Webinar @ OpenACC.org - January, 25, 2024

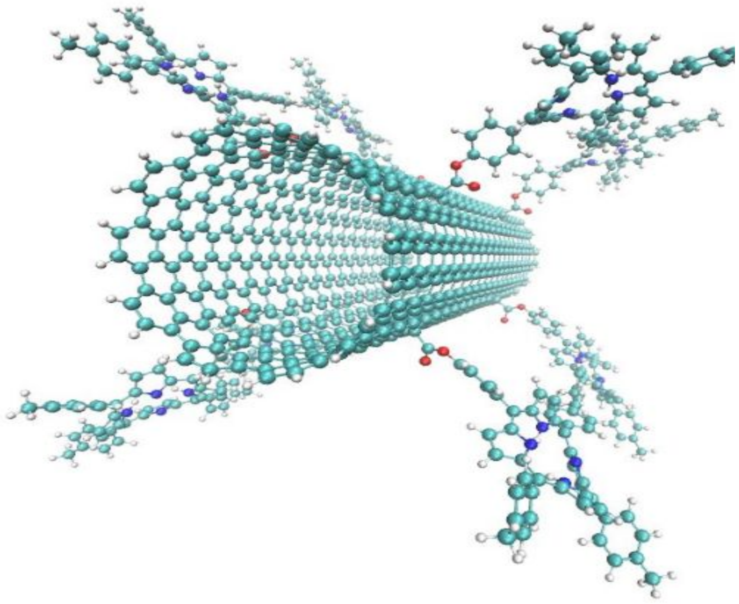
QUANTUM ESPRESSO on GPUs

Porting strategy and results

Fabrizio Ferrari Ruffino, CNR-IOM

QUANTUMESPRESSO MAX IOM ISTITUTO OFFICINA DEI MATERIALI CINECA SISSA

QUANTUM ESPRESSO



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.*

QUANTUM ESPRESSO is a community code.

Outline

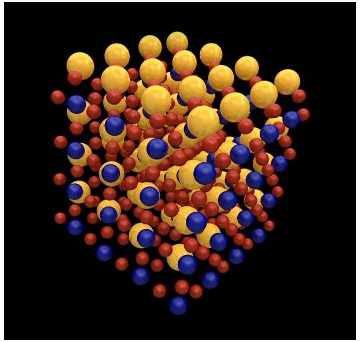
- **Quantum mechanics** for **materials**;
- the **QUANTUM ESPRESSO** (**QE**) suite;
- **porting strategy** and constraints;
- **multiple standards** in QE;
- toward a portable **FFTXlib**, library for 3D FFTs;
- results and state of the art of **QE performance**;
- summary and outlook

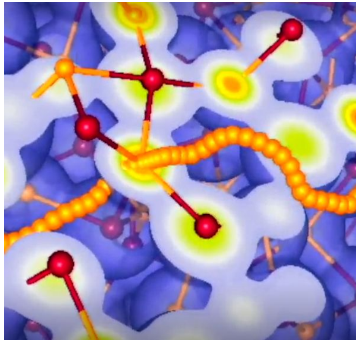
QUANTUM ESPRESSO

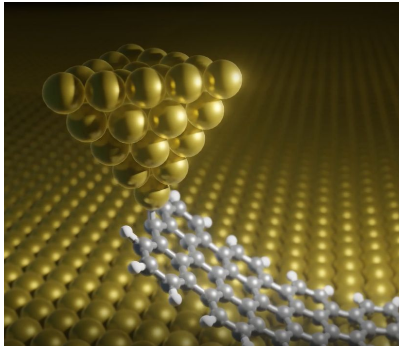
AB INITIO QUANTUM MECHANICS

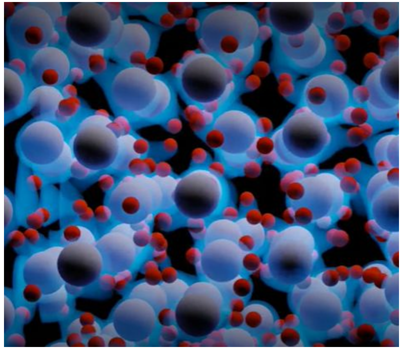
no input parameters for material modeling

