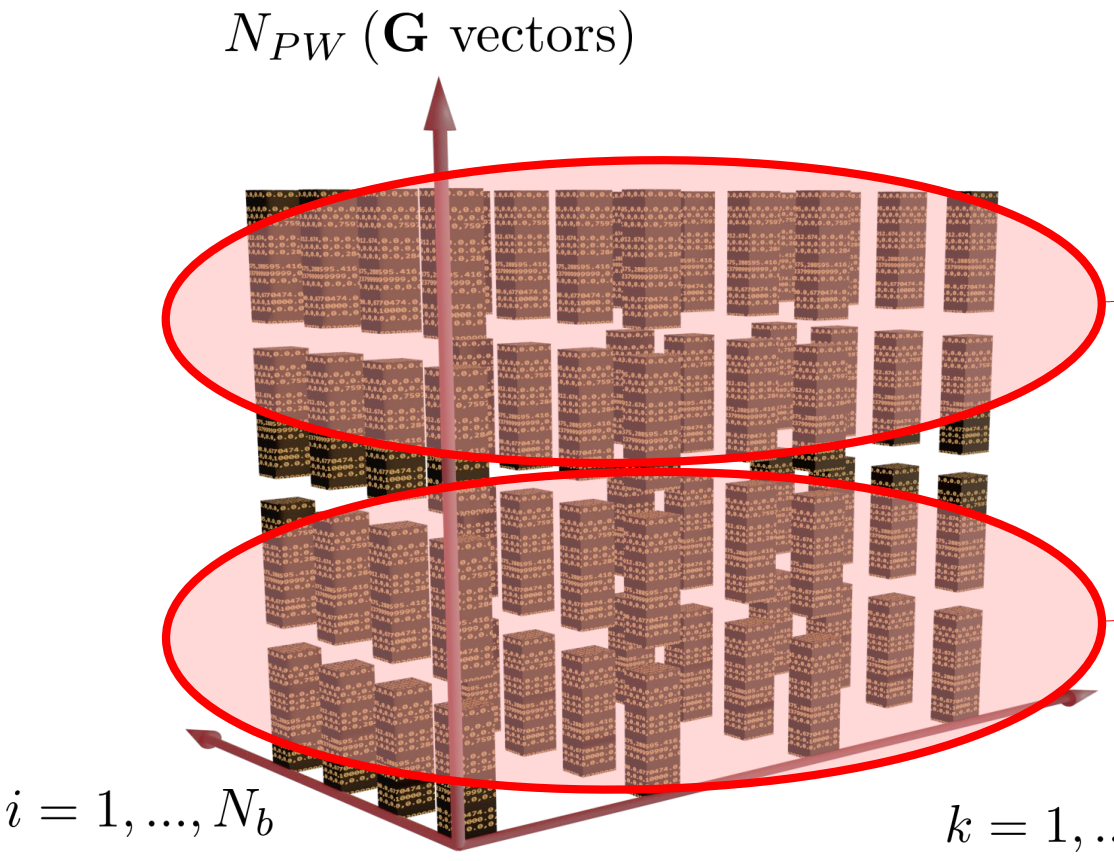


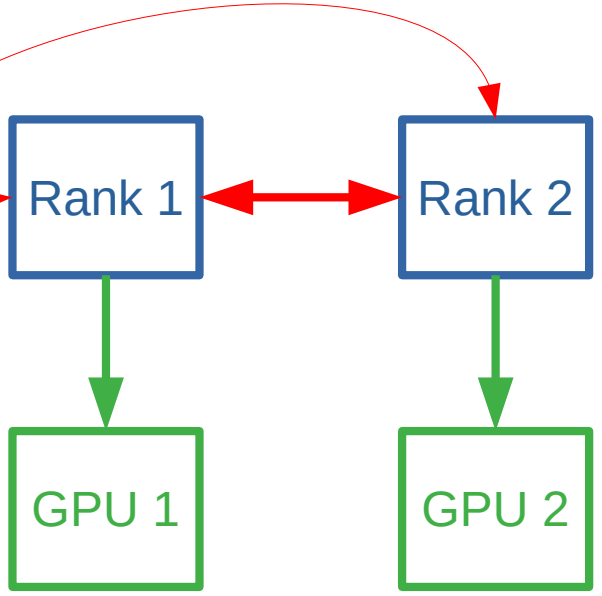
It is convenient to use one MPI per GPU

Exercise 5: running with GPUs

GPU parallelism
What is happening?

