

# Quantum ESPRESSO GPU on Marconi100

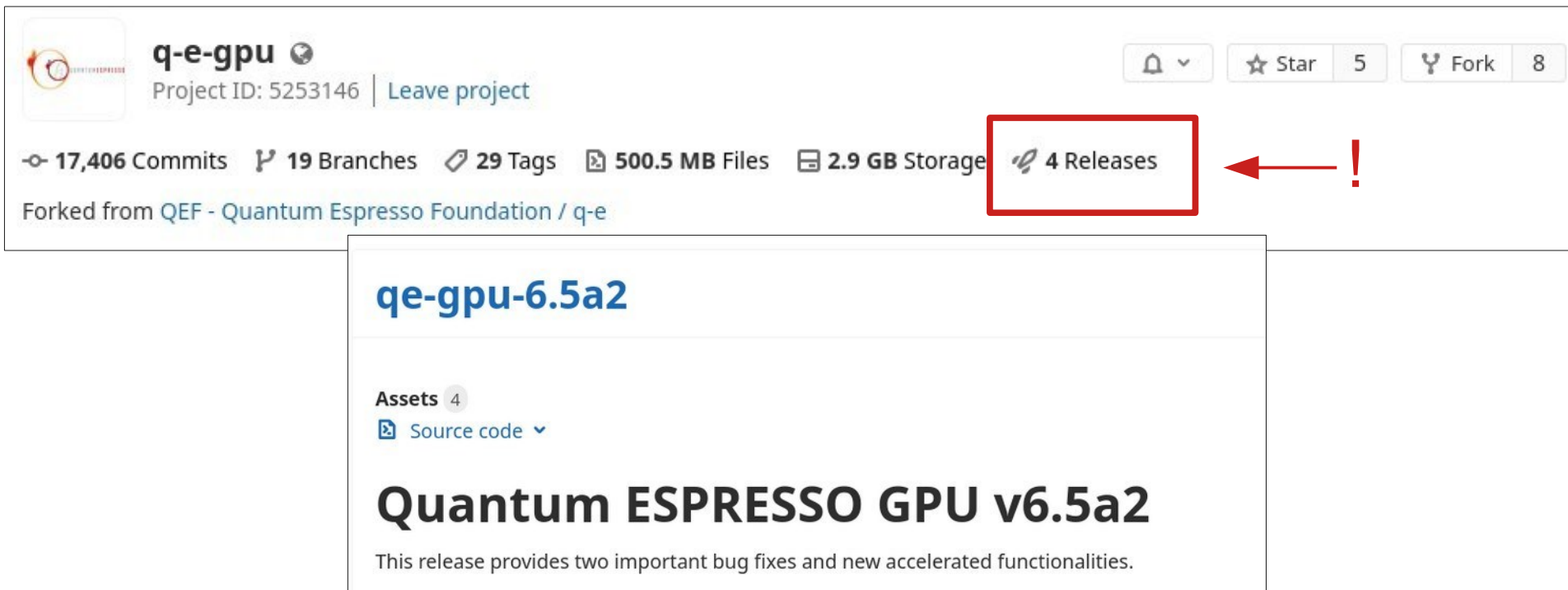
Pietro Bonfà


Department of Mathematical, Physical and Computer Sciences,  
University of Parma; Centro S3, CNR-Istituto Nanoscienze




# GPU enabled version of QuantumESPRESSO









The last release of QE-GPU is available at

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





**q-e-gpu**   
Project ID: 5253146 | [Leave project](#)

  Star 5  Fork 8

 17,406 Commits  19 Branches  29 Tags  500.5 MB Files  2.9 GB Storage  **4 Releases**  

Forked from [QEF - Quantum Espresso Foundation / q-e](#)

## qe-gpu-6.5a2

Assets 4  
 [Source code](#) 

### Quantum ESPRESSO GPU v6.5a2

This release provides two important bug fixes and new accelerated functionalities.

# GPU enabled version of QuantumESPRESSO

The last release of QE-GPU is available at

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

GitLab Projects Groups More

QEF - Quantum Espresso Foundation > q-e-gpu > Wiki > Home

Home  
Last edited by Pietro 2 months ago

New page Page history Edit

## Quantum ESPRESSO GPU

This repository hosts the experimental GPU accelerated version of Quantum ESPRESSO. This project aims at developing, testing and stabilizing the GPU accelerated version, which will eventually be merged into the official repository (QEF/q-e).

**Quick links:**

- Releases
- Optimized install instructions
- Tasks ported to GPU
- Benchmarks

**Overview**

Currently only pw.x can take advantage from offloading computations to GPU cards. It provides all the functionalities of the equivalent CPU-accelerated version.

