New developments of the QUANTUM ESPRESSO code: a combined CUF-OpenACC approach

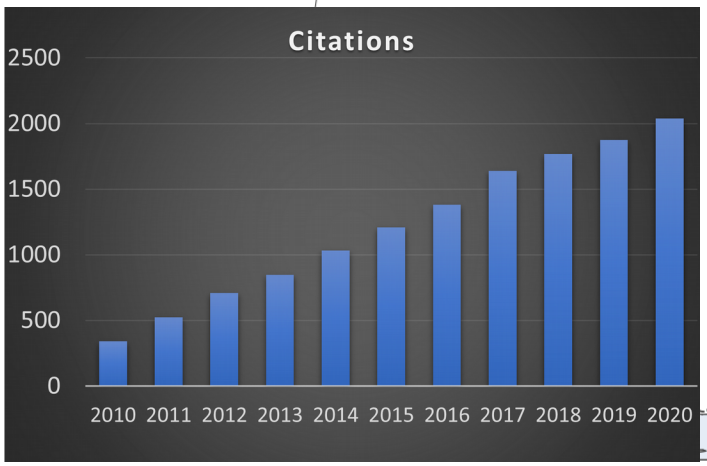
Ivan Carnimeo

Scuola Internazionale Superiore di Studi Avanzati (SISSA) – Trieste, Italy

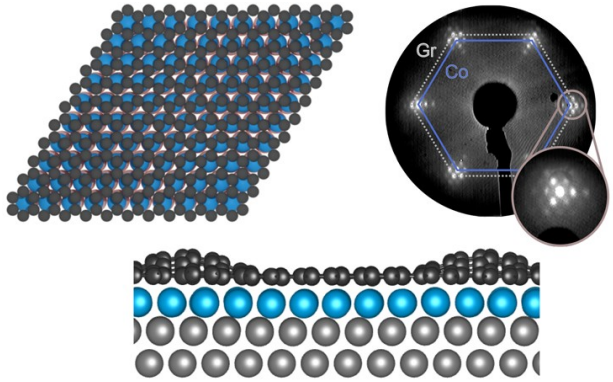
QUANTUM ESPRESSO

QUANTUM ESPRESSO™ (QE) 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.

QUANTUM ESPRESSO™ is an open initiative involving a **large community** of developers and contributors from different regions of the world



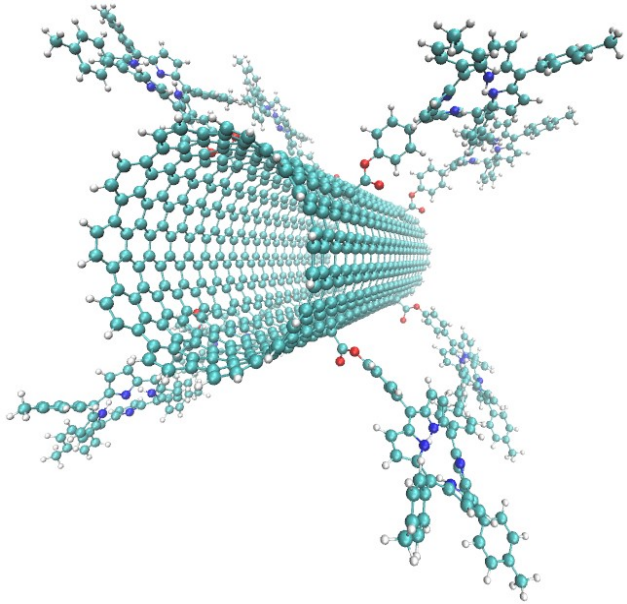
Pwscf, the core of QE, computes the electronic structure of molecules and materials by solving the Kohn-Sham (KS) equations, within the **Density Functional Theory** framework:



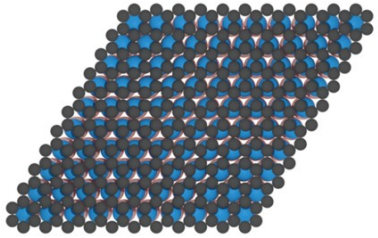
$$\hat{H}^{KS}\psi_{ik}(\mathbf{r}) = \varepsilon_{ik}\psi_{ik}(\mathbf{r})$$

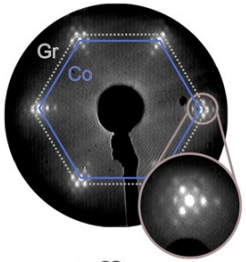
using Plane-Wave and Pseudopotential methods:

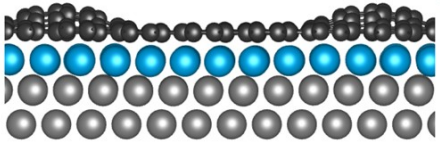
$$\psi_{ik}(\mathbf{r}) = \sum_{\mathbf{G}}^{N_{PW}} C_{\mathbf{G},ik} \frac{e^{i\mathbf{G}\mathbf{r}}}{\sqrt{\Omega}}$$