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



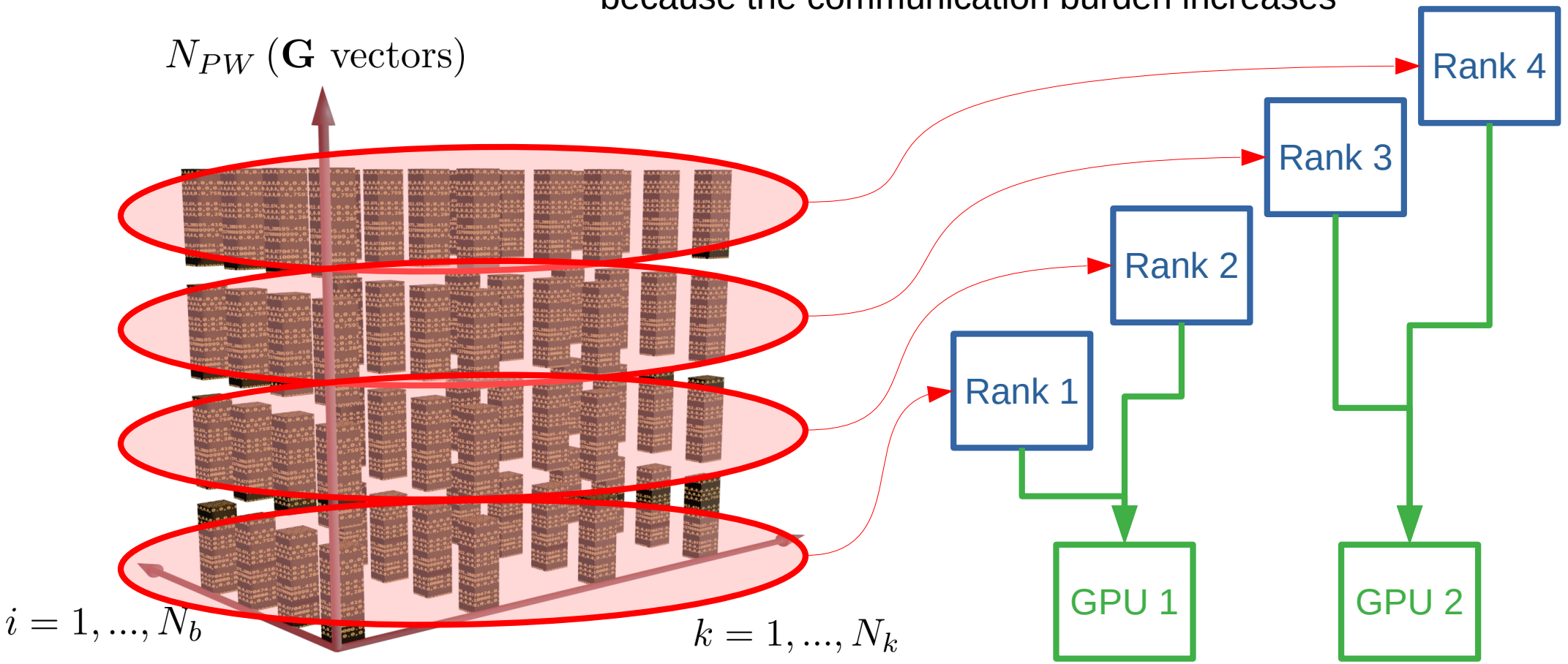


It is convenient to use one MPI per GPU

Exercise 5: running with 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 3: (very) basic concepts about GPUs

