MAX DRIVING THE EXASCALE TRANSITION

UNIVERSITÀ DI PARMA

GPU accelerated Quantum ESPRESSO

Pietro Bonfà
Department of Mathematical, Physical and Computer Sciences, University of Parma, Italy

14 Jan. 2020 - Jülich

Commissione europea

EUROPEAN CENTER OF EXCELLENCE - A H2020 E-INFRASTRUCTURE

H2020-INFRAEDI-2018-1 G.A. 824143

Topics

What’s in this presentation:

✓ Evolution of **GPU acceleration** in QE: strategies and motivation.

✓ **Domain Specific Libraries**, that you may want to check for your own codes.

✓ A few details about the programming model used for QE.

✗ How to run QE on GPU accelerated platforms

✗ How to develop QE on GPU accelerated platforms

Acknowledgements

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

Collaboration and support from: J. Romero, M. Marić, M. Fatica, E. Phillips (NVIDIA)
F. Spiga (ARM), A. Chandran (FZJ), I. Girotto (ICTP), Y. Luo (ANL), T. Kurth (NVIDIA), B. Cook (NERSC),
F. Ferrari, P. Giannozzi (Univ. Udine), A. Ferretti (CNR-S3 Modena),P. Delugas, S. De Gironcoli (SISSA).

MaX CoE

Mission: materials science codes ready for exascale computing.

siesta

QUANTUMESPRESSO

www.flapw.de

Fleur

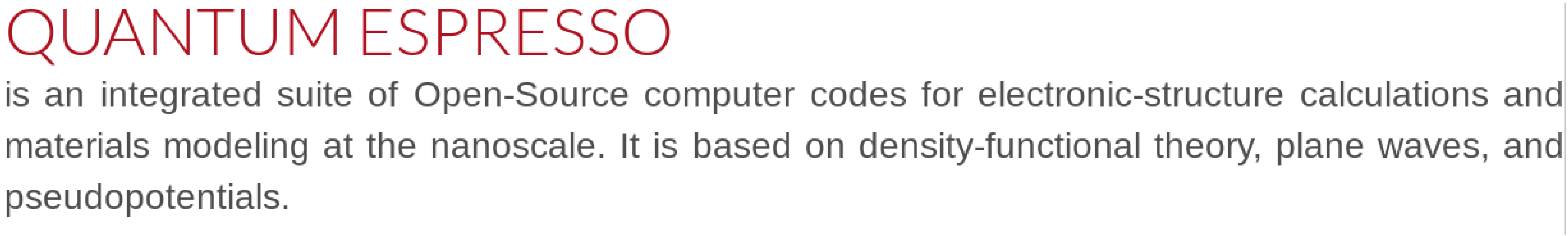
Yambo

CP2K

SIRIUS

BigDFT

What is QE



QUANTUM ESPRESSO

is an integrated suite of Open-Source computer codes for electronic-structure calculations and materials modeling at the nanoscale. It is based on density-functional theory, plane waves, and pseudopotentials.

What is QE

Ground-state calculations:

- Self-consistent total energies, forces, stresses;
- Kohn-Sham orbitals;
- Separable norm-conserving and ultrasoft (Vanderbilt) pseudo-potentials, PAW (Projector Augmented Waves);
- Several exchange-correlation functionals: from LDA to generalized-gradient corrections (PW91, PBE, B88-P86, BLYP) to meta-GGA, exact exchange (HF) and hybrid functionals (PBE0, B3LYP, HSE);
- VdW corrections (DFT-D) or nonlocal VdW functionals (vdw-DF);
- Hubbard U (DFT+U);
- Berry's phase polarization;
- Spin-orbit coupling and noncollinear magnetism.

Structural Optimization:

- GDIIS with quasi-Newton BFGS preconditioning;
- Damped dynamics.

Transition states and minimum energy paths

- Nudged Elastic Band method;
- Meta-Dynamics, using the PLUMED plug-in.

Ab-initio molecular dynamics

- Car-Parrinello Molecular Dynamics (CP package);
- Born-Oppenheimer Molecular Dynamics (PWscf package).

Response properties (density-functional perturbation theory):

- Phonon frequencies and eigenvectors at any wavevector;
- Full phonon dispersions; inter-atomic force constants in real space;
- Translational and rotational acoustic sum rules;
- Effective charges and dielectric tensors;
- Electron-phonon interactions;
- Third-order anharmonic phonon lifetimes, using the D3Q package;
- Infrared and (non-resonant) Raman cross-sections;
- EPR and NMR chemical shifts, using the QE-GIPAW package.
- Phonons for 2D heterostructures (reference)

Spectroscopic properties:

- $K-$, $L1$ and $L2,3$-edge X-ray Absorption Spectra (XSpectra package);
- Time-Dependent Density Functional Perturbation Theory (TurboTDDFT package);
- Electronic excitations with Many-Body Perturbation Theory, using the YAMBO package;
- Electronic excitations with Many-Body Perturbation Theory (GWL package).

Quantum Transport:

- Ballistic Transport (PWCOND package);
- Coherent Transport from Maximally Localized Wannier Functions, using the WanT code;
- Maximally-localized Wannier functions and transport properties, using the WANNIER90 code.
- Kubo-Greenwood electrical conductivity using the KGEC code.

What is QE

Ground-state calculations:

- Self-consistent total energies, forces, stresses;
- Kohn-Sham orbitals;
- Separable norm-conserving and ultrasoft (Vanderbilt) pseudo-potentials, PAW (Projector Augmented Waves);
- Several exchange-correlation functionals: from LDA to generalized-gradient corrections (PW91, PBE, B88-P86, BLYP) to meta-GGA, exact exchange (HF) and hybrid functionals (PBE0, B3LYP, HSE);
- VdW corrections (DFT-D) or nonlocal VdW functionals (vdw-DF);
- Hubbard U (DFT+U);
- Berry's phase polarization;
- Spin-orbit coupling and noncollinear magnetism.

package);
AMBO package;
ge).

Transition states and minimum energy paths

- Nudged Elastic Band method;
- Meta-Dynamics, using the PLUMED plug-in.

Ab-initio molecular dynamics

- Car-Parrinello Molecular Dynamics (CP package);
- Born-Oppenheimer Molecular Dynamics (PWscf package).

Quantum Transport:

- Ballistic Transport (PWCOND package);
- Coherent Transport from Maximally Localized Wannier Functions, using the WanT code;
- Maximally-localized Wannier functions and transport properties, using the WANNIER90 code.
- Kubo-Greenwood electrical conductivity using the KGEC code.

What is QE

Ground-state calculations:

- Self-consistent total energies, forces, stresses;
- Kohn-Sham orbitals;
- Separable norm-conserving and ultrasoft (Vanderbilt) pseudo-potentials, PAW (Projector Augmented Waves);
- Several exchange-correlation functionals: from LDA to generalized-gradient corrections (PW91,

B3LYP,

Structural Optimization:

- GDIIS with quasi-Newton BFGS preconditioning;
- Damped dynamics.

- Damped dynamics.

Transition states and minimum energy paths

- Nudged Elastic Band method;
- Meta-Dynamics, using the PLUMED plug-in.

Ab-initio molecular dynamics

- Car-Parrinello Molecular Dynamics (CP package);
- Born-Oppenheimer Molecular Dynamics (PWscf package).

Response properties (density-functional perturbation theory):

- Phonon frequencies and eigenvectors at any wavevector;
- Full phonon dispersions; inter-atomic force constants in real space;
- Translational and rotational acoustic sum rules;
- Effective charges and dielectric tensors;
- Electron-phonon interactions;
- Third-order anharmonic phonon lifetimes, using the D3Q package;
- Infrared and (non-resonant) Raman cross-sections;
- EPR and NMR chemical shifts, using the QE-GIPAW package.
- Phonons for 2D heterostructures (reference)

Spectroscopic properties:

- *K*−, *L*1 and *L*2,3-edge X-ray Absorption Spectra (XSpectra package);
- Time-Dependent Density Functional Perturbation Theory (TurboTDDFT package);
- Electronic excitations with Many-Body Perturbation Theory, using the YAMBO package;
- Electronic excitations with Many-Body Perturbation Theory (GWL package).

Quantum Transport:

- Ballistic Transport (PWCOND package);
- Coherent Transport from Maximally Localized Wannier Functions, using the WanT code;
- Maximally-localized Wannier functions and transport properties, using the WANNIER90 code.
- Kubo-Greenwood electrical conductivity using the KGEC code.

What is QE

Ground-state calculations:

- Self-consistent total energies, forces, stresses;
- Kohn-Sham orbitals;
- Separable norm-conserving and ultrasoft (Vanderbilt) pseudo-potentials, PAW (Projector Augmented Waves);
- Several exchange-correlation functionals: from LDA to generalized-gradient corrections (PW91, PBE, B88-P86, BLYP) to meta-GGA, exact exchange (HF) and hybrid functionals (PBE0, B3LYP, HSE);
- VdW corrections (DFT-D) or nonlocal VdW functionals (vdw-DF);
- Hubbard U (DFT+U);
- Berry's phase polarization;
- Spin-orbit coupling and noncollinear magnetism.

Transition states and minimum energy paths

- Nudged Elastic Band method;
- Meta-Dynamics, using the PLUMED plug-in.

Ab-initio molecular dynamics

- Car-Parrinello Molecular Dynamics (CP package);
- Born-Oppenheimer Molecular Dynamics (PWscf package).

Response properties (density-functional perturbation theory):

- Phonon frequencies and eigenvectors at any wavevector;
- Full phonon dispersions; inter-atomic force constants in real space;
- Translational and rotational acoustic sum rules;
- Effective charges and dielectric tensors;
- Electron-phonon interactions;
- Third-order anharmonic phonon lifetimes, using the D3Q package;
- Infrared and (non-resonant) Raman cross-sections;
- EPR and NMR chemical shifts, using the QE-GIPAW package.
- Phonons for 2D heterostructures (reference)

Spectroscopic properties:

- *K*−, *L*1 and *L*2,3-edge X-ray Absorption Spectra (XSpectra package);
- Time-Dependent Density Functional Perturbation Theory (TurboTDDFT package);
- Electronic excitations with Many-Body Perturbation Theory, using the YAMBO package;
- Electronic excitations with Many-Body Perturbation Theory (GWL package).

Quantum Transport:

- Ballistic Transport (PWCOND package);
- Coherent Transport from Maximally Localized Wannier Functions, using the WanT code;
- Maximally-localized Wannier functions and transport properties, using the WANNIER90 code.
- Kubo-Greenwood electrical conductivity using the KGEC code.

What is QE

Ground-state calculations:

- Self-consistent total energies, forces, stresses;
- Kohn-Sham orbitals;
- Separable norm-conserving and ultrasoft (Vanderbilt) pseudo-potentials, PAW (Projector Augmented Waves);
- Several exchange-correlation functionals: from LDA to generalized-gradient corrections (PW91, PBE, B88-P86, BLYP) to meta-GGA, exact exchange (HF) and hybrid functionals (PBE0, B3LYP, HSE);
- VdW corrections (DFT-D) or nonlocal VdW functionals (vdw-DF);
- Hubbard U (DFT+U);
- Berry's phase polarization;
- Spin-orbit coupling and noncollinear magnetism.

Structural Optimization:

- GDIIS with quasi-Newton BFGS preconditioning;
- Damped dynamics.

Ab-initio molecular dynamics

- Car-Parrinello Molecular Dynamics (CP package);
- Born-Oppenheimer Molecular Dynamics (PWscf package).

Response properties (density-functional perturbation theory):

- Phonon frequencies and eigenvectors at any wavevector;
- Full phonon dispersions; inter-atomic force constants in real space;
- Translational and rotational acoustic sum rules;
- Effective charges and dielectric tensors;
- Electron-phonon interactions;
- Third-order anharmonic phonon lifetimes, using the D3Q package;
- Infrared and (non-resonant) Raman cross-sections;
- EPR and NMR chemical shifts, using the QE-GIPAW package.
- Phonons for 2D heterostructures (reference)

Spectroscopic properties:

- *K*−, *L*1 and *L*2,3-edge X-ray Absorption Spectra (XSpectra package);
- Time-Dependent Density Functional Perturbation Theory (TurboTDDFT package);
- Electronic excitations with Many-Body Perturbation Theory, using the YAMBO package;
- Electronic excitations with Many-Body Perturbation Theory (GWL package).

Quantum Transport:

- Ballistic Transport (PWCOND package);
- Coherent Transport from Maximally Localized Wannier Functions, using the WanT code;
- Maximally-localized Wannier functions and transport properties, using the WANNIER90 code.
- Kubo-Greenwood electrical conductivity using the KGEC code.

What is QE

Ground-state calculations:

- Self-consistent total energies, forces, stresses;
- Kohn-Sham orbitals;
- Separable norm-conserving and ultrasoft (Vanderbilt) pseudo-potentials, PAW (Projector Augmented Waves);
- Several exchange-correlation functionals: from LDA to generalized-gradient corrections (P PBE, B88-P86, BLYP) to meta-GGA, exact exchange (HF) and hybrid functionals (PBE0, HSE);
- VdW corrections (DFT-D) or nonlocal VdW functionals (vdw-DF);
- Hubbard U (DFT+U);
- Berry's phase polarization;
- Spin-orbit coupling and noncollinear magnetism.

Structural Optimization:

- GDIIS with quasi-Newton BFGS preconditioning;
- Damped dynamics.

Transition states and minimum energy paths

- Nudged Elastic Band method;
- Meta-Dynamics, using the PLUMED plug-in.

Ab-initio molecular dynamics

- Car-Parrinello Molecular Dynamics (CP package);
- Born-Oppenheimer Molecular Dynamics (PWscf package).

Response properties (density-functional perturbation theory):

- Phonon frequencies and eigenvectors at any wavevector;
- Full phonon dispersions; inter-atomic force constants in real space;
- Translational and rotational acoustic sum rules;
- Effective charges and dielectric tensors;
- Electron-phonon interactions;
- Third-order anharmonic phonon lifetimes, using the D3Q package;
- Infrared and (non-resonant) Raman cross-sections;
- EPR and NMR chemical shifts, using the QE-GIPAW package.
- Phonons for 2D heterostructures (reference)

- Ballistic Transport (PWCOND package);
- Coherent Transport from Maximally Localized Wannier Functions, using the WanT code;
- Maximally-localized Wannier functions and transport properties, using the WANNIER90 code.
- Kubo-Greenwood electrical conductivity using the KGEC code.

What is QE

Ground-state calculations:

- Self-consistent total energies, forces, stresses;
- Kohn-Sham orbitals;
- Separable norm-conserving and ultrasoft (Vanderbilt) pseudo-potentials, PAW (Projector Augmented Waves);
- Several exchange-correlation functionals: from LDA to gene PBE, B88-P86, BLYP) to meta-GGA, exact exchange (HF) a HSE);
- VdW corrections (DFT-D) or nonlocal VdW functionals (vdw-
- Hubbard U (DFT+U);
- Berry's phase polarization;
- Spin-orbit coupling and noncollinear magnetism.

Structural Optimization:

- GDIIS with quasi-Newton BFGS preconditioning;
- Damped dynamics.

Transition states and minimum energy paths

- Nudged Elastic Band method;
- Meta-Dynamics, using the PLUMED plug-in.

Ab-initio molecular dynamics

- Car-Parrinello Molecular Dynamics (CP package);
- Born-Oppenheimer Molecular Dynamics (PWscf package).

Response properties (density-functional perturbation theory):

- Phonon frequencies and eigenvectors at any wavevector;
- Full phonon dispersions; inter-atomic force constants in real space;
- Translational and rotational acoustic sum rules;
- Effective charges and dielectric tensors;

Spectroscopic properties:

- $K-$, $L1$ and $L2,3$-edge X-ray Absorption Spectra (XSpectra package);
- Time-Dependent Density Functional Perturbation Theory (TurboTDDFT package);
- Electronic excitations with Many-Body Perturbation Theory, using the YAMBO package;
- Electronic excitations with Many-Body Perturbation Theory (GWL package).

Quantum Transport:

- Ballistic Transport (PWCOND package);
- Coherent Transport from Maximally Localized Wannier Functions, using the WanT code;
- Maximally-localized Wannier functions and transport properties, using the WANNIER90 code.
- Kubo-Greenwood electrical conductivity using the KGEC code.

What is QE

Ground-state calculations:

- Self-consistent total energies, forces, stresses;
- Kohn-Sham orbitals;
- Separable norm-conserving and ultrasoft (Vanderbilt) pseudo-potentials, PAW (Projector Augmented Waves);
- Several exchange-correlation functionals: from LDA to generalized-gradient corrections (PW91, PBE, B88-P86, BLYP) to meta-GGA, exact exchange (HF) and hybrid functionals (PBE0, B3LYP, HSE);
- VdW corrections (DFT-D) or nonlocal VdW functionals (vdw-DF);
- Hubbard U (DFT+U);
- Berry's phase polarization;
- Spin-orbit coupling and noncollinear magnetism.

Structural Optimization:

- GDIIS with quasi-Newton BFGS preconditioning;
- Damped dynamics.

Transition states and minimum energy paths

- Nudged Elastic Band method;
- Meta-Dynamics, using the PLUMED plug-in.

Ab-initio molecular dynamics

- Car-Parrinello Molecular Dynamics (CP package);
- Born-Oppenheimer Molecular Dynamics (PWscf package).