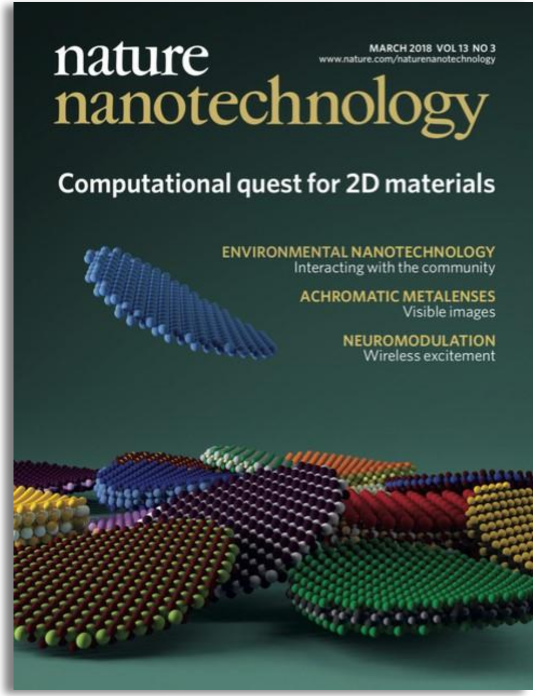
reduces costs, accelerates discoveries







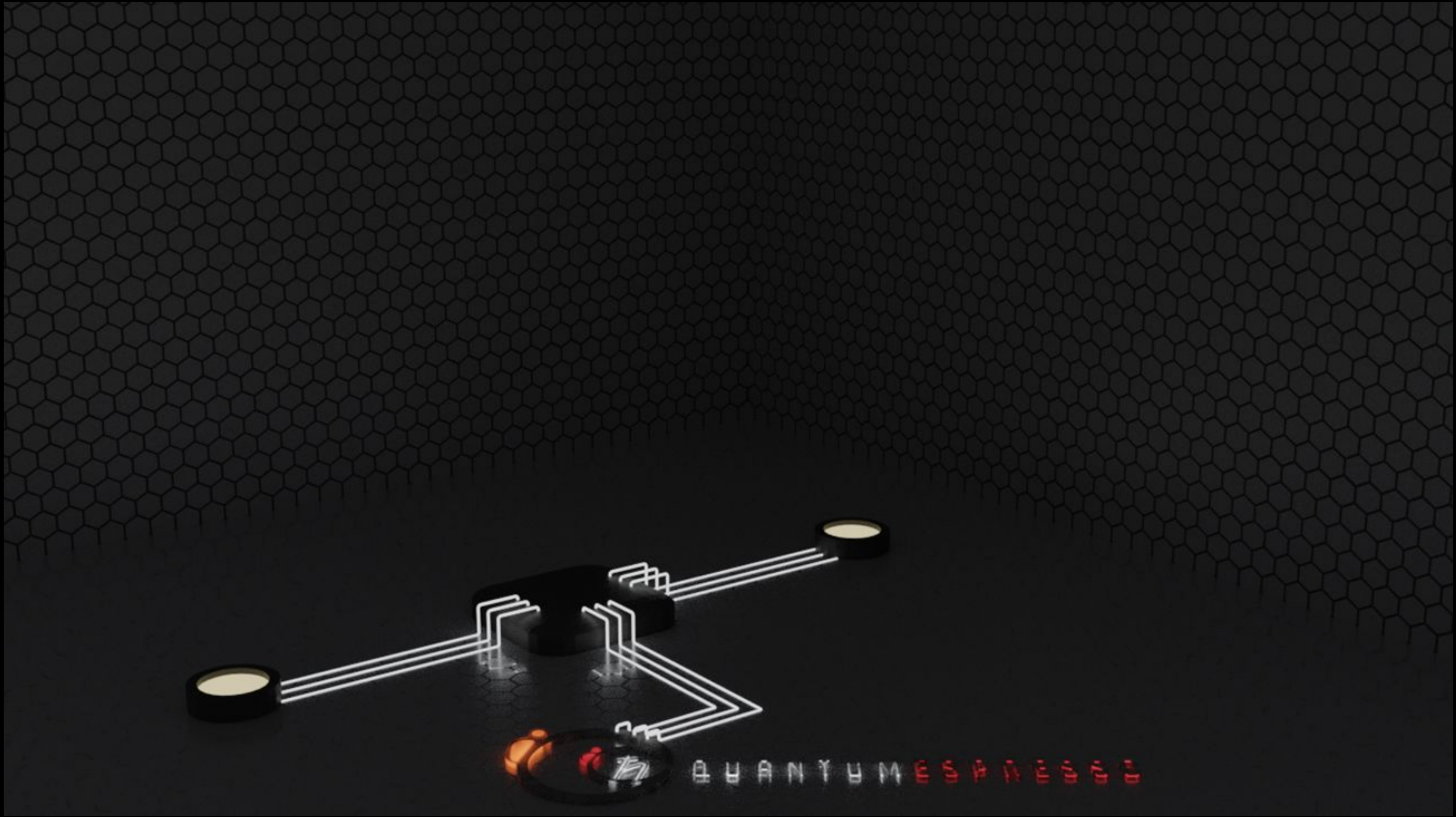




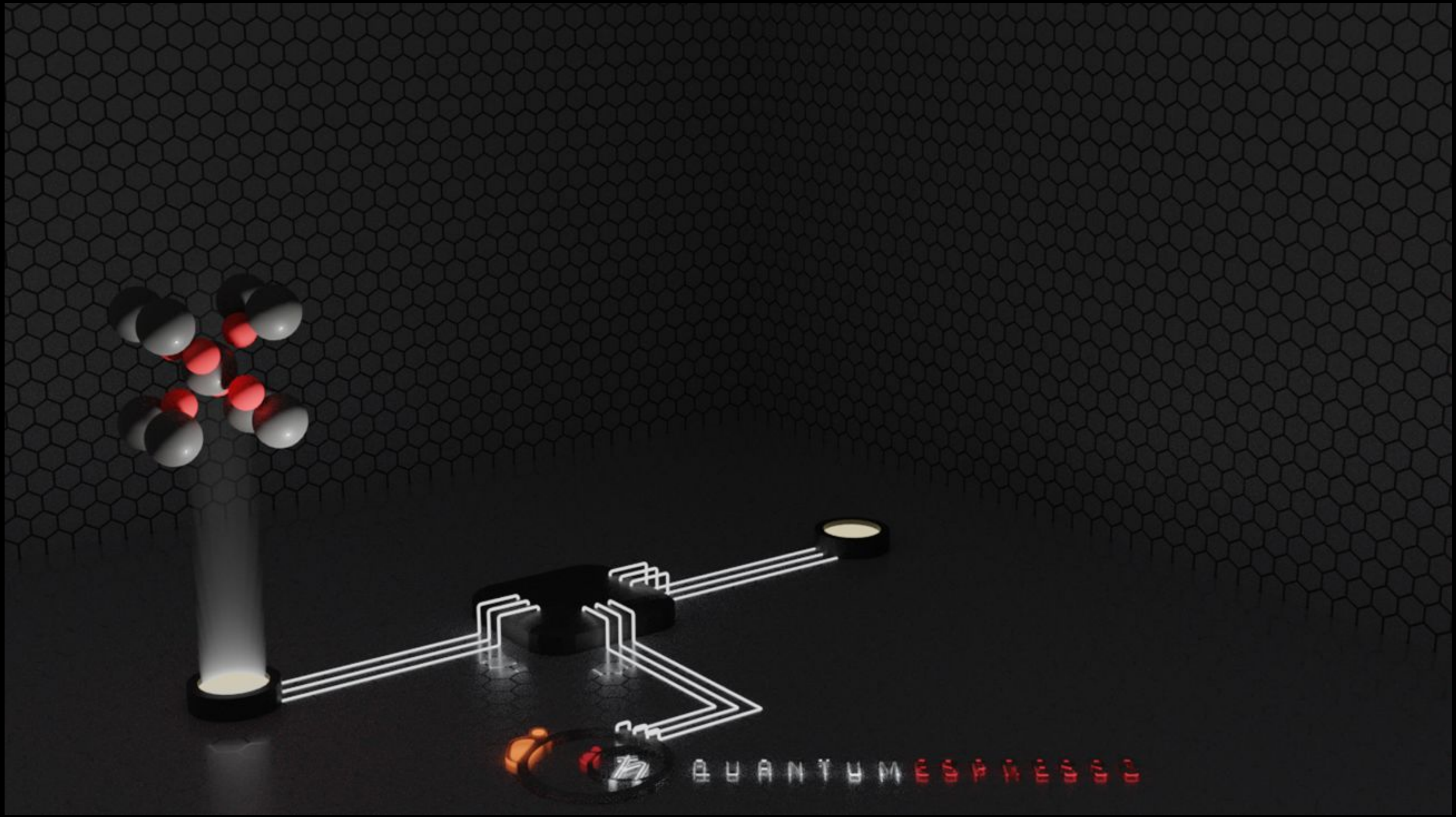
Two-dimensional materials from high-throughput computational exfoliation of experimentally known compounds, Nature Nanotechnology **13**, 246 (2018). doi:10.1038/s41565-017-0035-5

