

December 1st, 2023
OpenMP Users Monthly Teleconferences



OpenMP offloaded Quantum ESPRESSO

Ferrari-Ruffino Fabrizio

CNR-IOM

Bellentani Laura

HPC - CINECA

QUANTUM ESPRESSO ON ACCELERATORS

Quantum mechanics for materials

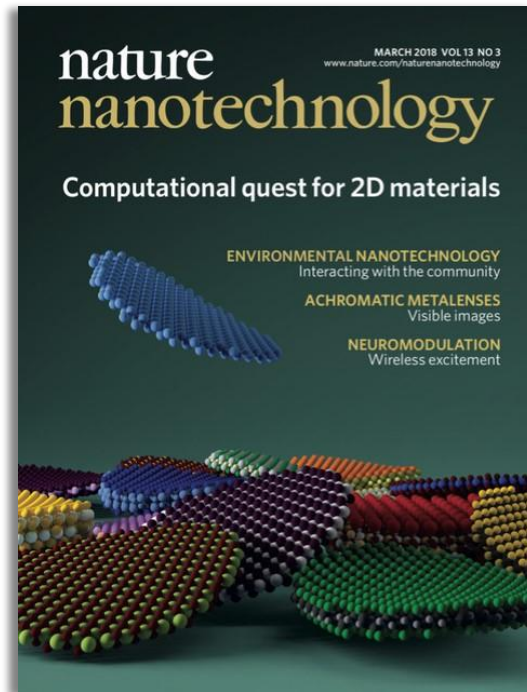
AB INITIO QUANTUM MECHANICS

no input parameters for material modeling

reduces costs,
accelerates discoveries



QUANTUM ESPRESSO is an integrated **parallel suite of Open-Source computer codes** for electronic-structure calculations and materials modeling at the nanoscale.



Two-dimensional materials from high-throughput computational exfoliation of experimentally known compounds, Nature Nanotechnology **13**, 246 (2018). [doi:10.1038/s41565-017-0035-5](https://doi.org/10.1038/s41565-017-0035-5)