**Versioning**

Clone repository

Accelerated Features

- Install QE GPU JUWELS JSC
- Install QE GPU on DAVIDE
- ALILEO
- mmmit
- PizDaint (hybrid)
- AndRun

Wiki

**Quick links:**

- Releases
- Optimized install instructions
- Tasks ported to GPU
- Benchmarks

# Compiling QE GPU

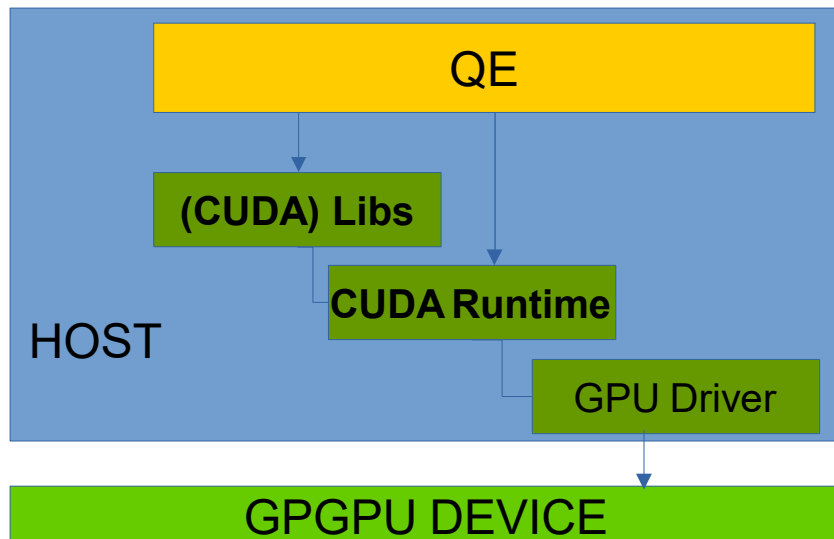
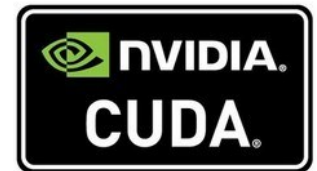
Compiling is as simple as...

```
./configure --with-cuda=XX --with-cuda-runtime=Y.y --with-cuda-cc=ZZ --enable-openssl [ --with-scalapack=no ]
```

where `XX` is the location of the CUDA Toolkit (in HPC environments it is generally `$CUDA_HOME`, be sure that this variable is not empty with a simple `echo $CUDA_HOME`), `Y.y` is the version of the CUDA Toolkit (`Y` and `y` are the two numbers identifying major and minor release, e.g. `9.0`) and `ZZ` is the compute capability (cc) of the card. This information can be found on the internet using the model name of the GPU card or by using `pgaccelinfo` command.

**Openmp is required** in order to successfully compile the accelerated version.

PGI®



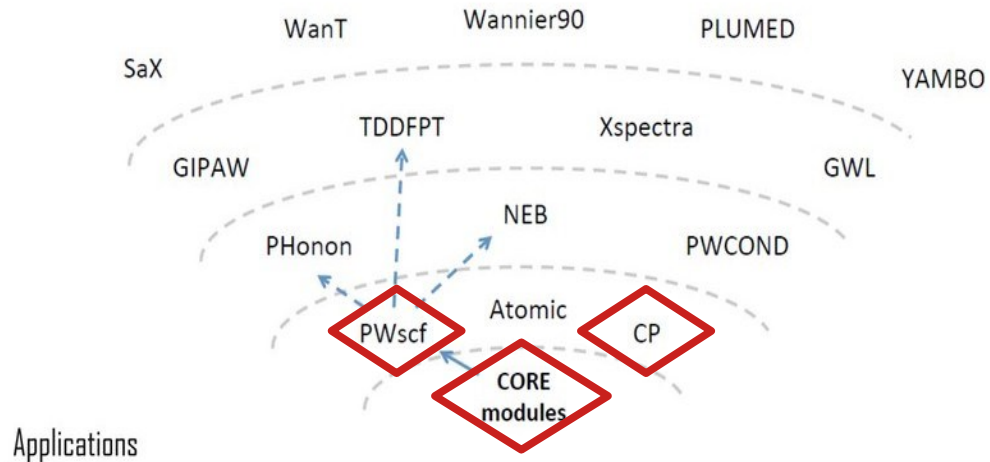
Cuda Toolkit is generally in `$CUDA_HOME`.

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

```
[pbonfa00@login02 scf]$ pgaccelinfo | grep cc
PGI Default Target:      -ta=tesla:cc70
PGI Default Target:      -ta=tesla:cc70
PGI Default Target:      -ta=tesla:cc70
PGI Default Target:      -ta=tesla:cc70
```

# What does QE GPU provide

Programming model Objective



Applications

Domain Specific Libraries

Directive based programming

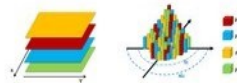
Optimize memory duplication, allocation and synchronization.

LAXLib

$$A\mathbf{v} = \lambda B\mathbf{v}$$

TASK: parallel linear algebra  
LIBS: ELPA, MKL, cuBLAS, cuSOLVER, ESSL, ...

FFTXlib



TASK: Parallel distributed FFT  
LIBS: FFTW, MKL, ESSL, cuFFT, ...

KS\_Solvers

$$|\delta\psi_i\rangle = \frac{1}{D - \epsilon_i}(H - \epsilon_i)|\psi_i\rangle$$

TASK: Iterative solvers  
LIBS: LAXLib, MKL, cuBLAS, ...