Response properties (density-functional perturbation theory):

- Phonon frequencies and eigenvectors at any wavevector;
- Full phonon dispersions; inter-atomic force constants in real space;
- Translational and rotational acoustic sum rules;
- Effective charges and dielectric tensors;
- Electron-phonon interactions;
- Third-order anharmonic phonon lifetimes, using the D3Q package;
- Infrared and (non-resonant) Raman cross-sections;
- EPR and NMR chemical shifts, using the QE-GIPAW package.
- Phonons for 2D heterostructures (reference)

Quantum Transport:

- Ballistic Transport (PWCOND package);
- Coherent Transport from Maximally Localized Wannier Functions, using the WanT code;
- Maximally-localized Wannier functions and transport properties, using the WANNIER90 code.
- Kubo-Greenwood electrical conductivity using the KGEC code.

What is QE

Ground-state calculations:

- Self-consistent total energies, forces, stresses;
- Kohn-Sham orbitals;
- Separable norm-conserving and ultrasoft (Vanderbilt) pseudo-potentials, PAW (Projector Augmented Waves);
- Several exchange-correlation functionals: from LDA to generalized-gradient corrections (PW91, PBE, B88-P86, BLYP) to meta-GGA, exact exchange (HF) and hybrid functionals (PBE0, B3LYP, HSE);
- VdW corrections (DFT-D) or nonlocal VdW functionals (vdw-DF);
- Hubbard U (DFT+U);
- Berry's phase polarization;
- Spin-orbit coupling and noncollinear magnetism.

Structural Optimization:

- GDIIS with quasi-Newton BFGS preconditioning;
- Damped dynamics.

Transition states and minimum energy paths

- Nudged Elastic Band method;
- Meta-Dynamics, using the PLUMED plug-in.

Ab-initio molecular dynamics

- Car-Parrinello Molecular Dynamics (CP package);
- Born-Oppenheimer Molecular Dynamics (PWscf package).

Response properties (density-functional perturbation theory):

- Phonon frequencies and eigenvectors at any wavevector;
- Full phonon dispersions; inter-atomic force constants in real space;
- Translational and rotational acoustic sum rules;
- Effective charges and dielectric tensors;
- Electron-phonon interactions;
- Third-order anharmonic phonon lifetimes, using the D3Q package;
- Infrared and (non-resonant) Raman cross-sections;
- EPR and NMR chemical shifts, using the QE-GIPAW package.
- Phonons for 2D heterostructures (reference)

Spectroscopic properties:

- *K*−, *L*1 and *L*2,3-edge X-ray Absorption Spectra (XSpectra package);
- Time-Dependent Density Functional Perturbation Theory (TurboTDDFT package);
- Electronic excitations with Many-Body Perturbation Theory, using the YAMBO package;
- Electronic excitations with Many-Body Perturbation Theory (GWL package).

Quantum Transport:

- Ballistic Transport (PWCOND package);
- Coherent Transport from Maximally Localized Wannier Functions, using the WanT code;
- Maximally-localized Wannier functions and transport properties, using the WANNIER90 code.
- Kubo-Greenwood electrical conductivity using the KGEC code.

...and several post processing tools by many research groups!

What is QE

- QUANTUM ESPRESSO is an initiative coordinated by the QUANTUM ESPRESSO Foundation, with the participation of SISSA, CINECA, ICTP, EPFL and *many partners in Europe and worldwide*.

- QUANTUM ESPRESSO is not a single application for quantum simulations; it is rather a distribution of packages performing different tasks and meant to be interoperable.

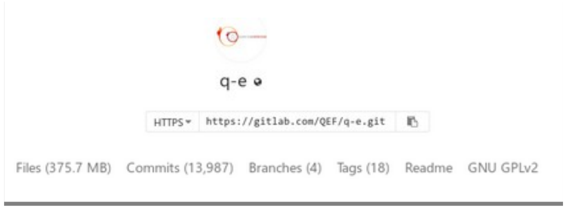
- Free as in GPLv2 and open development.

What is QE

- Runs from standalone workstation to massively parallel systems.

```
$ ./configure && make all
```

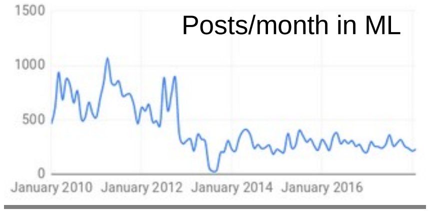
- Large scientific user base, vehicle for new methods, new theories and new science.
  - 600k lines of Fortran
  - V6.5 -> 10k downloads
  - >50 contributors
  - 1600+ registered users
  - …



q-e

HTTPS https://gitlab.com/QEF/q-e.git

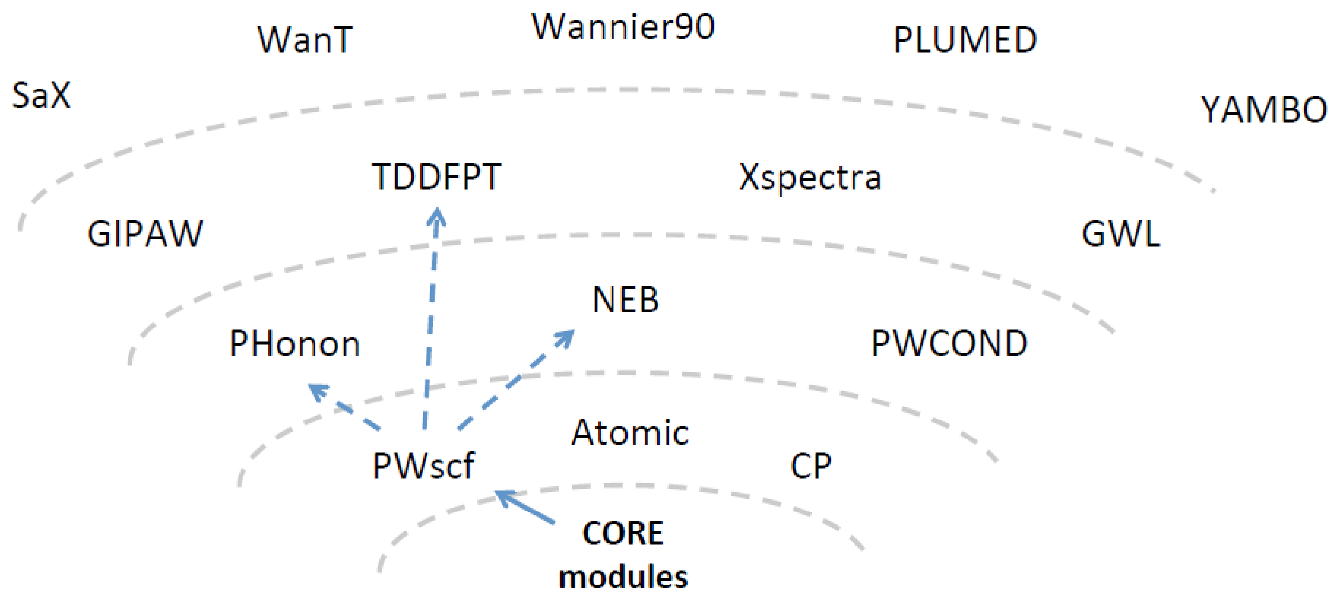
Files (375.7 MB) Commits (13,987) Branches (4) Tags (18) Readme GNU GPLv2



Posts/month in ML

- Simplify transition of new science to HPC environment.

What is QE


WanT
Wannier90
PLUMED
SaX
YAMBO
TDDFPT
Xspectra
GIPAW
GWL
NEB
PHonon
PWCOND
Atomic
PWscf
CP
CORE modules

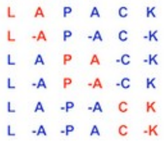
What is QE

Some of the time consuming workloads of many packages are already encapsulated in a number of libraries, namely

LAXLib

FFTXlib

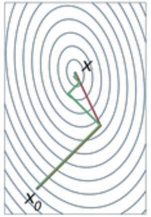
KS_Solvers



X
Y
P0
P1
P2
P3

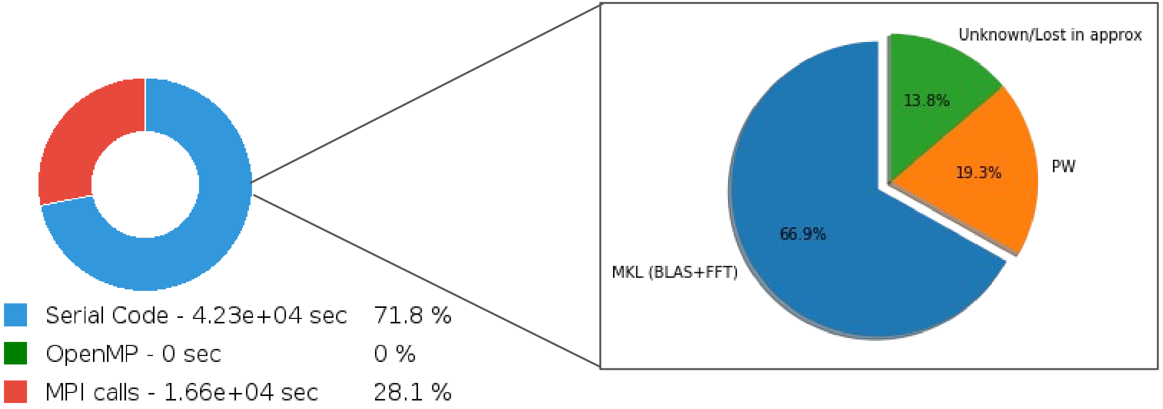
FFTW, MKL, ESSL, ...

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



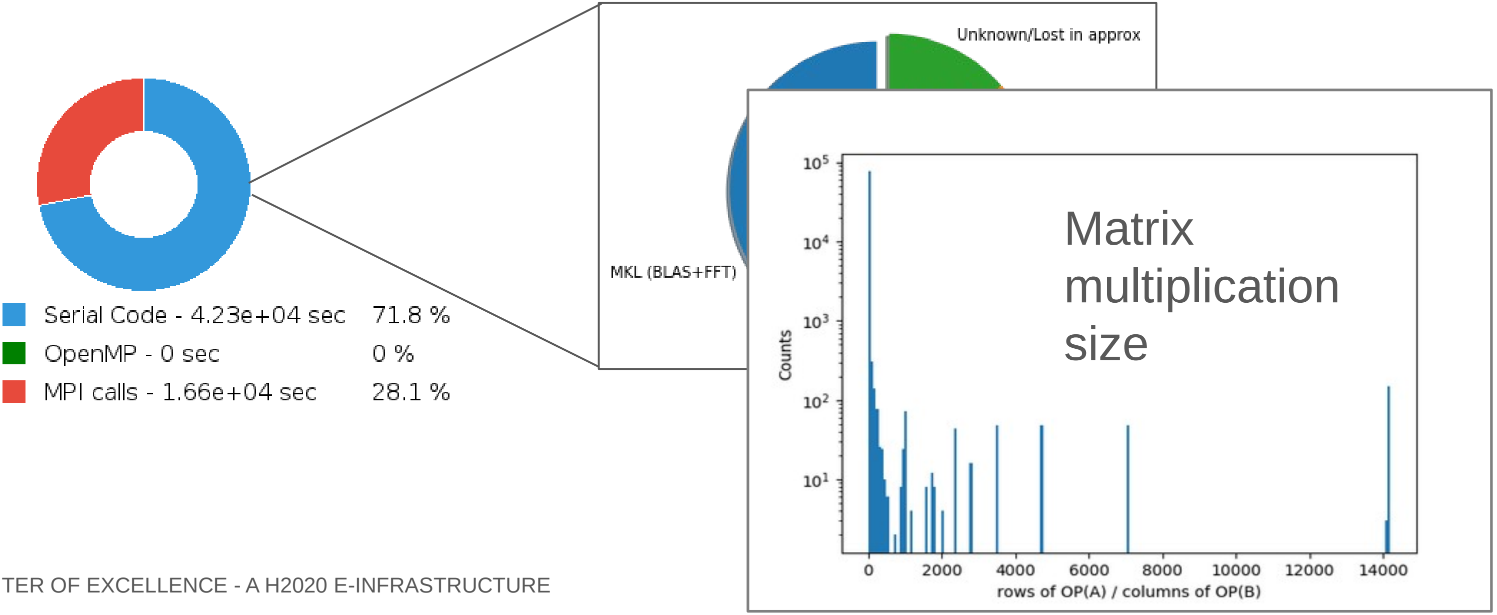
Profiling

PWscf (CPU version) running on a single KNL node with 64 MPI processes (best time to solution).



Profiling

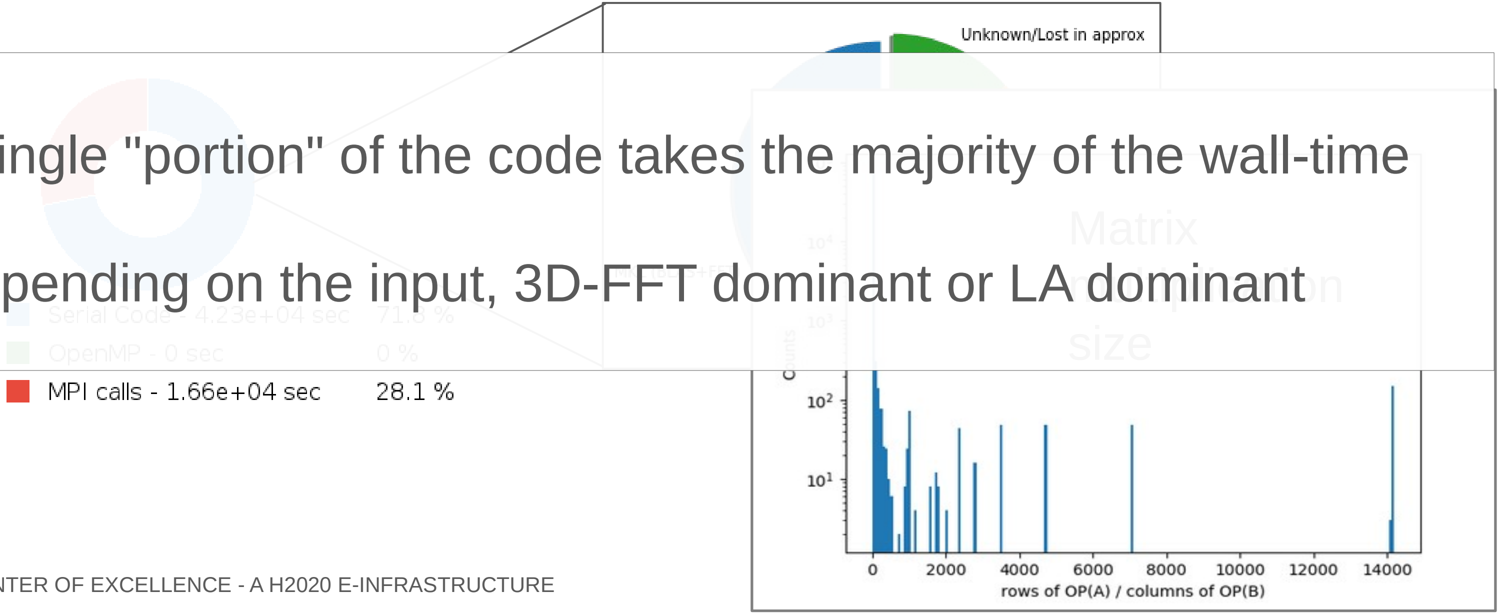
PWscf (CPU version) running on a single KNL node with 64 MPI processes (best time to solution).



Matrix multiplication size

Profiling

PWscf (CPU version) running on a single KNL node with 64 MPI processes (best time to solution).

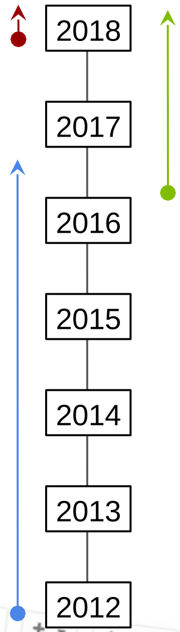


Not a single "portion" of the code takes the majority of the wall-time

Depending on the input, 3D-FFT dominant or LA dominant

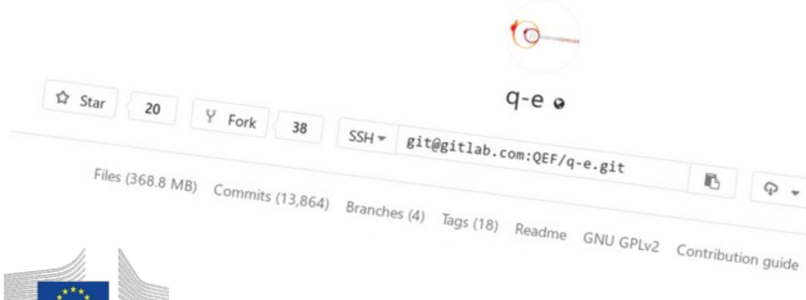
Past and present QE GPU ports

Porting effort carried out by MaX and supported by NVIDIA.



Independent CUDA Fortran based port of QE 6.1 (pw.x) developed by F. Spiga and NVIDIA. Provides much better performance, limited features implemented.