QUANTUM ESPRESSO workflow







molecular geometry

Input

KS Solvers

Iterative solution of the KS equations

Make KS potential

Output

Chem/Phys properties

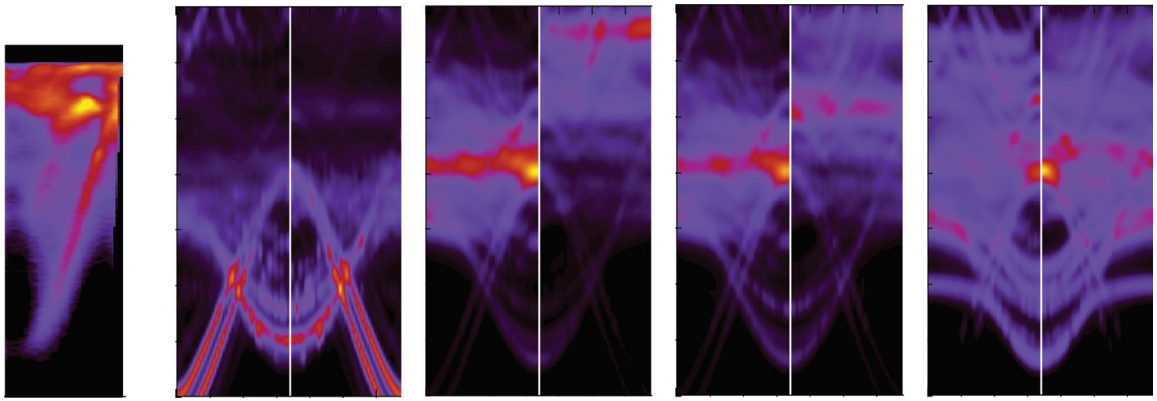
Energy

Forces

Stress

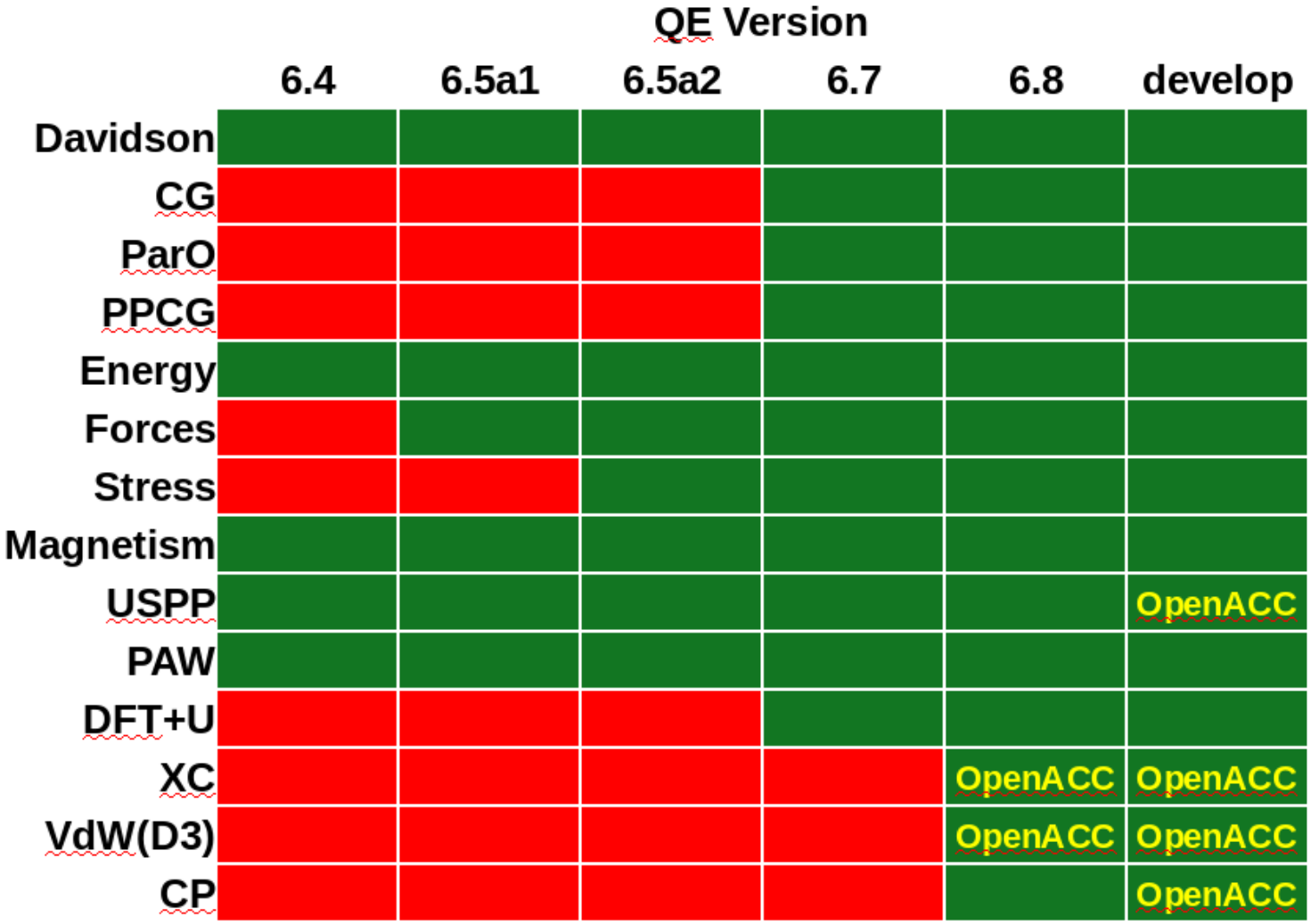
Band structure

Magnetism



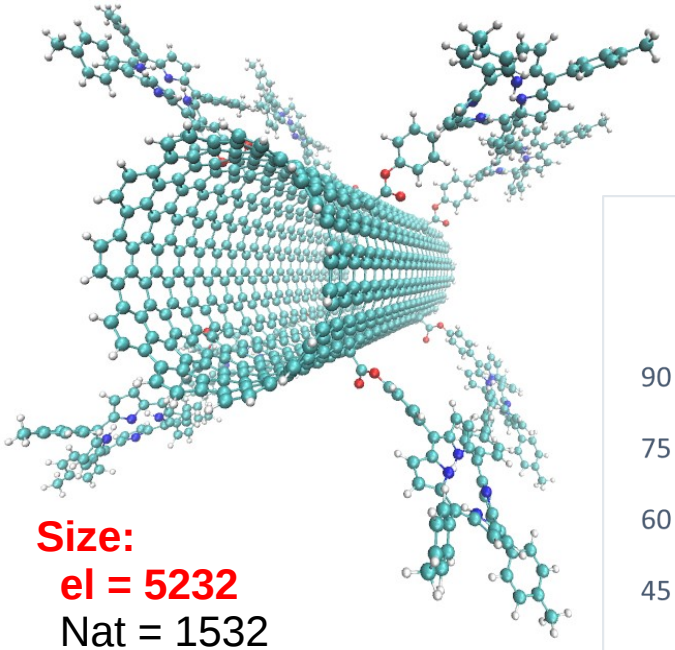
QUANTUM ESPRESSO on GPU

All the most important features of QUANTUM ESPRESSO™ have been ported to GPU-accelerated architectures in the last years using **CUDA Fortran**

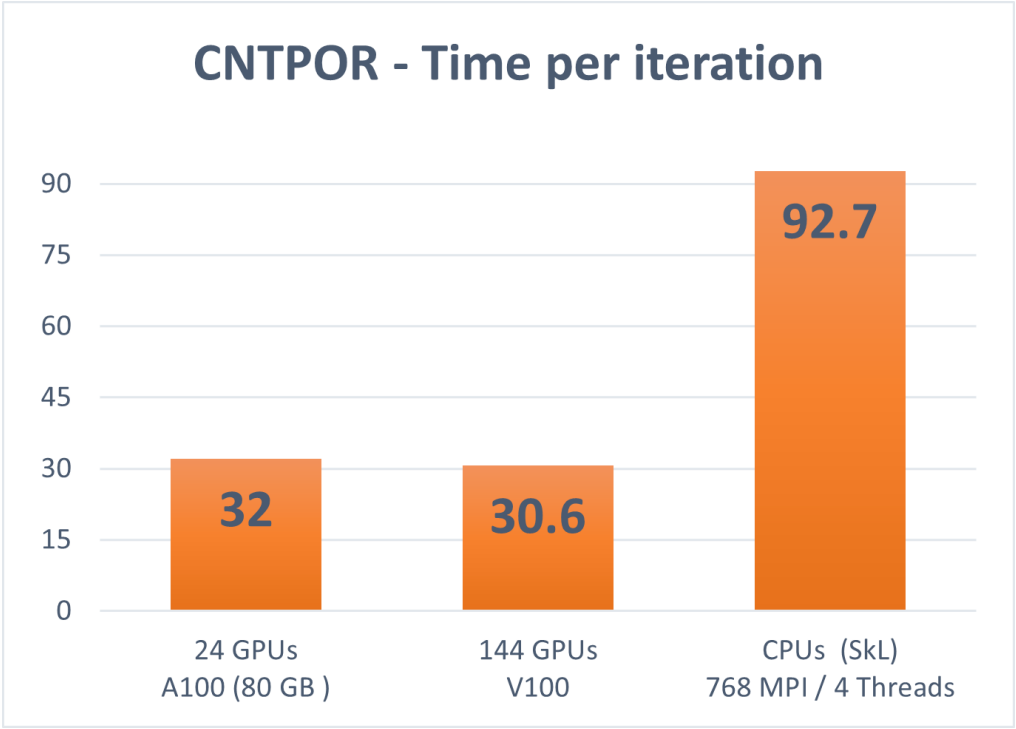


The most recent developments have been done using **OpenACC**

QUANTUM ESPRESSO on GPU



Size:
el = 5232
Nat = 1532
Ecut = US/25/200 Ry
NPW = 27M
Nbnd = 2616
Nks = Gamma only



The **CUDA Fortran** version of the code has a significantly better performance than the CPU version, but also comes with some drawbacks that we want to tackle using **OpenACC**

Data transfer

```
  SUBROUTINE using_vkb(intento)
      !
      !USE uspp, ONLY : vkb, vkb_d
      implicit none
      INTEGER, INTENT(IN) :: intento
      IF (size(vkb)==0) return
#if defined(__CUDA)  || defined(__CUDA_GNU)
      IF (vkb_ood) THEN
          IF (intento < 2) vkb = vkb_d
          vkb_ood = .false.
      ENDIF
      IF (intento > 0)    vkb_d_ood = .true.
#endif
  END SUBROUTINE using_vkb
  !
  SUBROUTINE using_vkb_d(intento)
      !
      !USE uspp, ONLY : vkb, vkb_d
      implicit none
      INTEGER, INTENT(IN) :: intento
      IF (size(vkb)==0) return
#if defined(__CUDA) || defined(__CUDA_GNU)
      !
      IF (vkb_d_ood) THEN
          IF (intento < 2) vkb_d = vkb
          vkb_d_ood = .false.
      ENDIF
      IF (intento > 0)    vkb_ood = .true.
#else
      CALL upf_error('using_vkb_d', 'no GPU support', 1)
#endif
  END SUBROUTINE using_vkb_d
```

If we synchronize host and device variables every time one of the copies changes, the computational burden of the calculations explodes.

We use flags that keep track whether the last change on a given variable has occurred on the host or device copy

Very efficient but redundant code!!!