Explicit accelerator programming

Optimize computational efficiency and concurrency

# What does QE GPU provide

What can be done with the accelerated version of pw.x

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
V6.5a1	A	A	W	A	A	A	A	A	W	W
V6.5a2	A	A	A	A	A	A	A	A	W	W

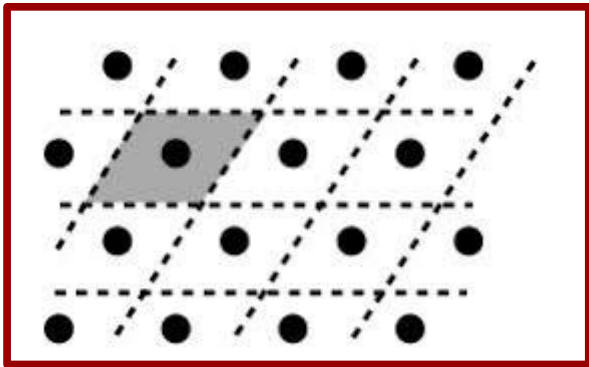
**A**ccelerated, **W**orking, **U**navailable, **B**roken

\* Acceleration obtained from other parts of the code.

# QE in the homogeneous HPC world

You know how to run QE efficiently on a HPC machine:

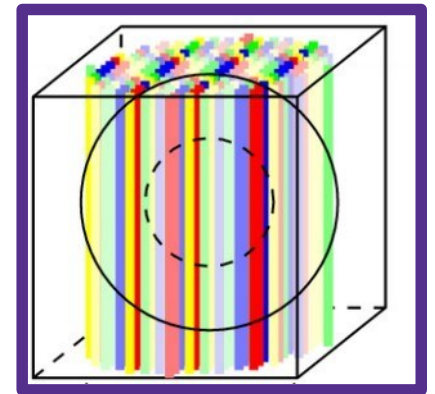
```
mpirun pw.x -npool X -ndiag Y -ntg Z -inp pw.in
```



$$A = \begin{bmatrix} 1 & 2 \\ 3 & -4 \end{bmatrix}$$

Step 1

$$\det(A - \lambda I) = 0$$

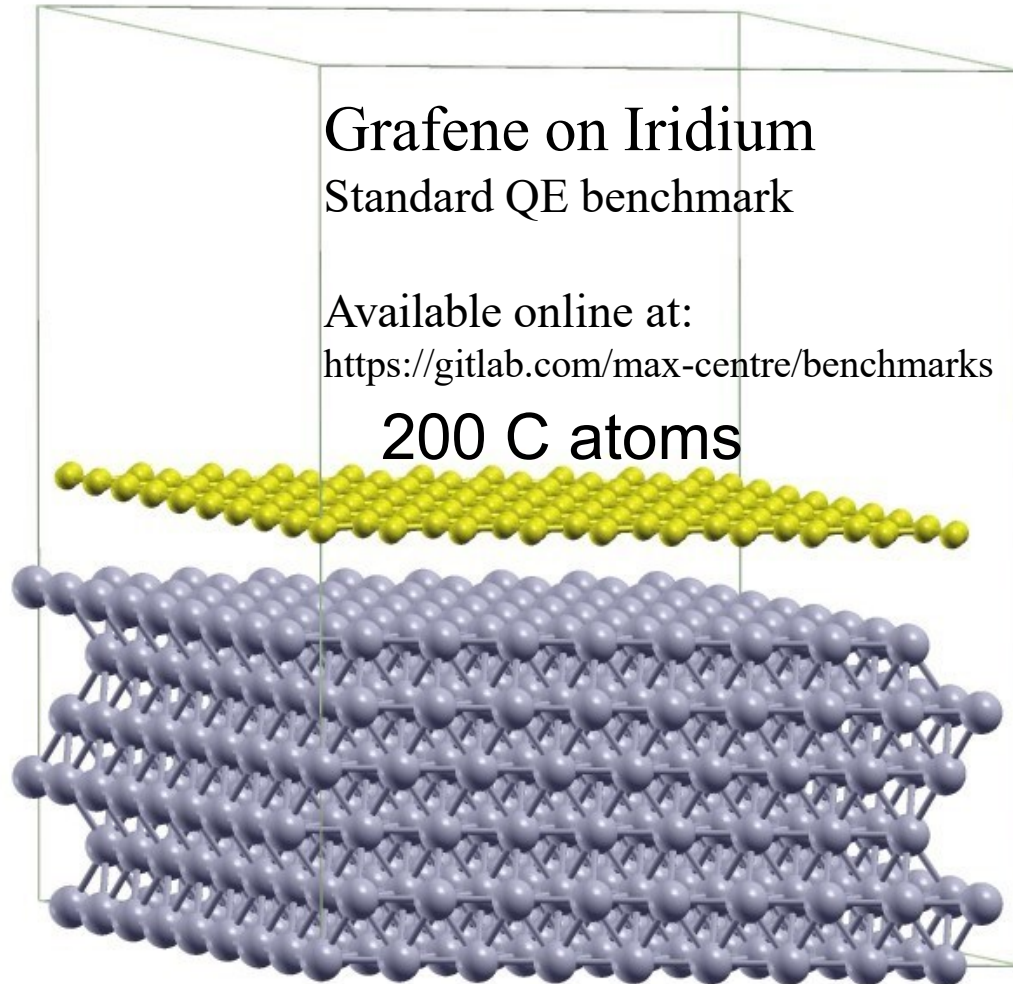


# QE in the homogeneous HPC world

```
mpirun pw.x -npool X -ndiag Y -ntg Z -inp pw.in
```

```
&control
  calculation = 'scf'
  prefix='GRIR'
  restart_mode='from_scratch'
  pseudo_dir='./',
/
&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
```