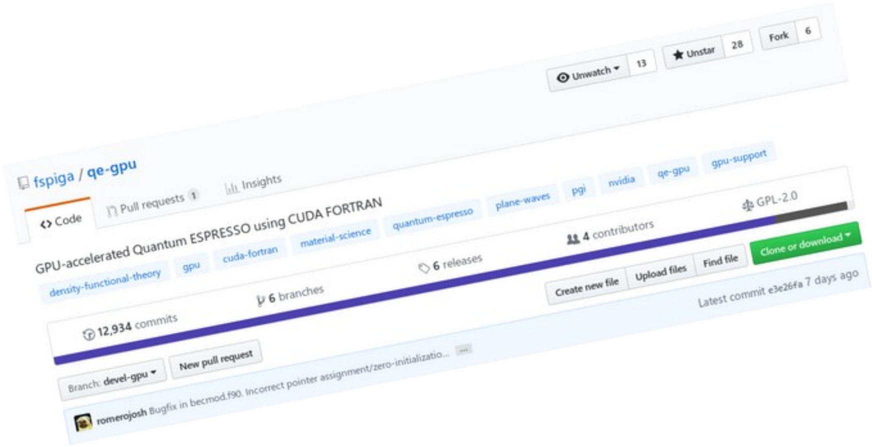
CUDA C based plugin for QE 5.x (pw.x) developed by F. Spiga and I. Girotto.



q-e

Star 20 Fork 38 SSH git@gitlab.com:QEF/q-e.git

Files (368.8 MB) Commits (13,864) Branches (4) Tags (18) Readme GNU GPLv2 Contribution guide CI/CD configuration



fspiga / qe-gpu

GPU-accelerated Quantum ESPRESSO using CUDA FORTRAN

density-functional-theory gpu cuda-fortran material-science quantum-espresso plane-waves pgi nvidia qe-gpu gpu-support

12,934 commits 6 branches 6 releases 4 contributors GPL-2.0

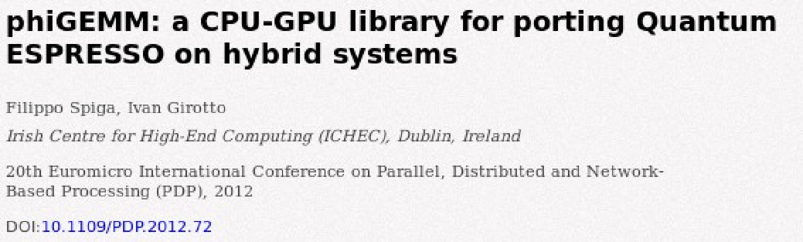
Branch: devel-gpu

romerojosh Bugfix in becmod.f90. Incorrect pointer assignment/zero-initializatio... Latest commit e3e26fa 7 days ago

Commissione europea

QE-GPU-Plugin

✓✓ Self contained



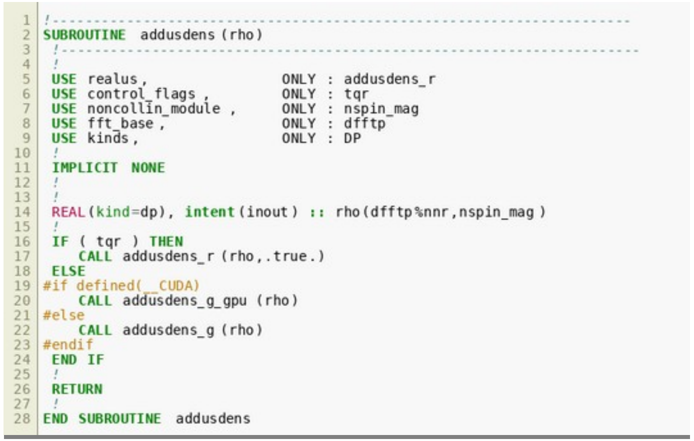
phiGEMM: a CPU-GPU library for porting Quantum ESPRESSO on hybrid systems

Filippo Spiga, Ivan Girotto

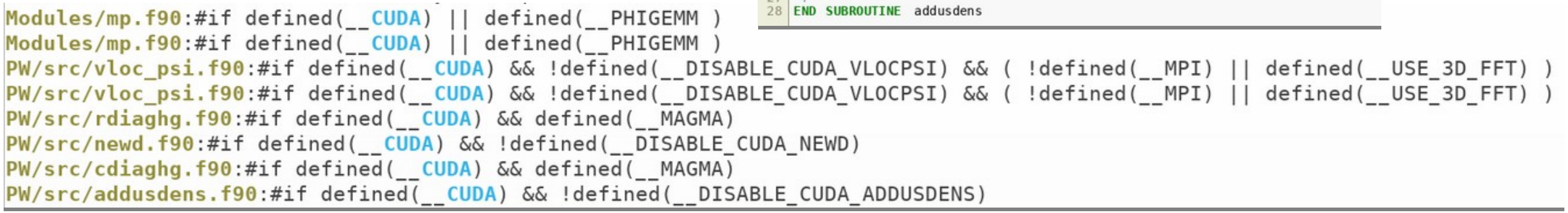
Irish Centre for High-End Computing (ICHEC), Dublin, Ireland

20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2012

DOI:10.1109/PDP.2012.72



```
!-----------------------------------------------------------------------
SUBROUTINE addusdens(rho)
  !----------------------------------------------------------------------
  !
  USE realus,               ONLY : addusdens_r
  USE control_flags,        ONLY : tqr
  USE noncollin_module,     ONLY : nspin_mag
  USE fft_base,             ONLY : dfftp
  USE kinds,                ONLY : DP
  !
  IMPLICIT NONE
  !
  !
  REAL(kind=dp), intent(inout) :: rho(dfftp%nnr,nspin_mag)
  !
  IF ( tqr ) THEN
     CALL addusdens_r(rho,.true.)
  ELSE
#if defined(__CUDA)
     CALL addusdens_g_gpu(rho)
#else
     CALL addusdens_g(rho)
#endif
  END IF
  !
  RETURN
  !
END SUBROUTINE addusdens
```

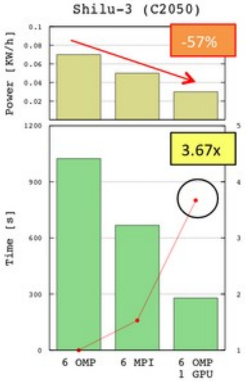


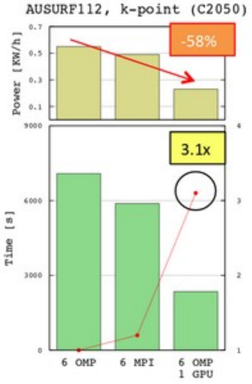
```
Modules/mp.f90:#if defined(__CUDA) || defined(__PHIGEMM )
Modules/mp.f90:#if defined(__CUDA) || defined(__PHIGEMM )
PW/src/vloc_psi.f90:#if defined(__CUDA) && !defined(__DISABLE_CUDA_VLOCPSI) && ( !defined(__MPI) || defined(__USE_3D_FFT) )
PW/src/vloc_psi.f90:#if defined(__CUDA) && !defined(__DISABLE_CUDA_VLOCPSI) && ( !defined(__MPI) || defined(__USE_3D_FFT) )
PW/src/rdiaghg.f90:#if defined(__CUDA) && defined(__MAGMA)
PW/src/newd.f90:#if defined(__CUDA) && !defined(__DISABLE_CUDA_NEWD)
PW/src/cdiaghg.f90:#if defined(__CUDA) && defined(__MAGMA)
PW/src/addusdens.f90:#if defined(__CUDA) && !defined(__DISABLE_CUDA_ADDUSDENS)
```

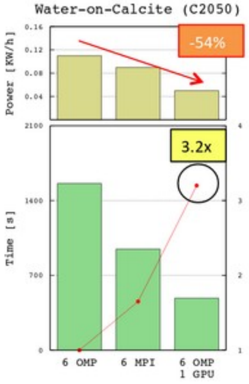
QE-GPU-Plugin

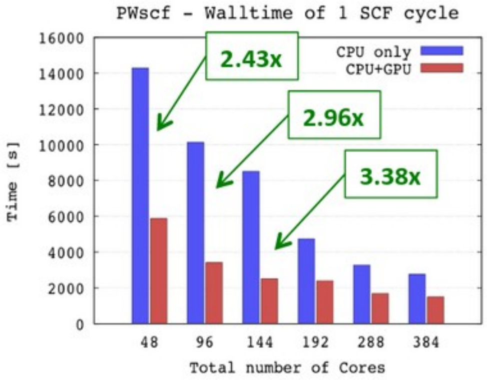
✓✓ Self contained

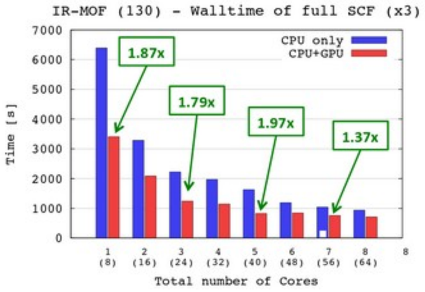
✓ Good performance

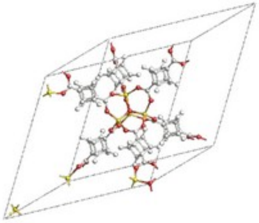








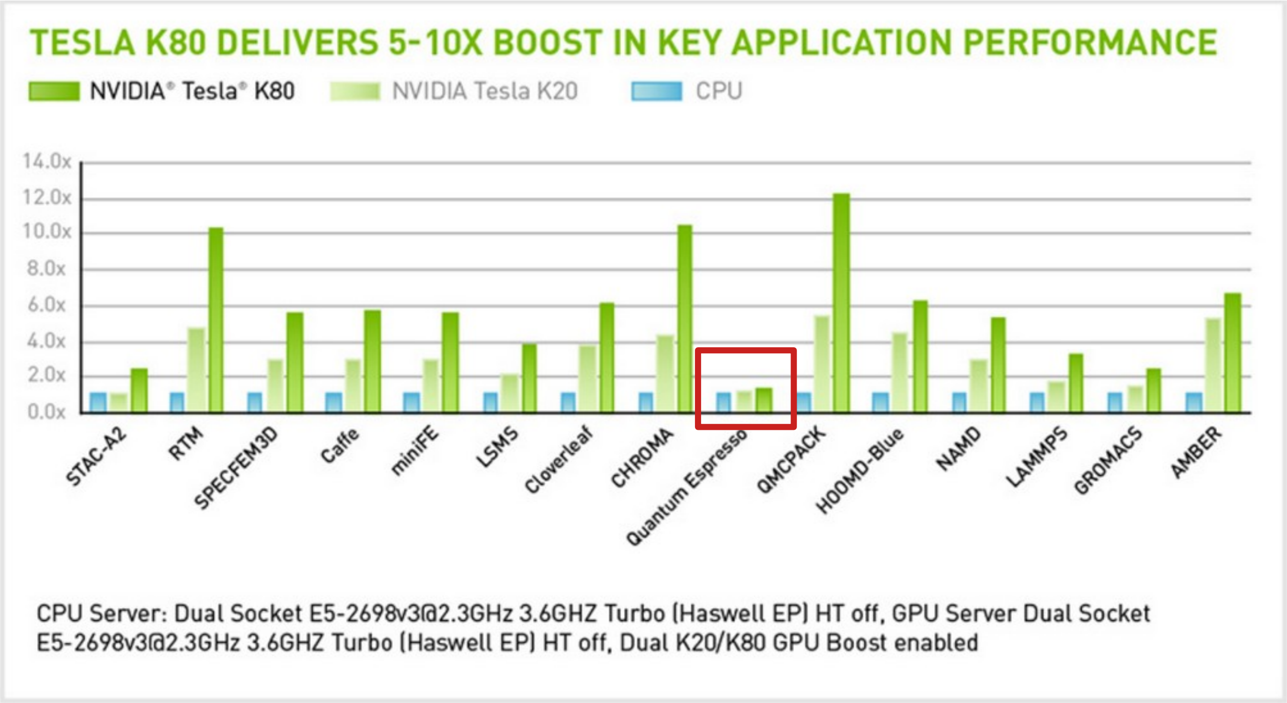




QE-GPU-Plugin

✓✓ Self contained

✓ Good performance



TESLA K80 DELIVERS 5-10X BOOST IN KEY APPLICATION PERFORMANCE

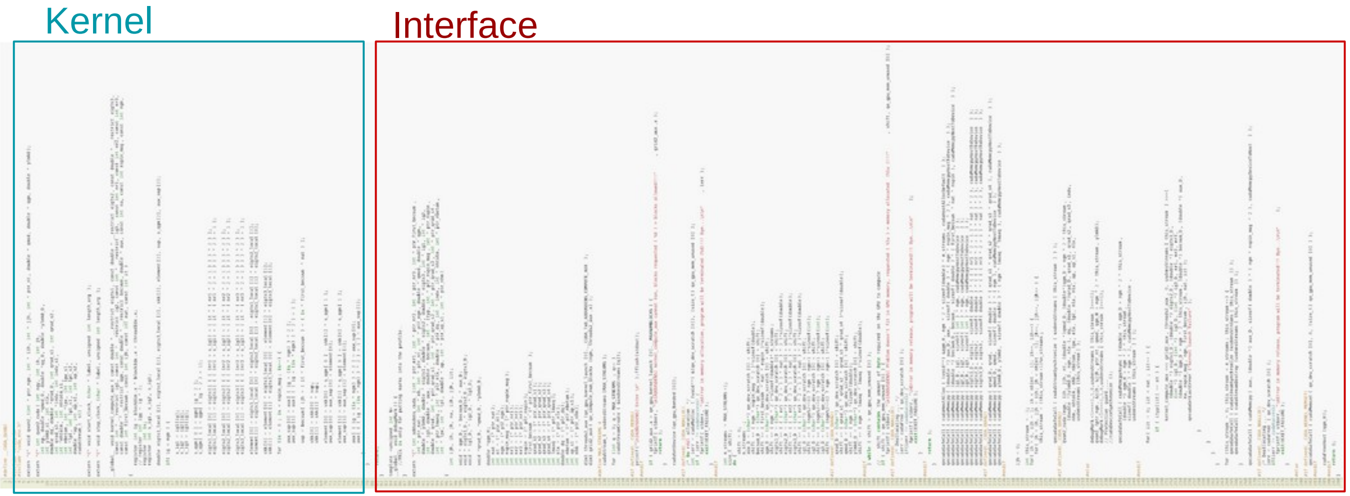
CPU Server: Dual Socket E5-2698v3@2.3GHz 3.6GHZ Turbo (Haswell EP) HT off, GPU Server Dual Socket E5-2698v3@2.3GHz 3.6GHZ Turbo (Haswell EP) HT off, Dual K20/K80 GPU Boost enabled

QE-GPU-Plugin

✓✓ Self contained

✓ Good performance

✗ Boilerplate code



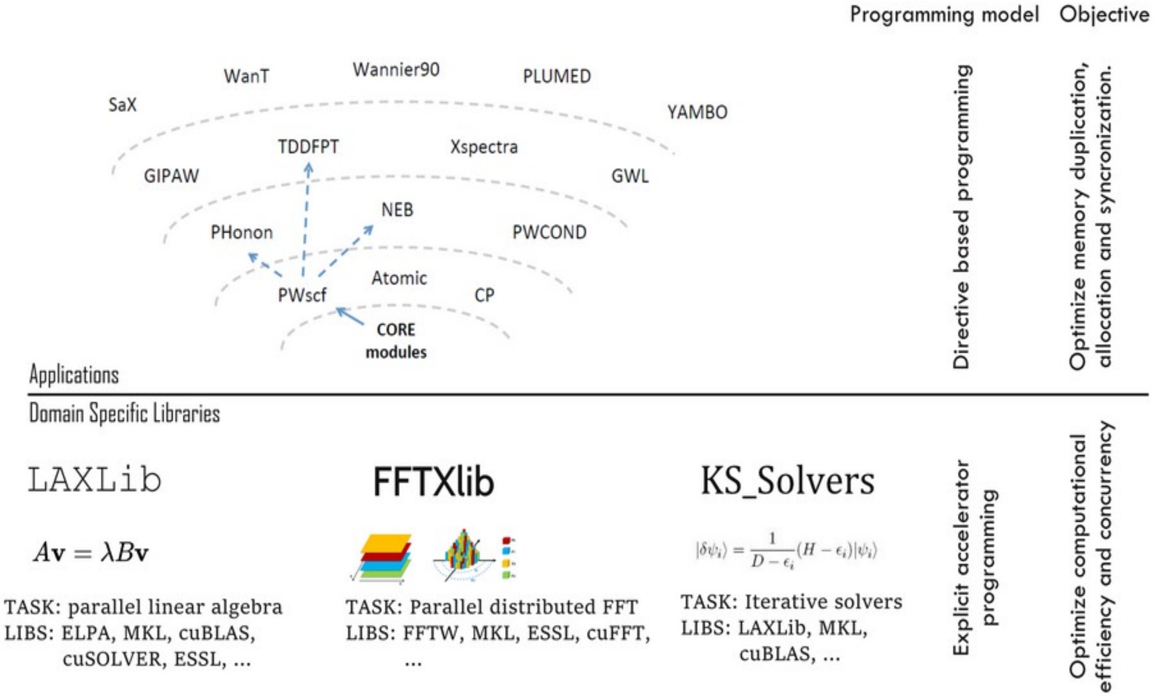
Porting strategy

Desiderata:

- Fortran: CUDA Fortran / OpenACC / OpenMP ≥ 4.5.
- Explicit memory management:
  - derived types
  - generic hybrid architecture support.
- Preserve modularity.
- Preserve user experience.