Quantum mechanics for materials

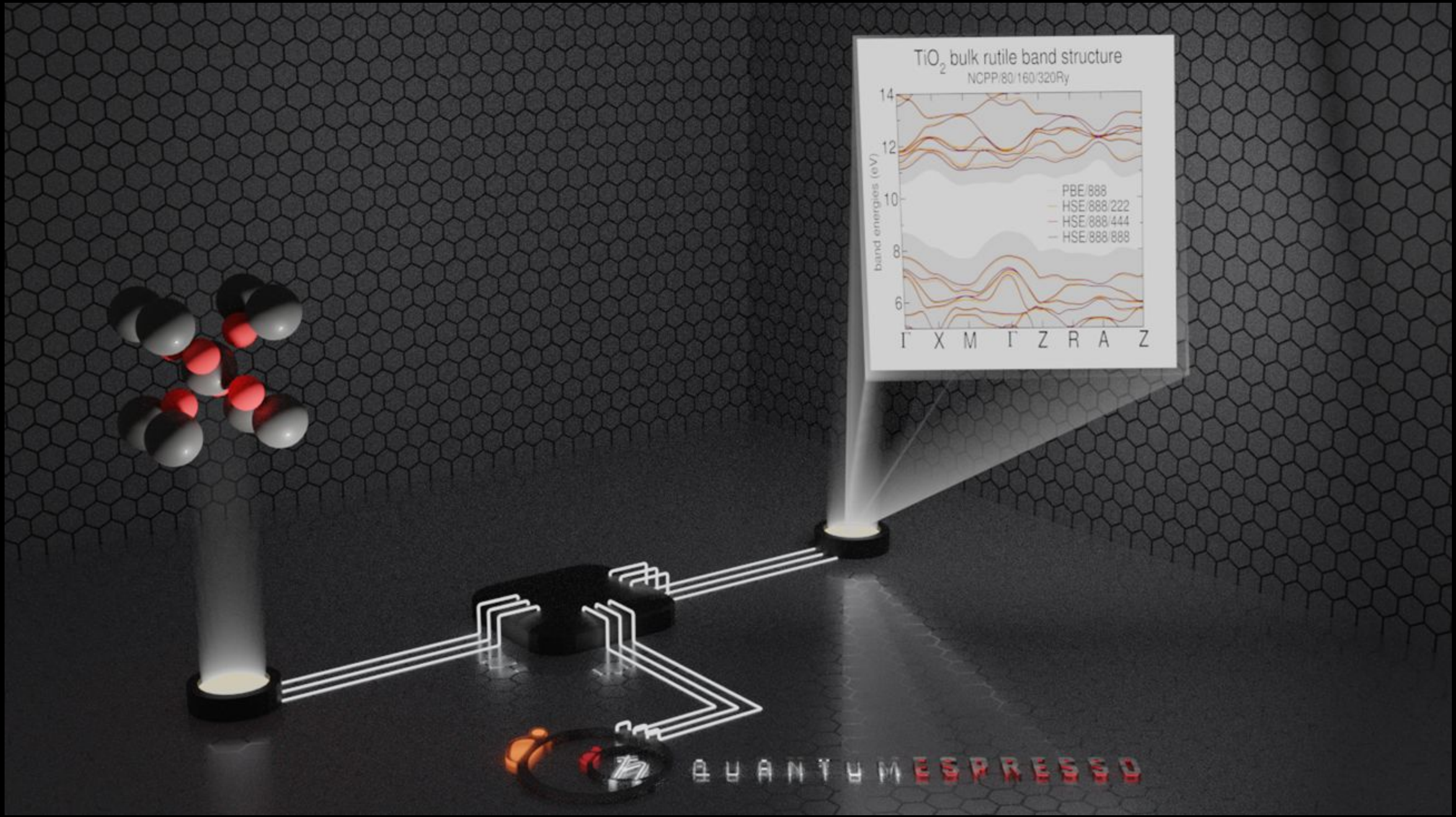
Quantum mechanics for materials



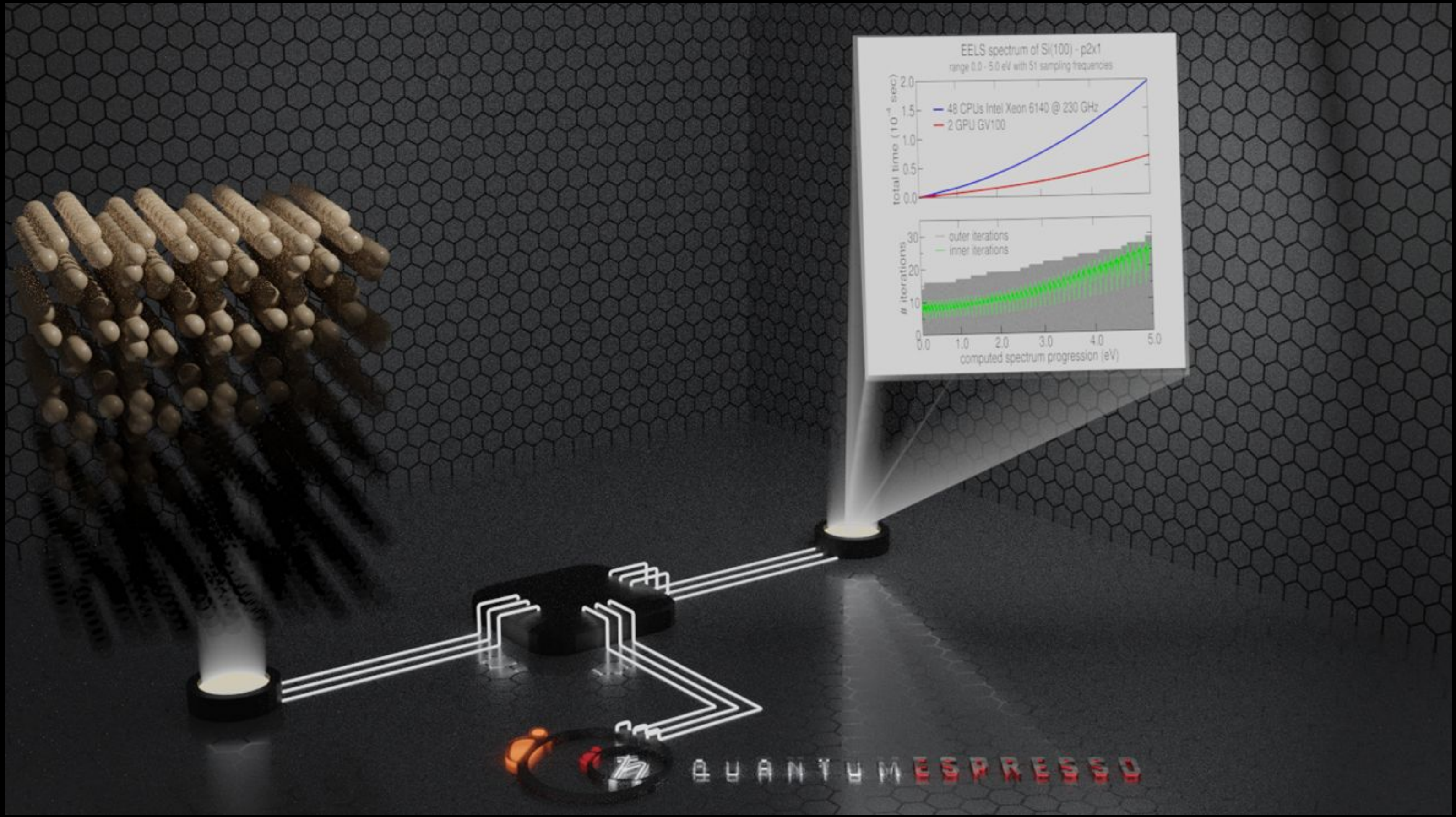
Quantum mechanics for materials



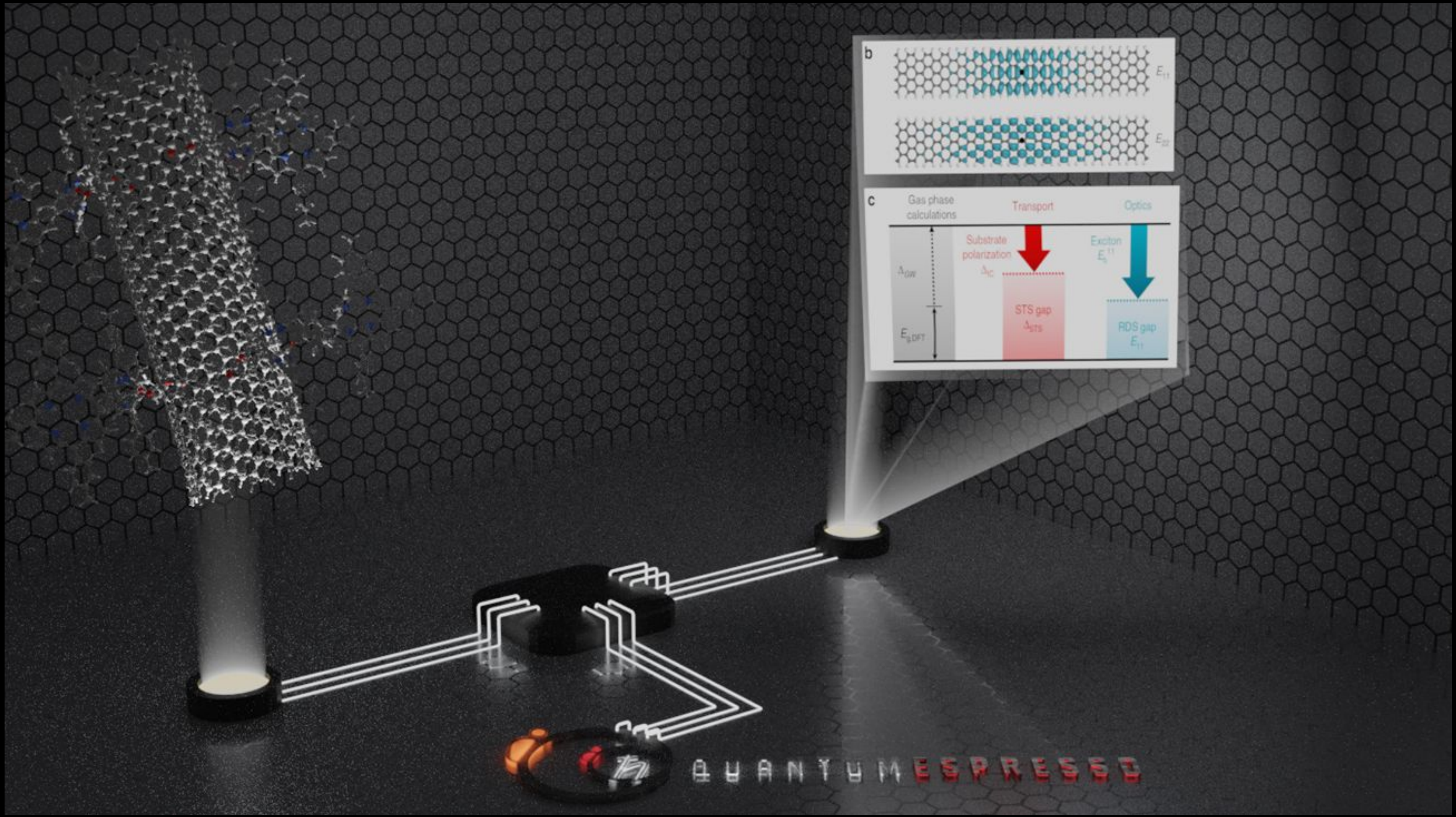
Quantum mechanics for materials



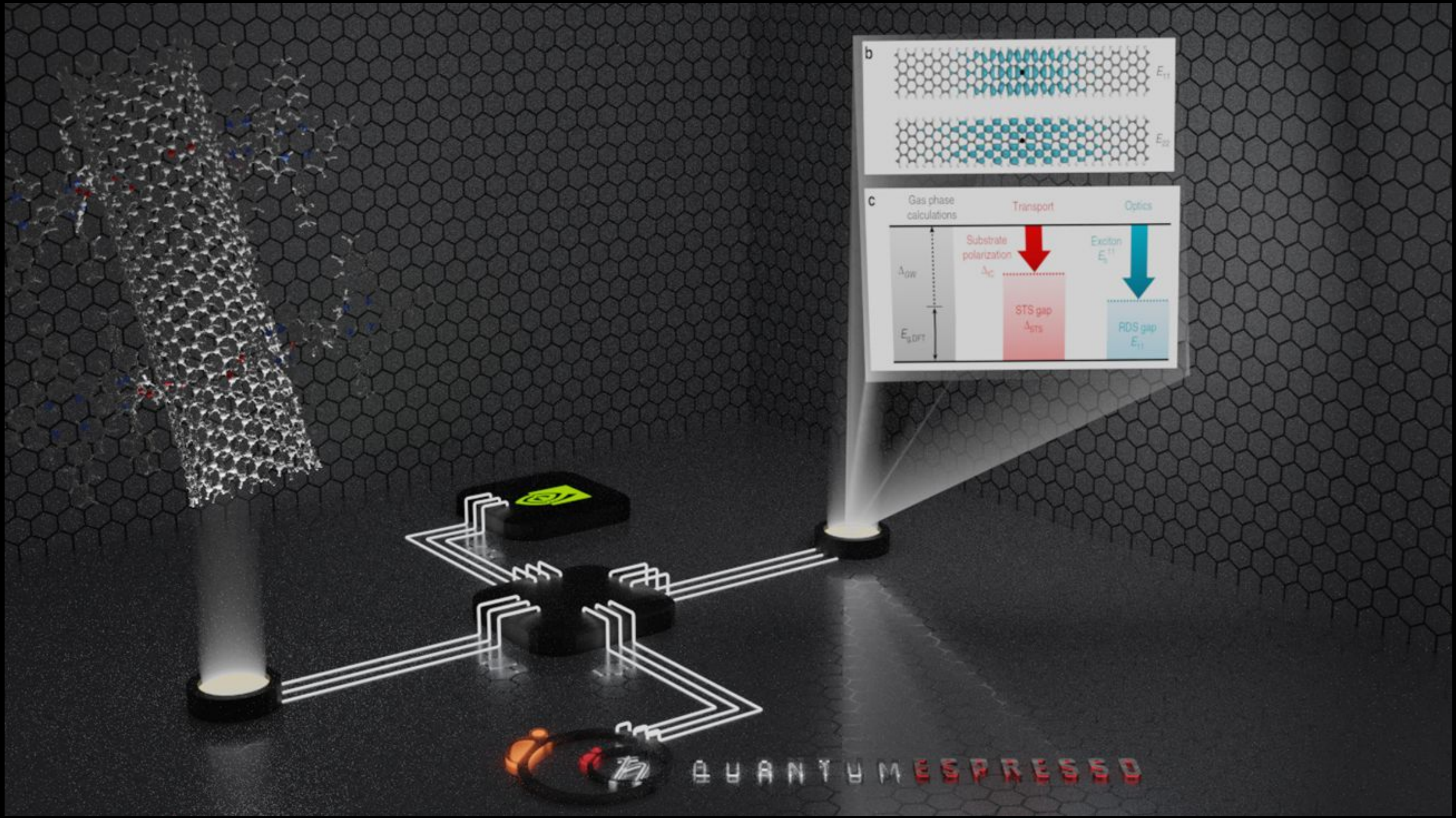
Quantum mechanics for materials



Quantum mechanics for materials