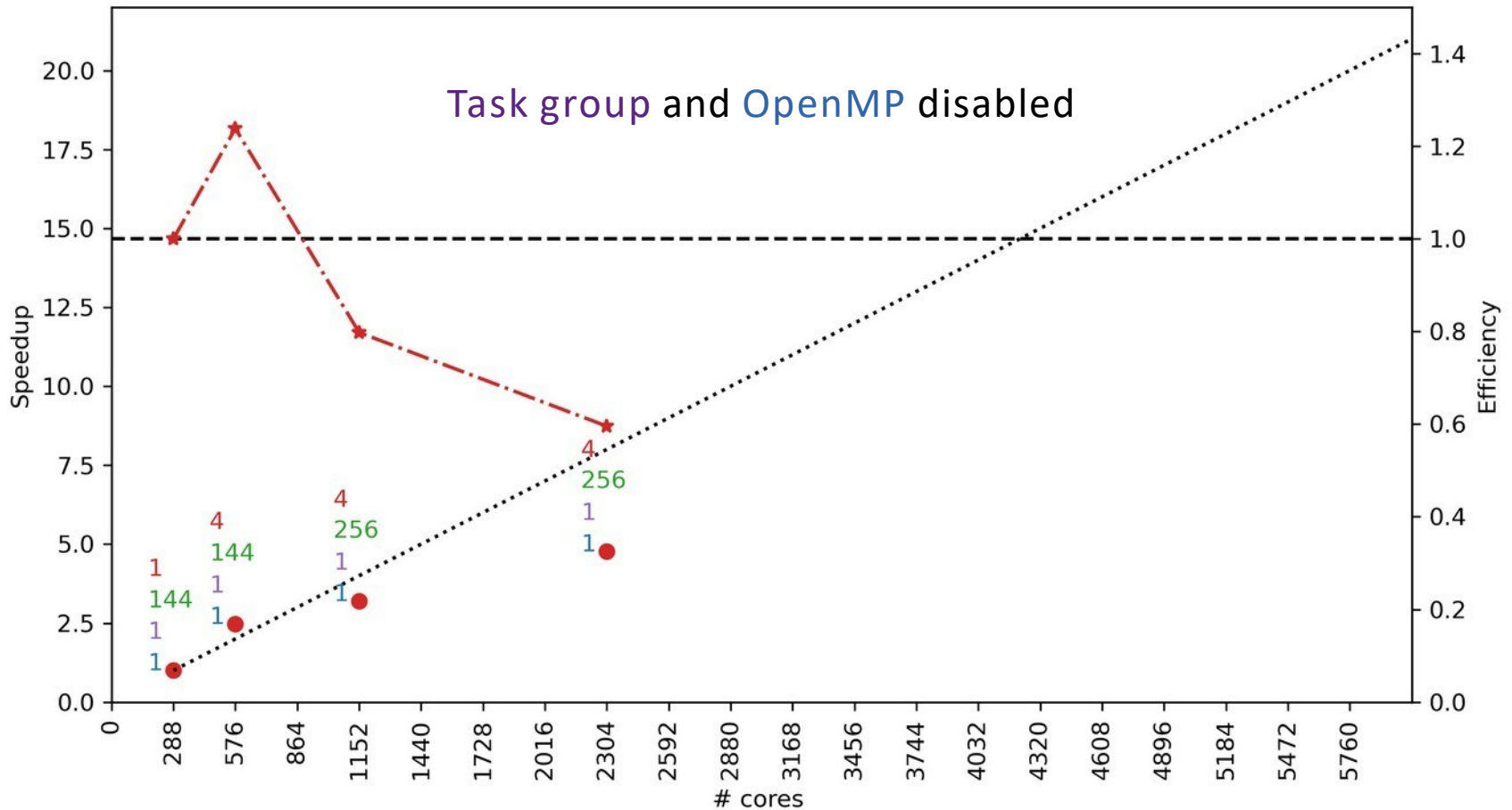
486 Ir atoms





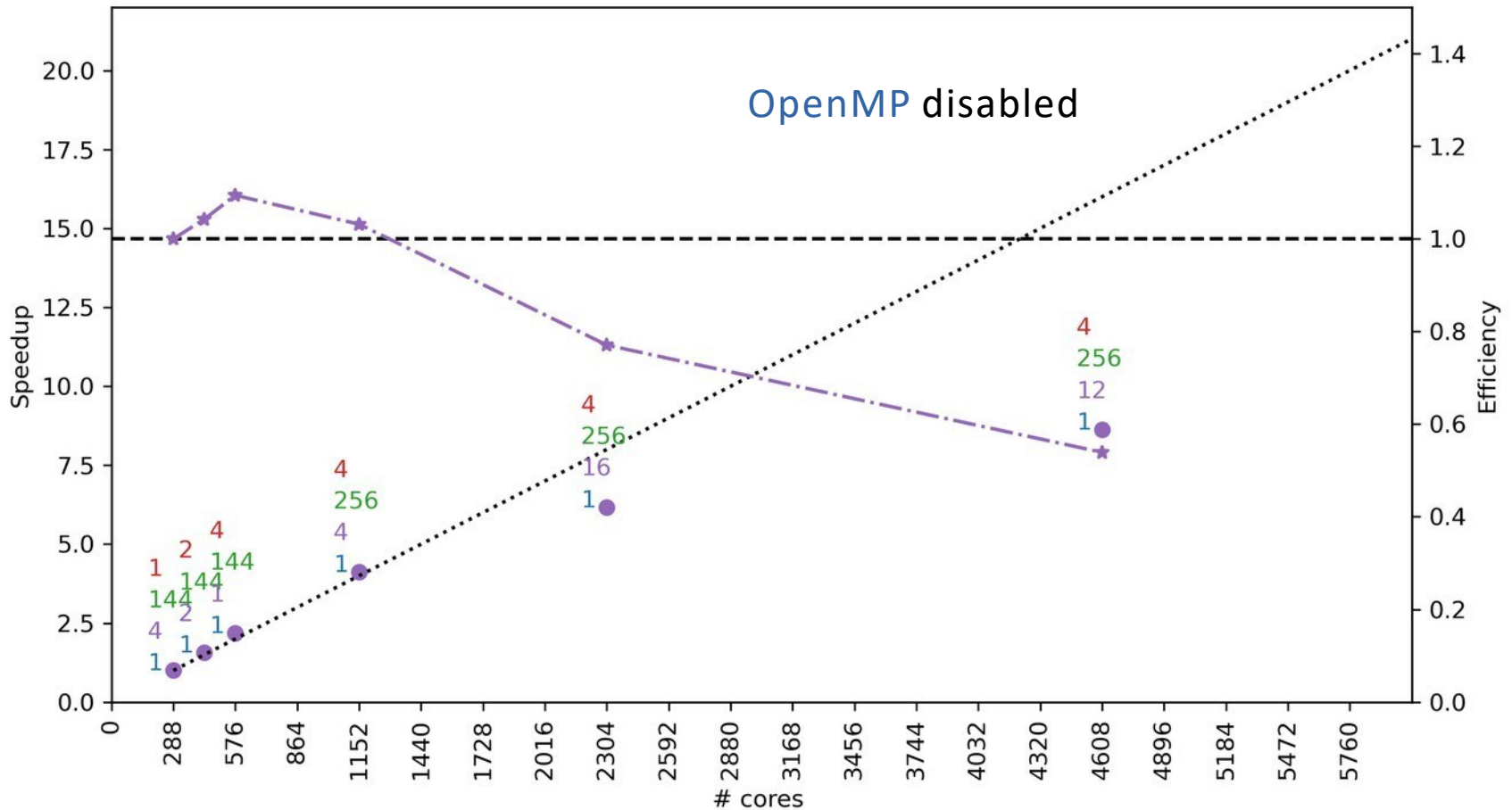
# QE in the homogeneous HPC world

```
mpirun pw.x -npool X -ndiag Y -ntg Z -inp pw.in
```



# QE in the homogeneous HPC world

```