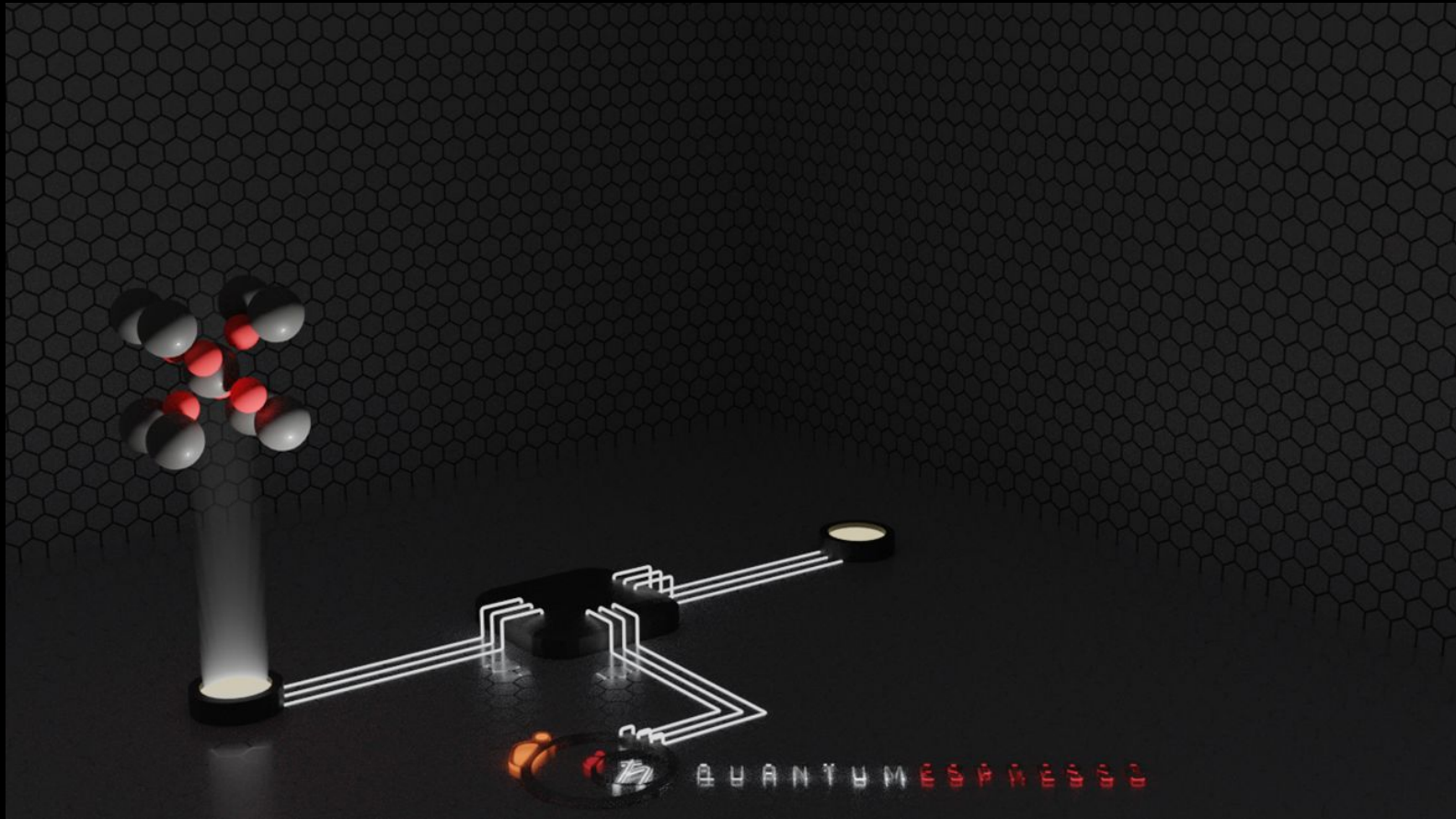
Quantum mechanics for materials



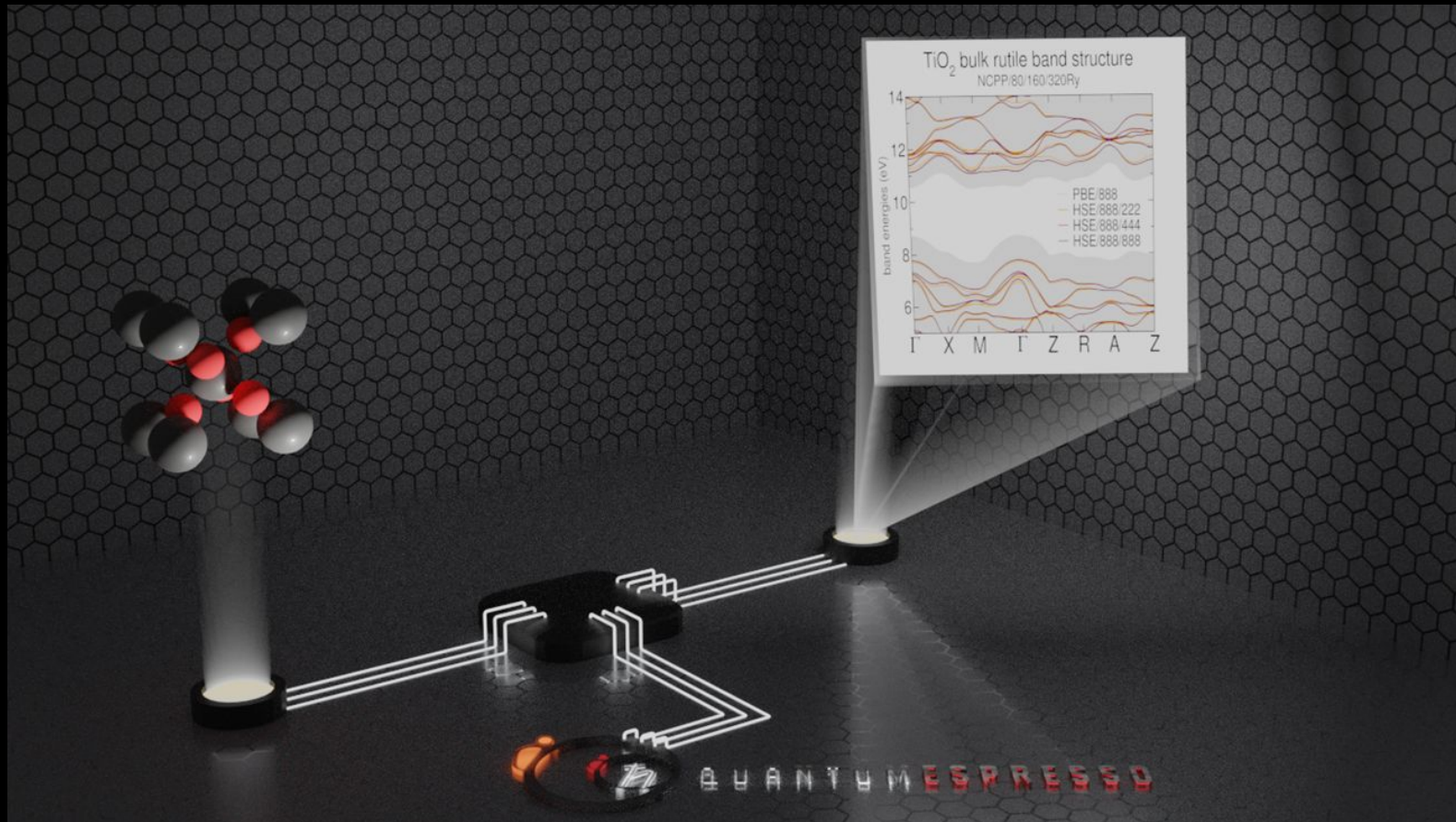
Quantum mechanics for materials



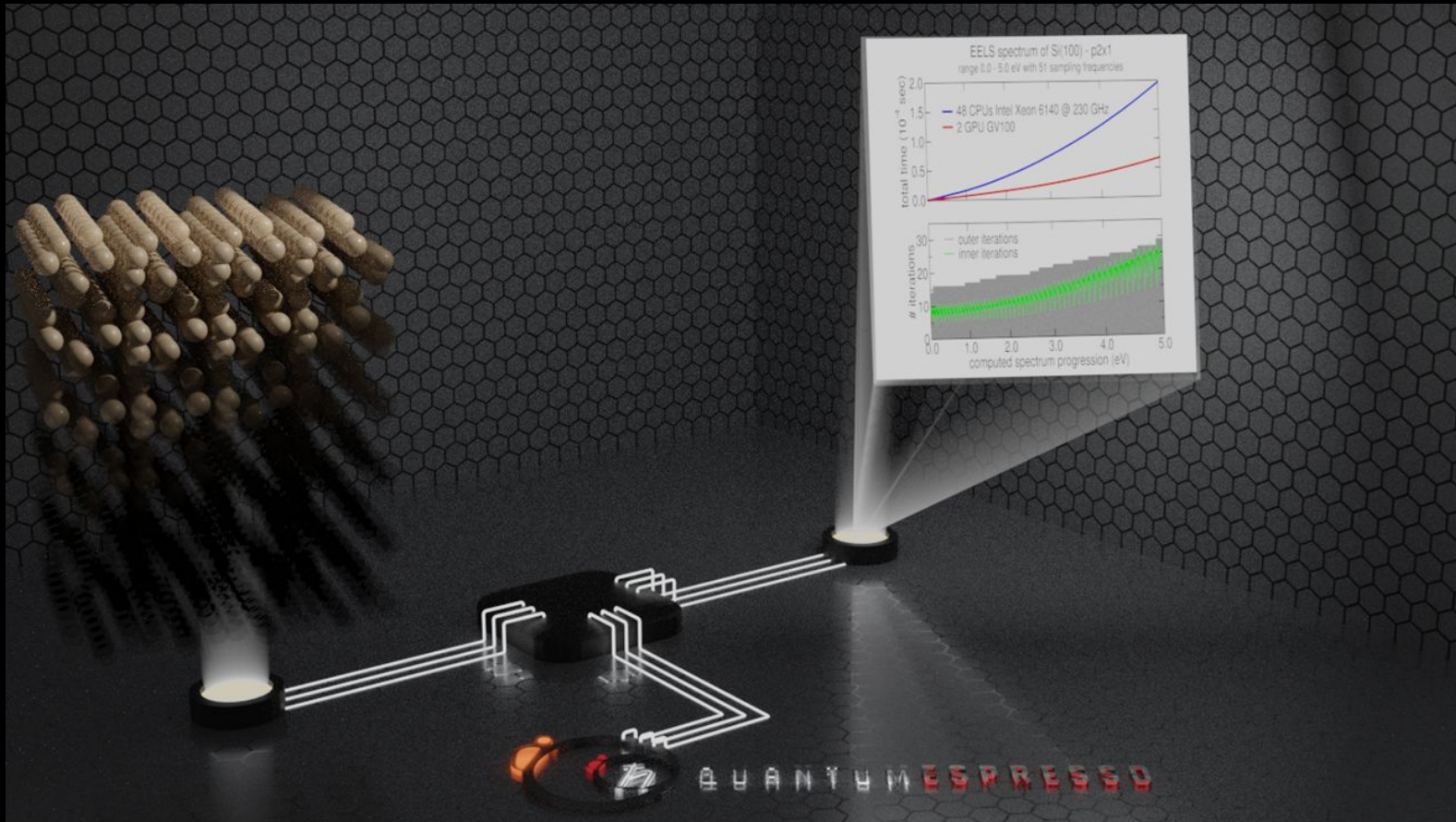
Quantum mechanics for materials



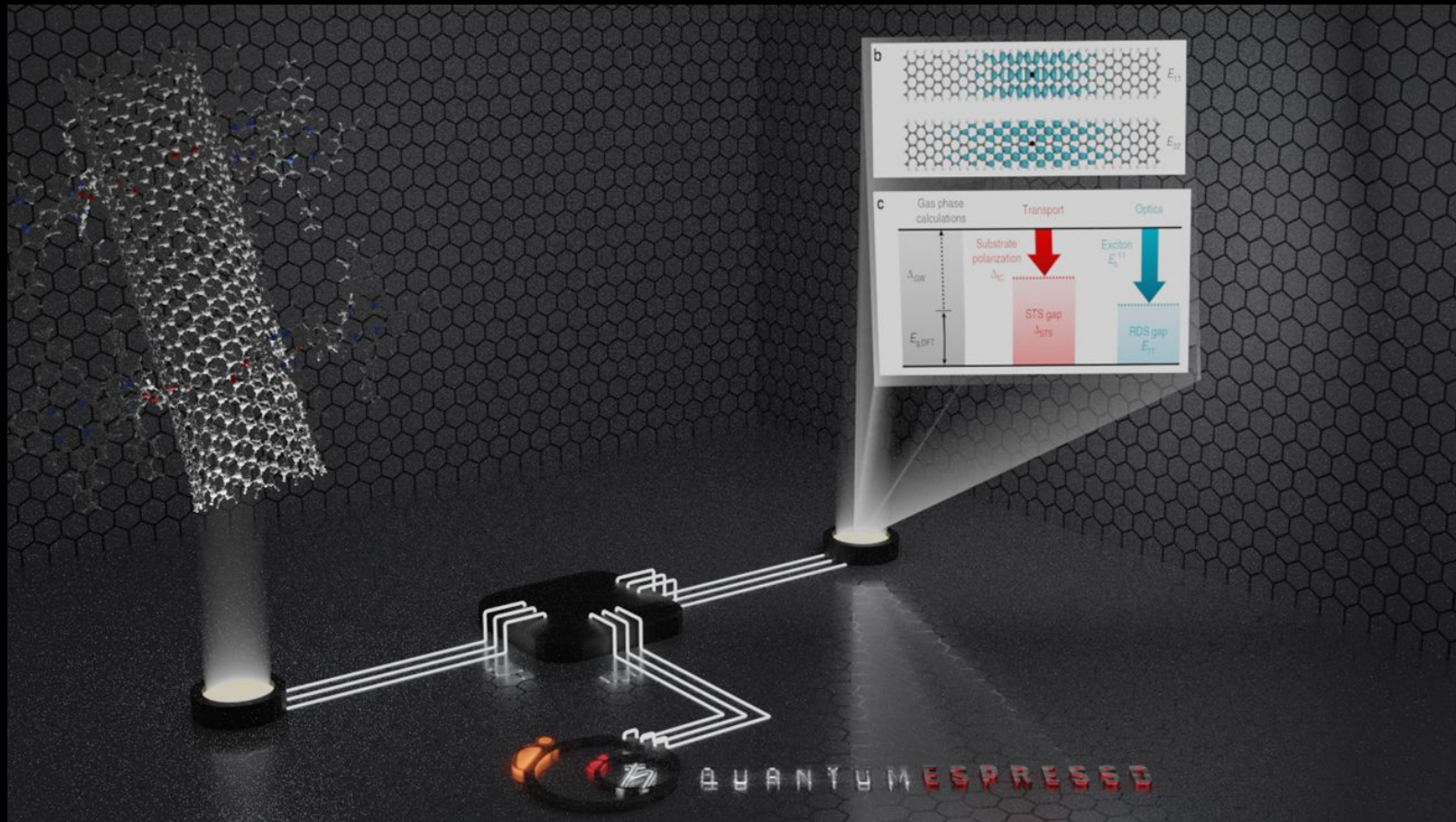
Quantum mechanics for materials



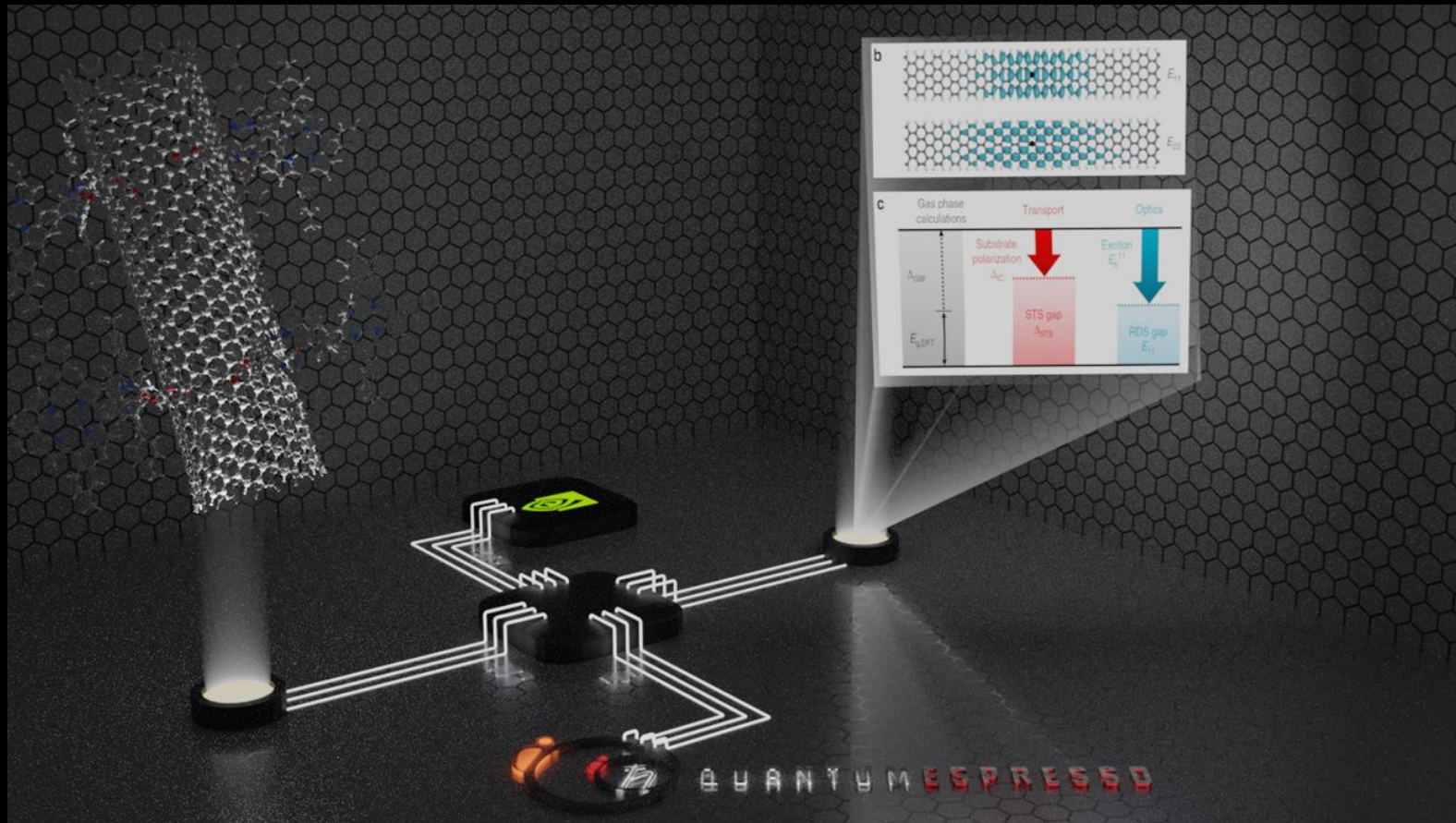
Quantum mechanics for materials



Quantum mechanics for materials



Quantum mechanics for materials



Materials design at the Exascale



LIGHTHOUSE CODES



DOMAIN EXPERTS & CODE DEVELOPERS



HPC EXPERTS & DATA CENTRES



TECHNOLOGY & CO-DESIGN PARTNERS



Coe for HPC applications in material science

exploit **frontier** HPC
for material science research in strong
link with **scientific communities**

CODE PORTING

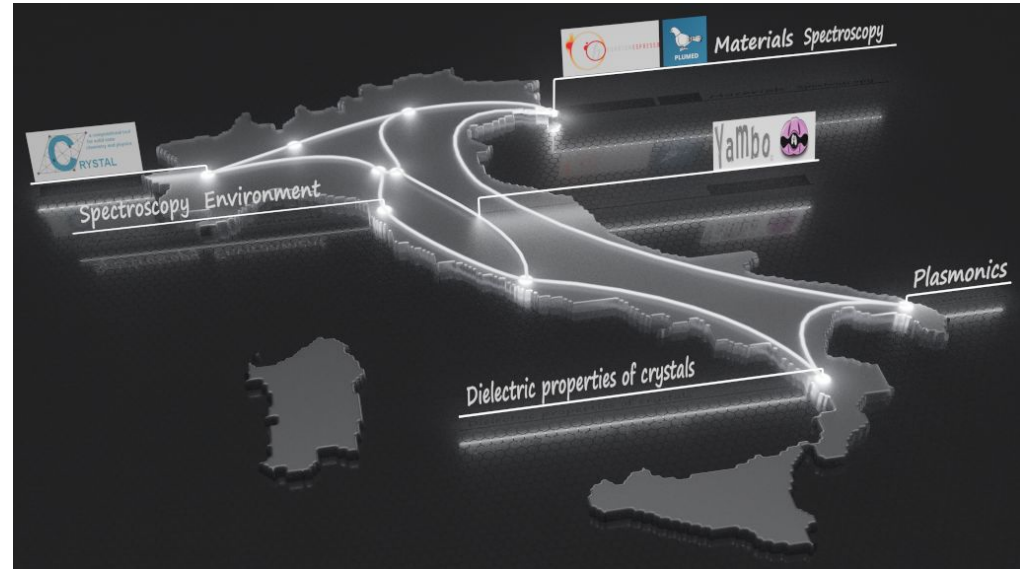
CO-DESIGN

HTC ECOSYSTEM

ICSC National Research Centre

for High Performance Computing, Big Data and Quantum Computing

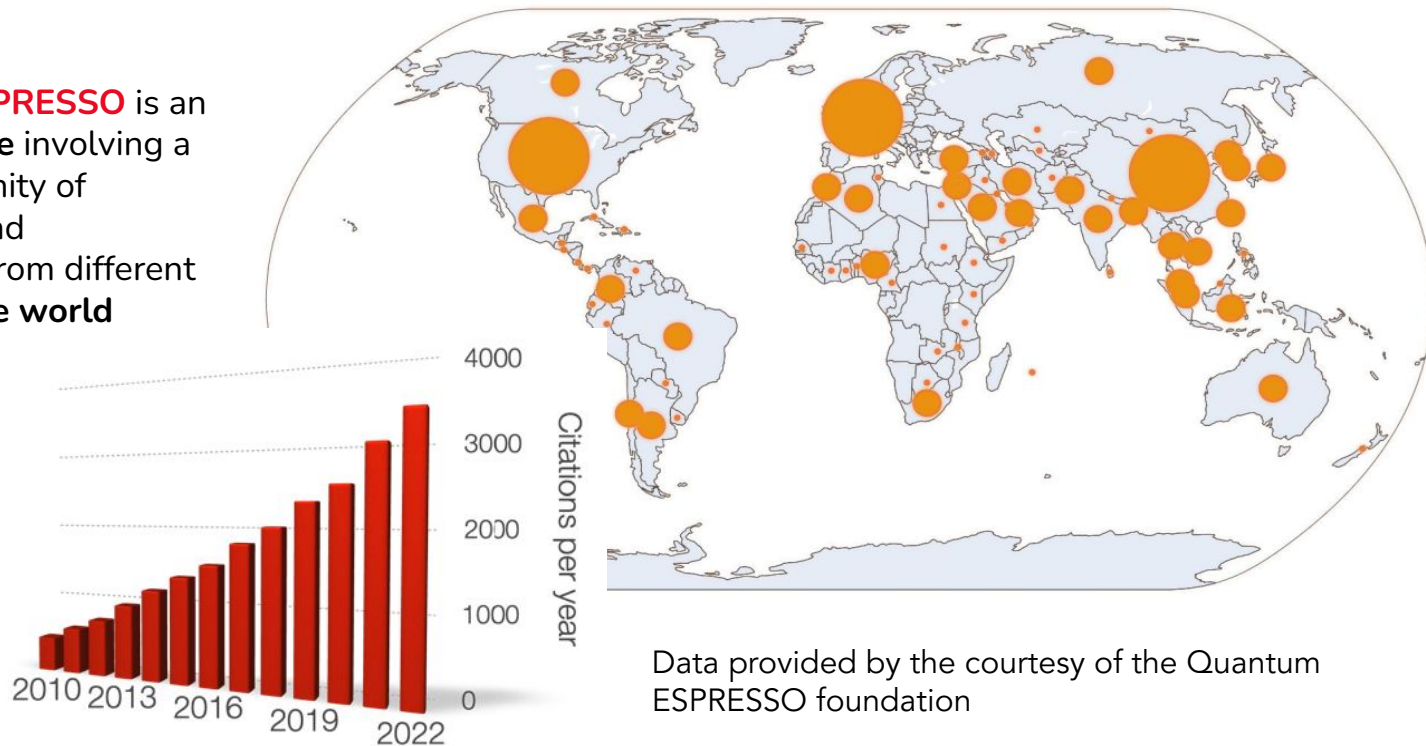
Flagship codes



The Quantum ESPRESSO suite

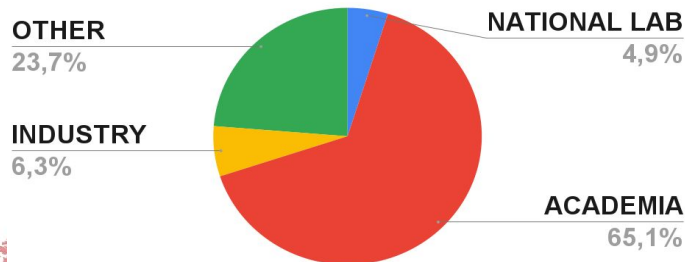
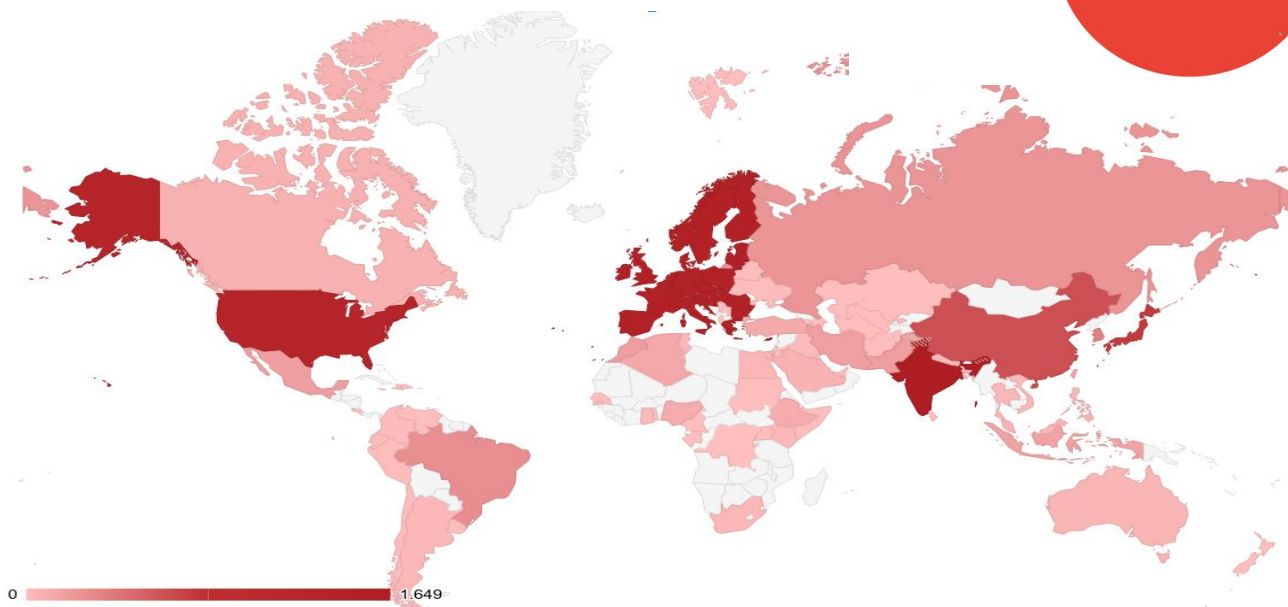
Geographic distribution of the authors of the articles citing the main reference articles as QuantumESPRESSO

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



The Quantum ESPRESSO suite

35000+ download of the code from the website in 2022, mostly from Europe, USA, India and China



Geographic distribution and main professional fields of people who have downloaded QE from the website since the beginning of 2022

The Quantum ESPRESSO suite

DENSITY FUNCTIONAL THEORY

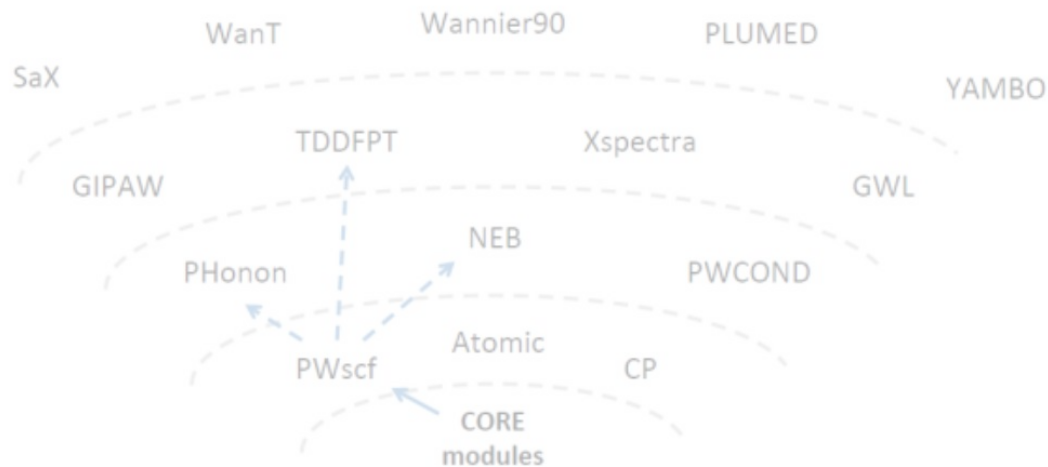
$$\left[-\frac{\hbar^2}{2m} \nabla^2 + V_s(\mathbf{r}) \right] \varphi_i(\mathbf{r}) = \varepsilon_i \varphi_i(\mathbf{r})$$

PLANE WAVES & PSEUDOPOTENTIAL

$$\varphi_\alpha(\mathbf{r}) = \frac{1}{\sqrt{\Omega}} \exp[iG_\alpha \cdot \mathbf{r}]$$

DUAL SPACE TECHNIQUE

$$\psi(\mathbf{r}) \rightarrow \psi(\mathbf{k}) \rightarrow \psi(\mathbf{r})$$



The Quantum ESPRESSO suite

DENSITY FUNCTIONAL THEORY

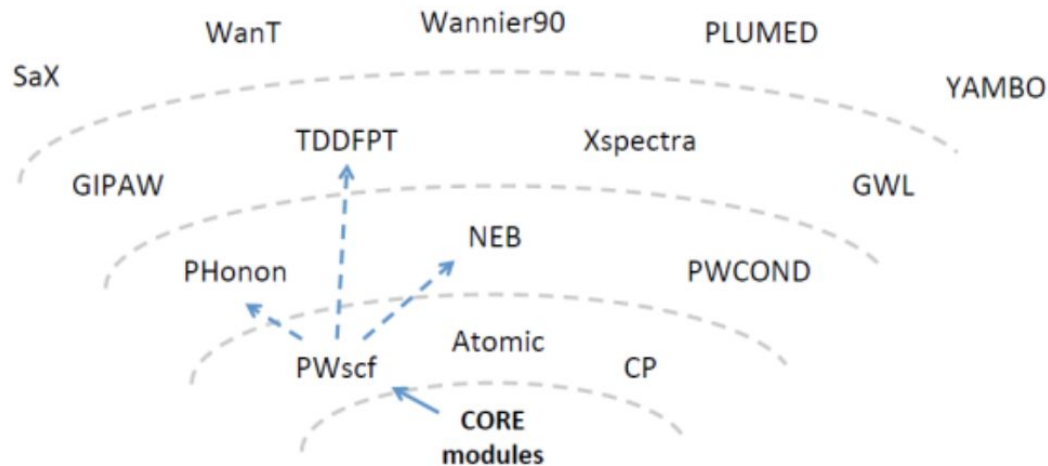
$$\left[-\frac{\hbar^2}{2m} \nabla^2 + V_s(\mathbf{r}) \right] \varphi_i(\mathbf{r}) = \varepsilon_i \varphi_i(\mathbf{r})$$

PLANE WAVES & PSEUDOPOTENTIAL

$$\varphi_\alpha(\mathbf{r}) = \frac{1}{\sqrt{\Omega}} \exp[iG_\alpha \cdot \mathbf{r}]$$

DUAL SPACE TECHNIQUE

$$\psi(\mathbf{r}) \rightarrow \psi(\mathbf{k}) \rightarrow \psi(\mathbf{r})$$



Quantum ESPRESSO on HPC clusters

PW

Input

Output

Calculation of the electronic density ρ

Build the
KS Hamiltonian \hat{H}

Calculate KS
orbitals (ψ)

FFTs
~ 40-60%

Diagonalization
~ 20-30%

gemm/PW-loops
~ 20-40%

Pools:

0

1

...