P. Bonfa'

Code duplication

```
IF ( use_gpu ) THEN
   IF ( nkb > 0 ) CALL using_vkb_d(2)
   IF ( nkb > 0 ) CALL init_us_2_gpu( ngk(ik), igk_k_d(1,ik), xk(1,ik), vkb_d )
ELSE
   IF ( nkb > 0 ) CALL using_vkb(2)
   IF ( nkb > 0 ) CALL init_us_2( ngk(ik), igk_k(1,ik), xk(1,ik), vkb )
END IF
```

In many cases we have to duplicate memory allocations creating both host and device copies of the variables.

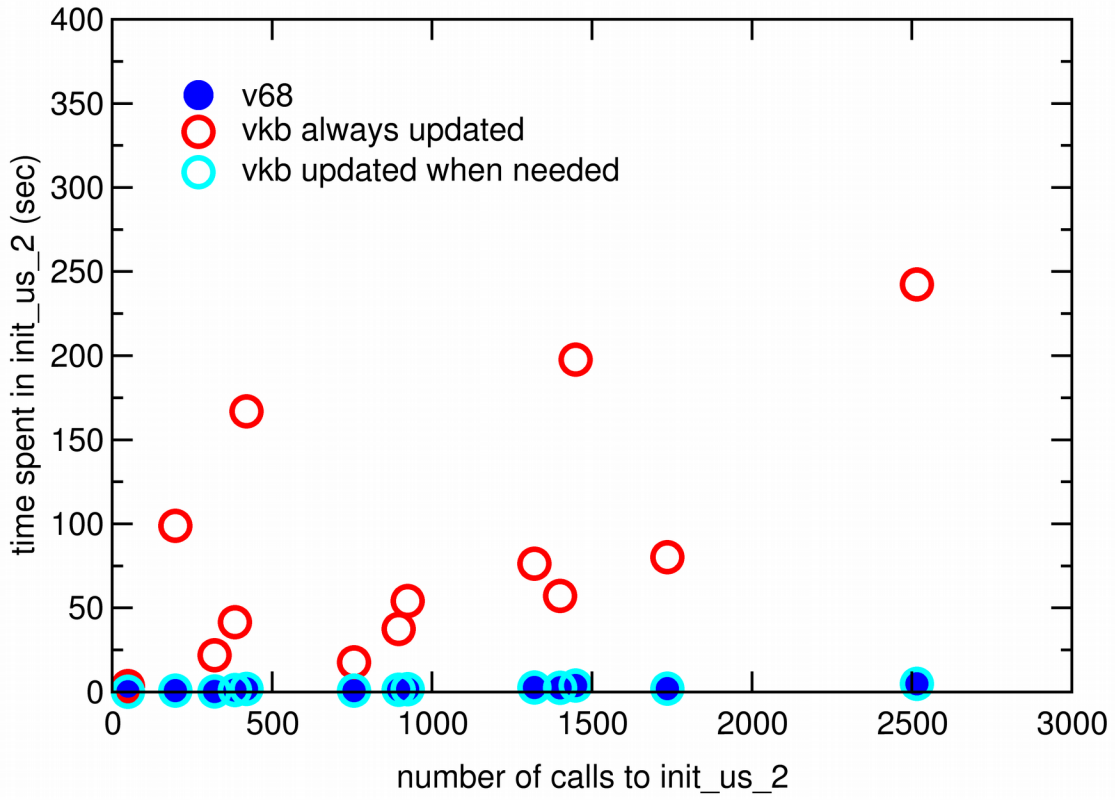
This lead to a frequent **duplication of subroutines and code in general**, that has a strong impact on the maintenance of the code, and makes it more difficult to add new features

USPP

OpenACC allows to reduce code duplication

```
IF ( nkb > 0 ) CALL init_us_2( ngk(ik), igk_k(1,ik), xk(1,ik), vkb, .true. )
```

(e.g. by declaring vkb as present inside init_us_2) but we have to be very careful with synchronizations as the data transfer burden quickly explodes



Blue and cyan marks:
host – device data synchronizations are done only when strictly needed

Red hollow marks:
host – device data synchronizations are done every time one of the two copies is changed

VdW (D3)

In this part of the code we have many complex nested loops

```
do iat=na_smax,na_e
  do jat=2, n
    do kat=1, n
      if((jat.ge.iat).or.(kat.ge.jat)) cycle
      linij=lin(iat,jat)
      ijvec1=xyz(1,jat)-xyz(1,iat)
      ijvec2=xyz(2,jat)-xyz(2,iat)
      ijvec3=xyz(3,jat)-xyz(3,iat)
```

```
            do ktaux=repmin1,repmax1
              do ktauy=repmin2,repmax2
                do ktauz=repmin3,repmax3
                  ktau1=ktaux*lat(1,1)+ktauy*lat(1,2)+ktauz*lat(1,3)
                  ktau2=ktaux*lat(2,1)+ktauy*lat(2,2)+ktauz*lat(2,3)
                  ktau3=ktaux*lat(3,1)+ktauy*lat(3,2)+ktauz*lat(3,3)
                  rik2=(ikvec1+ktau1)*(ikvec1+ktau1)+(ikvec2+ktau2)*(ikvec2+ktau2)+(ikvec3+ktau3)*(ikvec3+ktau3)
                  if (rik2.gt.abcthr)cycle

                  dumvec1=jkvec1+ktau1-jtau1
                  dumvec2=jkvec2+ktau2-jtau2
                  dumvec3=jkvec3+ktau3-jtau3
```

VdW (D3)