mpirun pw.x -npool X -ndiag Y -ntg Z -inp pw.in
```



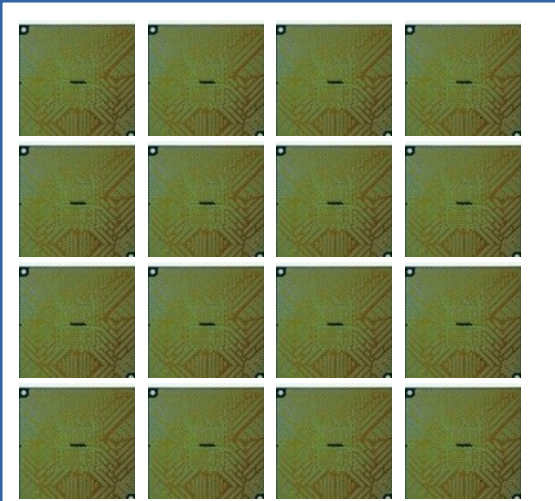


# QE in the heterogeneous HPC world

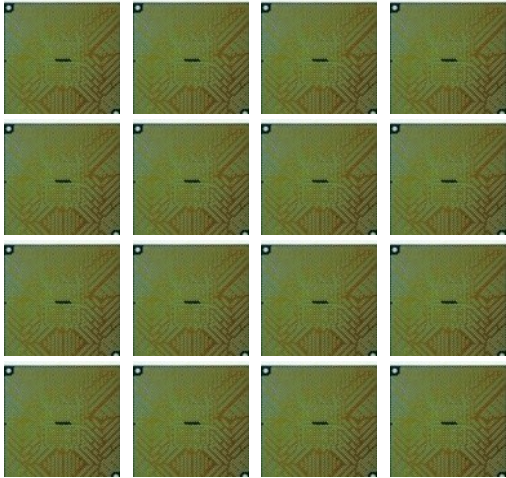
7.8 TFlops



About ten times more powerful!