R&G:

1

2

3

4

5

...

k_0

k_{N+1}

k_1

k_{N+2}

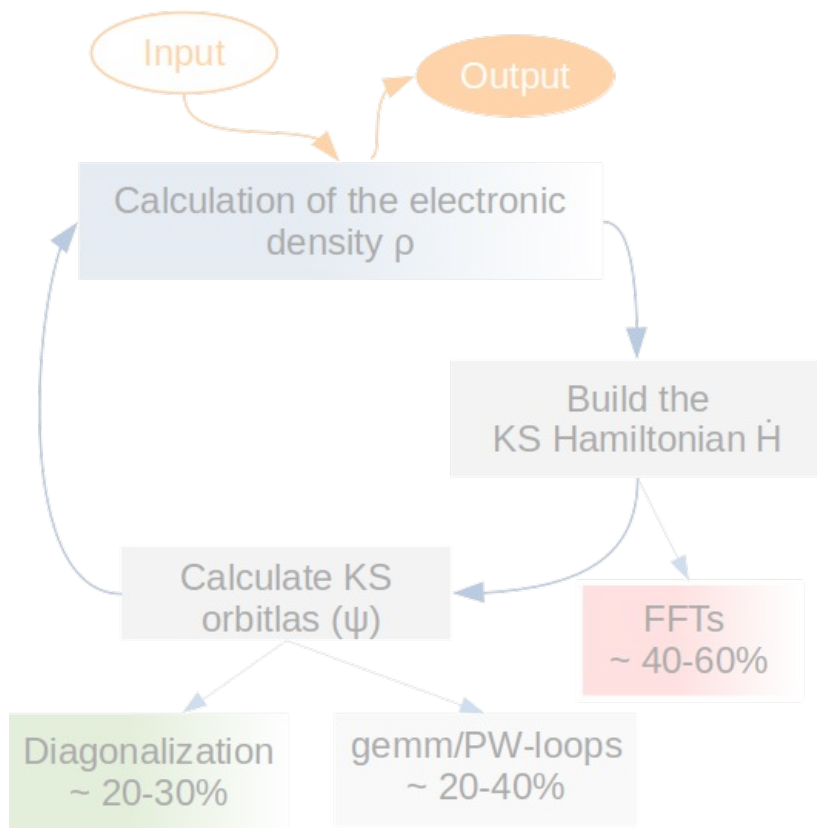
...

...

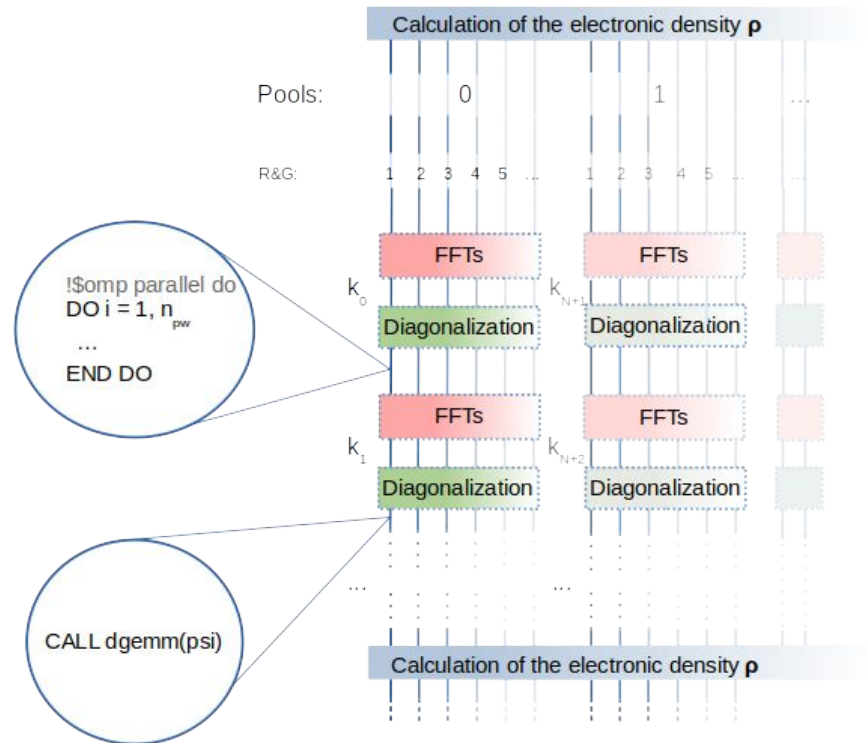
CALL dgemm(psi)