- Prepare full featured and well tested **CUDA-enabled libraries** containing performance critical kernels.
- **Directive based acceleration** of QE’s applications.

Porting strategy


Programming model
Objective
WanT
Wannier90
PLUMED
SaX
YAMBO
TDDFPT
Xspectra
GIPAW
GWL
NEB
PHonon
PWCOND
Atomic
PWscf
CP
CORE modules
Applications
Domain Specific Libraries
Directive based programming
Optimize memory duplication, allocation and syncronization.
Explicit accelerator programming
Optimize computational efficiency and concurrency
LAXLib
FFTXlib
KS_Solvers

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

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

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

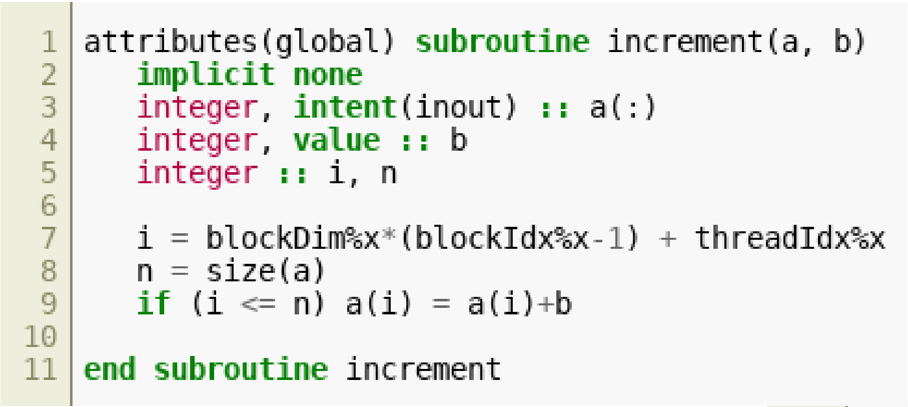
$$|\delta\psi_i\rangle = \frac{1}{D-\epsilon_i}(H-\epsilon_i)|\psi_i\rangle$$

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

CUDA Fortran

- 😁 **Fortran** equivalent for CUDA C++
- 😁 Syntax is similar to CUDA, but more concise.
- 😁 Full set of libraries and **interfaces**.
- ☹ Complete syntax only on PGI compilers.
- ☹ Partial implementation on IBM compilers, useless for QE.

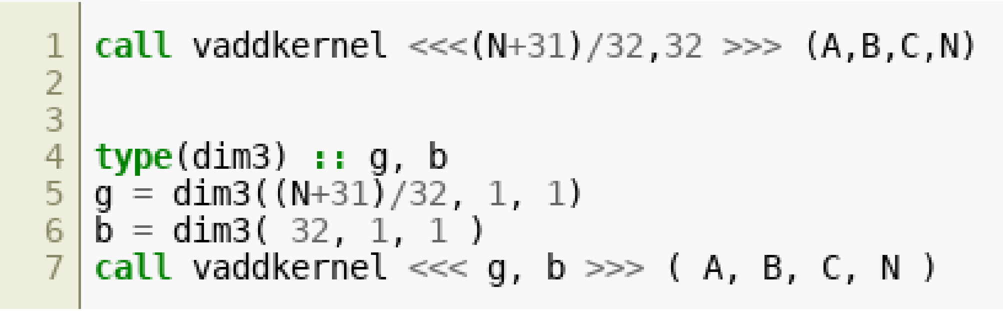
CUDA Fortran



```fortran
attributes(global) subroutine increment(a, b)
   implicit none
   integer, intent(inout) :: a(:)
   integer, value :: b
   integer :: i, n

   i = blockDim%x*(blockIdx%x-1) + threadIdx%x
   n = size(a)
   if (i <= n) a(i) = a(i)+b

end subroutine increment
```

CUDA kernels in Fortran

CUDA equivalent syntax



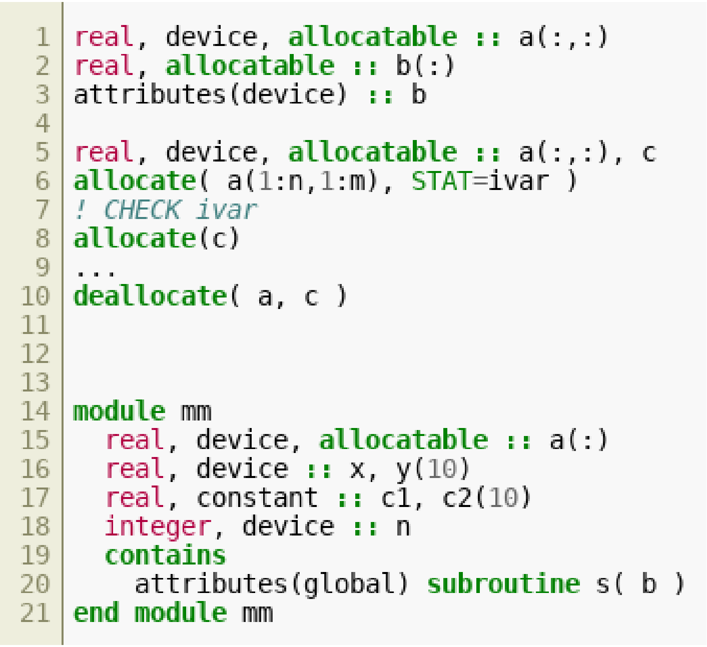
```fortran
call vaddkernel <<<(N+31)/32,32 >>> (A,B,C,N)


type(dim3) :: g, b
g = dim3((N+31)/32, 1, 1)
b = dim3( 32, 1, 1 )
call vaddkernel <<< g, b >>> ( A, B, C, N )
```

Full support for Fortran intrinsic types.

CUDA Fortran

Allocation done by the host, according to “device” attribute

Just copy (no need for cuda APIs for sync. copies).

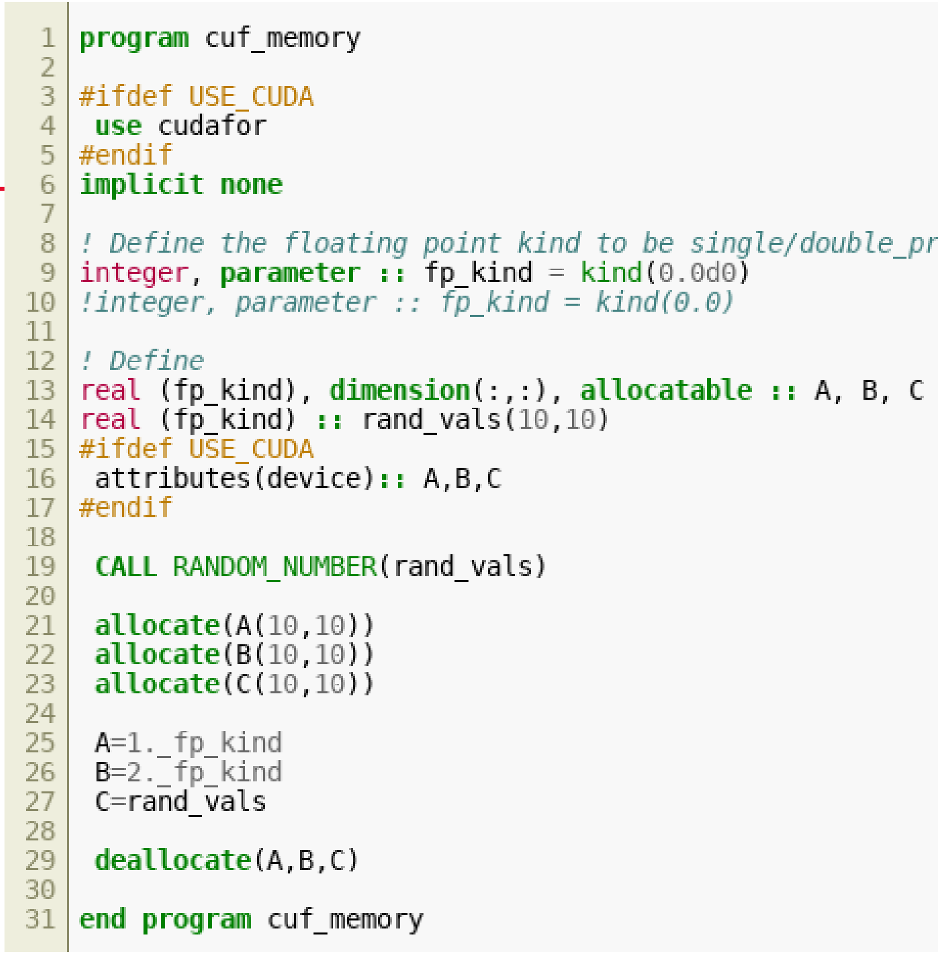


```fortran
real, device, allocatable :: a(:,:)
real, allocatable :: b(:)
attributes(device) :: b

real, device, allocatable :: a(:,:), c
allocate( a(1:n,1:m), STAT=ivar )
! CHECK ivar
allocate(c)
...
deallocate( a, c )



module mm
  real, device, allocatable :: a(:)
  real, device :: x, y(10)
  real, constant :: c1, c2(10)
  integer, device :: n
  contains
    attributes(global) subroutine s( b )
end module mm
```

CUDA Fortran

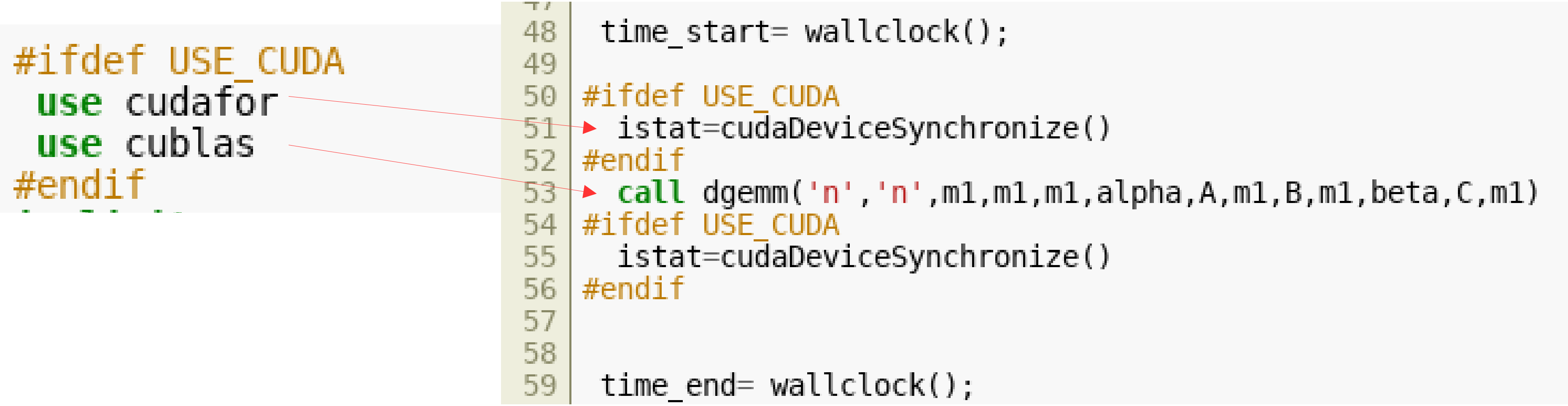
Allocation done by the host, according to “device” attribute

Just copy (no need for cuda APIs for sync. copies).



```fortran
program cuf_memory

#ifdef USE_CUDA
 use cudafor
#endif
implicit none

! Define the floating point kind to be single/double_pr
integer, parameter :: fp_kind = kind(0.0d0)
!integer, parameter :: fp_kind = kind(0.0)

! Define
real (fp_kind), dimension(:,:), allocatable :: A, B, C
real (fp_kind) :: rand_vals(10,10)
#ifdef USE_CUDA
 attributes(device):: A,B,C
#endif

 CALL RANDOM_NUMBER(rand_vals)

 allocate(A(10,10))
 allocate(B(10,10))
 allocate(C(10,10))

 A=1._fp_kind
 B=2._fp_kind
 C=rand_vals

 deallocate(A,B,C)

end program cuf_memory
```

CUDA Fortran



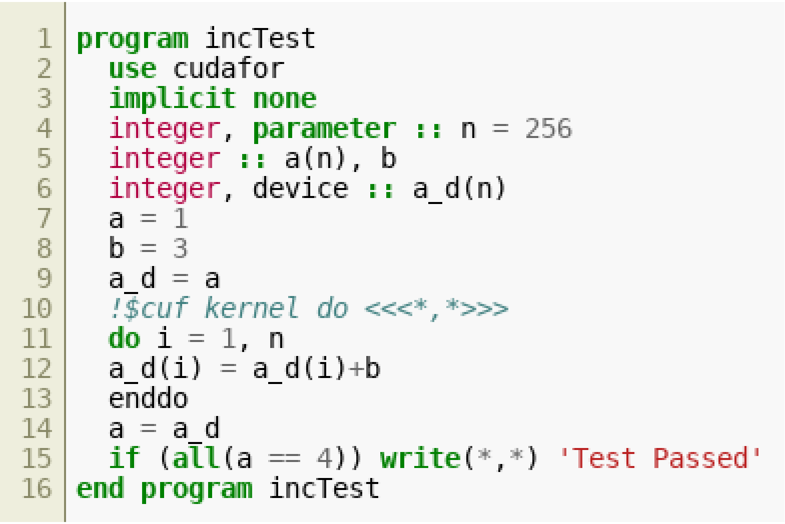
```
#ifdef USE_CUDA
 use cudafor
 use cublas
#endif
```

```
 time_start= wallclock();

#ifdef USE_CUDA
  istat=cudaDeviceSynchronize()
#endif
  call dgemm('n','n',m1,m1,m1,alpha,A,m1,B,m1,beta,C,m1)
#ifdef USE_CUDA
  istat=cudaDeviceSynchronize()
#endif


 time_end= wallclock();
```

Fortran interfaces for most (all?!) NVIDIA Cuda Runtime and NVIDIA Libraries.

CUDA Fortran

Cuf kernels, directive based automatic kernel generation:



```
program incTest
  use cudafor
  implicit none
  integer, parameter :: n = 256
  integer :: a(n), b
  integer, device :: a_d(n)
  a = 1
  b = 3
  a_d = a
  !$cuf kernel do <<<*,*>>>
  do i = 1, n
  a_d(i) = a_d(i)+b
  enddo
  a = a_d
  if (all(a == 4)) write(*,*) 'Test Passed'
end program incTest
```

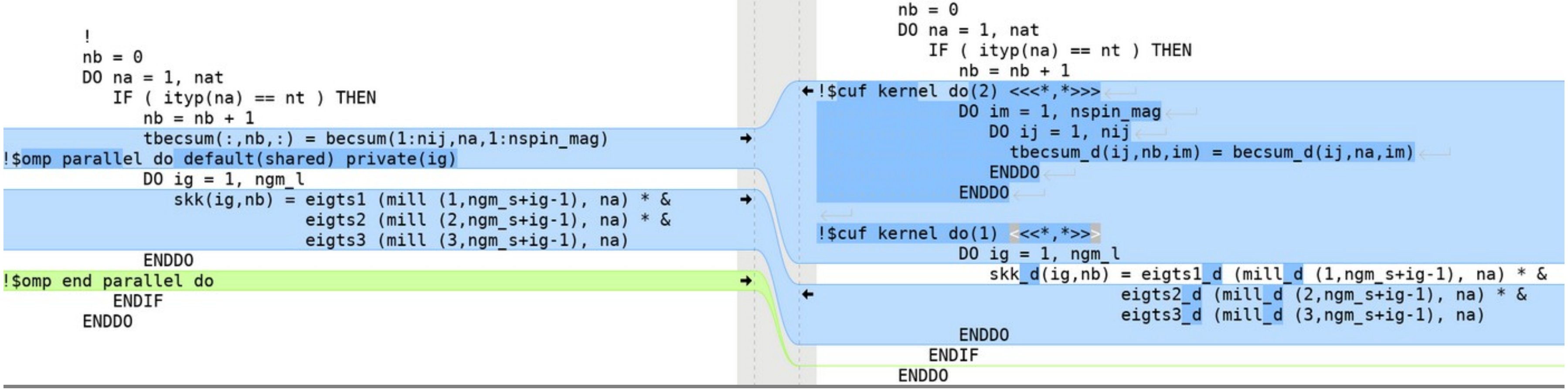
CUDA Fortran

Cuf kernels, a few simple rules:

- Scalars are private by default
- Reduction automatically detected (only scalars)
- Only loop based constructs (possibly nested)