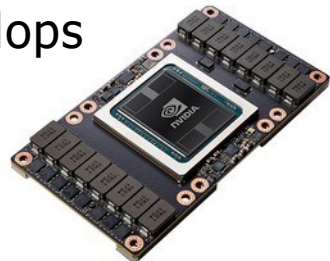


0.8 TFlops

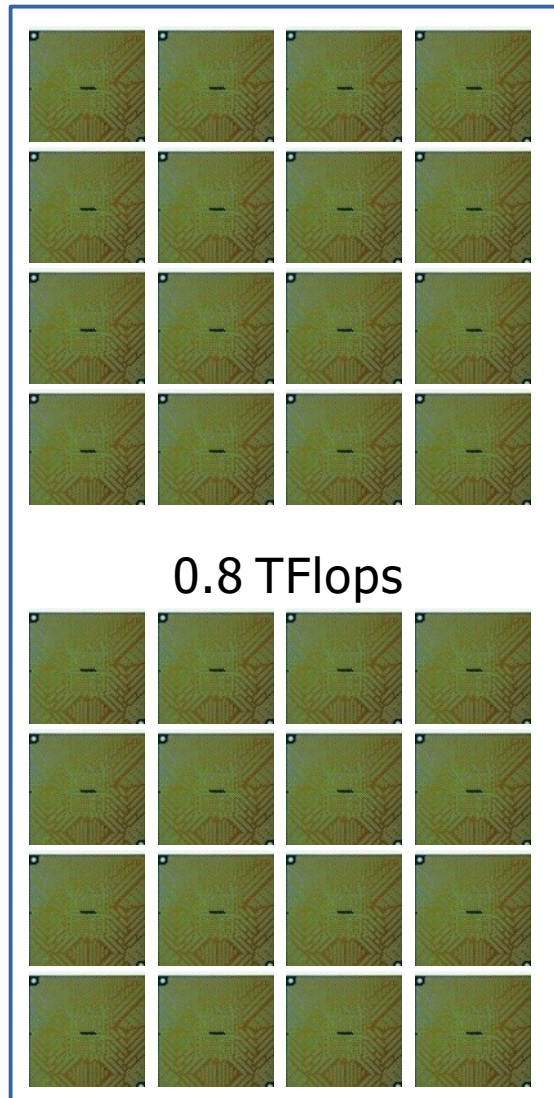
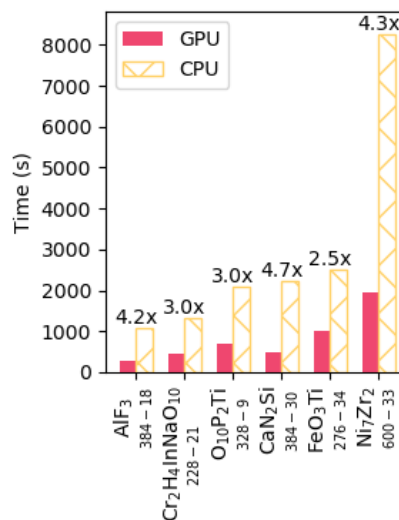
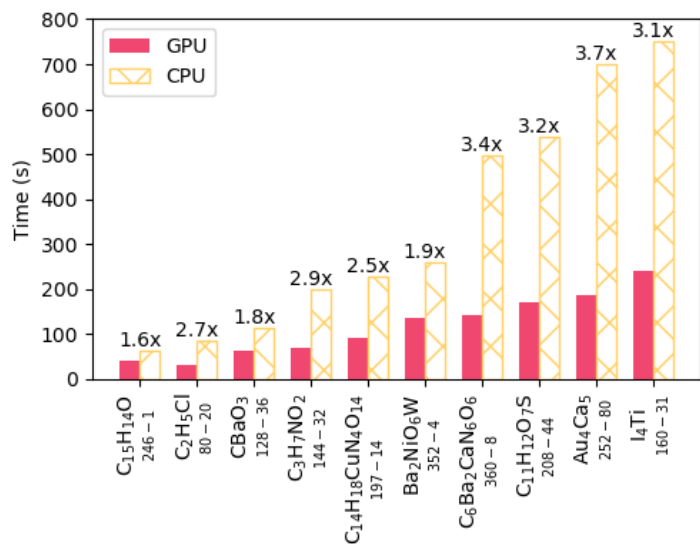


# QE in the heterogeneous HPC world

7.8 TFlops



About ten times more powerful!



Quantum ESPRESSO toward the exascale

P. Giannozzi *et al.*

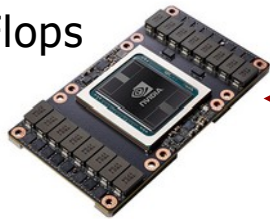
J. Chem. Phys. 152, 154105 (2020); DOI: 10.1063/5.0005082