As a reference, for a matrix size of 8192, the times required by a DGEMM on m100 cluster should be something around:

```
code_cpu.x
  Full time:       66.449
  Product time:    63.170
code_gpu.x
  Full time:       0.785
  Product time:    0.167
code_mix.x
  Full time:       4.236
  Product time:    0.365
```

You can find the related source files (code_cpu.f90, code_gpu.f90, code_mix.f90) in the folder of exercise3

Exercise 4: preparing QE (GPU version)

Exercise 4: preparing QE (GPU version)

Download the last release of the GPU accelerated version of QE

cd $CINECA_SCRATCH

cd Day-9/exercise4.GPU-setup/

wget https://gitlab.com/QEF/q-e-gpu/-/archive/qe-gpu-6.7/q-e-gpu-qe-gpu-6.7.tar.bz2

tar xjf q-e-gpu-qe-gpu-6.7.tar.bz2

mv q-e-gpu-qe-gpu-6.7 qe-gpu

cd qe-gpu

Exercise 4: preparing QE (GPU version)

For the GPU version you must load the CUDA, together with the HPC-SDK package

```
module purge
module load hpc-sdk/2020--binary spectrum_mpi/10.3.1--binary fftw/3.3.8--spectrum_mpi--10.3.1--binary cuda/11.0
```

You must also specify the cuda version when launching the configure script

```
./configure MPIF90=mpipgifort --enable-openmp --with-cuda=$CUDA_HOME --with-cuda-runtime=11.0 --with-cuda-cc=70
```

Check

```
setting DFLAGS... -D__PGI -D__CUDA -D__USE_CUSOLVER -D__FFTW -D__MPI
[...]
  BLAS_LIBS=-lblas
  LAPACK_LIBS=-L/cineca/prod/opt/compilers/hpc-sdk/2020/binary/Linux_ppc64le/2020/profilers/Nsight_Systems/host-linux-ppc64le -llapack -lblas
  FFT_LIBS=
```

Exercise 4: preparing QE (GPU version)

Compile again the code

```
make -j4 pw
```

Check that your installation works by running in parallel a quick random test from the test-suite

```
mpirun -np 2 PW/src/pw.x   -inp test-suite/pw_dft/dft1.in
```

You will find an error on pseudopotentials, but it is fine because it means that the installation works.

Exercise 5: running with GPUs

Job script (GPU)

```
#!/bin/bash
#SBATCH --ntasks-per-node=2     # number of MPI per node
#SBATCH --ntasks-per-socket=2   # number of MPI per socket
#SBATCH --cpus-per-task=8       # number of HW threads
#SBATCH --gres=gpu:2            # number of gpus per node
#SBATCH --mem=230000MB
#SBATCH --time 00:10:00         # format: HH:MM:SS
#SBATCH --reservation=s_tra_qe
#SBATCH -A tra21_qe
#SBATCH -p m100_usr_prod
#SBATCH -J qeschool

module load hpc-sdk/2020--binary spectrum_mpi/10.3.1--binary  fftw/3.3.8--spectrum_mpi--10.3.1--binary  cuda/11.0

export QE_ROOT=/m100_scratch/usertrain/a08trd1f/Day-9/exercise4.GPU-setup/qe-gpu/
export PW=$QE_ROOT/bin/pw.x
export OMP_NUM_THREADS=1 # This sets OpenMP parallelism

# Run pw.x with default options for npool and ndiag
mpirun  ${PW} -npool 1 -ndiag 1 -inp pw.CuO.scf.in | tee pw.CuO.scf.npool01.ndiag01.log
```

Exercise 5: running with GPUs

First launch the job as is.