Quantum mechanics for materials



Materials design at the Exascale

MAX

LIGHTHOUSE CODES
- QUANTUM ESPRESSO
- Yambo
- siesta
- Fleur (www.flapw.de)
- BigDFT

DOMAIN EXPERTS & CODE DEVELOPERS
- Consiglio Nazionale delle Ricerche
- SISSA
- ICN2 Institut Català de Nanociència i Nanotecnologia
- JÜLICH Forschungszentrum
- cea
- Universität Bremen

HPC EXPERTS & DATA CENTRES
- CINECA
- JÜLICH Forschungszentrum
- cea
- BSC
- IT4I
- Jožef Stefan Institute

TECHNOLOGY & CO-DESIGN PARTNERS
- AtoS
- SIPEARL The Silicon Pearl
- E4 Computer Engineering
- LEONARDO

Coe for HPC applications in material science

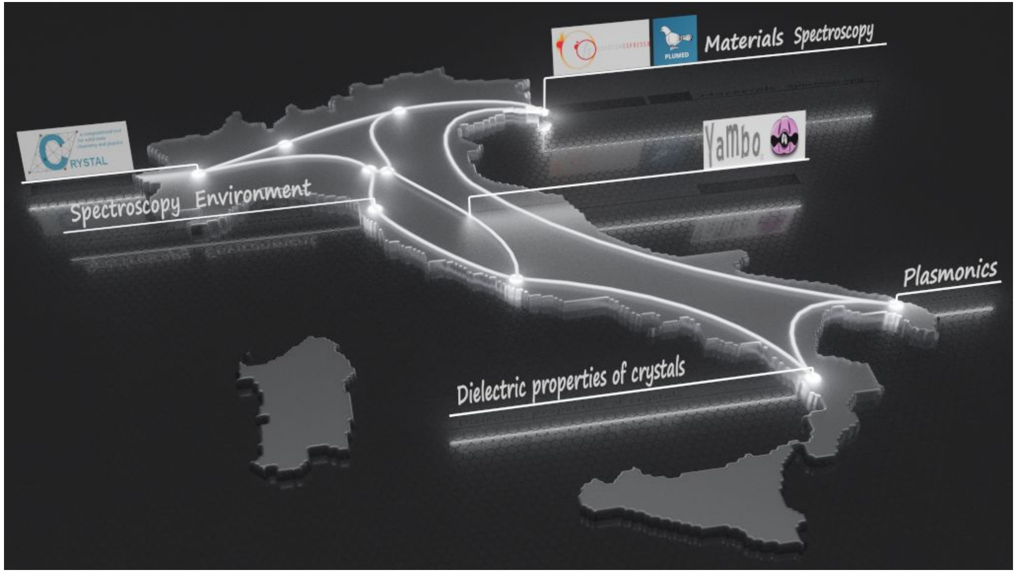
exploit **frontier HPC** for material science research in strong link with **scientific communities**