# QE in the heterogeneous HPC world

There are 4 GPUs per node on Marconi100!

```
mpirun -np 4 pw.x
```

7.8 TFlops



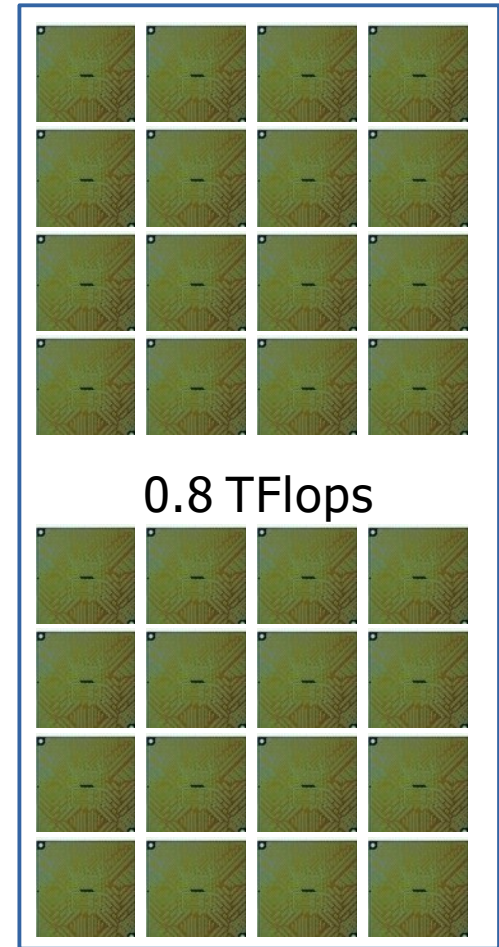
7.8 TFlops



7.8 TFlops



7.8 TFlops



0.8 TFlops

# QE in the heterogeneous HPC world

There are 4 GPUs per node on Marconi100!

```
mpirun -np 4 pw.x
```

7.8 TFlops



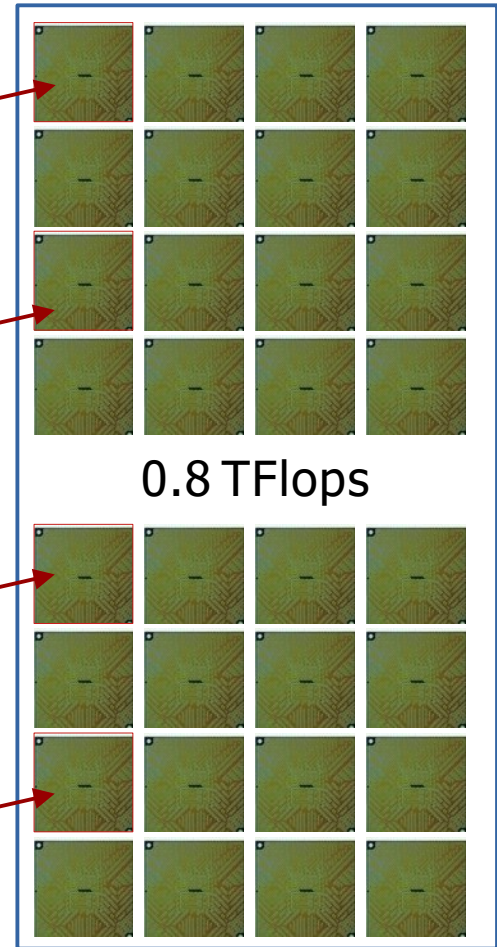
7.8 TFlops



7.8 TFlops



7.8 TFlops



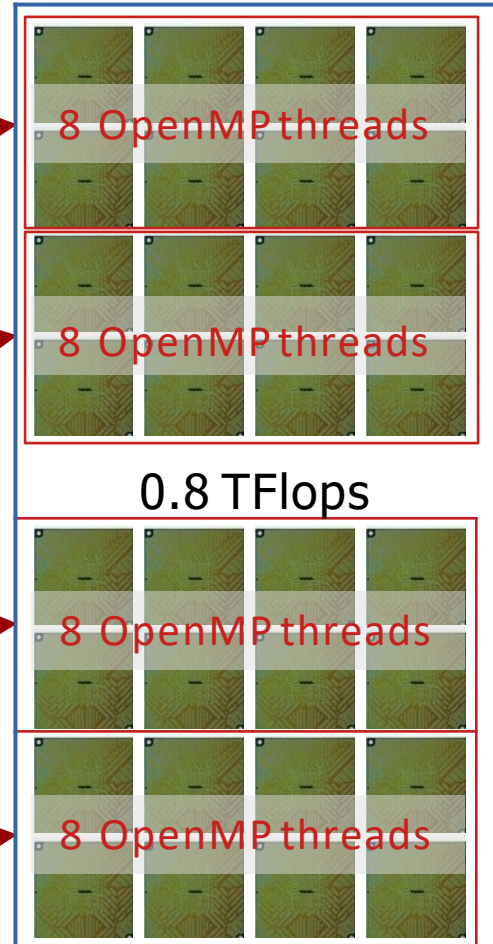
0.8 TFlops

# QE in the heterogeneous HPC world

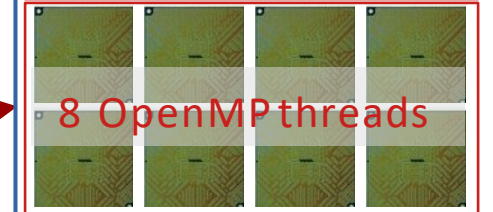
There are 4 GPUs per node on Marconi100!

```
OMP_NUM_THREADS=8  
mpirun -np 4 pw.x
```

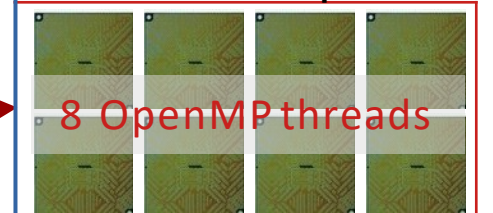
7.8 TFlops



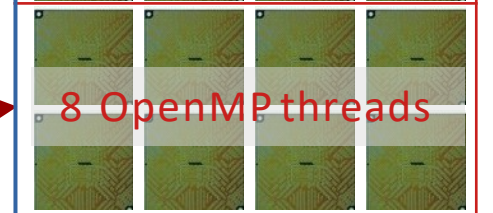
7.8 TFlops



7.8 TFlops



7.8 TFlops





# QE in the heterogeneous HPC world



One MPI process per GPU! `mpirun -np nGPU pw.x ...`

What about parallel execution options?