Then try to further improve the performance with OpenMP:

export OMP_NUM_THREADS=X (X=2, 4, 8)

with pool parallelism:

mpirun ${PW} -npool 2 -ndiag 1 -inp pw.CuO.scf.in | tee pw.CuO.scf.npool02.ndiag01.log

Exercise 5: running with GPUs

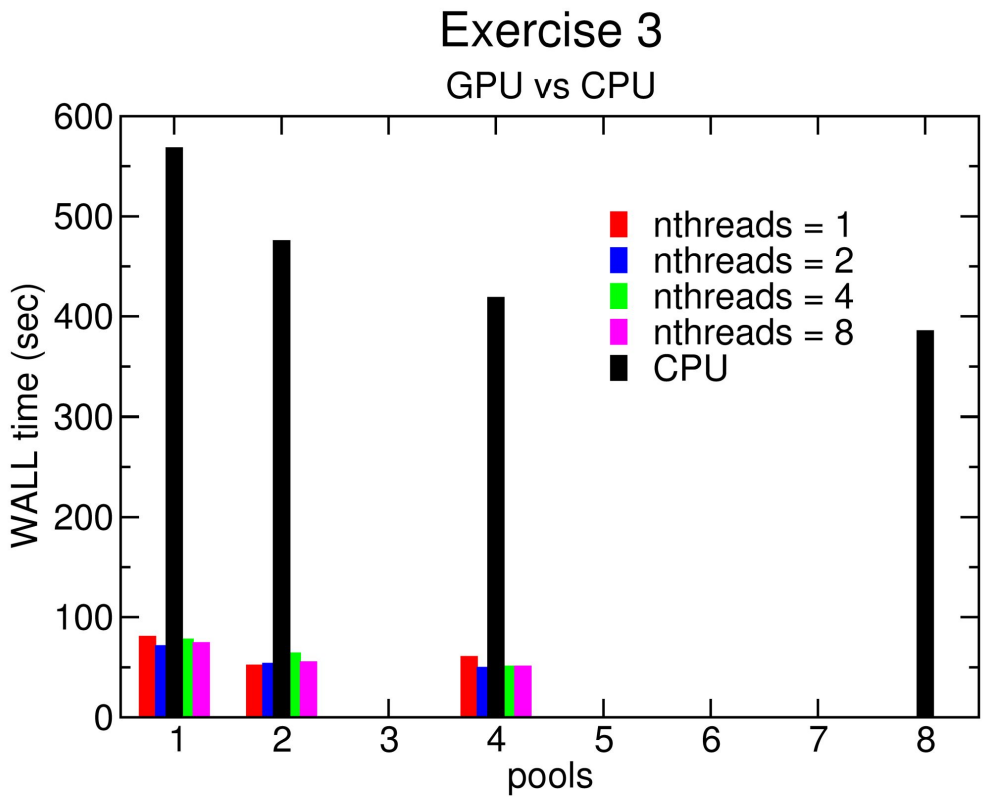
For small inputs, one can possibly obtain some additional performance by oversubscribing the GPU.