CODE PORTING → CO-DESIGN → HPC ECOSYSTEM

ICSC National Research Centre

for High Performance Computing, Big Data and Quantum Computing

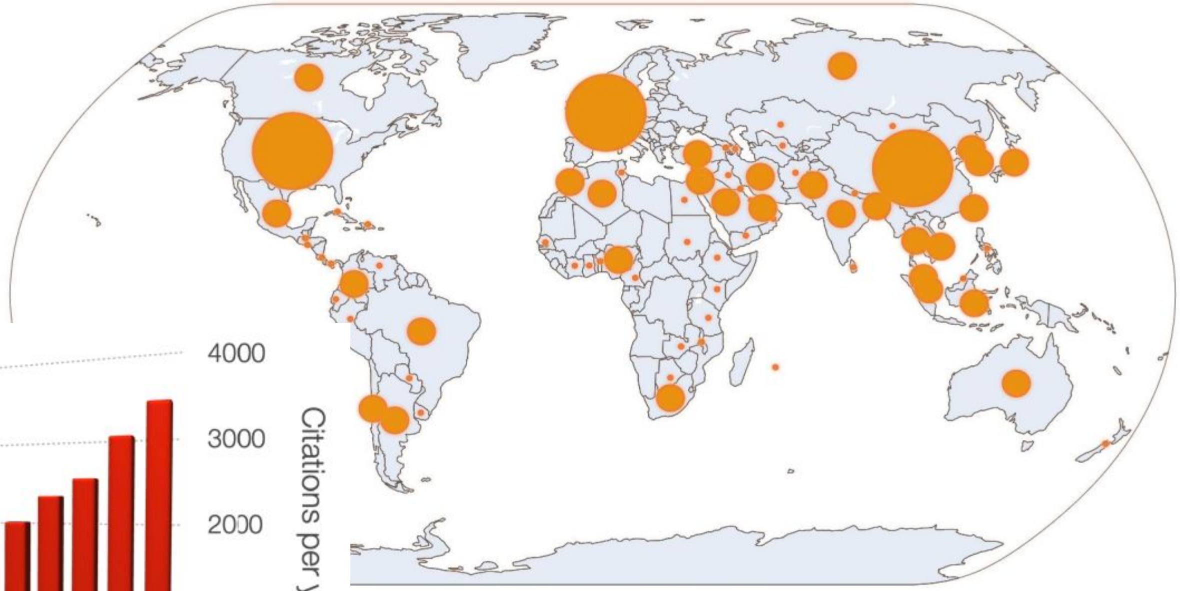
Flagship codes

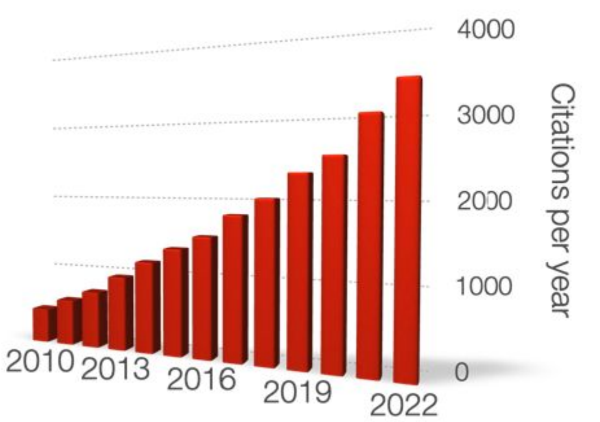


The Quantum ESPRESSO suite

Geographic distribution of the authors of the articles citing the main reference articles of Quantum ESPRESSO