Calculation of the electronic density ρ

Quantum ESPRESSO on HPC clusters

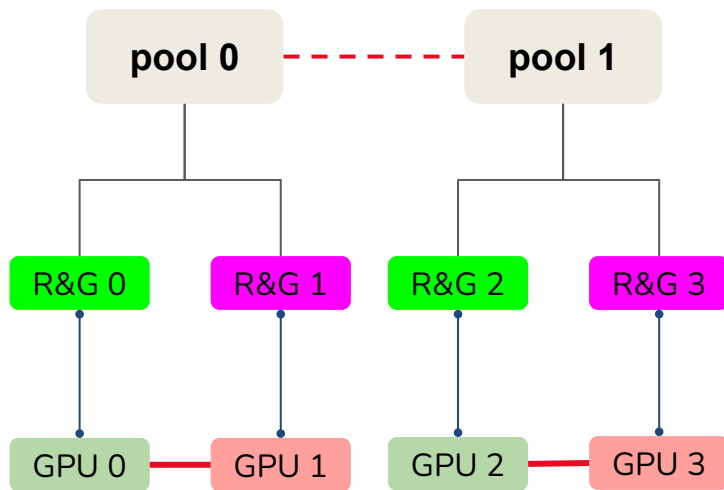


MPI + OpenMP

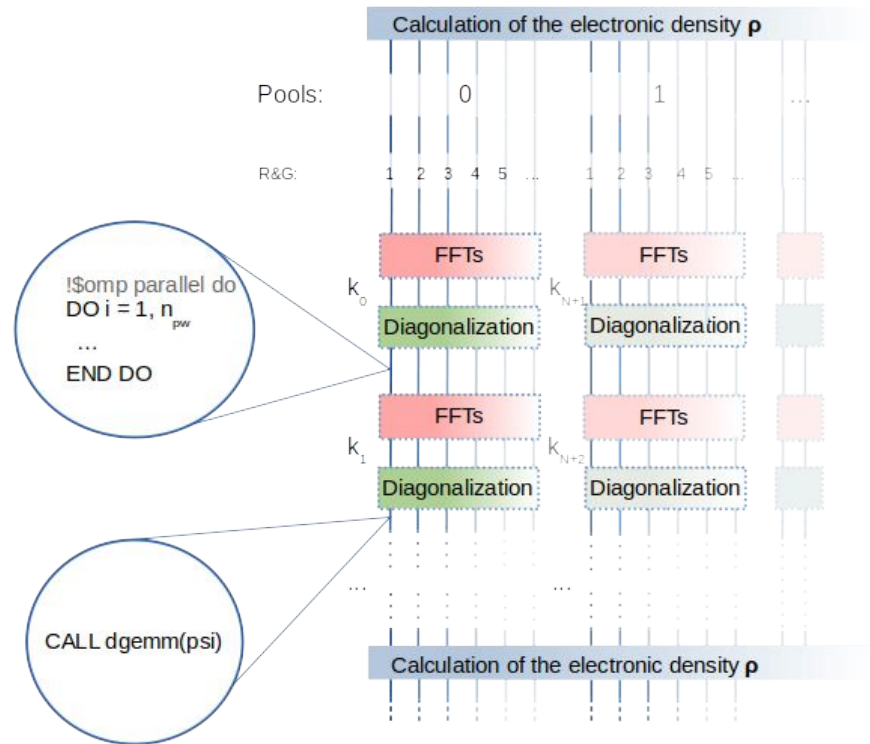


Quantum ESPRESSO on HPC clusters

MPI + OpenMP + GPU

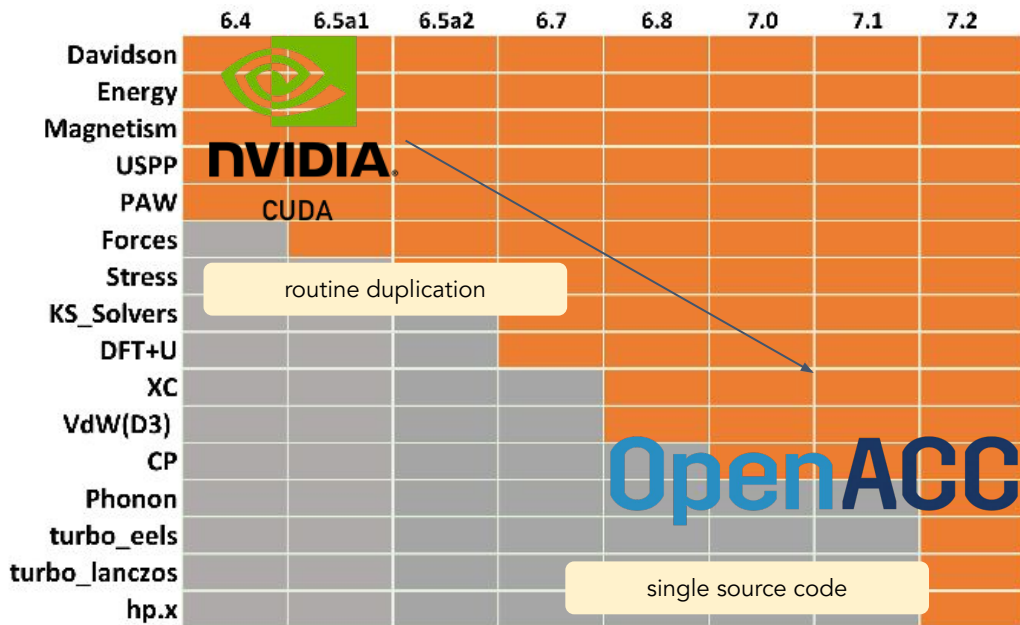


MPI + OpenMP



Towards a portable GPU version

The transition from CUDA to Openacc



DIRECTIVE-BASED
PROGRAMMING MODELS

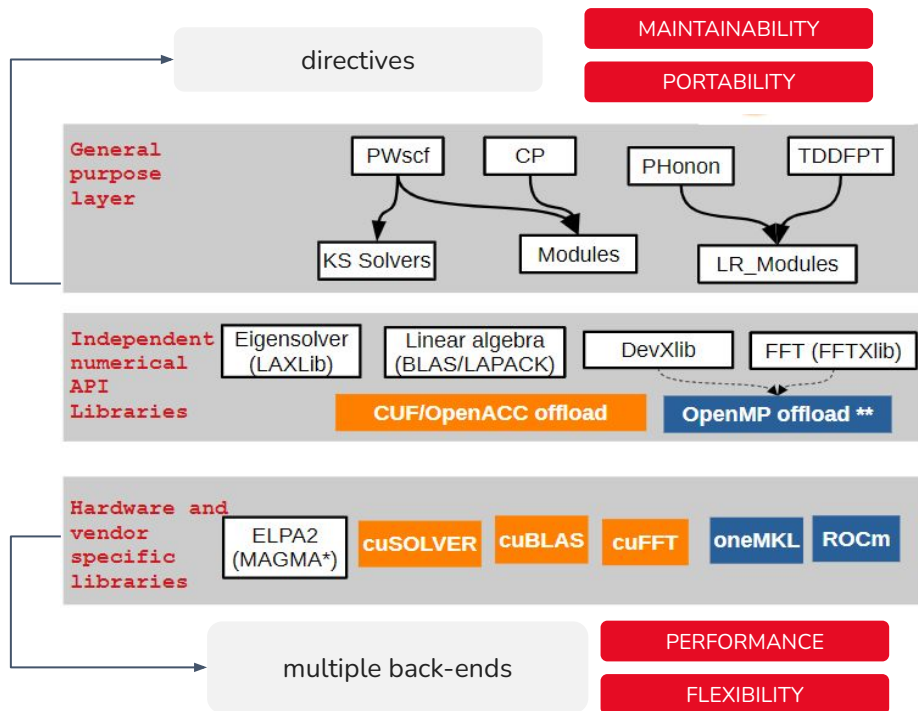
MAINTAINABLE

PORTABLE

SINGLE SOURCE CODE

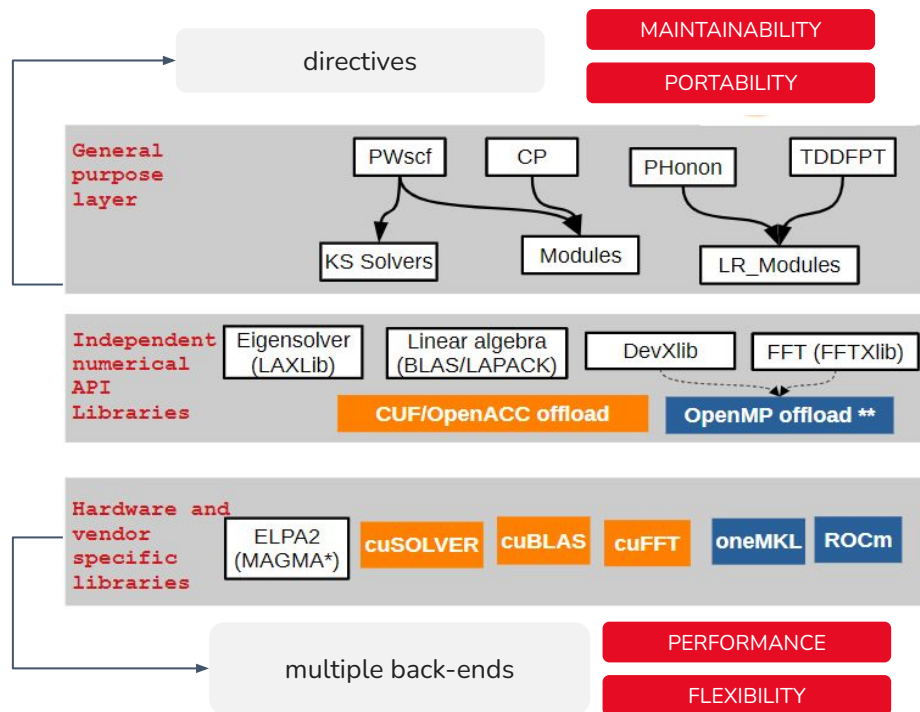
Towards a portable GPU version

Modularity supports interoperability and new programming models



Towards a portable GPU version

Modularity supports interoperability and new programming models

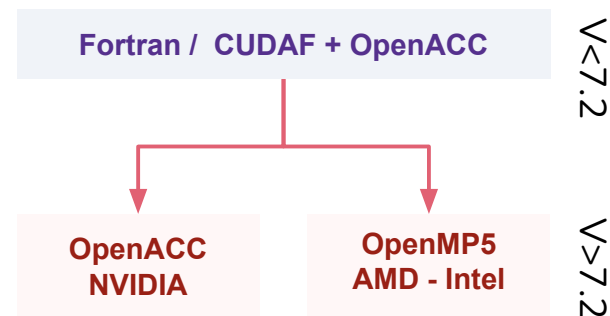
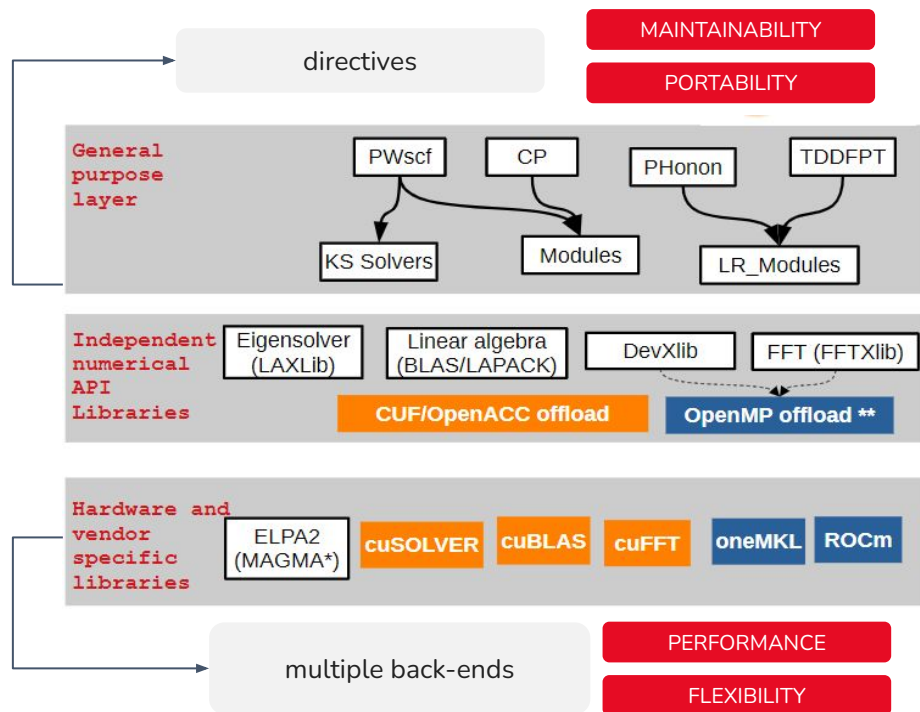


Fortran / CUDA + OpenACC

V<7.2

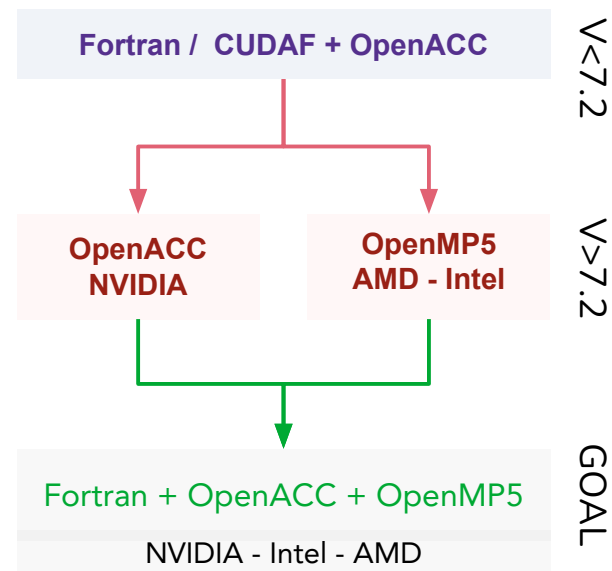
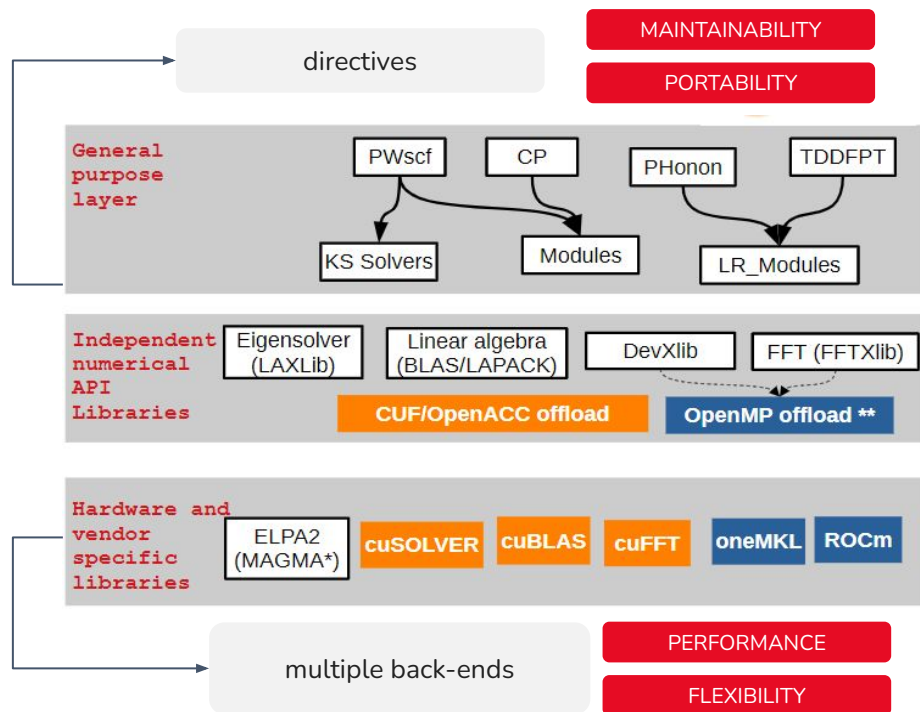
Towards a portable GPU version

Modularity supports interoperability and new programming models



Towards a portable GPU version

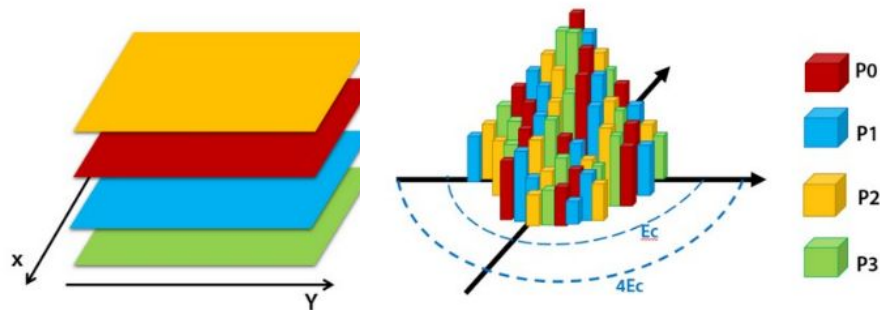
Modularity supports interoperability and new programming models



SLAB DECOMPOSITION

R planes distributed on z

G sticks distributed on x and y to balance the workload



MPI DISTRIBUTED 3D FFTs

Local 1D and 2D FFTs

+

MPI collective communications (All-to-all)

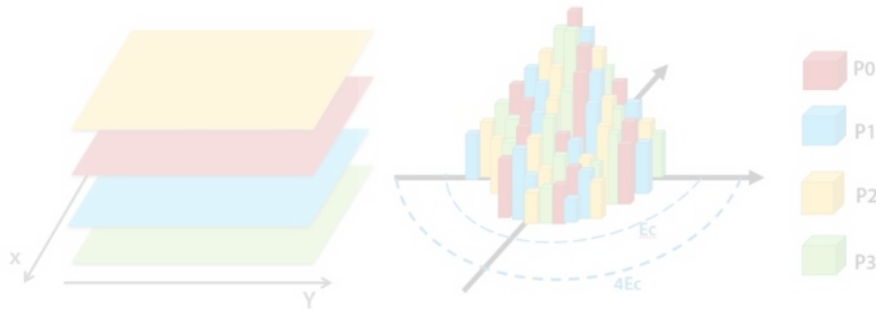


FFTXlib

SLAB DECOMPOSITION

R planes distributed on z

G sticks distributed on x and y to balance the workload

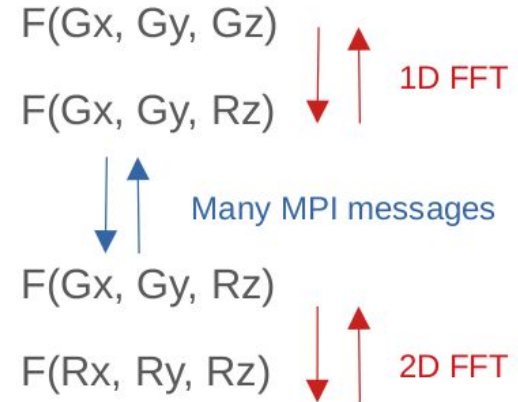


MPI DISTRIBUTED 3D FFTs

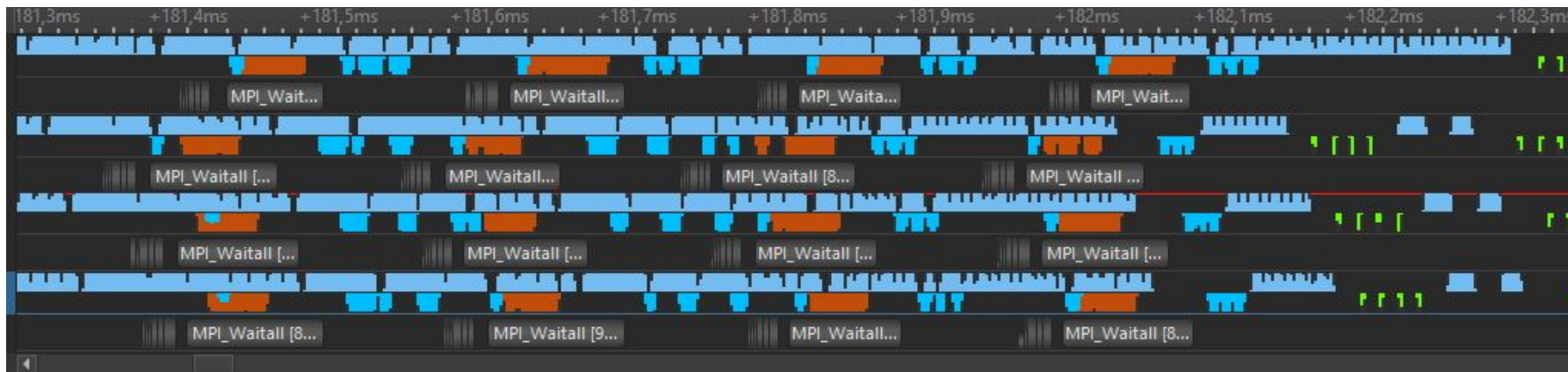
Local 1D and 2D FFTs

+

MPI collective communications (All-to-all)

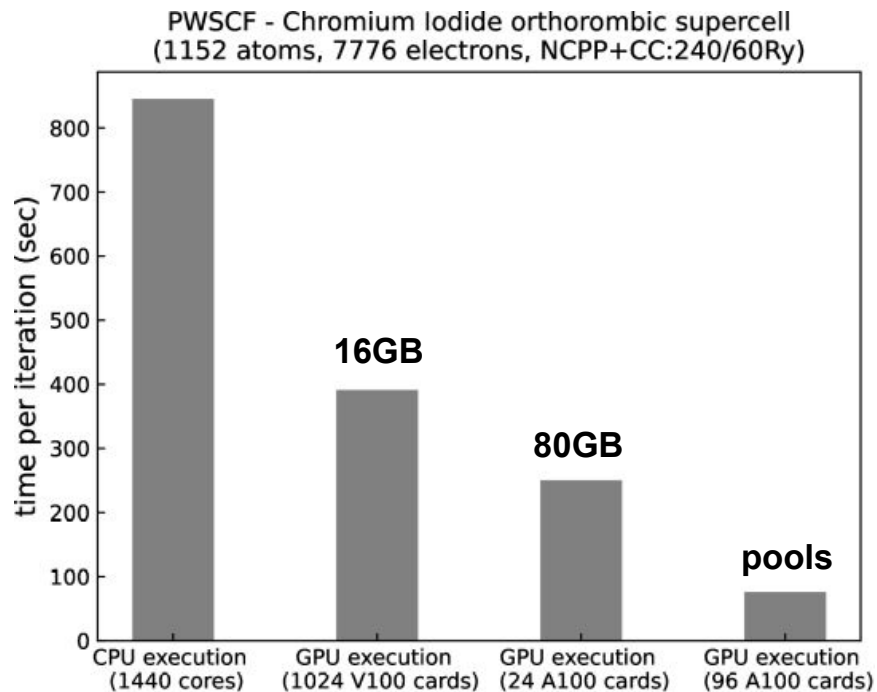


Batched FFTs for the GPU driver



- * batching of FFTs
- * non blocking MPI communications
- * overlap data/computation/communication by using streams
- * exploits GPUDirect

Performance assessment



- Significant improvement from GPU to CPU
- Time to solution decreases with GPU memory
- Pool distribution can be added to further speedup

Quantum ESPRESSO: one further step towards the exascale, I. Carnimeo et al., Journal of Chemical Theory and Computation
J. Chem. Theory Comput. **19**, 6992 (2023)

EXPANDING QE PORTABILITY WITH OpenMP

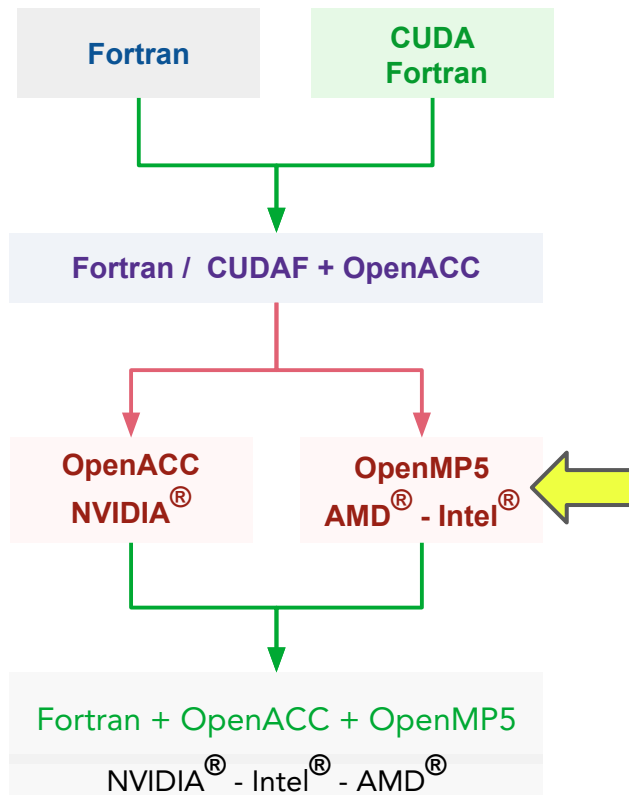
Outline

- **Porting roadmap** of QE and **present status** of the porting;
- **transition** from CUDA to **directive based** porting (openACC and OpenMP);
- results from first basic **OpenMP porting**;
- FFTs in QE (**FFTXlib** library);
- **streams** management in FFTXlib;
- results from running on **LUMI**;
- summary & outlook.