QE-GPU CUDA Fortran

✓ Single programming language: Fortran + CUDA Fortran

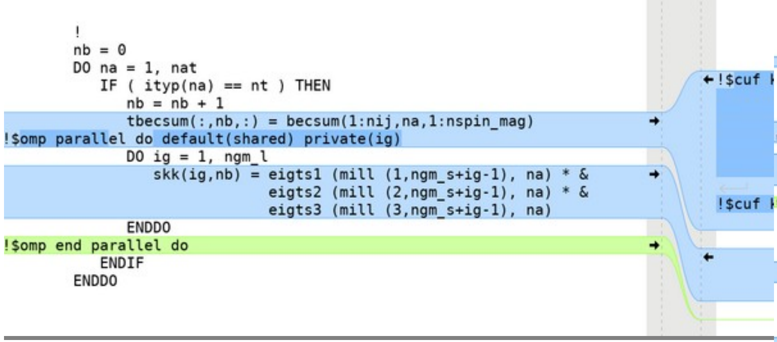


```
        !
        nb = 0
        DO na = 1, nat
           IF ( ityp(na) == nt ) THEN
              nb = nb + 1
              tbecsum(:,nb,:) = becsum(1:nij,na,1:nspin_mag)
!$omp parallel do default(shared) private(ig)
              DO ig = 1, ngm_l
                 skk(ig,nb) = eigts1 (mill (1,ngm_s+ig-1), na) * &
                              eigts2 (mill (2,ngm_s+ig-1), na) * &
                              eigts3 (mill (3,ngm_s+ig-1), na)
              ENDDO
!$omp end parallel do
           ENDIF
        ENDDO
```

```
        nb = 0
        DO na = 1, nat
           IF ( ityp(na) == nt ) THEN
              nb = nb + 1
!$cuf kernel do(2) <<<*,*>>>
              DO im = 1, nspin_mag
                 DO ij = 1, nij
                   tbecsum_d(ij,nb,im) = becsum_d(ij,na,im)
                 ENDDO
              ENDDO

!$cuf kernel do(1) <<<*,*>>>
              DO ig = 1, ngm_l
                 skk_d(ig,nb) = eigts1_d (mill_d (1,ngm_s+ig-1), na) * &
                                eigts2_d (mill_d (2,ngm_s+ig-1), na) * &
                                eigts3_d (mill_d (3,ngm_s+ig-1), na)
              ENDDO
           ENDIF
        ENDDO
```

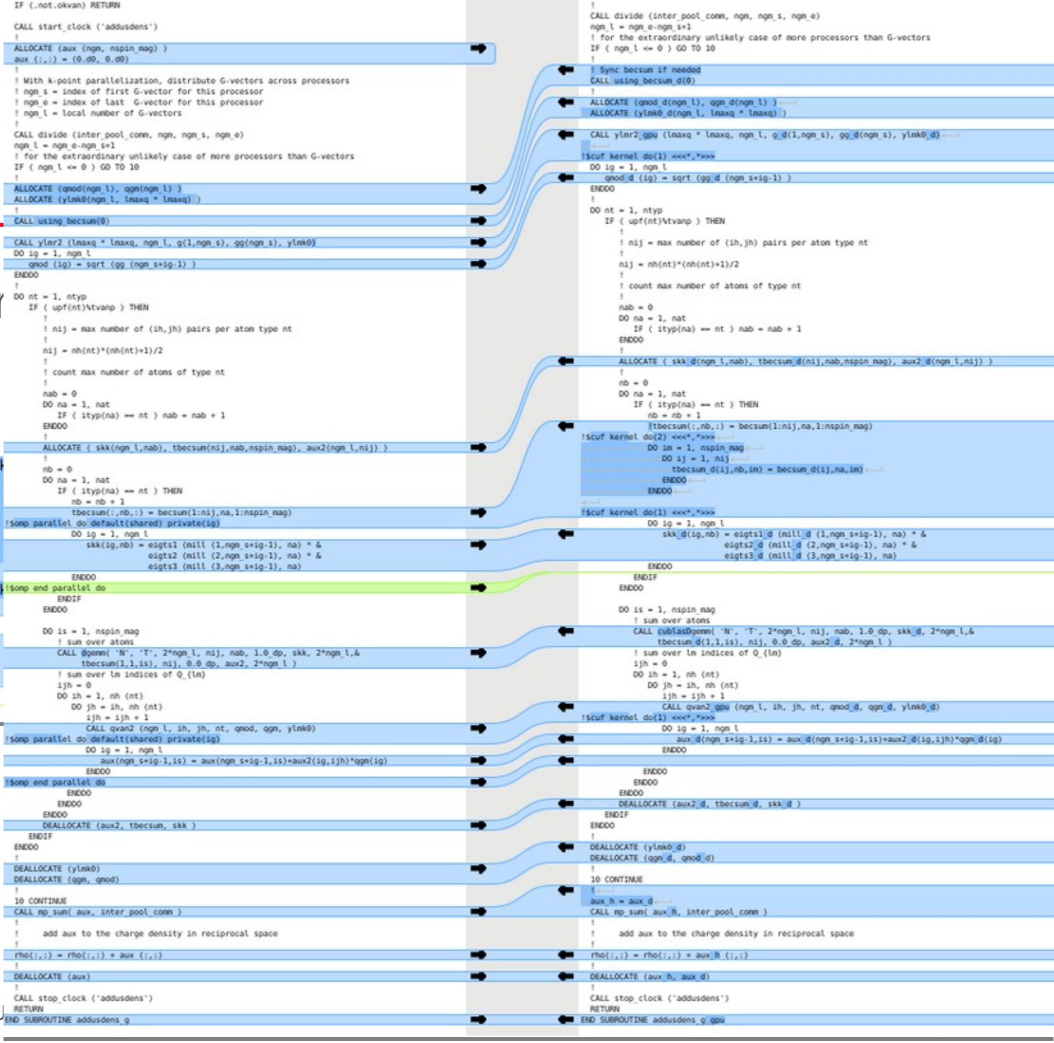
QE-GPU CUDA Fortran

✓ Single programming language: For



```
      !
      nb = 0
      DO na = 1, nat
         IF ( ityp(na) == nt ) THEN
            nb = nb + 1
            tbecsum(:,nb,:) = becsum(1:nij,na,1:nspin_mag)
!$omp parallel do default(shared) private(ig)
            DO ig = 1, ngm_l
               skk(ig,nb) = eigts1 (mill (1,ngm_s+ig-1), na) * &
                            eigts2 (mill (2,ngm_s+ig-1), na) * &
                            eigts3 (mill (3,ngm_s+ig-1), na)
            ENDDO
!$omp end parallel do
         ENDIF
      ENDDO
```

```
!$cuf k
!$cuf k
```



```
IF (.not.okvan) RETURN

CALL start_clock ('addusdens')
!
ALLOCATE (aux (ngm, nspin_mag) )
aux (:,:) = (0.d0, 0.d0)
!
! With k-point parallelization, distribute G-vectors across processors
! ngm_s = index of first G-vector for this processor
! ngm_e = index of last  G-vector for this processor
! ngm_l = local number of G-vectors
!
CALL divide (inter_pool_comm, ngm, ngm_s, ngm_e)
ngm_l = ngm_e-ngm_s+1
! for the extraordinary unlikely case of more processors than G-vectors
IF ( ngm_l <= 0 ) GO TO 10
!
ALLOCATE (qmod(ngm_l), qgm(ngm_l) )
ALLOCATE (ylmk0(ngm_l, lmaxq * lmaxq) )
!
CALL using_becsum(0)

CALL ylmr2 (lmaxq * lmaxq, ngm_l, g(1,ngm_s), gg(ngm_s), ylmk0)
DO ig = 1, ngm_l
   qmod (ig) = sqrt (gg (ngm_s+ig-1) )
ENDDO
!
DO nt = 1, ntyp
   IF ( upf(nt)%tvanp ) THEN
      !
      ! nij = max number of (ih,jh) pairs per atom type nt
      !
      nij = nh(nt)*(nh(nt)+1)/2
      !
      ! count max number of atoms of type nt
      !
      nab = 0
      DO na = 1, nat
         IF ( ityp(na) == nt ) nab = nab + 1
      ENDDO
      !
      ALLOCATE ( skk(ngm_l,nab), tbecsum(nij,nab,nspin_mag), aux2(ngm_l,nij) )
      !
      nb = 0
      DO na = 1, nat
         IF ( ityp(na) == nt ) THEN
            nb = nb + 1
            tbecsum(:,nb,:) = becsum(1:nij,na,1:nspin_mag)
!$omp parallel do default(shared) private(ig)
            DO ig = 1, ngm_l
               skk(ig,nb) = eigts1 (mill (1,ngm_s+ig-1), na) * &
                            eigts2 (mill (2,ngm_s+ig-1), na) * &
                            eigts3 (mill (3,ngm_s+ig-1), na)
            ENDDO
!$omp end parallel do
         ENDIF
      ENDDO

      DO is = 1, nspin_mag
         ! sum over atoms
         CALL dgemm( 'N', 'T', 2*ngm_l, nij, nab, 1.0_dp, skk, 2*ngm_l,&
              tbecsum(1,1,is), nij, 0.0_dp, aux2, 2*ngm_l )
         ! sum over lm indices of Q_(lm)
         ijh = 0
         DO ih = 1, nh (nt)
            DO jh = ih, nh (nt)
               ijh = ijh + 1
               CALL qvan2 (ngm_l, ih, jh, nt, qmod, qgm, ylmk0)
!$omp parallel do default(shared) private(ig)
               DO ig = 1, ngm_l
                  aux(ngm_s+ig-1,is) = aux(ngm_s+ig-1,is)+aux2(ig,ijh)*qgm(ig)
               ENDDO
!$omp end parallel do
            ENDDO
         ENDDO
      ENDDO
      DEALLOCATE (aux2, tbecsum, skk )
   ENDIF
ENDDO
!
DEALLOCATE (ylmk0)
DEALLOCATE (qgm, qmod)
!
10 CONTINUE
CALL mp_sum( aux, inter_pool_comm )
!
!     add aux to the charge density in reciprocal space
!
rho(:,:) = rho(:,:) + aux (:,:)
!
DEALLOCATE (aux)
!
CALL stop_clock ('addusdens')
RETURN
END SUBROUTINE addusdens_g
```

```
!
CALL divide (inter_pool_comm, ngm, ngm_s, ngm_e)
ngm_l = ngm_e-ngm_s+1
! for the extraordinary unlikely case of more processors than G-vectors
IF ( ngm_l <= 0 ) GO TO 10
!
! Sync becsum if needed
CALL using_becsum_d(0)
!
ALLOCATE (qmod_d(ngm_l), qgm_d(ngm_l) )
ALLOCATE (ylmk0_d(ngm_l, lmaxq * lmaxq) )

CALL ylmr2_gpu (lmaxq * lmaxq, ngm_l, g_d(1,ngm_s), gg_d(ngm_s), ylmk0_d)

!$cuf kernel do(1) <<<*,*>>>
DO ig = 1, ngm_l
   qmod_d (ig) = sqrt (gg_d (ngm_s+ig-1) )
ENDDO
!
DO nt = 1, ntyp
   IF ( upf(nt)%tvanp ) THEN
      !
      ! nij = max number of (ih,jh) pairs per atom type nt
      !
      nij = nh(nt)*(nh(nt)+1)/2
      !
      ! count max number of atoms of type nt
      !
      nab = 0
      DO na = 1, nat
         IF ( ityp(na) == nt ) nab = nab + 1
      ENDDO
      !
      ALLOCATE ( skk_d(ngm_l,nab), tbecsum_d(nij,nab,nspin_mag), aux2_d(ngm_l,nij) )
      !
      nb = 0
      DO na = 1, nat
         IF ( ityp(na) == nt ) THEN
            nb = nb + 1
            !tbecsum(:,nb,:) = becsum(1:nij,na,1:nspin_mag)
!$cuf kernel do(2) <<<*,*>>>
            DO im = 1, nspin_mag
               DO ij = 1, nij
                  tbecsum_d(ij,nb,im) = becsum_d(ij,na,im)
               ENDDO
            ENDDO

!$cuf kernel do(1) <<<*,*>>>
            DO ig = 1, ngm_l
               skk_d(ig,nb) = eigts1_d (mill_d (1,ngm_s+ig-1), na) * &
                              eigts2_d (mill_d (2,ngm_s+ig-1), na) * &
                              eigts3_d (mill_d (3,ngm_s+ig-1), na)
            ENDDO
         ENDIF
      ENDDO

      DO is = 1, nspin_mag
         ! sum over atoms
         CALL cublasDgemm( 'N', 'T', 2*ngm_l, nij, nab, 1.0_dp, skk_d, 2*ngm_l,&
              tbecsum_d(1,1,is), nij, 0.0_dp, aux2_d, 2*ngm_l )
         ! sum over lm indices of Q_(lm)
         ijh = 0
         DO ih = 1, nh (nt)
            DO jh = ih, nh (nt)
               ijh = ijh + 1
               CALL qvan2_gpu (ngm_l, ih, jh, nt, qmod_d, qgm_d, ylmk0_d)
!$cuf kernel do(1) <<<*,*>>>
               DO ig = 1, ngm_l
                  aux_d(ngm_s+ig-1,is) = aux_d(ngm_s+ig-1,is)+aux2_d(ig,ijh)*qgm_d(ig)
               ENDDO

            ENDDO
         ENDDO
      ENDDO
      DEALLOCATE (aux2_d, tbecsum_d, skk_d )
   ENDIF
ENDDO
!
DEALLOCATE (ylmk0_d)
DEALLOCATE (qgm_d, qmod_d)
!
10 CONTINUE
!
aux_h = aux_d
CALL mp_sum( aux_h, inter_pool_comm )
!
!     add aux to the charge density in reciprocal space
!
rho(:,:) = rho(:,:) + aux_h (:,:)
!
DEALLOCATE (aux_h, aux_d)
!
CALL stop_clock ('addusdens')
RETURN
END SUBROUTINE addusdens_g_gpu
```

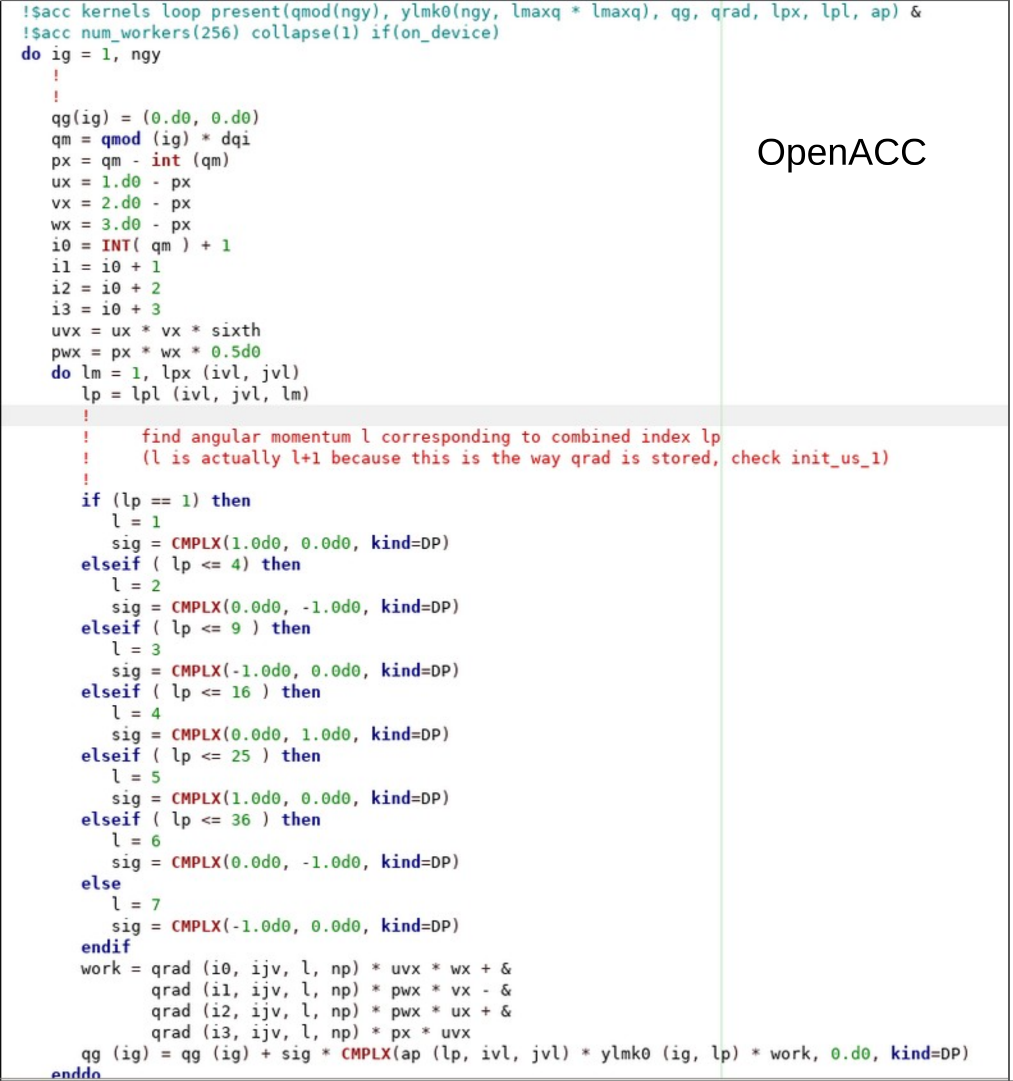
QE-GPU CUDA Fortran

✓ Single programming language: Fortran + CUDA Fortran

✓ Very good performance (shown at the end of this presentation).