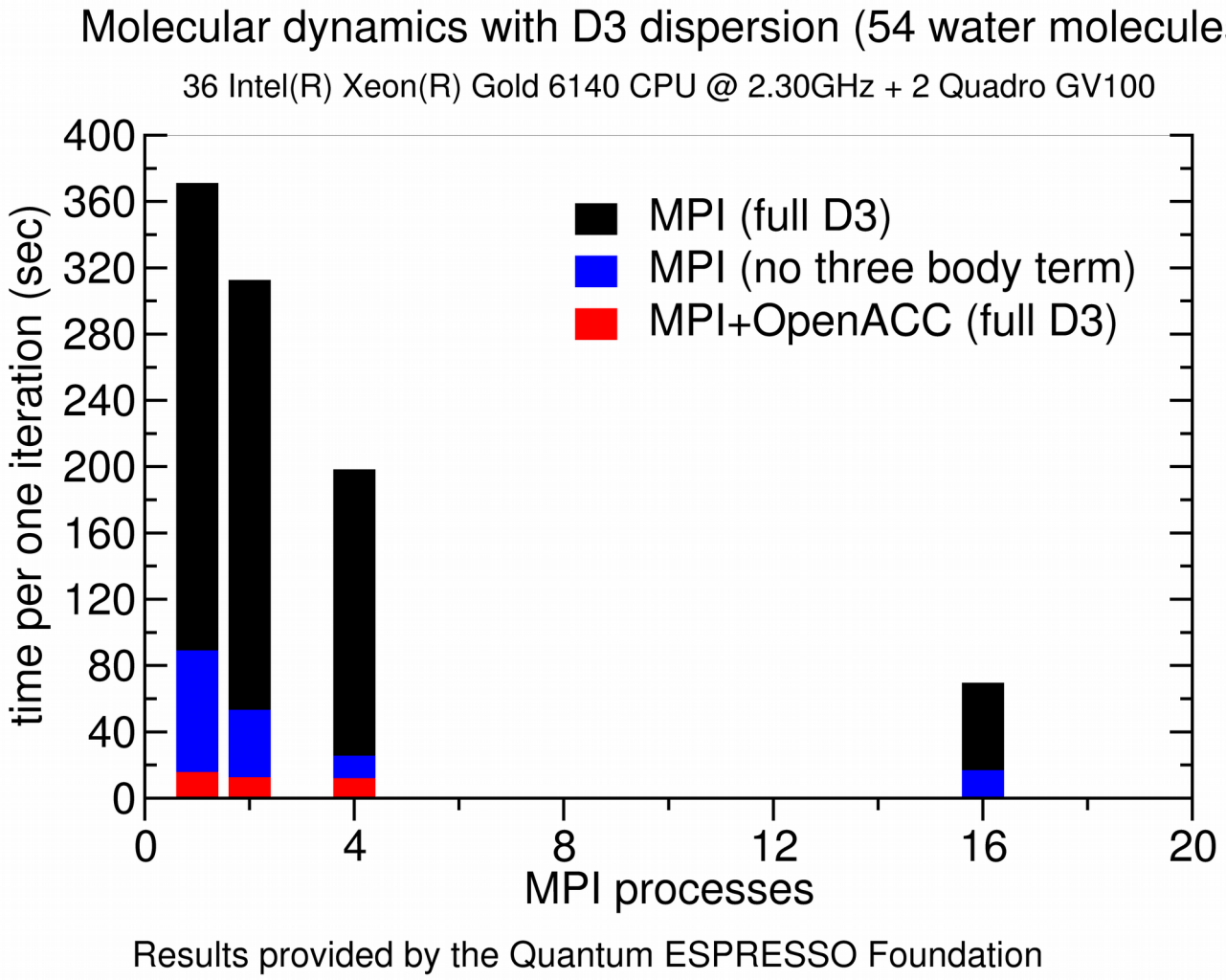
For complex and nested loops **OpenACC** provides more flexible optimization directives than CUF

```
!$acc data copyin(xyz(1:3,1:n),iz(1:n),lat(1:3,1:3),r0ab(1:max_elem,1:max_elem),c6save(1:n*(n+1)),dc6ij(1:n,1:n)) &
!$acc&           copy(dc6i(1:n),drij(-rep_v3:rep_v3,-rep_v2:rep_v2,-rep_v1:rep_v1,1:n*(n+1)/2))
!$acc parallel vector_length(32)
!$acc loop collapse(3) gang private(ijvec1,ijvec2,ijvec3, ikvec1,ikvec2,ikvec3, jkvec1,jkvec2,jkvec3, c6ij,c6ik,c6jk,c9, linij,linik,linjk, &
!$acc&                              jtau1,jtau2,jtau3, rij2,rr0ij, repmin1,repmin2,repmin3,repmax1,repmax2,repmax3 ) &
!$acc&                      reduction(+:eabc)
      do iat=na_smax,na_e
        do jat=2, n
          do kat=1, n
            if((jat.ge.iat).or.(kat.ge.jat)) cycle
            linij=lin(iat,jat)
            ijvec1=xyz(1,jat)-xyz(1,iat)
            ijvec2=xyz(2,jat)-xyz(2,iat)
            ijvec3=xyz(3,jat)-xyz(3,iat)
```

```
!$acc loop vector collapse(3) private(ktau1,ktau2,ktau3, dumvec1,dumvec2,dumvec3, rik2,rjk2,rr0ik,rr0jk, &
!$acc&                                 geomean,geomean2,geomean3,r0av,damp9,ang,dc6_rest,dfdmp,r,dang,tmp1,dc9) &
!$acc&                         reduction(+:eabc)
                do ktaux=repmin1,repmax1
                  do ktauy=repmin2,repmax2
                    do ktauz=repmin3,repmax3
                      ktau1=ktaux*lat(1,1)+ktauy*lat(1,2)+ktauz*lat(1,3)
                      ktau2=ktaux*lat(2,1)+ktauy*lat(2,2)+ktauz*lat(2,3)
                      ktau3=ktaux*lat(3,1)+ktauy*lat(3,2)+ktauz*lat(3,3)
                      rik2=(ikvec1+ktau1)*(ikvec1+ktau1)+(ikvec2+ktau2)*(ikvec2+ktau2)+(ikvec3+ktau3)*(ikvec3+ktau3)
                      if (rik2.gt.abcthr)cycle

                      dumvec1=jkvec1+ktau1-jtau1
                      dumvec2=jkvec2+ktau2-jtau2
                      dumvec3=jkvec3+ktau3-jtau3
```