On the porting roadmap

- ◆ J. Chem. Phys. **152**, 154105 (2020)

- ◆ J. Chem. Theory Comput. **19**, 6992 (2023)



- ◆ Until v 6.8;

- ◆ from v 7.0;

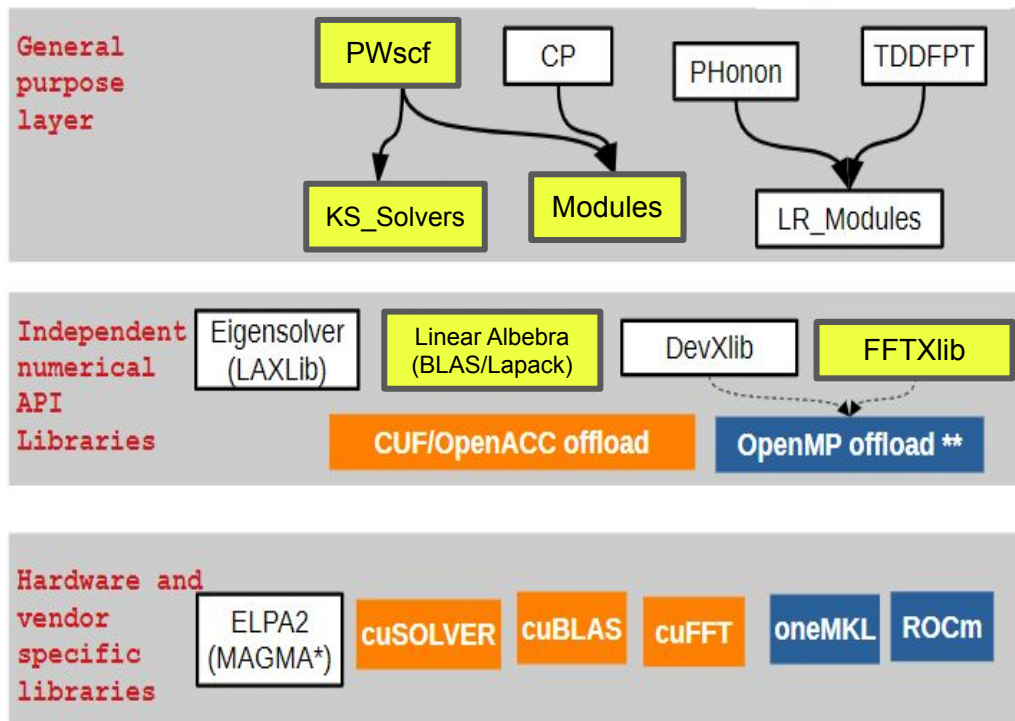
- ◆ under development;

- ◆ current goal.

QE & OpenMP5

- Main parts of **FFTXlib** ported for **Intel®** with **OpenMP5** (Giacomo Rossi);
- Intel® Hackathon (May 2022);
- first scratch of **PWscf** porting with omp on **Intel®** DevCloud (June 2022);
- **AMD®** collaboration (Ossian O'Reilly) for **low-level libraries** porting (starting July 2022);
- **LUMI** available to QE developers (October 2022);
- **MAX-3** kick-off - preliminary porting (Modena, February 2023);
- **develop_omp5** branch on **QE official repository**: <https://gitlab.com/QEF/q-e> (July 2023);
- **LUMI Hackathon** at CSC (Helsinki, September 2023).

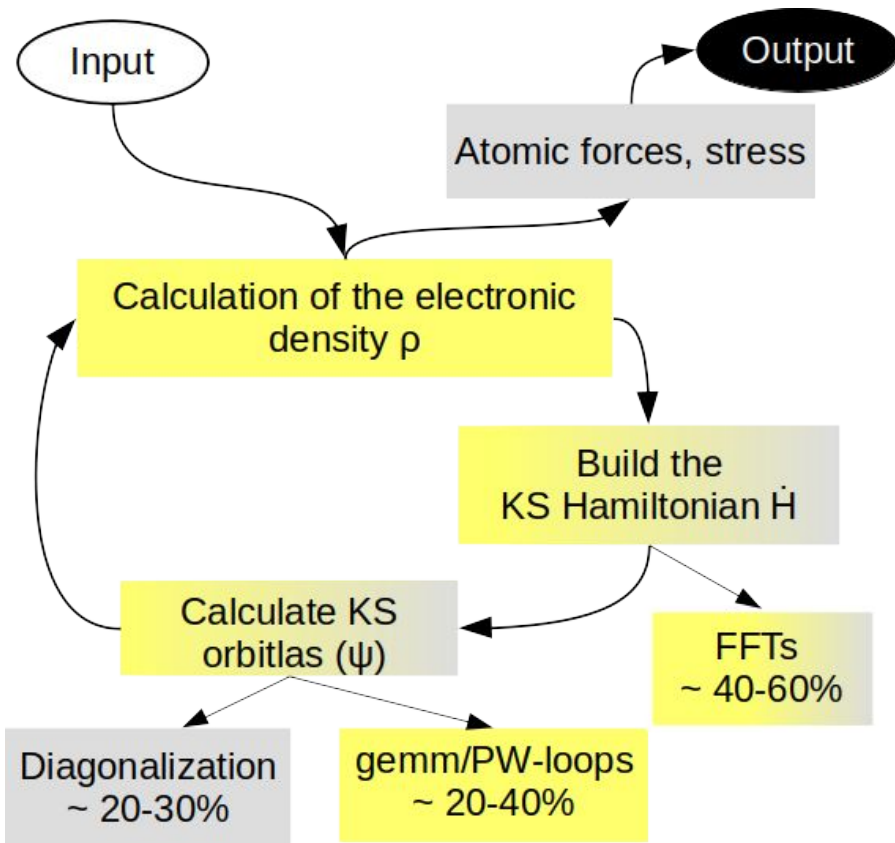
OMP porting of QE



Basic features:

- **loop** offloading;
- **global variables** offloading and pinning;
- manage different **backends** (linear algebra and FFTs);
- **streams** and/or **tasks** (for async batched FFTs).

Status of OMP porting in PWscf



Ported:

- **FFTs** (cpu driver);
- **KS_Solver** (except diagonalization);
- Interfaces for **mathematical libraries**;
- qe instrumentation routines (**rocprof**) have been added.

To be ported:

- diagonalization (zhegv);
- forces, stress;
- codes other than PW.

OpenMP5 Offload

CUF only

Host to Device

```
if ( use_gpu ) then  
  arg_d = arg  
endif
```

Routine calls

```
if ( use_gpu ) then  
  call abc( arg_d )  
else  
  call abc( arg )  
endif
```

Interfaces

```
interface abc  
  subroutine abc_cpu( v )  
  subroutine abc_gpu( v_d )  
end interface
```

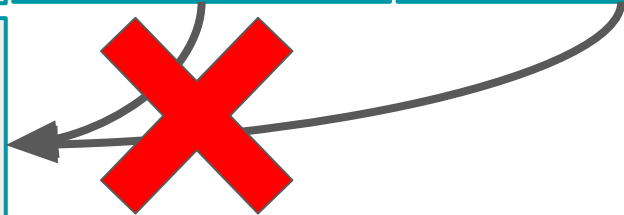
OpenMP5 Offload

	CUF only	CUF interfaces OpenACC parent code
Host to Device	<pre>if (use_gpu) then arg_d = arg endif</pre>	<pre>!\$acc update device(arg)</pre>
Routine calls	<pre>if (use_gpu) then call abc(arg_d) else call abc(arg) endif</pre>	<pre>!\$acc host_data use_device(arg) call abc(arg) !\$acc end host_data</pre>
Interfaces	<pre>interface abc subroutine abc_cpu(v) subroutine abc_gpu(v_d) end interface</pre>	

OpenMP5 Offload

	CUF only	CUF interfaces OpenACC parent code	OpenACC only
Host to Device	<pre>if (use_gpu) then arg_d = arg endif</pre>	<pre>!\$acc update device(arg)</pre>	
Routine calls	<pre>if (use_gpu) then call abc(arg_d) else call abc(arg) endif</pre>	<pre>!\$acc host_data use_device(arg) call abc(arg) !\$acc end host_data</pre>	<pre>call abc_acc(arg)</pre>
Interfaces	<pre>interface abc subroutine abc_cpu(v) subroutine abc_gpu(v_d) end interface</pre>		<pre>subroutine abc_acc(v)</pre>

OpenMP5 Offload

	CUF only	CUF interfaces OpenACC parent code	OpenACC only	OpenACC + OpenMP5
Host to Device	<pre>if (use_gpu) then arg_d = arg endif</pre>	!\$acc update device(arg)		!\$acc update device(arg) !\$omp target update to(arg)
Routine calls	<pre>if (use_gpu) then call abc(arg_d) else call abc(arg) endif</pre>	!\$acc host_data use_device(arg) call abc(arg) !\$acc end host_data	call abc_acc(arg)	<pre>#if def __OPENACC call abc_acc(arg) #elif def __OPENMP call abc_omp(arg) #endif</pre>
Interfaces	interface abc subroutine abc_cpu(v) subroutine abc_gpu(v_d) end interface			

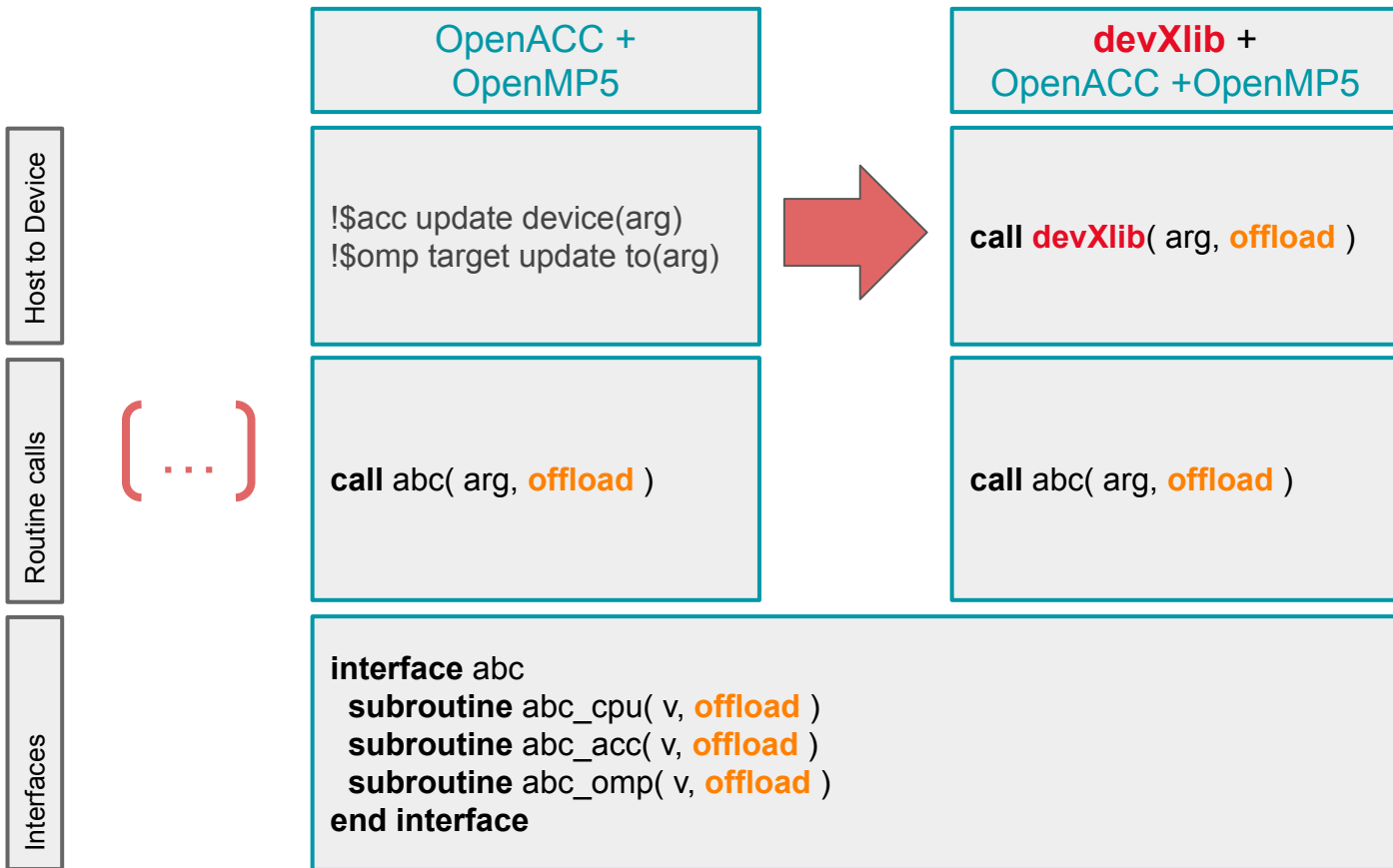
OpenMP5 Offload

	CUF only	CUF interfaces OpenACC parent code	OpenACC only	OpenACC + OpenMP5
Host to Device	if (use_gpu) then arg_d = arg endif	!\$acc update device(arg)		!\$acc update device(arg) !\$omp target update to(arg)
Routine calls	if (use_gpu) then call abc(arg_d) else call abc(arg) endif	!\$acc host_data use_device(arg) call abc(arg) !\$acc end host_data	call abc_acc(arg)	#if def __OPENACC call abc_acc(arg) #elif def __OPENMP call abc_omp(arg) #endif
Interfaces	interface abc subroutine abc_cpu(v) subroutine abc_gpu(v_d) end interface		subroutine abc_acc(v)	subroutine abc_acc(v) subroutine abc_omp(v)

OpenMP5 Offload

	CUF only	CUF interfaces OpenACC parent code	OpenACC only	OpenACC + OpenMP5
Host to Device	if (use_gpu) then arg_d = arg endif	!\$acc update device(arg)		!\$acc update device(arg) !\$omp target update to(arg)
Routine calls	if (use_gpu) then call abc(arg_d) else call abc(arg) endif	!\$acc host_data use_device(arg) call abc(arg) !\$acc end host_data	call abc(arg, offload)	call abc(arg, offload)
Interfaces	interface abc subroutine abc_cpu(v) subroutine abc_gpu(v_d) end interface		interface abc subroutine abc_cpu(v, offload) subroutine abc_acc(v, offload) subroutine abc_omp(v, offload) end interface	

OpenMP5 Offload



The Yambo group in Modena is developing a portable library (**devXlib**) to manage porting to multiplatform heterogeneous architectures

Main developers:
A. Ferretti (CNR-NANO)
N. Spallanzani (CNR-NANO)
G. Rossi (Intel)

devXlib

- MAX component for wrapping device-oriented routines and utilities
- Fortran 90/95 + iso_c_binding from F2003
- Source code is generated starting from Jinja2 sources via Python scripts
- Main components:
 - Device buffers
 - Device memcpy / memset
 - Device allocation*
 - Device linear algebra
 - Auxiliary functions
 - complex conjugate
 - scalar add
 - scale



OpenMP offload enabling
wasn't difficult: just added
few lines into the original
source code

```
#if defined( __DXL_CUDA)
  ALLOCATE (good%space(d) , stat=info)
#elif defined( __DXL_OPENMP_GPU)
  cloc = omp_target_alloc(d, omp_get_default_device())
  CALL c_f_pointer( cloc, good%space, [ d ] )
#else
  ALLOCATE (good%space(d) , stat=info)
#endif
```

```
!DEV_ACC data present(A,X,Y)
!DEV_ACC host_data use_device(A,X,Y)
!DEV_OMPGPU target variant dispatch use_device_ptr(A,X,Y)
#ifdef __DXL_CUBLAS
  call cublasZGEMV(TRANS,M,N,ALPHA,A,LDA,X,INCX,BETA,Y,INCX)
#else
  call ZGEMV(TRANS,M,N,ALPHA,A,LDA,X,INCX,BETA,Y,INCX)
#endif
!DEV_OMPGPU end target variant dispatch
!DEV_ACC end host_data
!DEV_ACC end data
```

1. [max-centre / Components / deviceXlib · GitLab](https://max-centre.github.io/Components/deviceXlib)

Wrappers instead of interfaces

```
!-----  
SUBROUTINE wave_r2g( f_in, f_out, dfft, igk, howmany_set, omp_mod )  
!-----  
!! Wave function FFT from R to G-space.  
!  
USE fft_helper_subroutines, ONLY: fftx_psi2c_gamma, fftx_psi2c_k  
#if defined( _OPENMP_GPU )  
USE fft_helper_subroutines, ONLY: fftx_psi2c_gamma_omp, fftx_psi2c_k_omp  
#endif  
USE control_flags, ONLY: many_fft  
!  
IMPLICIT NONE  
!  
...  
!  
omp_offload = .FALSE.  
omp_map      = .FALSE.  
#if defined( _OPENMP_GPU )  
IF (PRESENT(omp_mod)) THEN  
  omp_offload = omp_mod>=0 ! run FFT on device (data already mapped)  
  omp_map      = omp_mod>=1 ! map data and run FFT on device  
ENDIF  
#endif  
!
```

```
!  
!$acc host data use_device(f_in)  
IF (PRESENT(howmany_set)) THEN  
  IF(omp_offload) THEN  
#if defined( _OPENMP_GPU )  
    IF(omp_map) THEN  
      !$omp target data map(tofrom:f_in)  
      CALL fwfft_y_omp( 'Wave', f_in, dfft, howmany=howmany_set(3) )  
      !$omp end target data  
    ELSE  
      CALL fwfft_y_omp( 'Wave', f_in, dfft, howmany=howmany_set(3) )  
    ENDIF  
  #endif  
  ELSE  
    CALL fwfft( 'Wave', f_in, dfft, howmany=howmany_set(3) )  
  ENDIF  
!  
  ELSE  
    IF(omp_offload) THEN  
#if defined( _OPENMP_GPU )  
      IF(omp_map) THEN  
        !$omp target data map(tofrom:f_in)  
        CALL fwfft_y_omp( 'Wave', f_in, dfft )  
        !$omp end target data  
      ELSE  
        CALL fwfft_y_omp( 'Wave', f_in, dfft )  
      ENDIF  
    #endif  
    ELSE  
      CALL fwfft( 'Wave', f_in, dfft )  
    ENDIF  
  ENDIF  
!  
!$acc end host_data  
!  
IF (gamma_only) THEN  
!
```

Some notes

- We need **both offloaded and non-offloaded** low level routines (e.g. FFTXlib, LAXlib) at the same time;
- **dispatch** construct currently **not available on all compilers**: not an option at the moment (in general);
- we use **wrappers with offloading switch** to sort CPU and GPU low level library calls;
- **duplication** (“*_omp*”) of **low level routines** still necessary (avoidable? In the future?);
- ***omp target*** for GPU **protected** from openACC and from CPU *omp*.