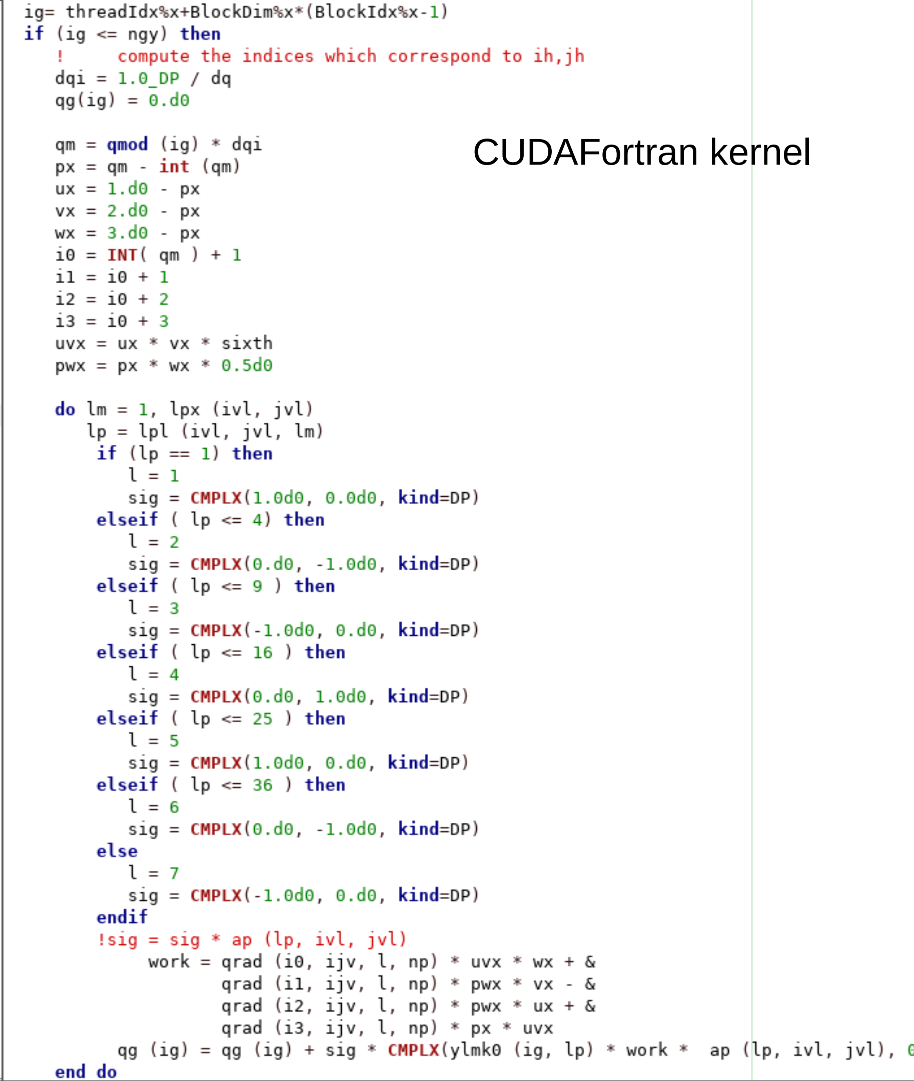
✗ Code duplication:

- Performance (eg. preserve cache blocking optimizations)
- Preserve both CPU and GPU version.



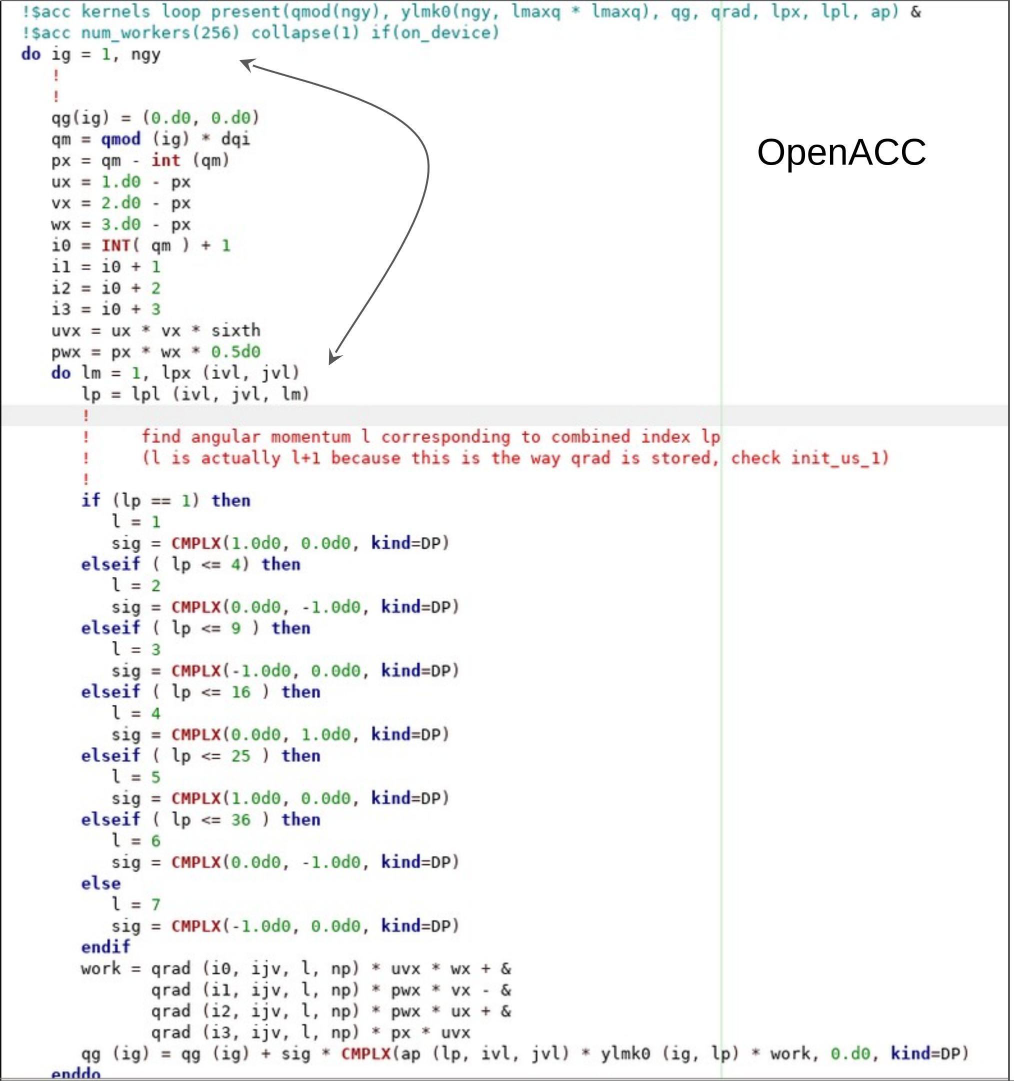
OpenACC

```
!$acc kernels loop present(qmod(ngy), ylmk0(ngy, lmaxq * lmaxq), qg, qrad, lpx, lpl, ap) &
!$acc num_workers(256) collapse(1) if(on_device)
do ig = 1, ngy
   !
   !
   qg(ig) = (0.d0, 0.d0)
   qm = qmod (ig) * dqi
   px = qm - int (qm)
   ux = 1.d0 - px
   vx = 2.d0 - px
   wx = 3.d0 - px
   i0 = INT( qm ) + 1
   i1 = i0 + 1
   i2 = i0 + 2
   i3 = i0 + 3
   uvx = ux * vx * sixth
   pwx = px * wx * 0.5d0
   do lm = 1, lpx (ivl, jvl)
      lp = lpl (ivl, jvl, lm)
      !
      !     find angular momentum l corresponding to combined index lp
      !     (l is actually l+1 because this is the way qrad is stored, check init_us_1)
      !
      if (lp == 1) then
         l = 1
         sig = CMPLX(1.0d0, 0.0d0, kind=DP)
      elseif ( lp <= 4) then
         l = 2
         sig = CMPLX(0.0d0, -1.0d0, kind=DP)
      elseif ( lp <= 9 ) then
         l = 3
         sig = CMPLX(-1.0d0, 0.0d0, kind=DP)
      elseif ( lp <= 16 ) then
         l = 4
         sig = CMPLX(0.0d0, 1.0d0, kind=DP)
      elseif ( lp <= 25 ) then
         l = 5
         sig = CMPLX(1.0d0, 0.0d0, kind=DP)
      elseif ( lp <= 36 ) then
         l = 6
         sig = CMPLX(0.0d0, -1.0d0, kind=DP)
      else
         l = 7
         sig = CMPLX(-1.0d0, 0.0d0, kind=DP)
      endif
      work = qrad (i0, ijv, l, np) * uvx * wx + &
             qrad (i1, ijv, l, np) * pwx * vx - &
             qrad (i2, ijv, l, np) * pwx * ux + &
             qrad (i3, ijv, l, np) * px * uvx
      qg (ig) = qg (ig) + sig * CMPLX(ap (lp, ivl, jvl) * ylmk0 (ig, lp) * work, 0.d0, kind=DP)
   enddo
```



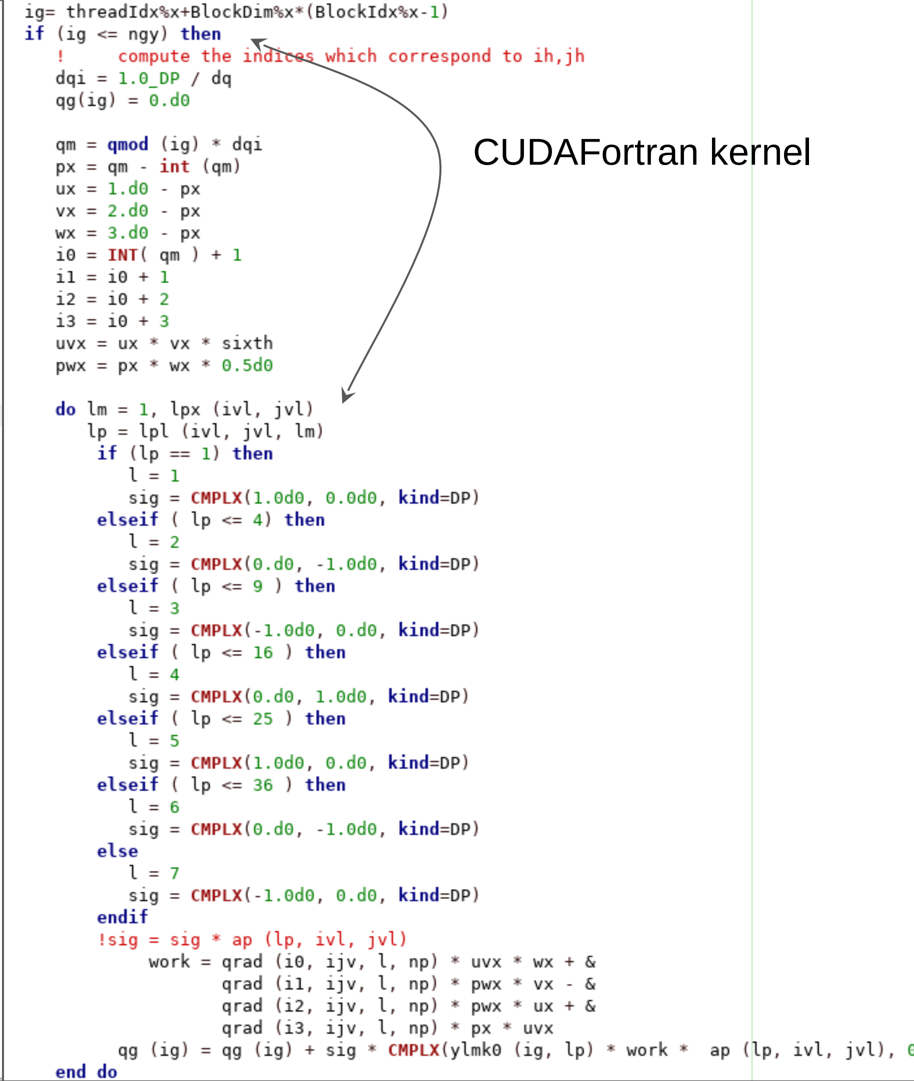
CUDAFortran kernel

```
ig= threadIdx%x+BlockDim%x*(BlockIdx%x-1)
if (ig <= ngy) then
   !     compute the indices which correspond to ih,jh
   dqi = 1.0_DP / dq
   qg(ig) = 0.d0

   qm = qmod (ig) * dqi
   px = qm - int (qm)
   ux = 1.d0 - px
   vx = 2.d0 - px
   wx = 3.d0 - px
   i0 = INT( qm ) + 1
   i1 = i0 + 1
   i2 = i0 + 2
   i3 = i0 + 3
   uvx = ux * vx * sixth
   pwx = px * wx * 0.5d0

   do lm = 1, lpx (ivl, jvl)
      lp = lpl (ivl, jvl, lm)
       if (lp == 1) then
          l = 1
          sig = CMPLX(1.0d0, 0.0d0, kind=DP)
       elseif ( lp <= 4) then
          l = 2
          sig = CMPLX(0.d0, -1.0d0, kind=DP)
       elseif ( lp <= 9 ) then
          l = 3
          sig = CMPLX(-1.0d0, 0.d0, kind=DP)
       elseif ( lp <= 16 ) then
          l = 4
          sig = CMPLX(0.d0, 1.0d0, kind=DP)
       elseif ( lp <= 25 ) then
          l = 5
          sig = CMPLX(1.0d0, 0.d0, kind=DP)
       elseif ( lp <= 36 ) then
          l = 6
          sig = CMPLX(0.d0, -1.0d0, kind=DP)
       else
          l = 7
          sig = CMPLX(-1.0d0, 0.d0, kind=DP)
       endif
       !sig = sig * ap (lp, ivl, jvl)
            work = qrad (i0, ijv, l, np) * uvx * wx + &
                   qrad (i1, ijv, l, np) * pwx * vx - &
                   qrad (i2, ijv, l, np) * pwx * ux + &
                   qrad (i3, ijv, l, np) * px * uvx
         qg (ig) = qg (ig) + sig * CMPLX(ylmk0 (ig, lp) * work *  ap (lp, ivl, jvl), 0
   end do
```



OpenACC

```
!$acc kernels loop present(qmod(ngy), ylmk0(ngy, lmaxq * lmaxq), qg, qrad, lpx, lpl, ap) &
!$acc num_workers(256) collapse(1) if(on_device)
do ig = 1, ngy
   !
   !
   qg(ig) = (0.d0, 0.d0)
   qm = qmod (ig) * dqi
   px = qm - int (qm)
   ux = 1.d0 - px
   vx = 2.d0 - px
   wx = 3.d0 - px
   i0 = INT( qm ) + 1
   i1 = i0 + 1
   i2 = i0 + 2
   i3 = i0 + 3
   uvx = ux * vx * sixth
   pwx = px * wx * 0.5d0
   do lm = 1, lpx (ivl, jvl)
      lp = lpl (ivl, jvl, lm)
      !
      !     find angular momentum l corresponding to combined index lp
      !     (l is actually l+1 because this is the way qrad is stored, check init_us_1)
      !
      if (lp == 1) then
         l = 1
         sig = CMPLX(1.0d0, 0.0d0, kind=DP)
      elseif ( lp <= 4) then
         l = 2
         sig = CMPLX(0.0d0, -1.0d0, kind=DP)
      elseif ( lp <= 9 ) then
         l = 3
         sig = CMPLX(-1.0d0, 0.0d0, kind=DP)
      elseif ( lp <= 16 ) then
         l = 4
         sig = CMPLX(0.0d0, 1.0d0, kind=DP)
      elseif ( lp <= 25 ) then
         l = 5
         sig = CMPLX(1.0d0, 0.0d0, kind=DP)
      elseif ( lp <= 36 ) then
         l = 6
         sig = CMPLX(0.0d0, -1.0d0, kind=DP)
      else
         l = 7
         sig = CMPLX(-1.0d0, 0.0d0, kind=DP)
      endif
      work = qrad (i0, ijv, l, np) * uvx * wx + &
             qrad (i1, ijv, l, np) * pwx * vx - &
             qrad (i2, ijv, l, np) * pwx * ux + &
             qrad (i3, ijv, l, np) * px * uvx
      qg (ig) = qg (ig) + sig * CMPLX(ap (lp, ivl, jvl) * ylmk0 (ig, lp) * work, 0.d0, kind=DP)
   enddo
```



CUDAFortran kernel

```
ig= threadIdx%x+BlockDim%x*(BlockIdx%x-1)
if (ig <= ngy) then
   !     compute the indices which correspond to ih,jh
   dqi = 1.0_DP / dq
   qg(ig) = 0.d0

   qm = qmod (ig) * dqi
   px = qm - int (qm)
   ux = 1.d0 - px
   vx = 2.d0 - px
   wx = 3.d0 - px
   i0 = INT( qm ) + 1
   i1 = i0 + 1
   i2 = i0 + 2
   i3 = i0 + 3
   uvx = ux * vx * sixth
   pwx = px * wx * 0.5d0

   do lm = 1, lpx (ivl, jvl)
      lp = lpl (ivl, jvl, lm)
       if (lp == 1) then
          l = 1
          sig = CMPLX(1.0d0, 0.0d0, kind=DP)
       elseif ( lp <= 4) then
          l = 2
          sig = CMPLX(0.d0, -1.0d0, kind=DP)
       elseif ( lp <= 9 ) then
          l = 3
          sig = CMPLX(-1.0d0, 0.d0, kind=DP)
       elseif ( lp <= 16 ) then
          l = 4
          sig = CMPLX(0.d0, 1.0d0, kind=DP)
       elseif ( lp <= 25 ) then
          l = 5
          sig = CMPLX(1.0d0, 0.d0, kind=DP)
       elseif ( lp <= 36 ) then
          l = 6
          sig = CMPLX(0.d0, -1.0d0, kind=DP)
       else
          l = 7
          sig = CMPLX(-1.0d0, 0.d0, kind=DP)
       endif
       !sig = sig * ap (lp, ivl, jvl)
            work = qrad (i0, ijv, l, np) * uvx * wx + &
                   qrad (i1, ijv, l, np) * pwx * vx - &
                   qrad (i2, ijv, l, np) * pwx * ux + &
                   qrad (i3, ijv, l, np) * px * uvx
         qg (ig) = qg (ig) + sig * CMPLX(ylmk0 (ig, lp) * work *  ap (lp, ivl, jvl), 0
   end do
```

Libraries

LAXLib

ELPA

LAPACK

FFTXlib

X
Y
Ec
4Ec
P0
P1
P2
P3

FFTW, MKL, ESSL, ...

devXlib

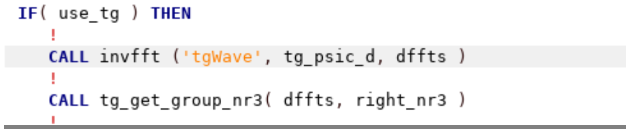
NVIDIA CUDA

OpenMP

CUDA FORTRAN

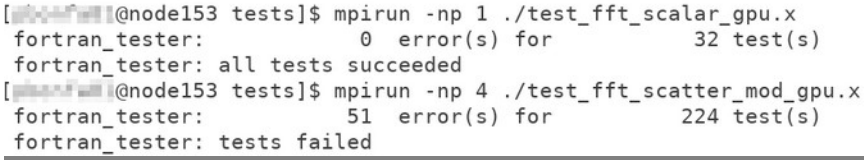
QE Libraries

- Full API support:



```
IF( use_tg ) THEN
   !
   CALL invfft ('tgWave', tg_psic_d, dffts )
   !
   CALL tg_get_group_nr3( dffts, right_nr3 )
   !
```

- Unit testing:



```
[[redacted]@node153 tests]$ mpirun -np 1 ./test_fft_scalar_gpu.x
 fortran_tester:              0  error(s) for          32 test(s)
 fortran_tester: all tests succeeded
[[redacted]@node153 tests]$ mpirun -np 4 ./test_fft_scatter_mod_gpu.x
 fortran_tester:             51  error(s) for         224 test(s)
 fortran_tester: tests failed
```

QE Libraries

LAXLib

ELPA

LAPACK

Solution of **dense eigenvalue problem** of real or complex hermitian matrices.