VdW (D3)

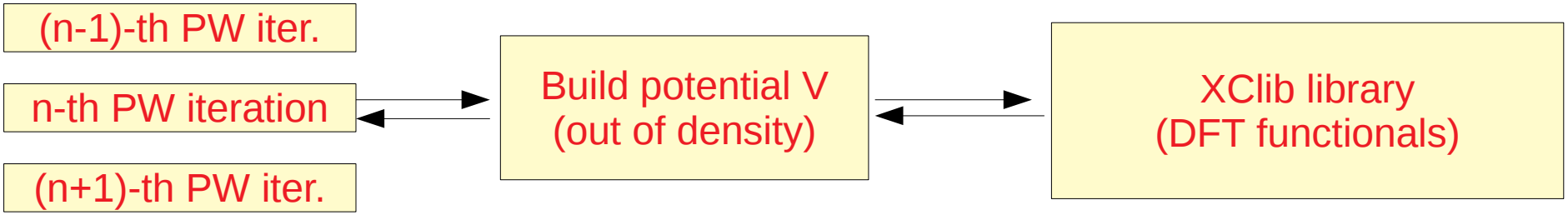


Black bars
DFT + D3 parallelized with MPI

Blue bars
DFT + D3 parallelized with MPI, excluding the three body energy term with `dftd3_threebody=.false.`

Red bars
DFT + D3 parallelized with MPI and accelerated with OpenACC

XC functionals




```
!----------------------------------------
MODULE exch_lda
!----------------------------------------
!! LDA exchange functionals.
!
CONTAINS
!
!----------------------------------------
SUBROUTINE slater( rs, ex, vx )
!$acc routine (slater) seq
  !--------------------------------------
  !! Slater exchange with alpha=2/3
  !
  USE kind_l,  ONLY: DP
  !
  IMPLICIT NONE
```

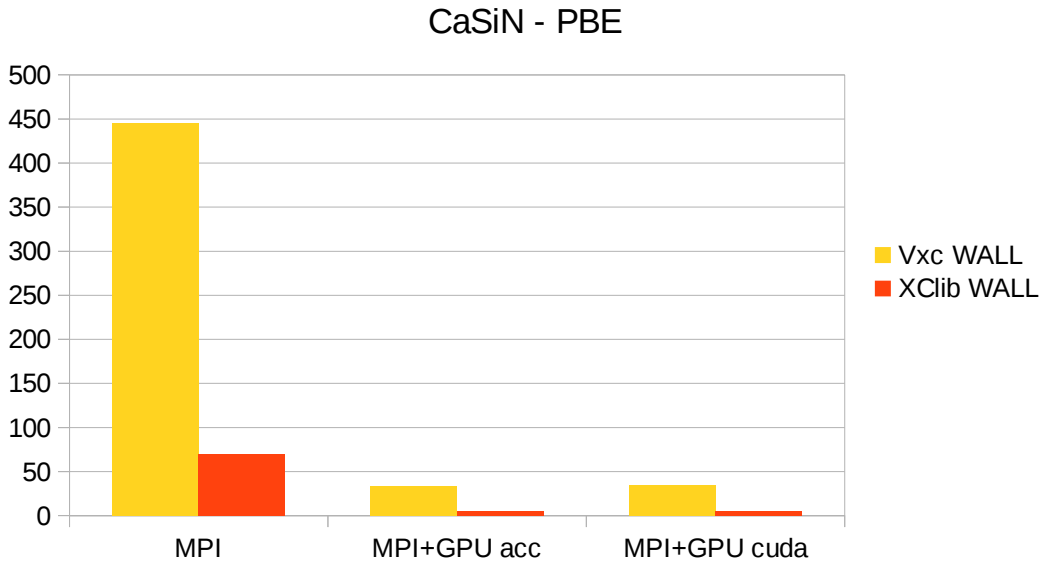
- **XClib**, library of functionals with large number of *kernels* and *acc routines*
- **Vxc**, potential calculation with calls to **XClib** and **FTTs**
- **XClib** and **Vxc** in OpenACC (no code duplication) with CUDA **FFTs** (already present)