Backends

Explicit
streams

gpu/cpu
interface

No need
c_bind

Linear Algebra

cuBlas



rocBlas



oneMKL



Fourier transforms

cuFFT



hipFFT



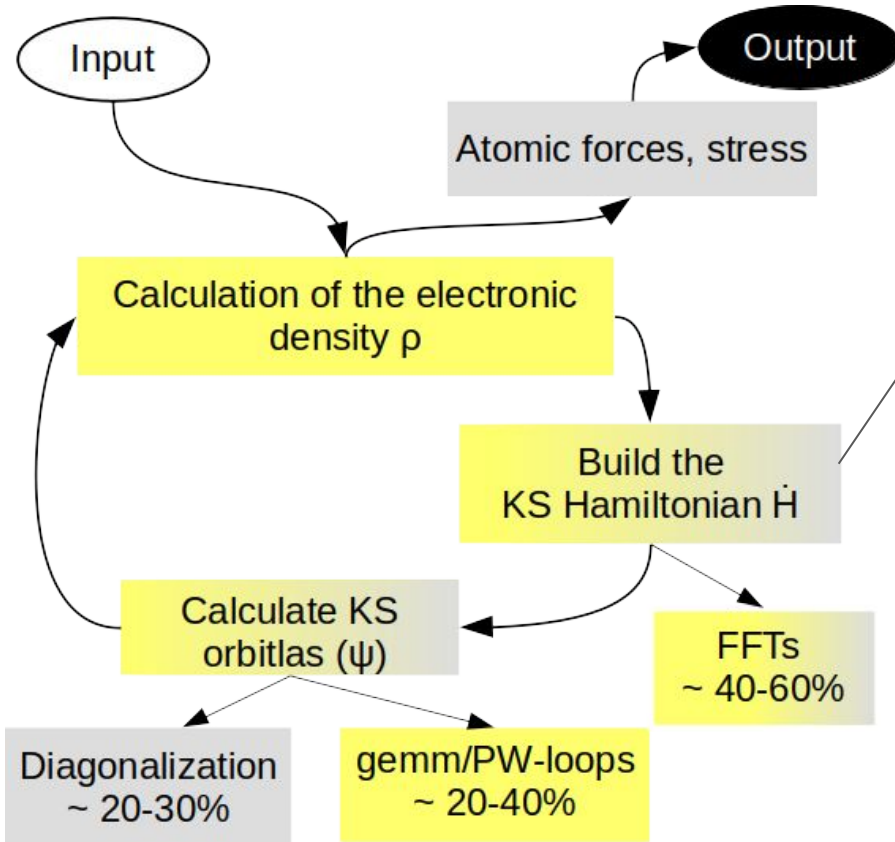
oneMKL



```

SUBROUTINE MYDGEMM2( TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, &
    BETA, C, LDC, OMP_OFFLOAD )
# if defined( _CUDA )
    use cudafor
    use cublas
# elif defined( _OPENMP_GPU )
# if defined( _ONEMKL )
    use onemkl_blas_gpu
# endif
# if defined( _ROCLAS )
    use roclblas_utils
# endif
# endif
CHARACTER*1, INTENT(IN) :: TRANSA, TRANSB
INTEGER, INTENT(IN) :: M, N, K, LDA, LDB, LDC
DOUBLE PRECISION, INTENT(IN) :: ALPHA, BETA
DOUBLE PRECISION :: A( LDA, * ), B( LDB, * ), C( LDC, * )
LOGICAL, INTENT(IN) :: OMP_OFFLOAD
# if defined( _CUDA )
    attributes(device) :: A, B, C
    CALL cublasdgemm( TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, &
        BETA, C, LDC )
# elif defined( _ONEMKL )
    IF ( OMP_OFFLOAD ) THEN
        !$omp target variant dispatch use_device_ptr(A, B, C)
        CALL dgemm( TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, &
            C, LDC )
        !$omp end target variant dispatch
    ELSE
        CALL dgemm( TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, &
            C, LDC )
    ENDIF
# elif defined( _ROCLAS )
    IF ( OMP_OFFLOAD ) CALL roclblas_dgemm( TRANSA, TRANSB, M, N, K, ALPHA, &
        A, LDA, B, LDB, BETA, C, LDC )
    IF ( .NOT. OMP_OFFLOAD ) CALL dgemm( TRANSA, TRANSB, M, N, K, ALPHA, A, &
        LDA, B, LDB, BETA, C, LDC )
# else
    CALL dgemm( TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC )
# endif
END SUBROUTINE MYDGEMM2
    
```

Exchange-Correlation library - XClib



- The XC potential is a **function of the density**;
- Calculated with a **loop** over the density grid;
- At each iteration (grid point) a **functional routine** is called.

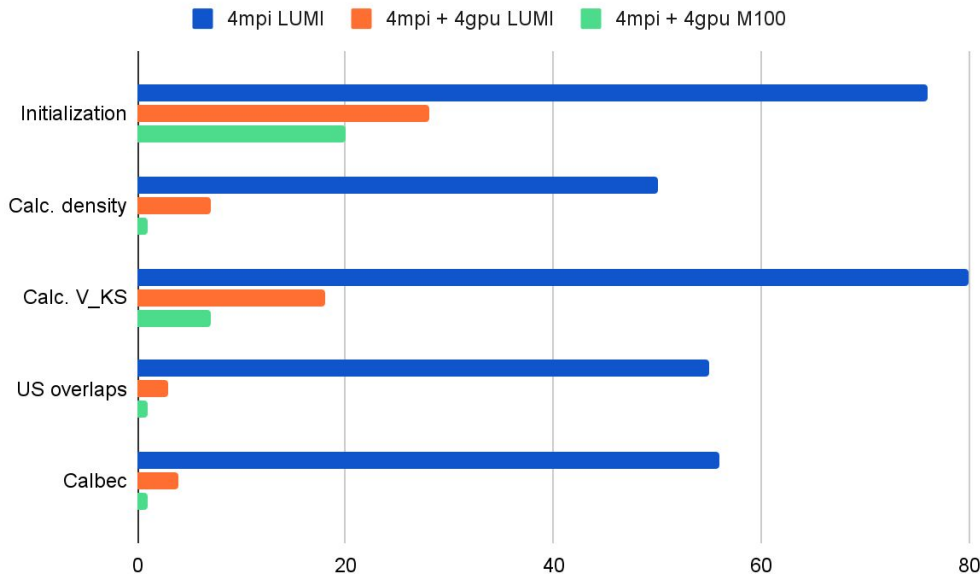
Exchange-Correlation library - XClib

```
#if defined(_OPENACC)
!$acc data present( rho_in, grho_in, sx_out, sc_out, v1x_out, v2x_out, v1c_out, v2c_out ) copy( err_out )
!$acc parallel loop
#elif defined(_OPENMP_GPU)
!$omp target teams distribute parallel do
#elif defined(_OPENMP)
!$omp parallel if(ntids==1) default(none) &
!$omp private( rho, grho, sx, sx_, xsr, v1x, v1x_, v1xsr, &
!$omp v2x, v2x_, v2xsr, sc, v1c, v2c, iflag, in_err ) &
!$omp shared( rho_in, grho_in, length, igcx, exx_started, &
!$omp grho_threshold_gga, rho_threshold_gga, gau_parameter, &
!$omp screening_parameter, exx_fraction, igcc, v1x_out, v2x_out, &
!$omp v1c_out, v2c_out, sx_out, sc_out, err_out )
!$omp do
#endif
DO ir = 1, length
!
in_err = 0
grho = grho_in(ir)
rho = ABS(rho_in(ir))
!
SELECT CASE( igcx )
CASE( 1 )
!
CALL becke88( rho, grho, sx, v1x, v2x )
!
CASE( 2 )
!
CALL ggax( rho, grho, sx, v1x, v2x )
!
CASE( 3 )
!
CALL pbex( rho, grho, 1, sx, v1x, v2x )
!
CASE( 4 )
!
```

```
!-----
MODULE exch_gga
!-----
!! GGA exchange functionals
!
CONTAINS
!
!-----
SUBROUTINE becke88( rho, grho, sx, v1x, v2x )
!-----
!! Becke exchange: A.D. Becke, PRA 38, 3098 (1988)
!! only gradient-corrected part, no Slater term included
!
USE kind_l, ONLY: DP
!
IMPLICIT NONE
!
#if defined(_OPENACC)
!$acc routine seq
#elif defined(_OPENMP_GPU)
!$omp declare target
#endif
!
REAL(DP), INTENT(IN) :: rho, grho
REAL(DP), INTENT(OUT) :: sx, v1x, v2x
!
! ... local variables
!
REAL(DP) :: rho13, rho43, xs, xs2, sa2b8, shm1, dd, dd2, ee
REAL(DP), PARAMETER :: beta=0.0042_DP
REAL(DP), PARAMETER :: third=1._DP/3._DP, two13=1.259921049894873_DP
! two13= 2^(1/3)
!
rho13 = rho**third
rho43 = rho13**4
!
```

Parent code porting - starting results

Reference benchmark: **Ausurf 112** (1 scf step). Starting results (**April 2023**) with **basic porting only**:

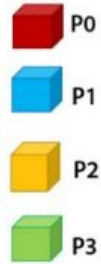
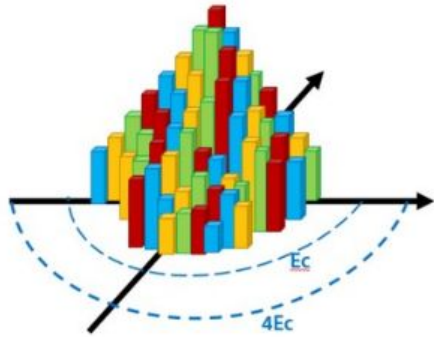


The total time on LUMI is still bigger than the reference one from M100.

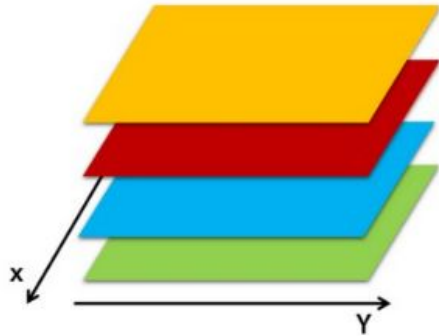
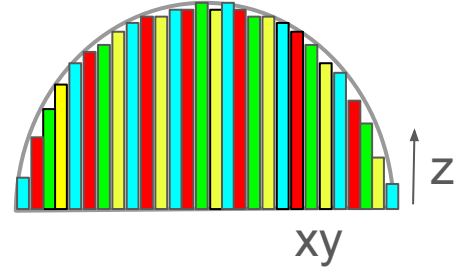
The main reasons:

- **diagonalization** backend to be implemented (~70%);
- batched **FFT**s (~15%);
- **smaller routines** to be ported (~15%)

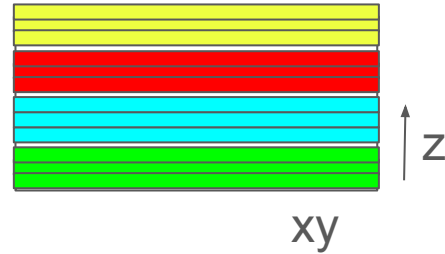
FFTXlib - slab decomposition



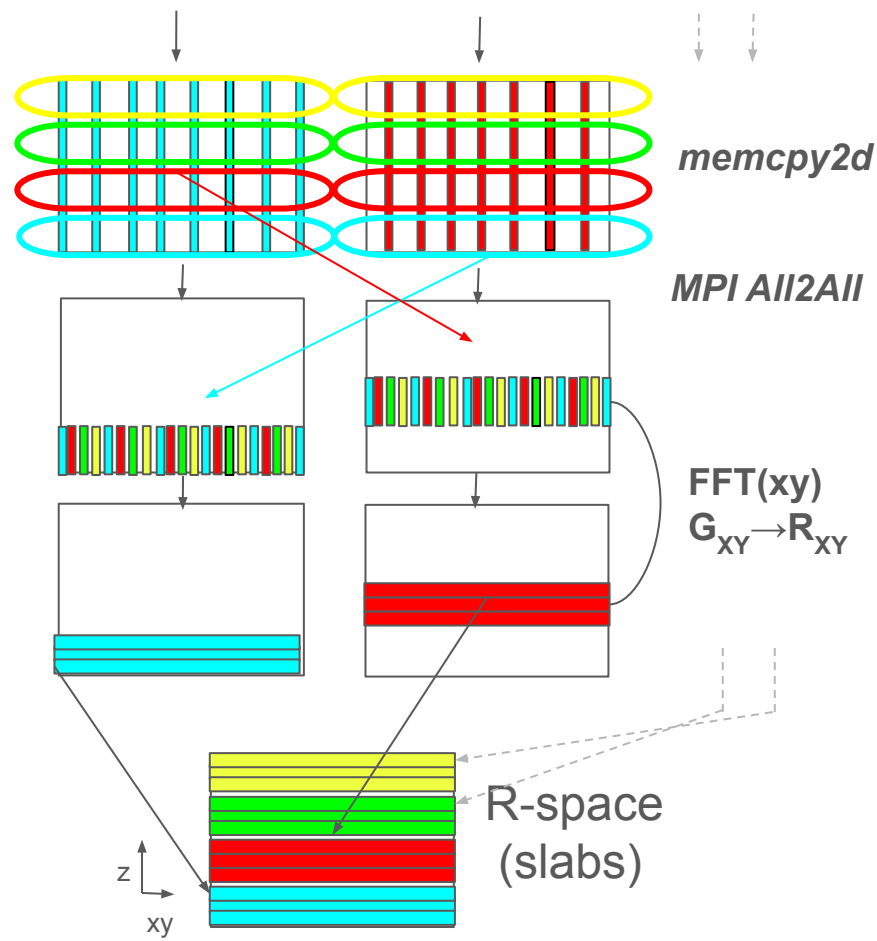
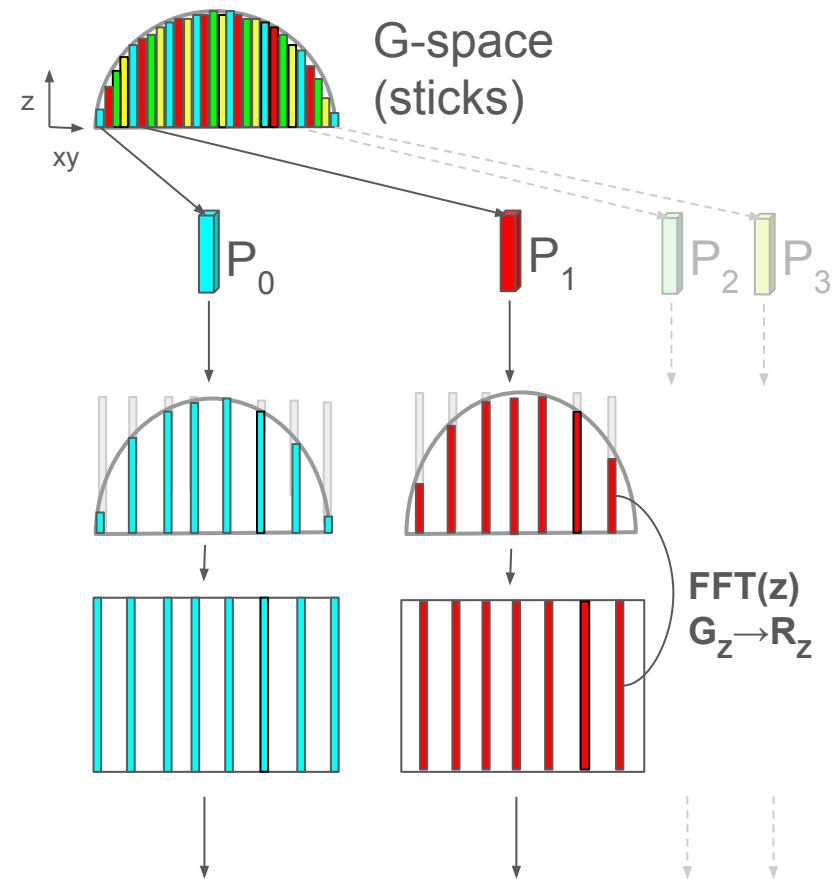
G-space
(sticks)