Both **serial** and **distributed parallel** implementation.

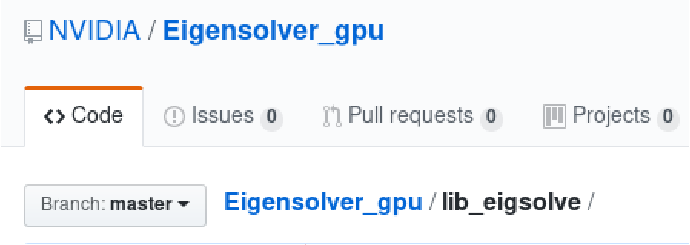
Extract **all eigenpairs** or a **subset**.

QE Libraries

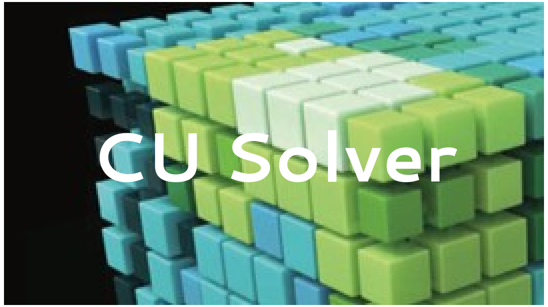
LAXLib

ELPA

LAPACK


NVIDIA / Eigensolver_gpu

Eigensolver_gpu / lib_eigsolve /



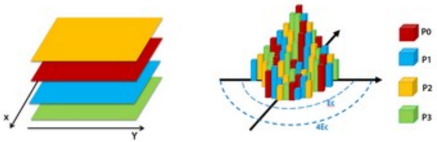
Initially custom code now part of CUDA 10.1.

Available at https://gitlab.com/max-centre/components

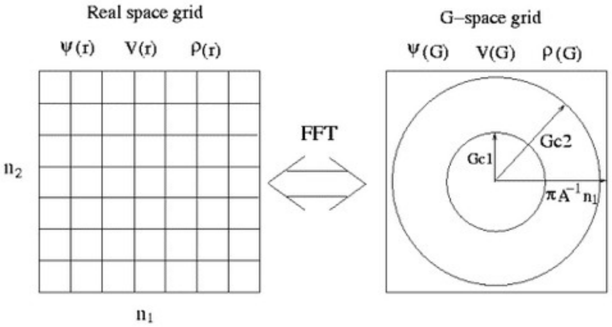
QE Libraries

FFTXlib

Sparse FFT in reciprocal space.
Parallel, distributed, accelerated.
Both pencil and slab decomposition.



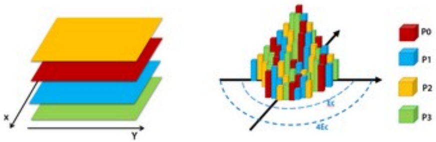
FFTW, MKL, ESSL, ...



Available at https://gitlab.com/max-centre/components

QE Libraries

FFTXlib



FFTW, MKL, ESSL, ...

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

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

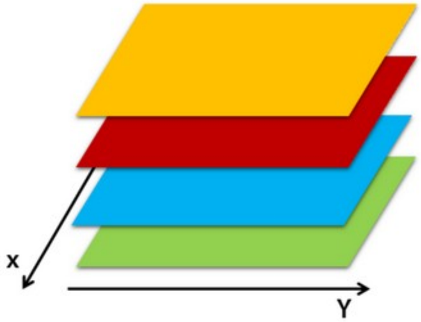
$$[v_{KS}\psi_{ik}](\boldsymbol{r}) = v_{KS}(\boldsymbol{r})\psi_{ik}(\boldsymbol{r})$$

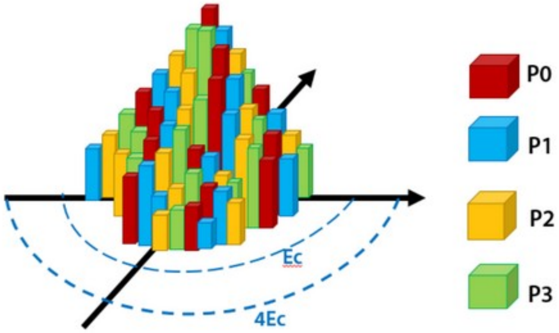
$$[v_{KS}\psi_{ik}](\boldsymbol{r}) \xrightarrow{FFT} [v_{KS}\psi_{ik}](\boldsymbol{G})$$

Available at https://gitlab.com/max-centre/components

QE Libraries

- For each band, FFT to real space, multiplicatio, FFT to reciprocal space.
  → Many independent small 3D FFTs (10^1 → 10^3)



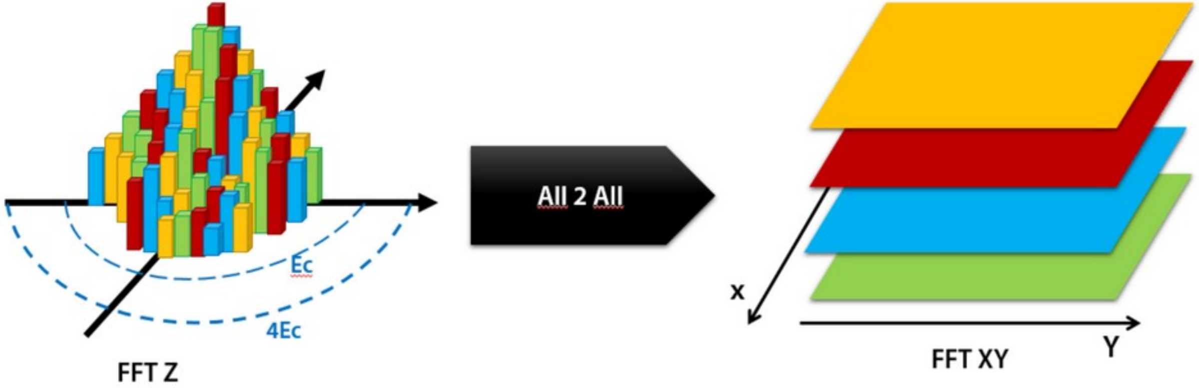


Images from E. Pascolo, M. Sc. thesis

QE Libraries

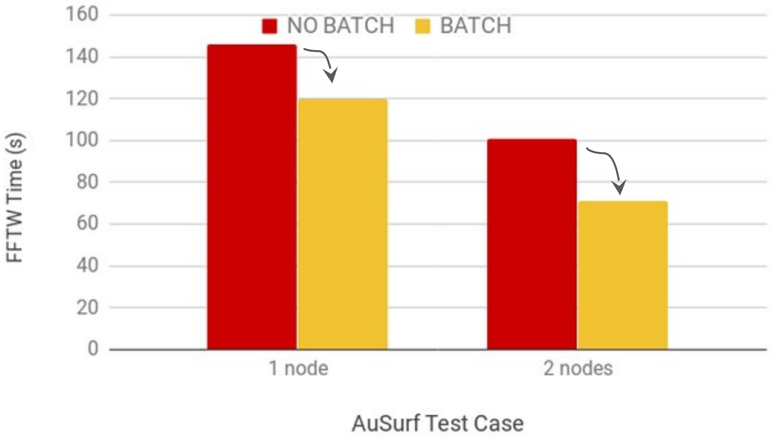
For each band, FFT to real space, multiplicatio, FFT to reciprocal space.

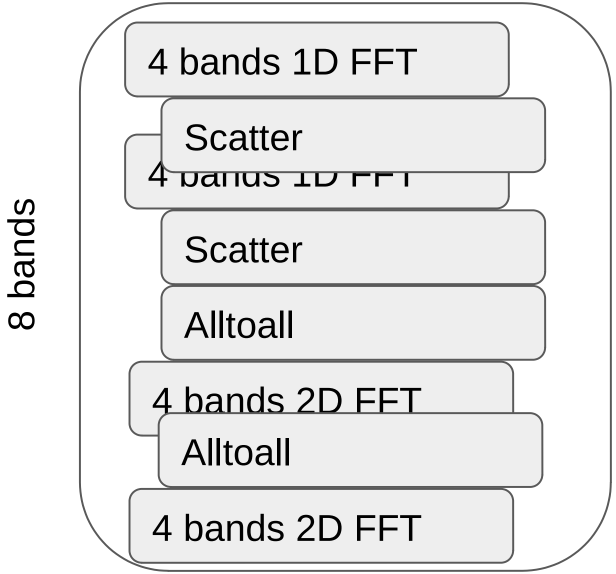
→ Many independent small 3D FFTs (10^1 → 10^3)



QE Libraries

- Many small 3D FFTs (10^1 → 10^3)
- Overlap of communication and computation
- Batched work





8 bands

4 bands 1D FFT

Scatter

4 bands 1D FFT

Scatter

Alltoall

4 bands 2D FFT

Alltoall

4 bands 2D FFT

QE Libraries

- QE allocates many small auxiliary workspaces. This impacts substantially the performances of the accelerated version of the code.
- Optimize memory allocation: GPU memory is limited.

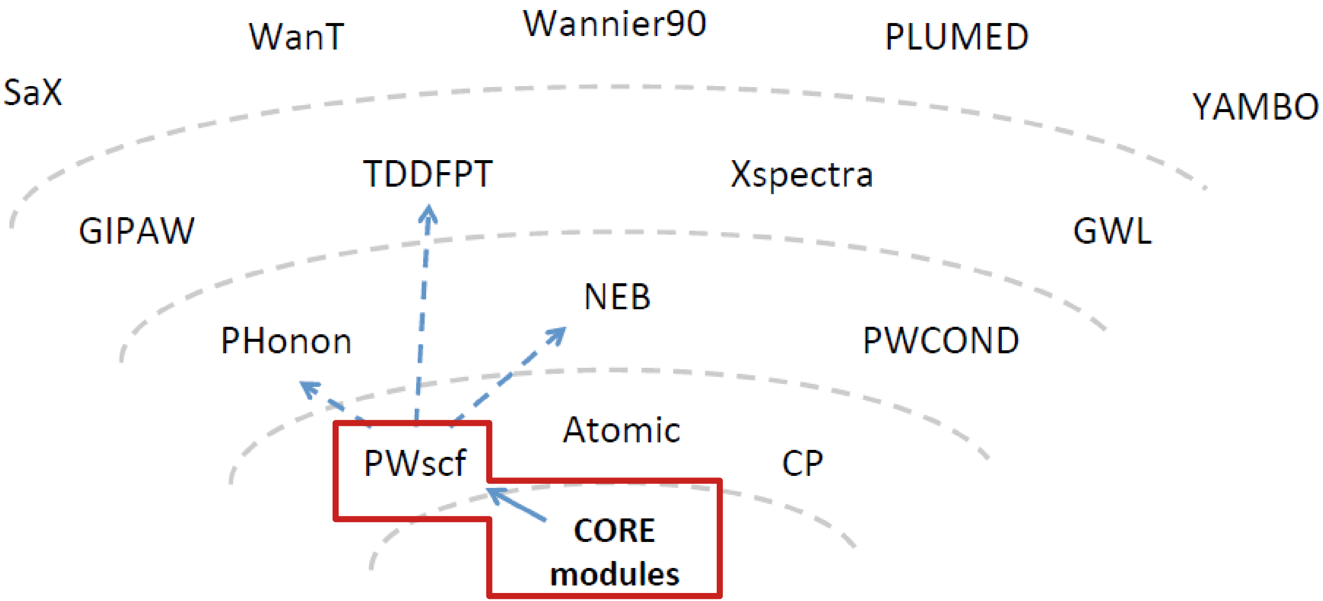
devXlib

Available at
https://gitlab.com/max-centre/components

```
USE buffer_module, ONLY : gpu_buffer
!
implicit none
!
REAL, POINTER :: work(:)
gpu_buffer%lock_buffer(work, 10, ierr)
[...]
gpu_buffer%release_buffer(work, ierr)
```

QE Codes

- GPU acceleration currently available for **PWscf**. CP and PHonon planned.



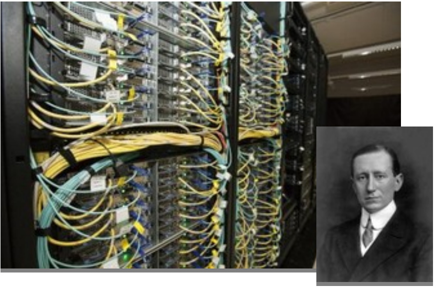
Current status and evolution

Application: *pw.x*

GPU version	Total Energy (K points)	Forces	Stress	Collinear Magnetism	Non-collinear magnetism	Gamma trick	US PP & PAW	EXX	DFT+U	All other functionalities
v5.4	**A**	**W**	**W**	**B** (?)	**U**	**A**	**A**	?	**W** (?)	**W** (?)
v6.1	**A**	**A**	**A**	**A**	**U**	**W** (*)	**A**	**U**	**U**	**U** (*)
v6.3	**A**	**W**	**W**	**A**	**A**	**A**	**A**	**W**	**W**	**W**
V6.5	**A**	**A**	**W**	**A**	**A**	**A**	**A**	**A**	**A**	**W**

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

Benchmarks



Marconi @ CINECA
Model: Xeon E5-2697 v4 (BDW) @ 2.30 GHz
Cores: 2x18 = **36**
RAM: 128 GB/node

Q3 2016
1.3 TFLOPs

Galileo @ CINECA
Model: Xeon E5-2630 v3 (HSW) @ 2.40 GHz
Cores: 2x8 = **16**
Accelerators: **2 x K80**
RAM: 128 GB/node

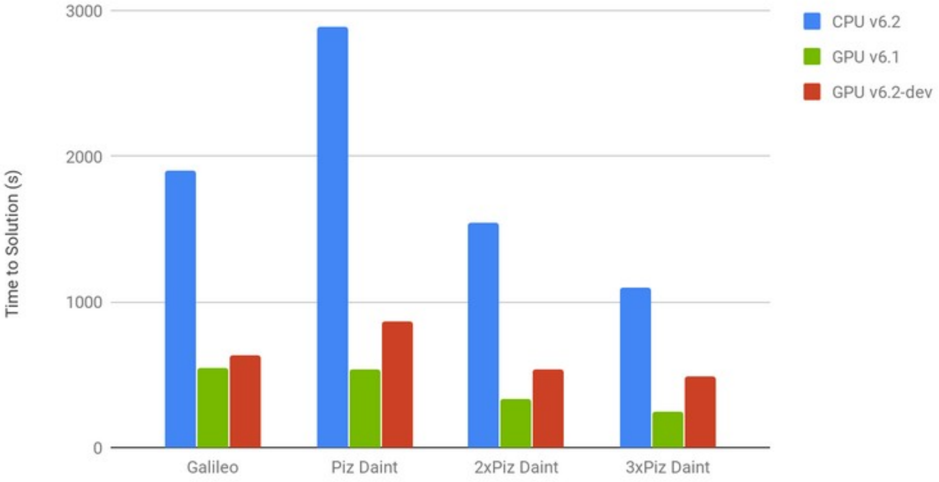
Q1 2015
0.6 + 2x2.9 TFLOPs

Piz Daint XC50 @ CSCS:
Model: Xeon E5-2690 v3 (HSW) @ 2.60 GHz
Cores: 1x12 = **12**
Accelerators: **1 x P100**
RAM: 64 GB/node

Q4 2016
0.5 + 4.7 TFLOPs

Benchmarks

Best time to solution obtained with *pw.x* v6.2, with and without GPU support, and with the GPU port of v6.1 done by NVIDIA.