XC functionals

1) 40% less code w.r.t. CUF

2) Same performance as CUF

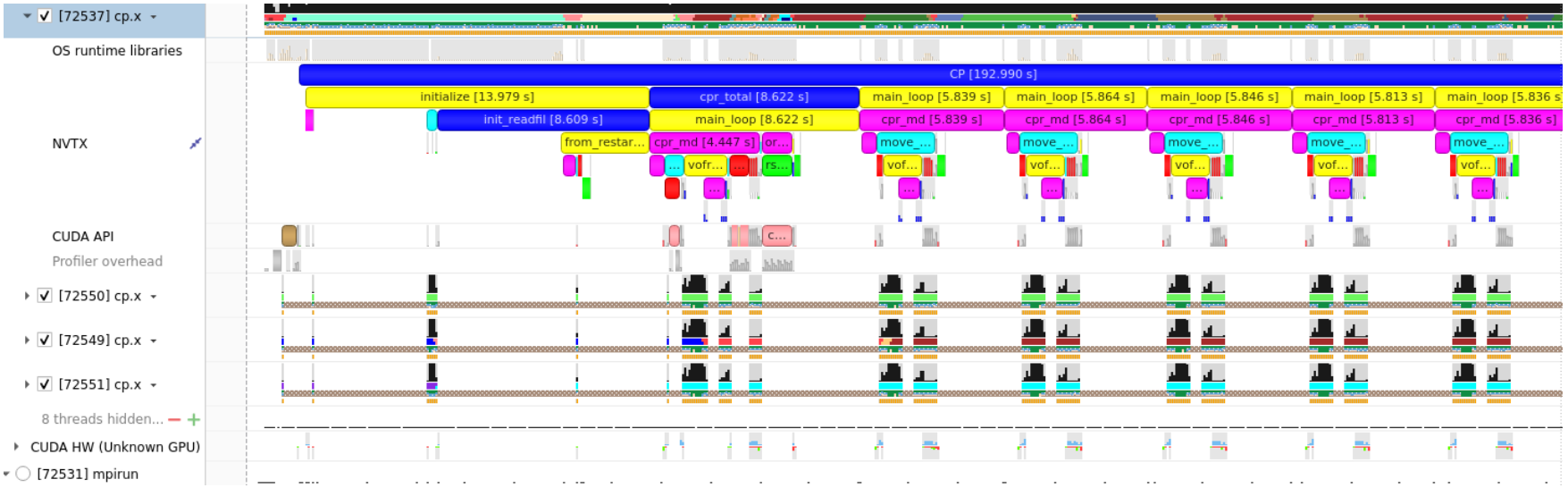
3) No need to rewrite large portions of code when passing from CPU to OpenACC



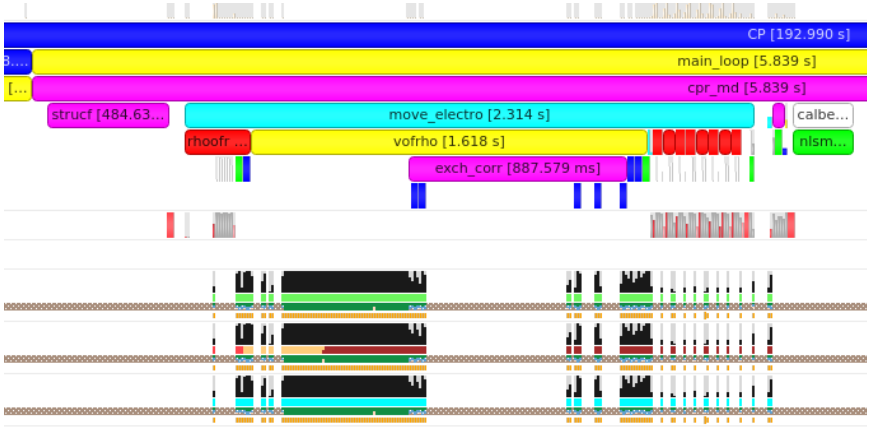
F. Ferrari Ruffino

Car-Parrinello MD

Car-Parrinello molecular dynamics using plane waves and pseudopotentials: most of the simulation time is spent in the main_loop repeated for each step of the dynamics



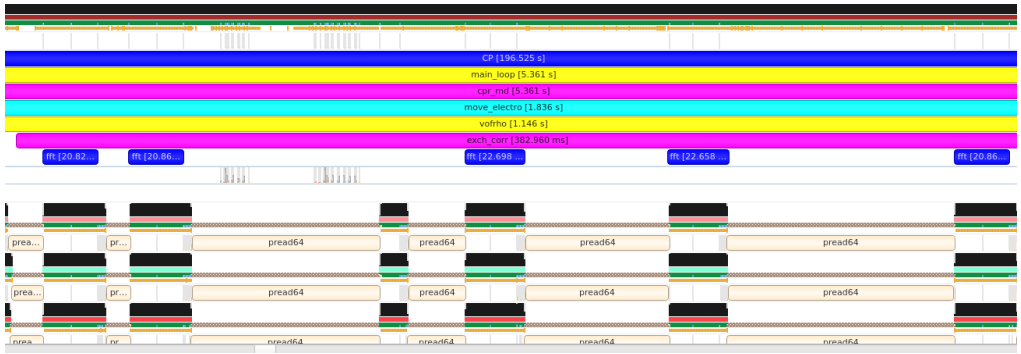
The code has been partially accelerated using CUDA Fortran but significant parts still run on CPU



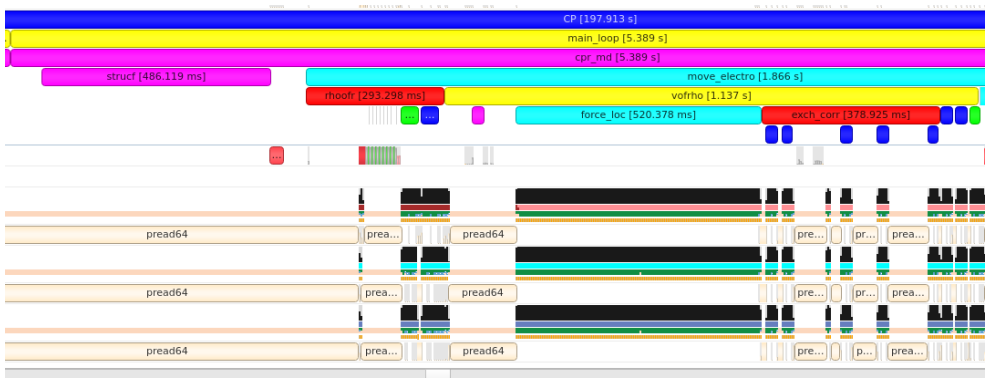
P. Delugas

Car-Parrinello MD

- The exch_corr part benefits from the work done in XClib with a 2X speedup without any code change