R-space
(slabs)



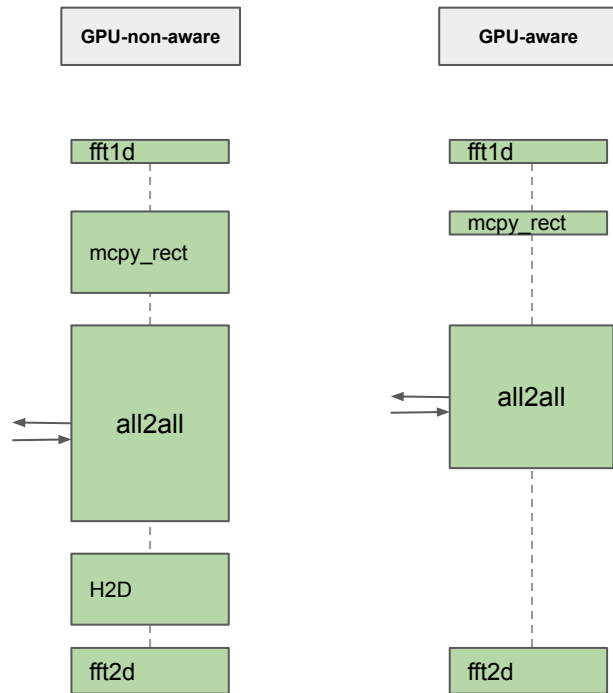
FFTXlib - slab decomposition



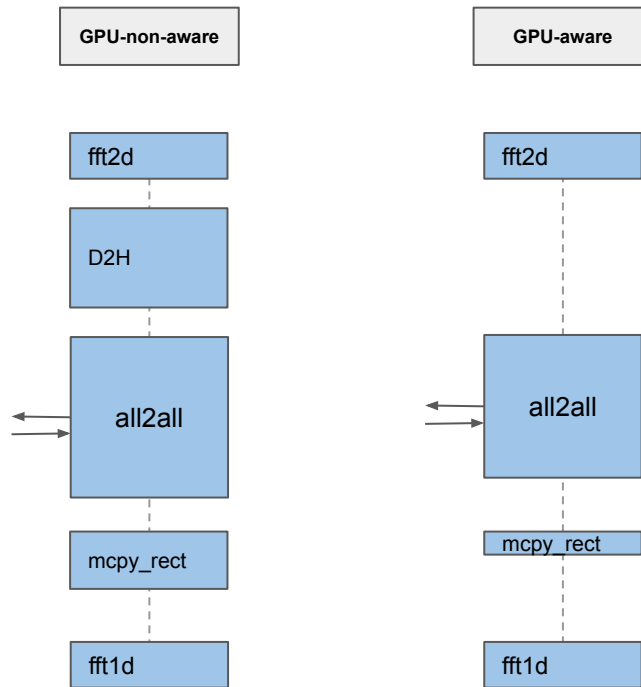
FFTXlib - standard flow

LUMI run -
4mpi,4gpu

Inverse

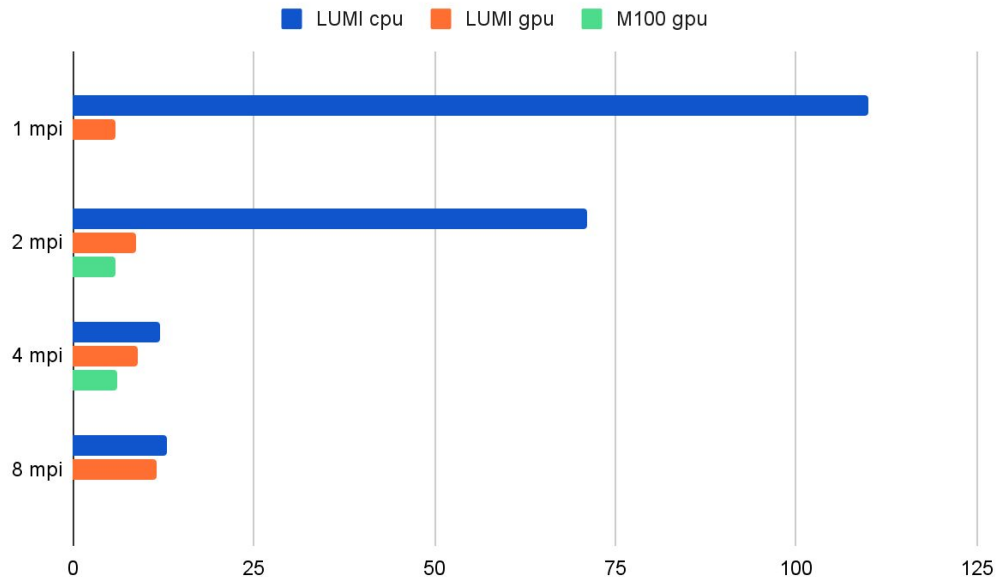


Direct



FFTXlib - basic porting

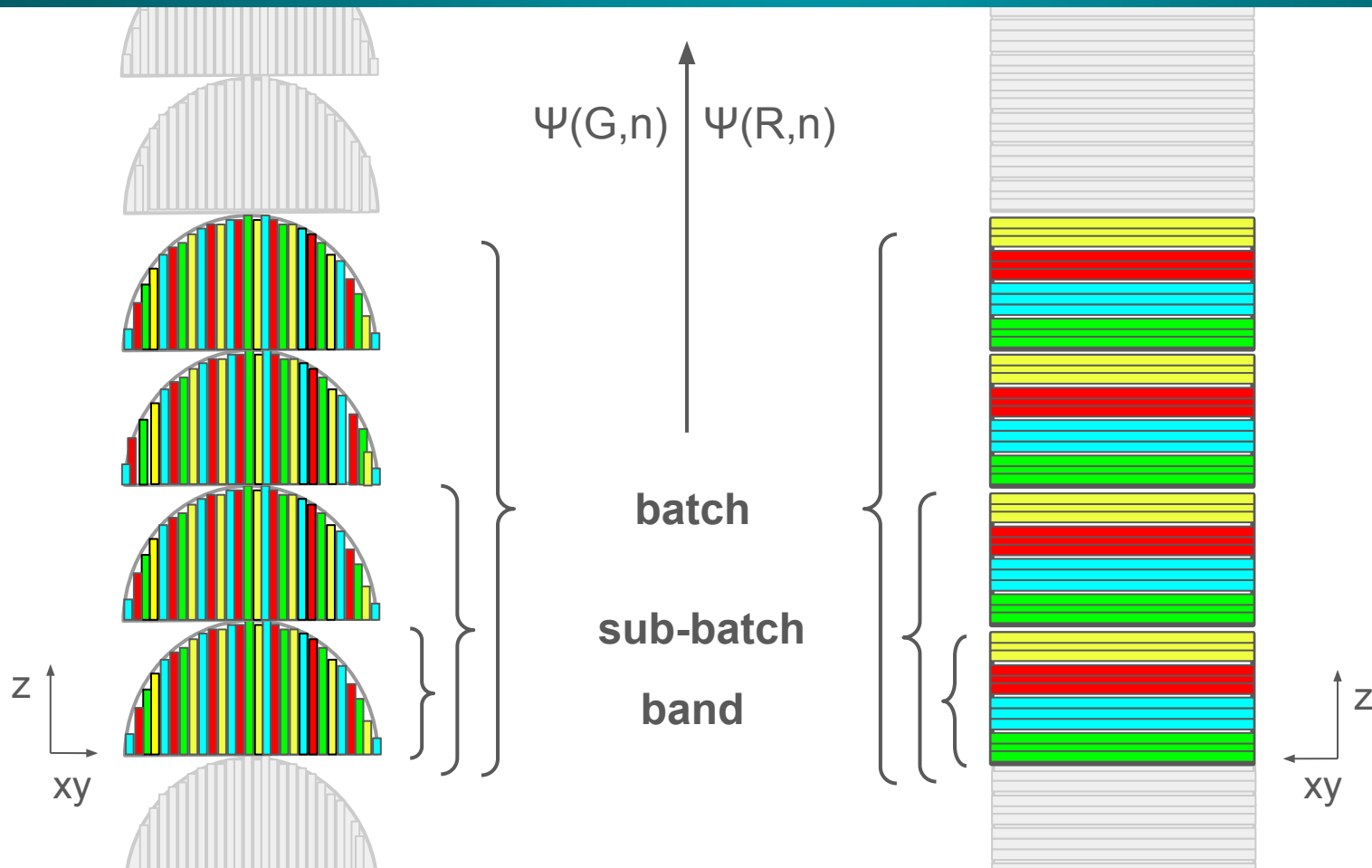
Reference benchmark: **Ausurf 112** (1 scf step). FFTXlib calls, **preliminary** results:



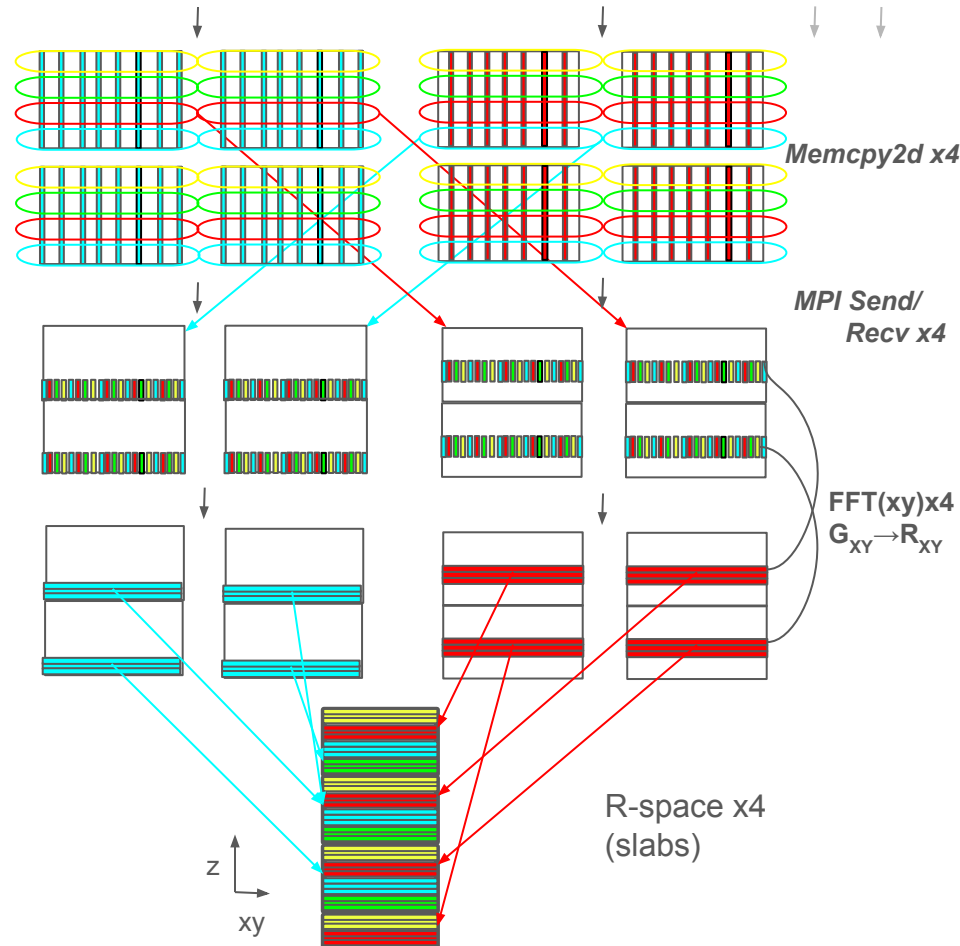
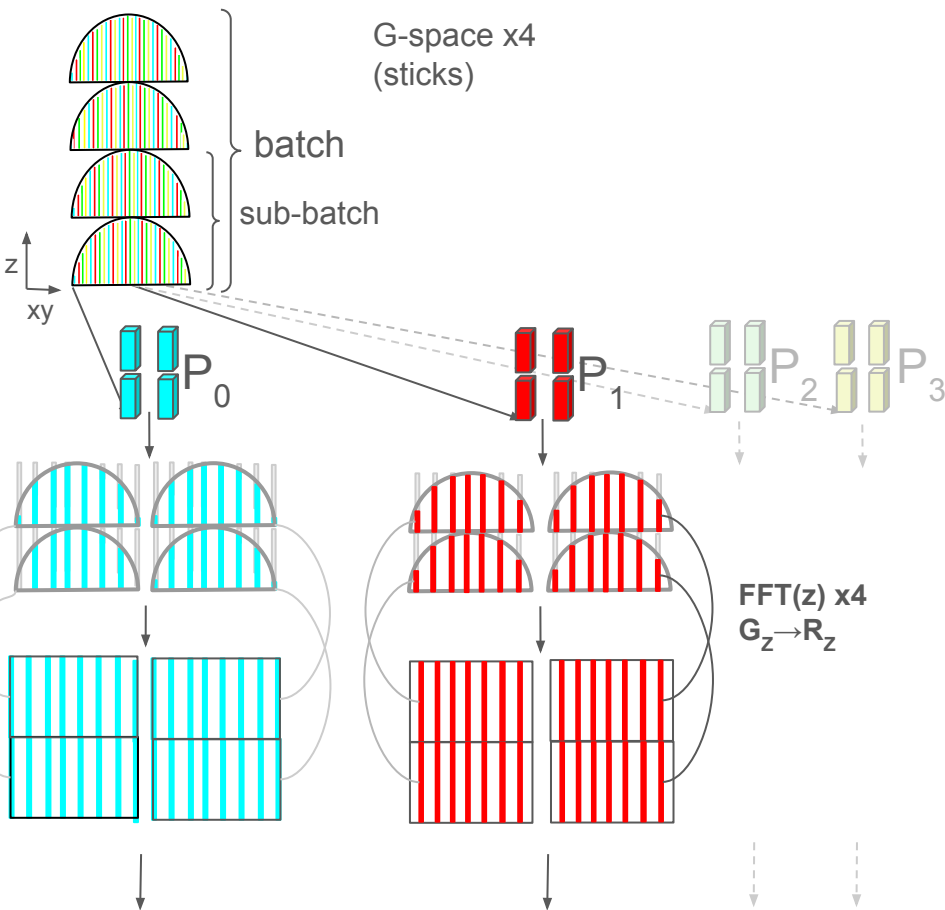
3d FFTs with SLAB decomposition (standard case):

- reference runs: M100 (**V100 gpus**)
- **overall match** between LUMI and M100;
- **H2D-D2H** part of the FFT looks a bit slower on LUMI side (still under investigation).