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

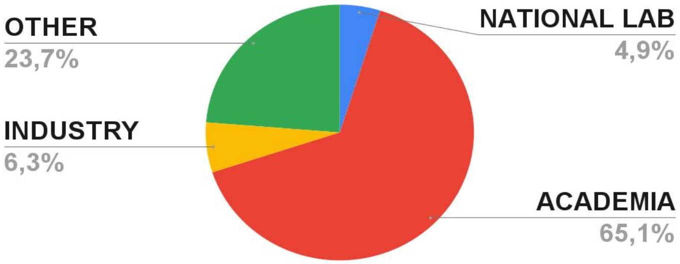


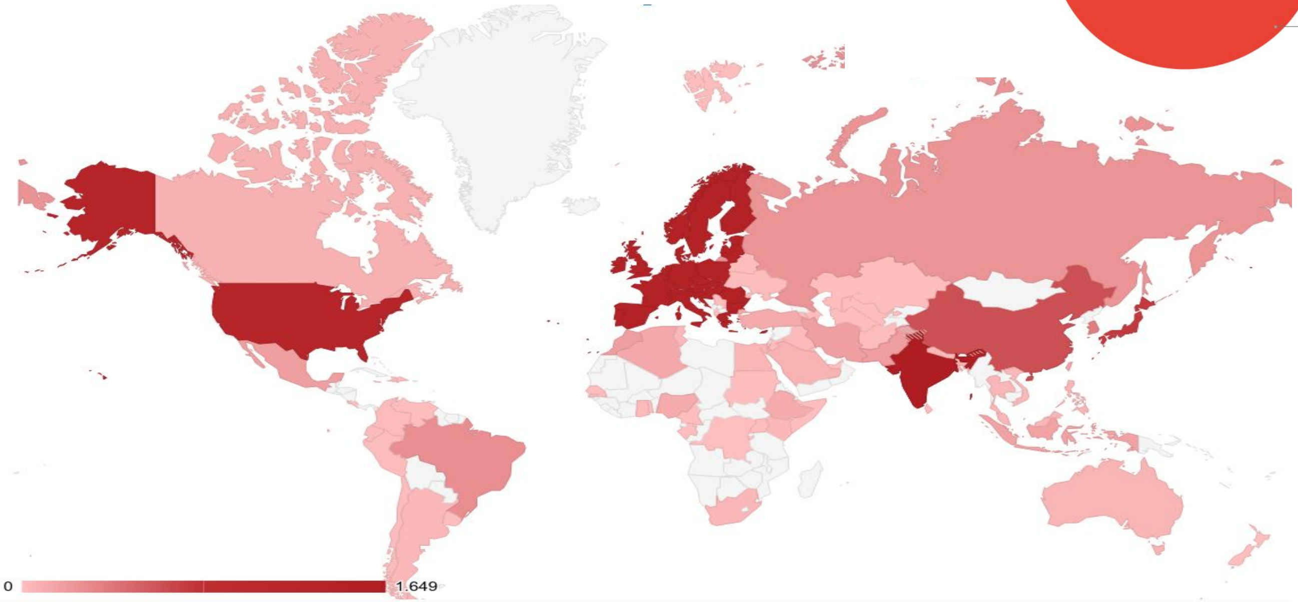


Data provided by the courtesy of the Quantum ESPRESSO foundation

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

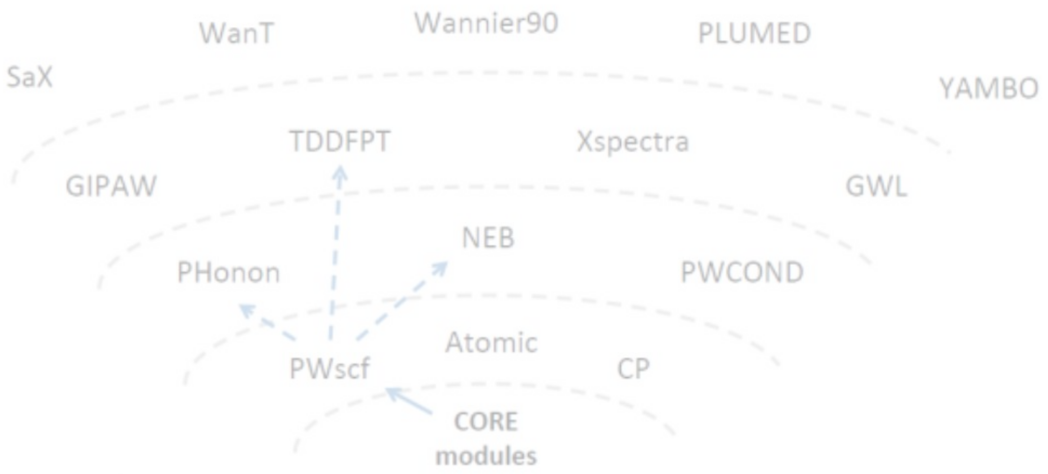
$$\left[-\frac{\hbar^2}{2m}\nabla^2 + V_s(\mathbf{r})\right]\varphi_i(\mathbf{r}) = \varepsilon_i\varphi_i(\mathbf{r}).$$

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

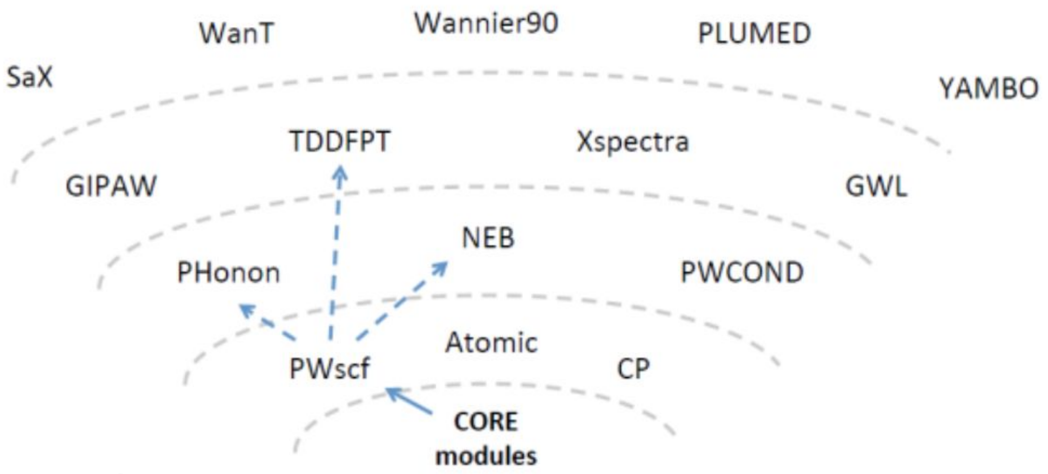
$$\left[-\frac{\hbar^2}{2m}\nabla^2 + V_s(\mathbf{r})\right]\varphi_i(\mathbf{r}) = \varepsilon_i\varphi_i(\mathbf{r}).$$