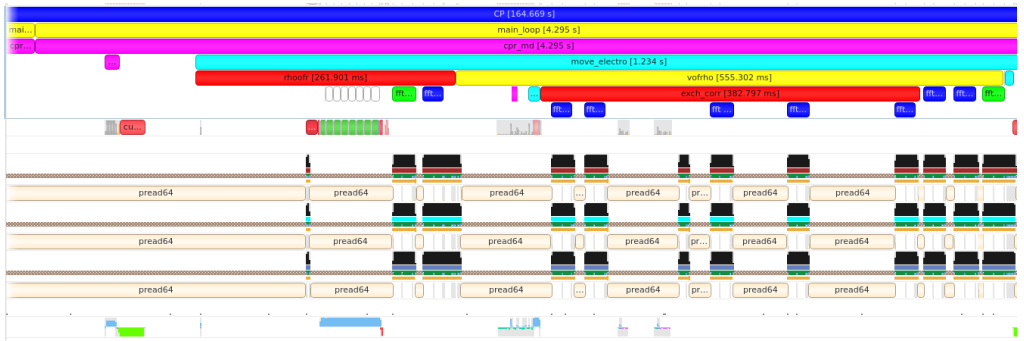


- Replacing the multithreading with OpenACC directives we observe a reduction from ~100 to few ms



- Currently the overall acceleration of each step is ~ 3X

- Further speedup can be achieved
  - Using accelerated FFTs
  - Improving data locality and syncronization



P. Delugas, F. Ferrari Ruffino, S. Orlandini

Car-Parrinello MD

For those parts of the code where **OpenACC** and **OpenMP** overlap, compilation directives (taken from the **Yambo Code**) are employed to choose one of the two, on the basis of the available architectures

```
1 / 1                                                    @XPS-13  32:130
1: pdelugas@login01:/m100_scratch/userexternal/pdelugas/q-e/CPV/src
! in the root directory of the present distribution,
! or http://www.gnu.org/copyleft/gpl.txt .
!
!
#if defined ( _OPENACC)
 #ifndef __OPENACC
  #define __OPENACC
 #endif
#endif

#if defined ( __OPENACC)
 #define DEV_ACC !$acc
 #define DEV_OMP !!!
 #define START_WSHARE DEV_ACC  kernels
 #define END_WSHARE   DEV_ACC end  kernels
#else
 #define DEV_ACC !!!
 #define DEV_OMP !$omp
 #define START_WSHARE DEV_OMP workshare
 #define END_WSHARE   DEV_OMP end workshare
#endif

!----------------------------------------------------------------------
SUBROUTINE vofrho_x( nfi, rhor, drhor, rhog, drhog, rhos, rhoc, tfirst, &
                     tlast, ei1, ei2, ei3, irb, eigrb, sfac, tau0, fion )
!----------------------------------------------------------------------
!     computes: the one-particle potential v in real space,
!               the total energy etot,
!               the forces fion acting on the ions,
!               the derivative of total energy to cell parameters h
```

```
1 / 1                                                    @XPS-13  32:130
1: pdelugas@login01:/m100_scratch/userexternal/pdelugas/q-e/CPV/src
      !
DEV_OMP do
DEV_ACC parallel loop
      DO ig = 1, SIZE(vtemp)
         vtemp(ig)=(0.d0,0.d0)
      END DO
      DO is=1,nsp
DEV_OMP do
DEV_ACC parallel loop
         DO ig=1,dffts%ngm
            vtemp(ig)=vtemp(ig)+CONJG(rhotmp(ig))*sfac(ig,is)*vps(ig,is)
         END DO
      END DO

DEV_OMP do reduction(+:zpseu)
DEV_ACC parallel loop reduction(+:zpseu)
      DO ig=1,dffts%ngm
         zpseu = zpseu + vtemp(ig)
      END DO
DEV_OMP end parallel
      !
DEV_ACC update self(vtemp(1))
      epseu = wz * DBLE(zpseu)
      !
      IF (gstart == 2) epseu = epseu - DBLE( vtemp(1) )
      !
      CALL mp_sum( epseu, intra_bgrp_comm )
      epseu = epseu * omega
      !
```

P. Delugas, P. Bonfa', A. Ferretti

Conclusions and perspectives

- CUF-OpenACC interoperability is one of the key aspects of QE acceleration
- OpenACC logic for handling device variables is helpful for reducing code duplication
- Portability of the code to different hardware architectures is very important
- Other important codes of the QUANTUM ESPRESSO suite will be accelerated (e.g. TDDFPT, PHonon)

Acknowledgements

QUANTUM ESPRESSO GPU development group

- Paolo Giannozzi, University of Udine
- Stefano Baroni, SISSA
- Pietro Delugas, SISSA
- Pietro Bonfa', University of Parma
- Ivan Carnimeo, SISSA
- Fabrizio Ferrari Ruffino, CNR-IOM
- Oscar Baseggio, SISSA
- Elena De Paoli, CNR-IOM

Nvidia Technical support

- Filippo Spiga, Nvidia
- Louis Stuber, Nvidia

CINECA Technical support

- Sergio Orlandini, CINECA