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

mpirun  ${PW} -npool 4 -ndiag 1 -inp pw.CuO.scf.in | tee pw.CuO.scf.npool04.ndiag01.log
```

Exercise 5: running with GPUs

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

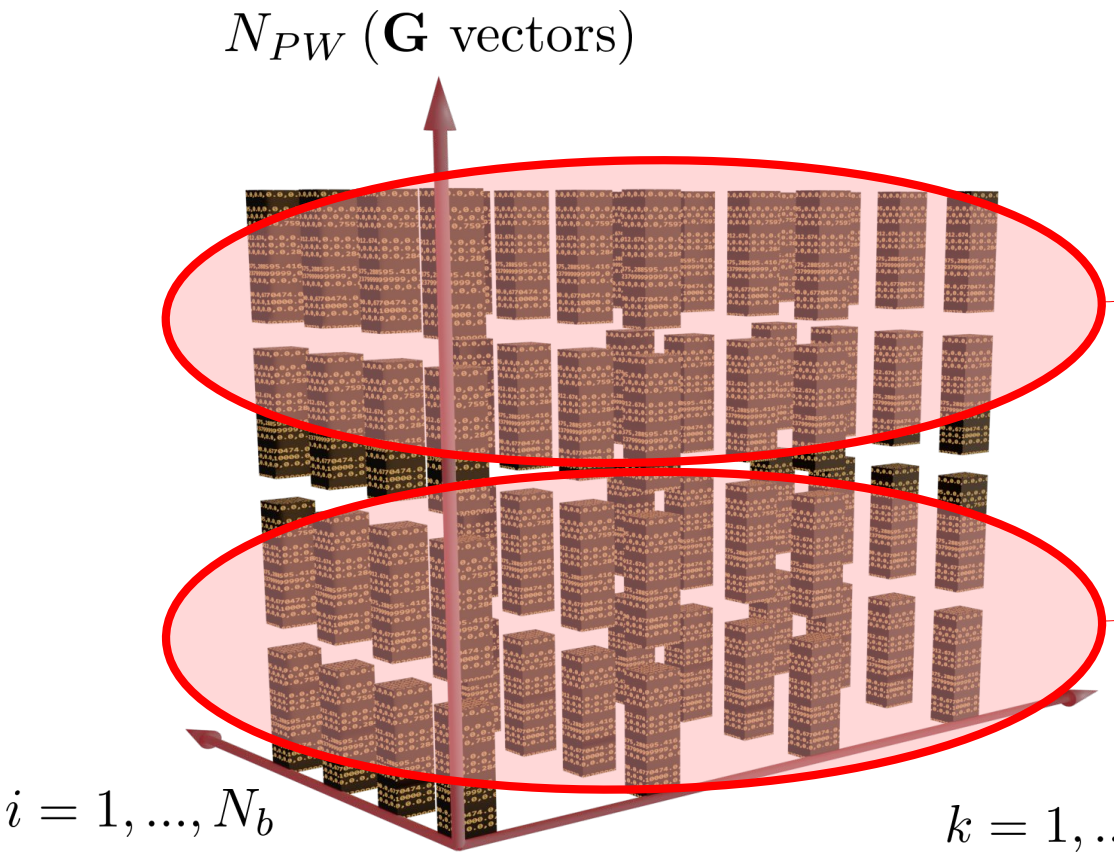


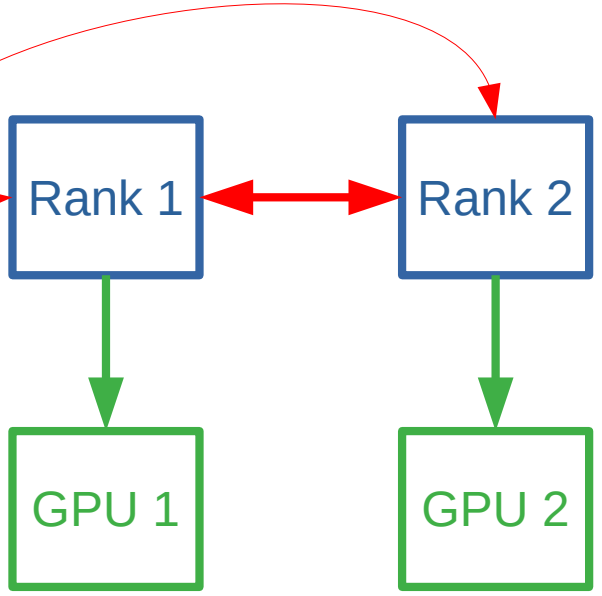
Evaluate the ratio between the best time to solution of your CPU and GPU tests.

Exercise 5: running with GPUs

GPU parallelism
What is happening?

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



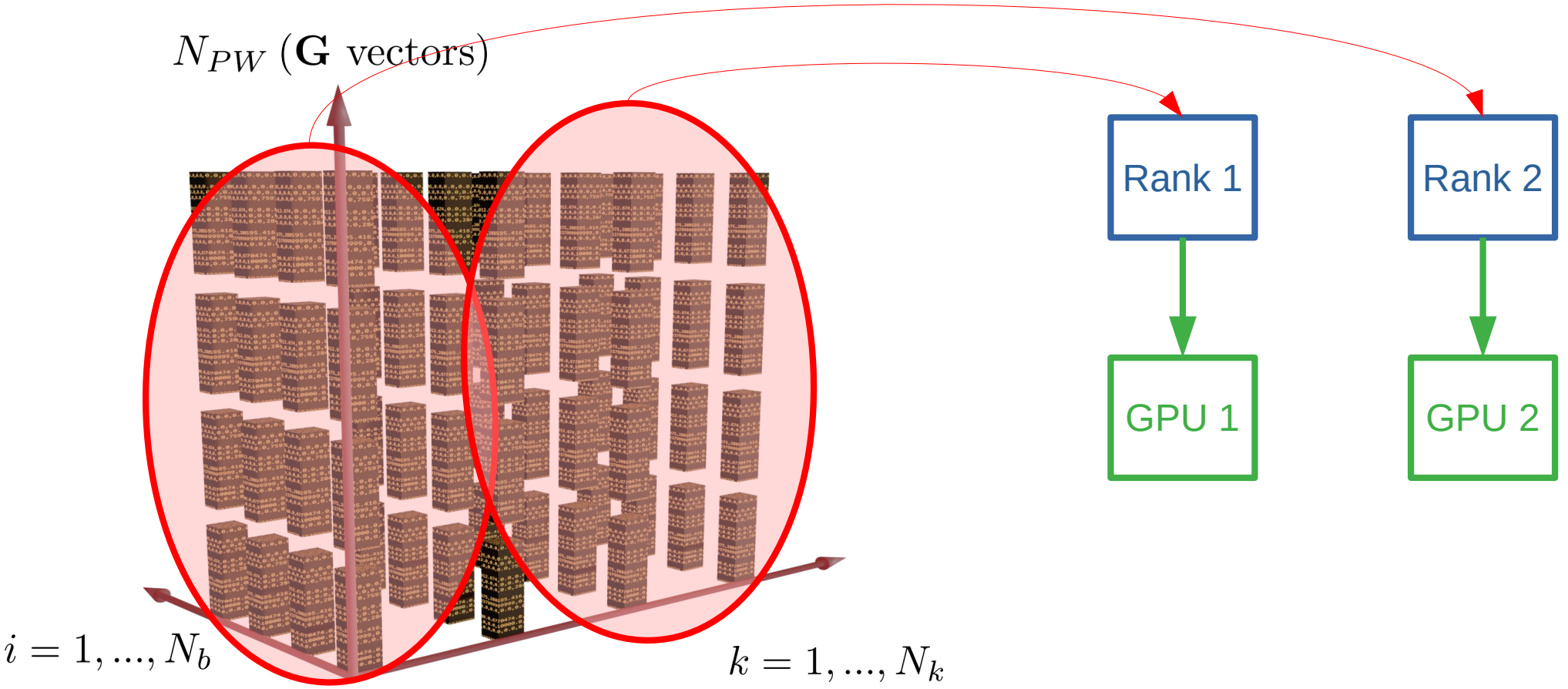


It is convenient to use one MPI per GPU

Exercise 5: running with GPUs

GPU parallelism
What is happening?

Again, using pools, will improve communications



QUANTUM ESPRESSO FOUNDATION

QUANTUMESPRESSO

MaX School on Advanced Materials and Molecular Modelling
with QUANTUM ESPRESSO

Thanks for your attention!

Ivan Carnimeo, Pietro Bonfa'
Paolo Pegolo, Mandana Safari, Riccardo Bertossa

MaX

ICTP The Abdus Salam International Centre for Theoretical Physics

CINECA

上海大学 SHANGHAI UNIVERSITY

Jožef Stefan Institute Ljubljana, Slovenia

cecam

SISSA