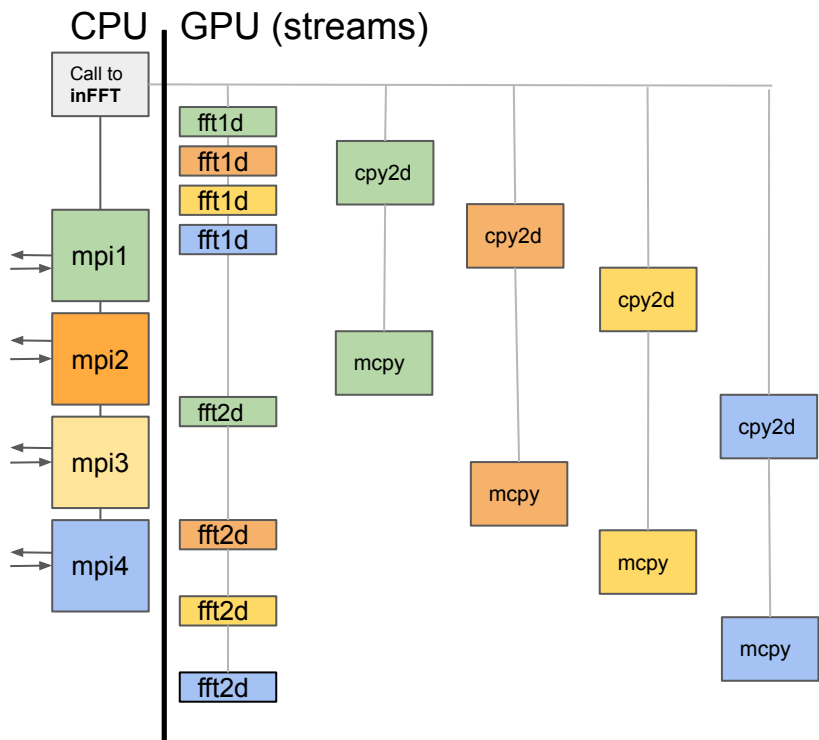
FFTXlib - many bands



FFTXlib - slab decomp. & many bands



Batched FFTs in QE - HIP



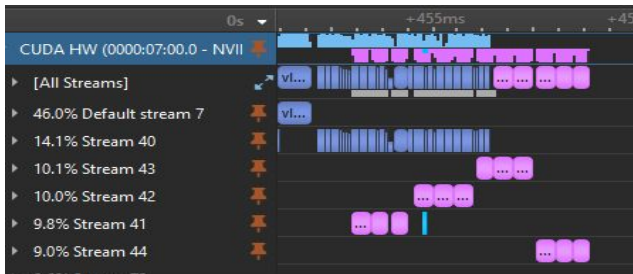
- **Batched** 3d-FFT of the **wave-function**;
- the input array divided in **4 batches** (on bands);
- 1 stream for **FFTs**, 4 streams for **data movements**;
- 4 **async mpi** communications (ISEND, IRECV).

Notes:

- **non-asynchronous memcpy**;
- memcpy operations **D2H/H2D** much more time consuming than FFT calls;
- memcpy operations **D2D** same order of FFT calls.

Hip streams & OMP

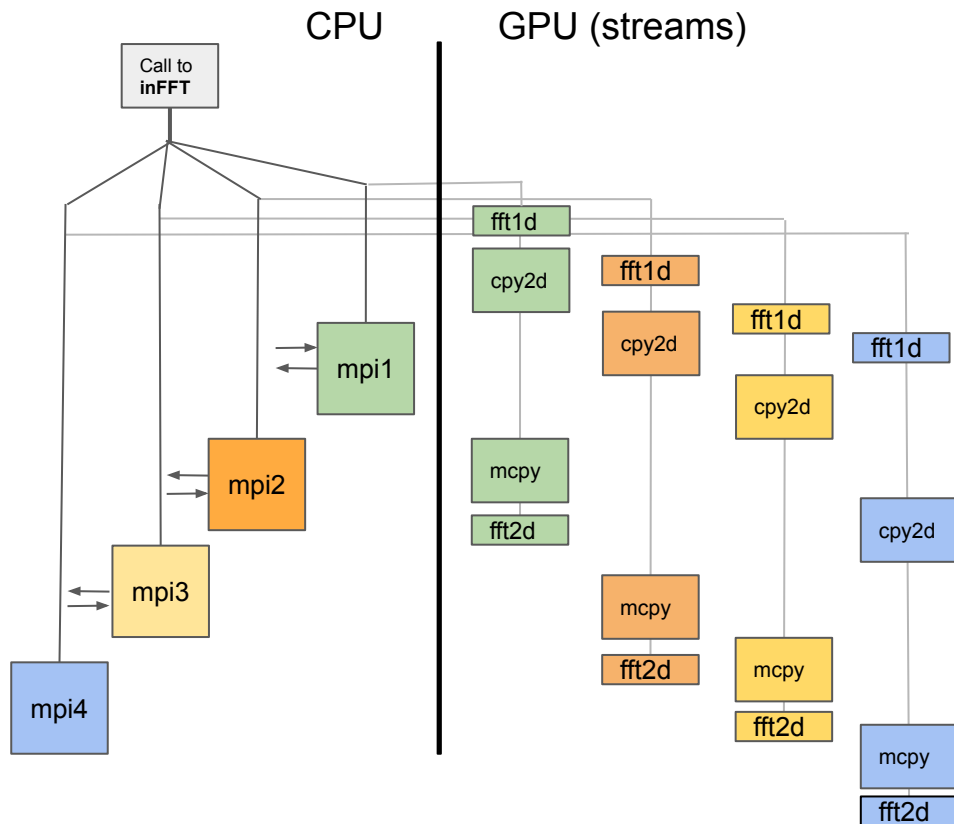
- Besides ffts and memcpy, few loops converted into hip-kernels



- Pinning variables in order to get asynchronicity with data movements from/to streams

```
!
#if defined(_OPENMP_GPU)
    pinned_alloc = omp_init_allocator(omp_default_mem_alloc, ntraits, traits)
    !$omp allocate(aux) allocator(pinned_alloc)
    ALLOCATE(aux(current_size), STAT=info)
    IF ( info /= 0 ) CALL fftx_error_( ' fft_buffers ', ' Allocation failed ', 4 )
    !$omp target enter data map(alloc:aux)
    !$omp allocate(aux2) allocator(pinned_alloc)
    ALLOCATE(aux2(current_size), STAT=info)
    IF ( info /= 0 ) CALL fftx_error_( ' fft_buffers ', ' Allocation failed ', 5 )
    !$omp target enter data map(alloc:aux2)
#else
```

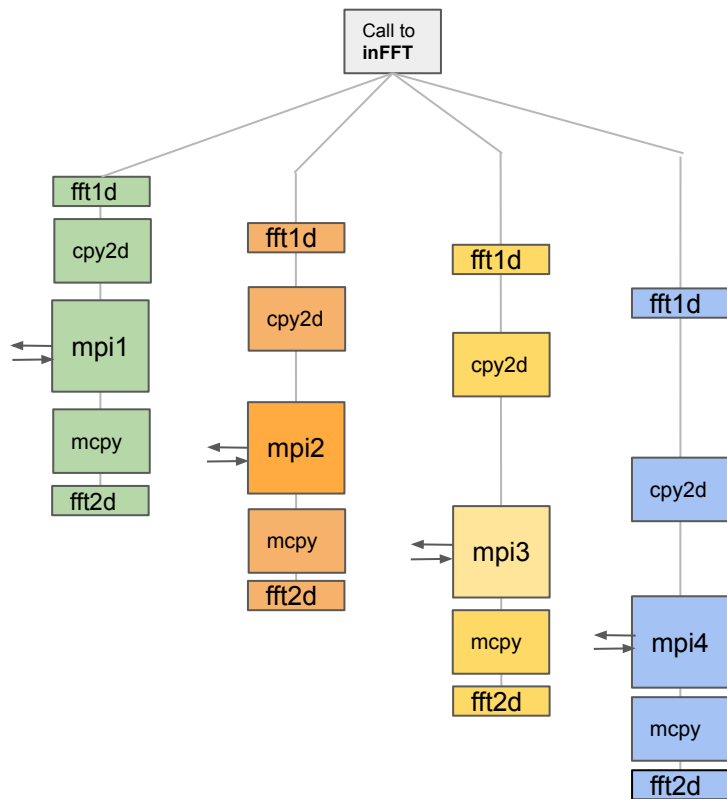
Batched FFTs in QE - OMP task+HIP



- Need to set up pure OMP porting of batched FFTs for the Intel[®] side;
- setting up a starting scheme by using **omp task** with hip streams and **detach** clause;

```
nov 28
!$omp parallel
!$omp single
DO j = 0, batchsize-1, dfft%subbatchsize
  currsz = min(dfft%subbatchsize, batchsize - j)
  !$omp task firstprivate(j,currsz) private(i) shared(ptr_callback) detach(event)
  DO i = 0, currsz - 1
    CALL cft_lz_omp( f((j+1)*dfft_nnr + 1:), sticks(me_p), n3, nx3, isgn, &
      aux(j*dfft_nnr + i*ncpx*nx3 + 1:), &
      stream=dfft%bstreams(j/dfft%subbatchsize+1) )
  ENDDO
  CALL fft_scatter_many_columns_to_planes_store_omp( dfft, aux(j*dfft_nnr+1:), nx3, &
    dfft_nnr, f(j*dfft_nnr+1:), &
    sticks, dfft%nr3p, isgn, currsz, &
    j/dfft%subbatchsize+1 )
  iadc = hipStreamAddCallback(dfft%bstreams(j/dfft%subbatchsize+1),ptr_callback ,c_loc(event), 0)
!$omp end task
ENDDO
```

Batched FFTs in QE - OMP task+dep



- Starting point: **oneMKL** does not get explicit streams as input;
- Simplest scheme given by **n tasks** associated to **n subbatches**;
- **still in progress**

Streams with multiple standards

- Full implementation of **hip-based streams** FFTs (both aware and non-aware);
- implementation of **omp task+hip streams**;
- **OpenMP tasks only** version in progress.

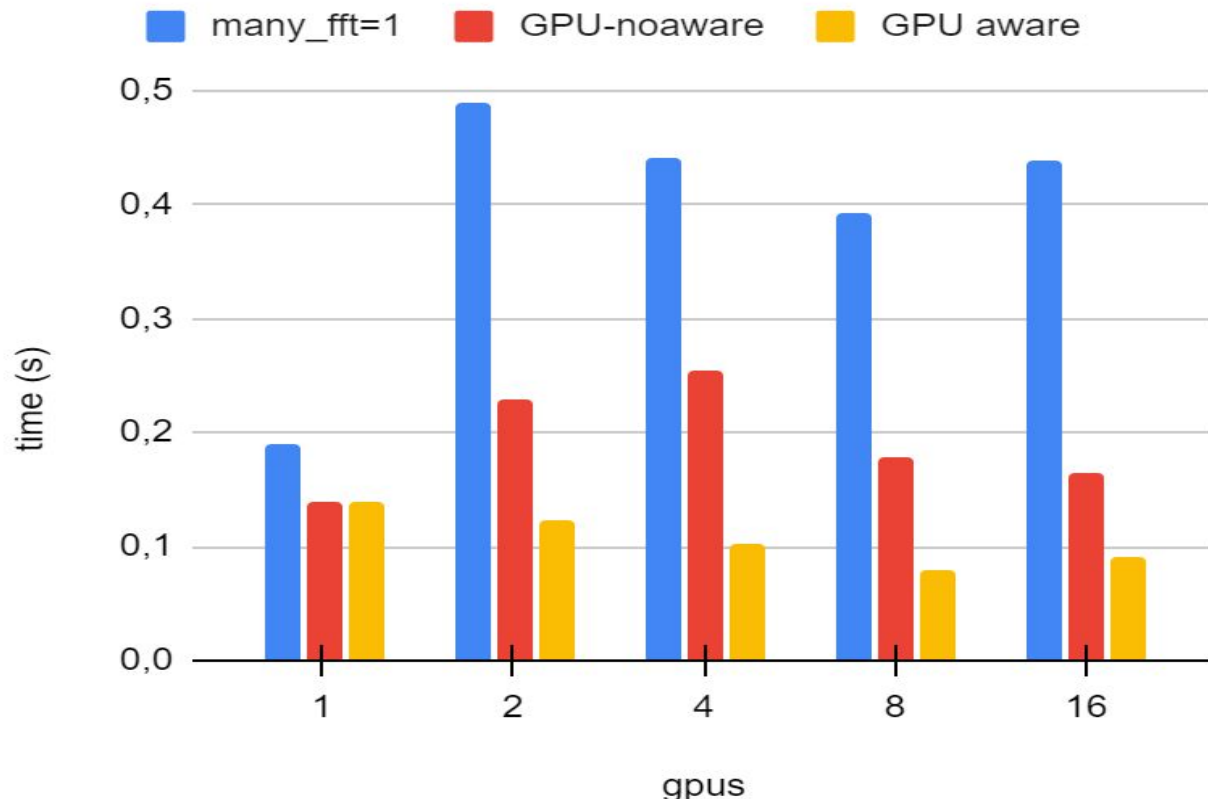
Streamed FFTXlib tested by **benchmarking** the **vloc_psi** routine of PWscf: it is the part of the scf iteration that relies the most on FFTs.

Next:

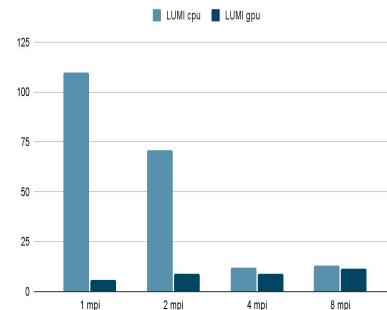
- Merge of hip streamed code with the cuda one;
- Aim: hip-cuda streams and omp tasks with minimal code duplication.

Results

vloc_psi/call



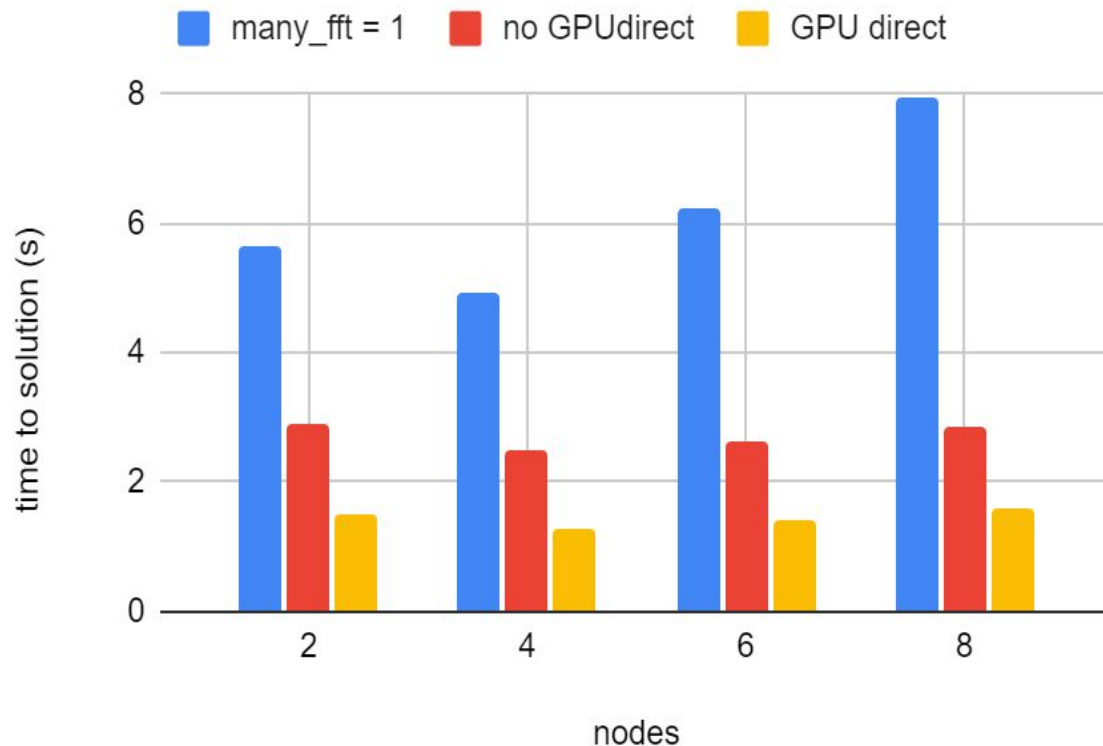
LUMI cpu, LUMI gpu-many_fft=1



- Gold surface;
- 112 atoms;
- ~1600 electrons.
- vloc_psi only.

Results (2)

cri3-small, vloc_psi / call



- CrI3;
- 480 atoms;
- ~2700 electrons;
- vloc_psi only.

What's next

- **Complete the porting** of PWscf minor routines;
 - complete the **openMP task** based version of the **batched FFTs** to enable them on **Intel®** architectures.
- **Medium-large size benchmarks** by the first half of the next year.
- Port QE codes other than PWscf (**PHonon**, **CP**, **EELS**, ...);
 - test **openMP** on **Nvidia®** architectures;
 - incorporate **DevXlib**.

Acknowledgments

QE dev group

- Pietro Delugas, SISSA
- Ivan Carnimeo, SISSA
- Fabrizio Ferrari Ruffino, CNR-IOM
- Oscar Baseggio, SISSA
- Riccardo Bertossa, SISSA
- Aurora Ponzi, CNR-IOM
- Stefano Baroni, SISSA, CNR-IOM
- Paolo Giannozzi, UniUD, CNR-IOM

CINECA

- Fabio Affinito
- Laura Bellentani
- Sergio Orlandini

QE Foundation

- Francesca Garofalo

AMD®

- Ossian O'Reilly
- Jakub Kurzak

ANL

- Ye Luo

Intel®

- Giacomo Rossi

Nvidia®

- Filippo Spiga
- Louis Stuber

THANK YOU