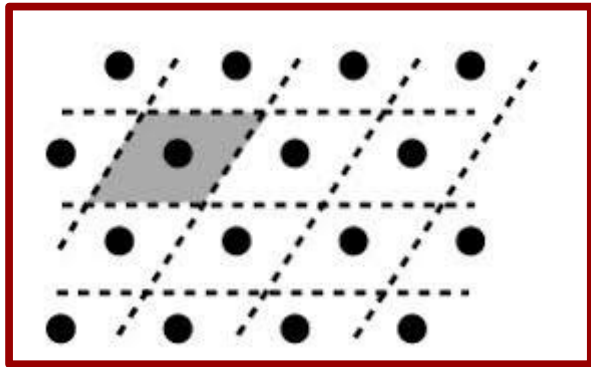
`mpirun -np nGPU pw.x -npool X -ndiag Y -ntg Z`

# QE in the heterogeneous HPC world

One MPI process per GPU! `mpirun -np nGPU pw.x ...`

What about parallel execution options?

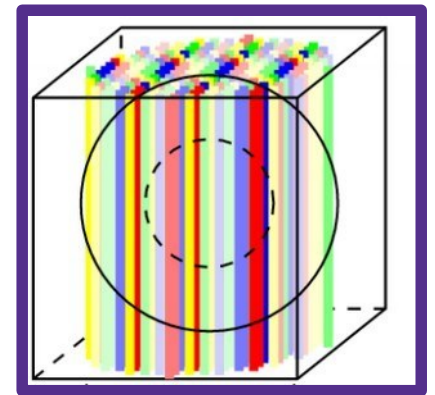
`mpirun -np nGPU pw.x -npool X -ndiag Y -ntg Z`



$$A = \begin{bmatrix} 1 & 2 \\ 3 & -4 \end{bmatrix}$$

Step 1

$$\det(A - \lambda I) = 0$$

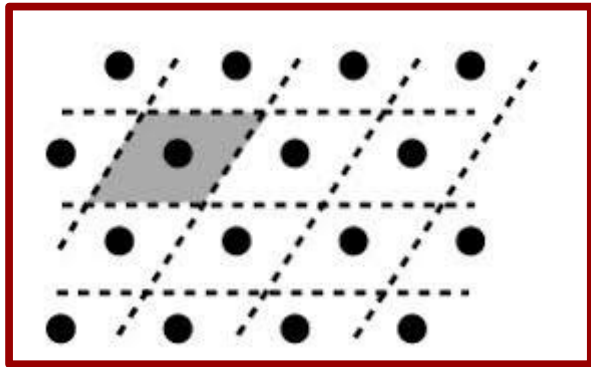


# QE in the heterogeneous HPC world

One MPI process per GPU! `mpirun -np nGPU pw.x ...`

What about parallel execution options?

`mpirun -np nGPU pw.x -npool X -ndiag Y -ntg Z`



$\begin{bmatrix} 1 & 2 \\ -4 & \end{bmatrix}$

Step 1

$\det(A - \lambda I) = 0$

GPU Solver uses only 1 device in v6.5a2



# QE in the heterogeneous HPC world



One MPI process per GPU! `mpirun -np nGPU pw.x ...`

What about parallel execution options?

```
mpirun -np nGPU pw.x -npool X -ndiag 1 -ntg 1
```

Subspace diagonalization in iterative solution of the eigenvalue problem:  
a serial algorithm will be used ✓

[...]

GPU acceleration is ACTIVE. ✓

# QE in the heterogeneous HPC world

K-point parallelization is very effective...  
...but what about memory?!



Only 16GB!

```
mpirun -np nGPU pw.x -npool X -ndiag 1 -ntg 1
```

Check memory estimator!

```
X=4      Estimated max dynamical RAM per process > 14.72 GB  
X=1      Estimated max dynamical RAM per process > 2.97 GB
```

Choose the largest value for **X** that fits available memory.

# Tips & Tricks

You run out of memory ... what to do?

- use more GPUs...
- reduce subspace dimension in Davidson algorithm
- Change diagonalization method

Feature X is slow!

→ Open an issue at <https://gitlab.com/QEF/q-e-gpu/-/issues>

```
&electrons
  conv_thr = 1.0d-9
  mixing_beta=0.3d0
  startingwfc='atomic'
  diago_david_ndim='2'
/
```

```
&electrons
  conv_thr = 1.0d-9
  mixing_beta=0.3d0
  startingwfc='atomic'
  diagonalization='cg'
/
```

# Take home message

A few things you should remember when running the GPU version of the code:

- *1MPI process per GPU,*
- CPU cores can (must!) be exploited with *OpenMP parallelism*
- Pool parallelism is very effective, but requires memory
- The dense eigenvalue problem is (as of v6.5a2) solved on 1 GPU, *use the serial eigensolver.*
- Check the Wiki, it's updated with a collaborative effort!
- More details: P. Giannozzi *et al.* J. Chem. Phys. 152, 154105 (2020)





DRIVING THE EXASCALE TRANSITION

**Follow us on:**

**THANKS**

 [company/max-centre/](https://www.linkedin.com/company/max-centre/)

 [@max\\_center2](https://twitter.com/max_center2)

 <http://www.max-centre.eu/>