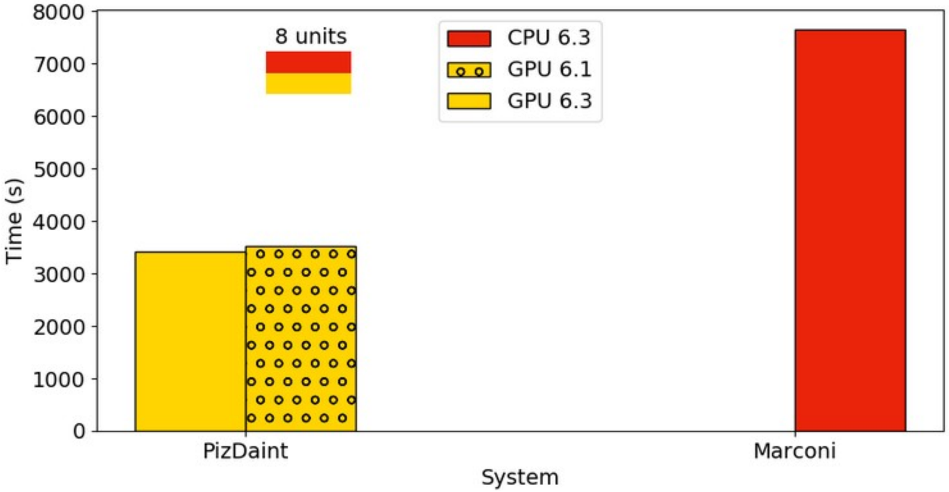
MnSi, bulk, ferromagnetic.
64 atoms, 365 KS states, USPP.

Piz Daint XC50 @ CSCS:
Processors: 12-cores Intel Haswell 2.60 GHz
Accelerators: 1 NVIDIA P100
RAM: 64 GB/node

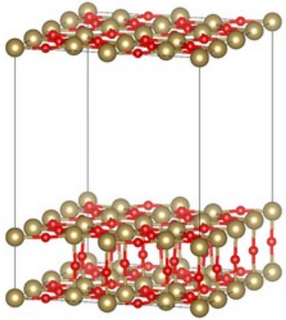
Galileo @ CINECA
Processors: 2*8-cores Intel Haswell 2.40 GHz
Accelerators: 2 NVIDIA K80
RAM: 128 GB/node

Benchmarks

Best time to solution obtained with *pw.x* v6.3, with and without GPU acceleration, and with the GPU port of *pw.x* v6.1 done by NVIDIA.



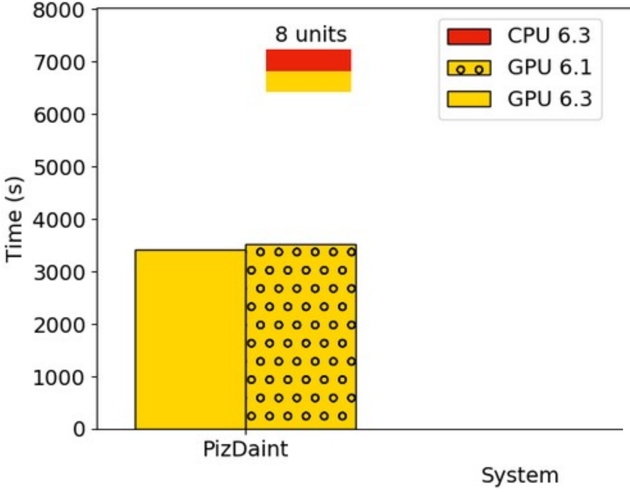
Ta_2O_5

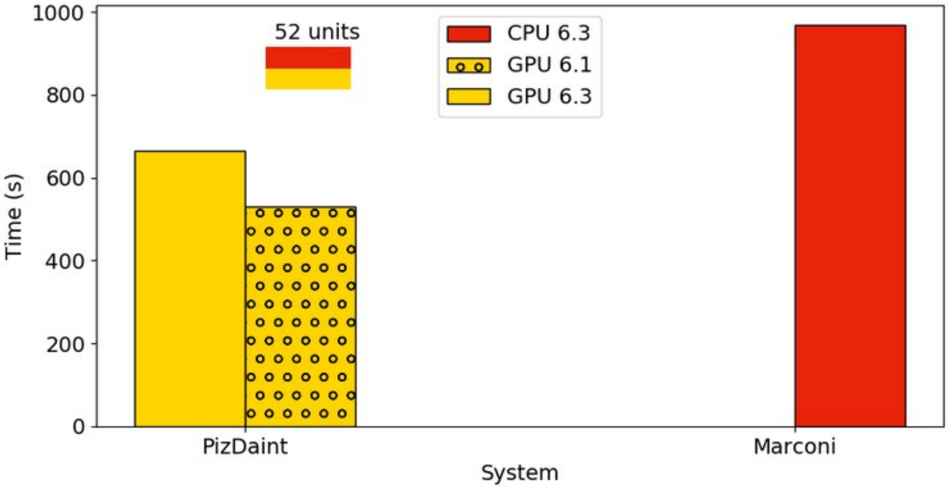


Large test case, 2D,
26 k-points, 96 atoms,
326 KS states.

Benchmarks

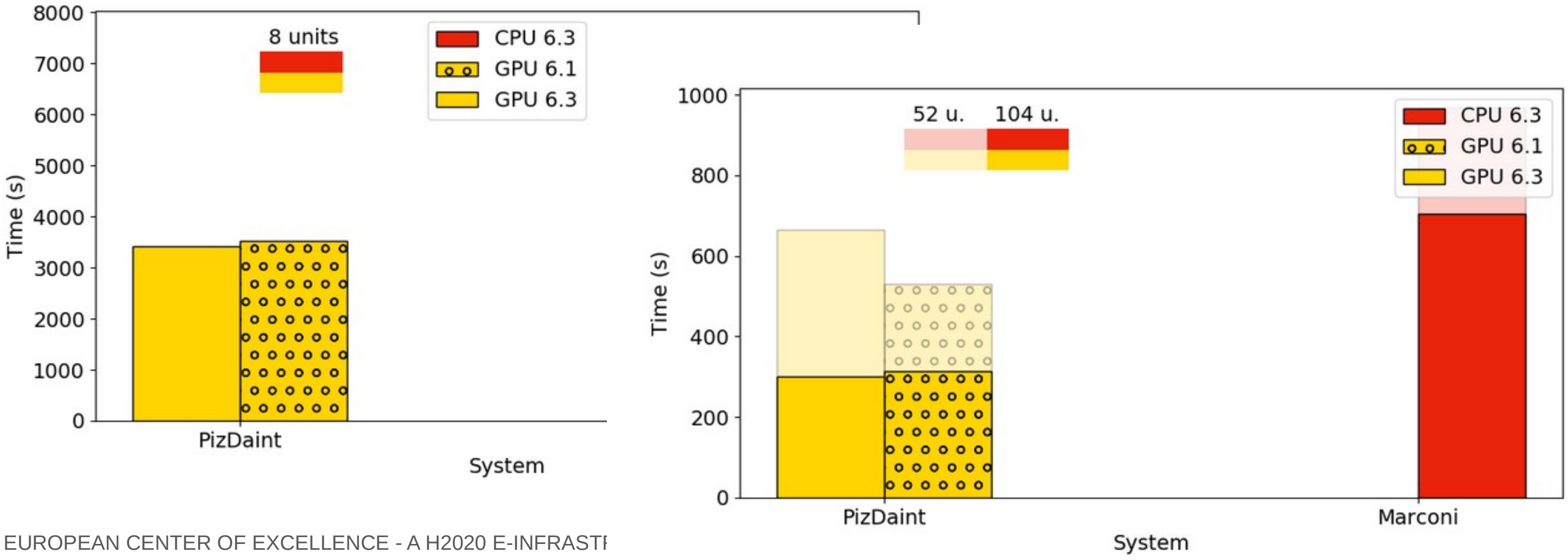
Best time to solution obtained with *pw.x* v6.3, with and without GPU acceleration, and with the GPU port of *pw.x* v6.1 done by NVIDIA.





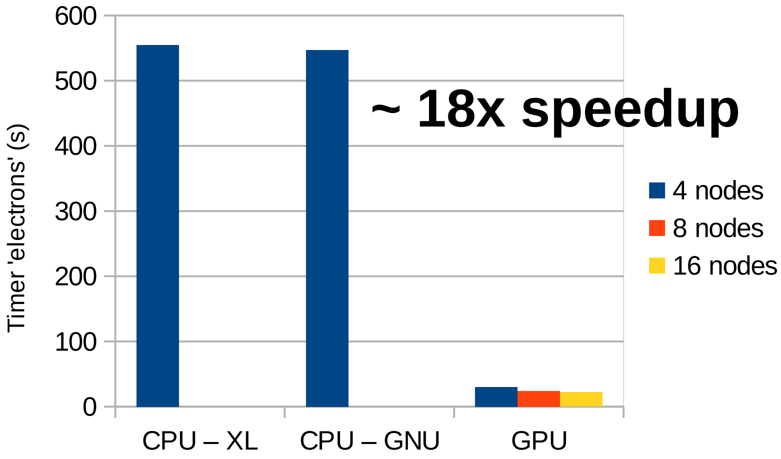
Benchmarks

Best time to solution obtained with *pw.x* v6.3, with and without GPU acceleration, and with the GPU port of *pw.x* v6.1 done by NVIDIA.



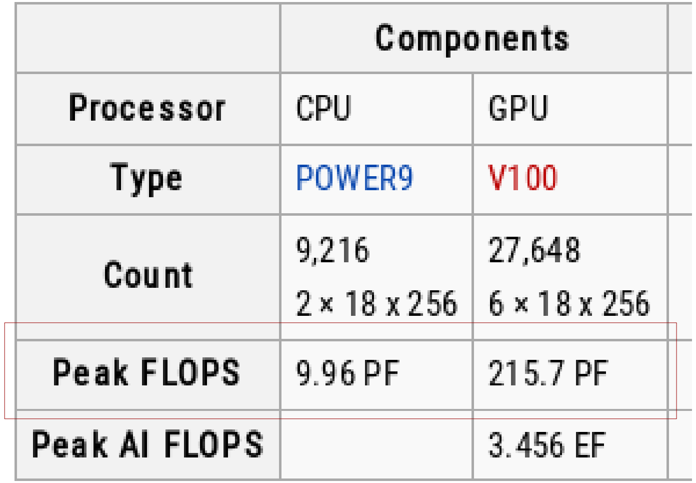
Benchmarks

Best time to solution obtained with *pw.x* v6.4, with and without GPU acceleration.



~ 18x speedup

Very large benchmark:
AuCONH benchmark: 586 atoms, 1531 KS



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

Courtesy of Dr. Ye Luo

HT Benchmarks

New Galileo @ CINECA
Model: Xeon E5-2697 v4 (BDW) @ 2.30 GHz
Cores: 1x18; RAM: 128 GB/node

+

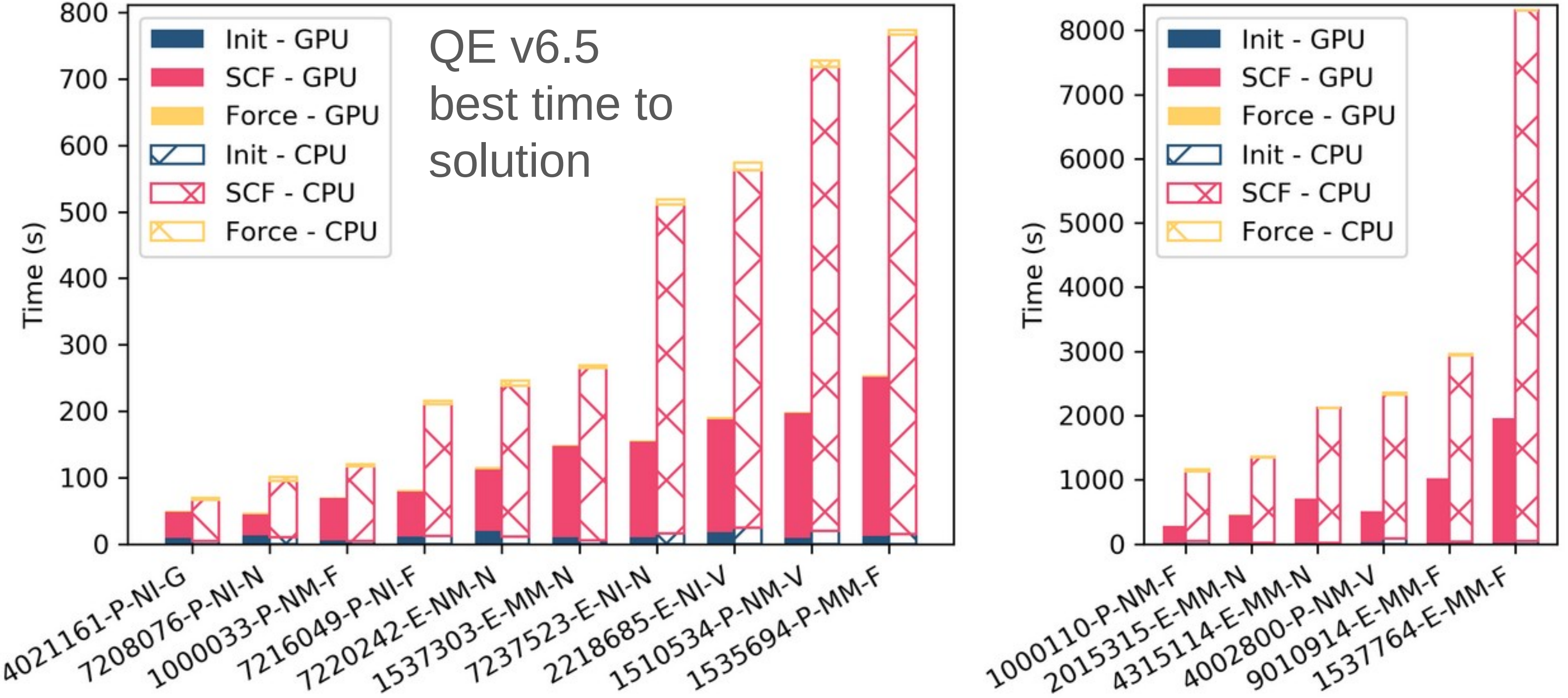
NVIDIA V100

Random structures from COD:

COD + QE Input generator materialscloud.org

- **M**agnetic / **N**on Magnetic
- **M**etal / **I**nsulator
- **E**fficiency / **A**ccuracy
- K-points grid:
  Very fine / **F**ine / **N**ormal / **G**amma

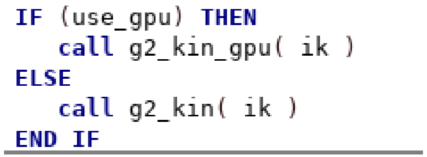
HT Benchmarks



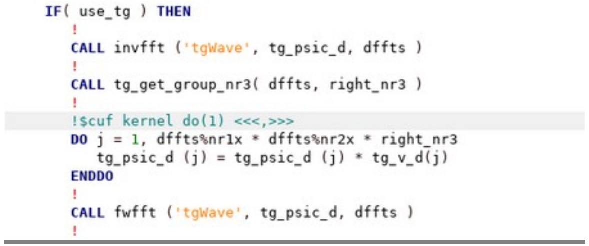
QE v6.5
best time to
solution

Porting other QE applications

- Preserve *all* functionalities
  - Feature testing already available...enough?
  - More feature and unit tests
  - Create verification scheme
- Preserve accelerated function modularity
  - For debugging
  - For code maintainability
  - For simpler development
- Directive based



```
IF (use_gpu) THEN
   call g2_kin_gpu( ik )
ELSE
   call g2_kin( ik )
END IF
```



```
IF( use_tg ) THEN
   !
   CALL invfft ('tgWave', tg_psic_d, dffts )
   !
   CALL tg_get_group_nr3( dffts, right_nr3 )
   !
   !$cuf kernel do(1) <<<,>>>
   DO j = 1, dffts%nr1x * dffts%nr2x * right_nr3
      tg_psic_d (j) = tg_psic_d (j) * tg_v_d(j)
   ENDDO
   !
   CALL fwfft ('tgWave', tg_psic_d, dffts )
   !
```

Lessons learnt

- Separation of concerns simplifies subsequent application porting
  - Reduces branches in the code.
  - Cleaner and simpler adoption of accelerated functions in new code.

- Modularization helps (duplicated) data management
  - Porting pushes modularization forward

- CUDA Fortran helped removing boilerplate.

- (Moving to GIT helped a lot)

Remaining challenges

- HW availability limits GPU programming:
  - Policy for contributions?
  - Ratio between CPU and GPU efforts?
- Different hybrid solutions about to appear.
- Is OpenMP the final answer?
- Source code duplication.

Remaining challenges

- HW availability limits GPU programming:
  - Policy for contributions?
  - Ratio between CPU and GPU efforts?
- Different hybrid solutions about to appear.
- Is OpenMP the final answer?
- Source code duplication.

Suggestions?
Thanks for your attention!