PLANE WAVES & PSEUDOPOTENTIAL

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

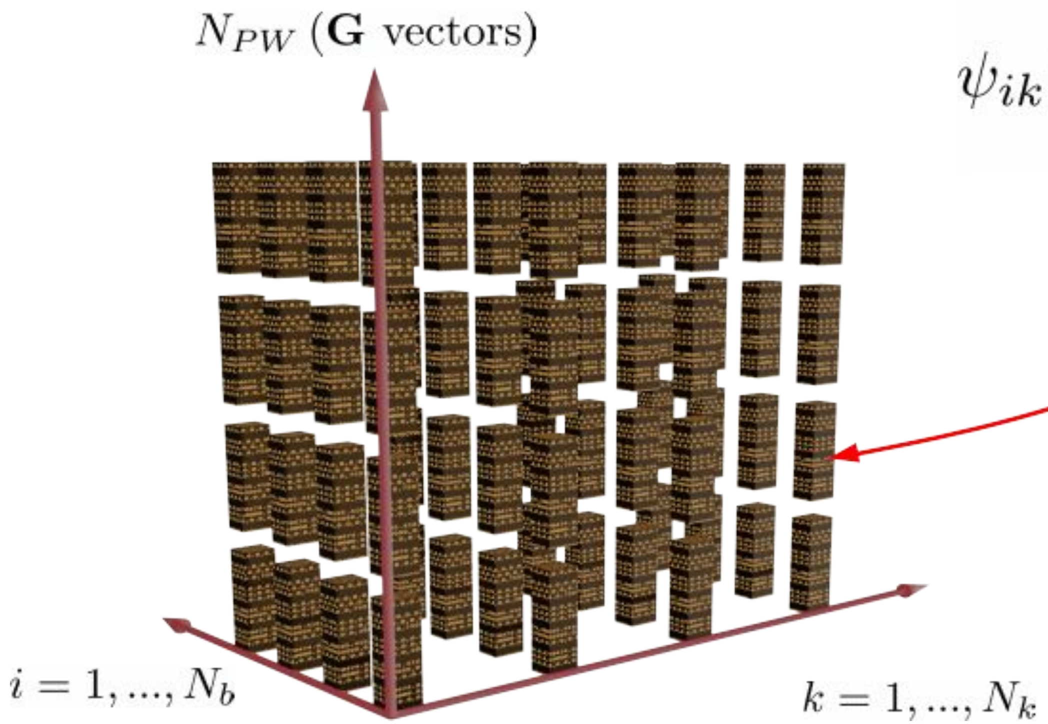
DUAL SPACE TECHNIQUE

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



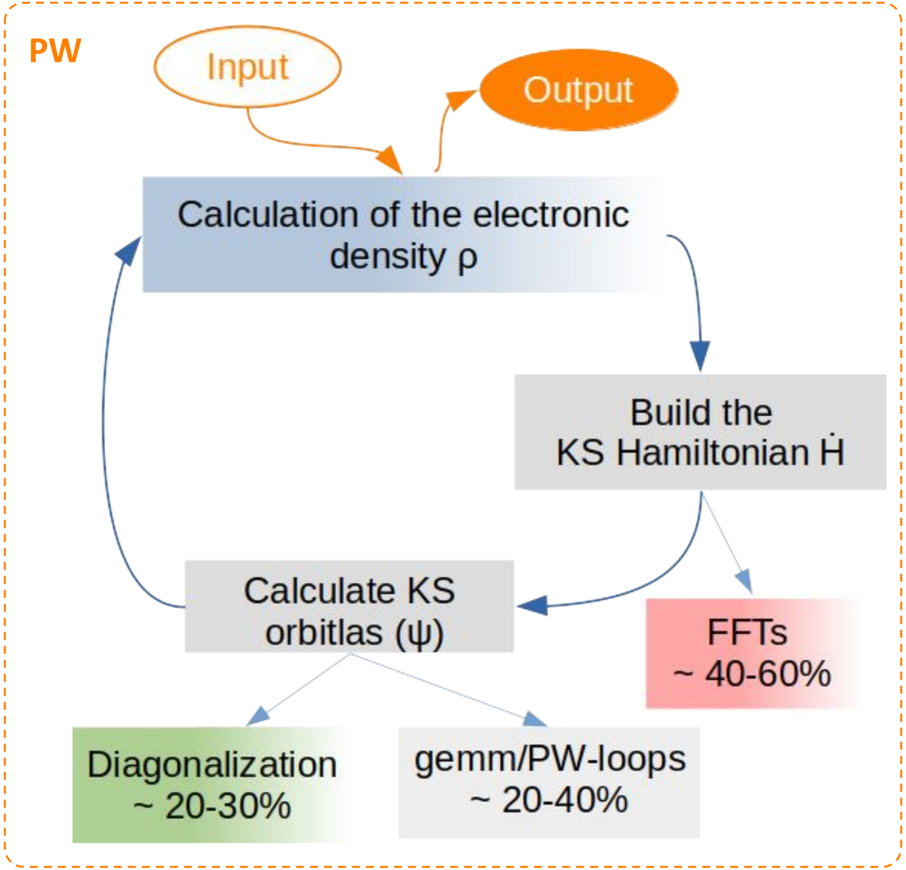
The parallel scheme of PWscf

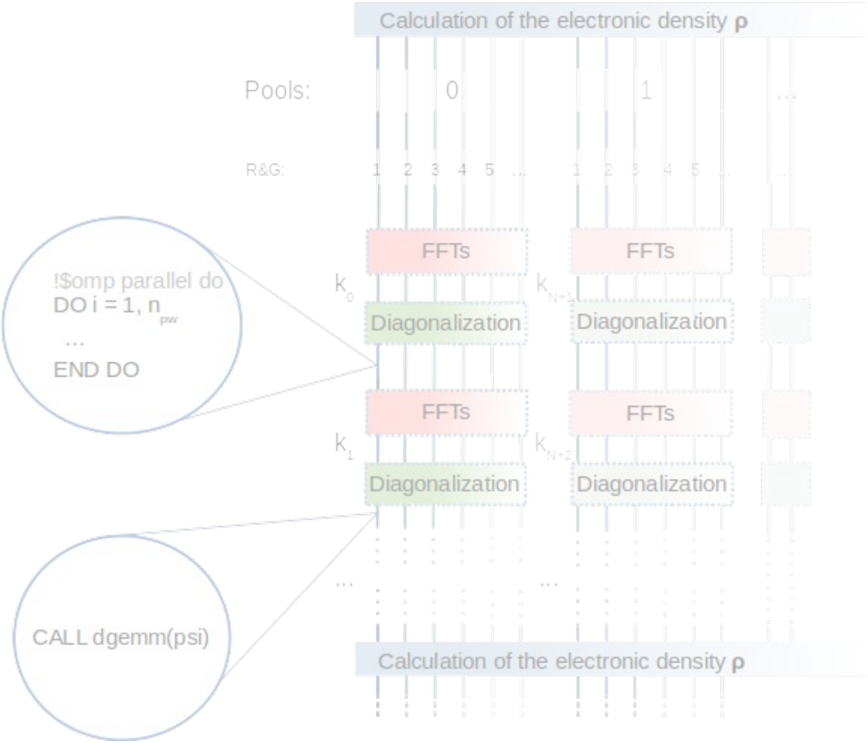
Data distribution



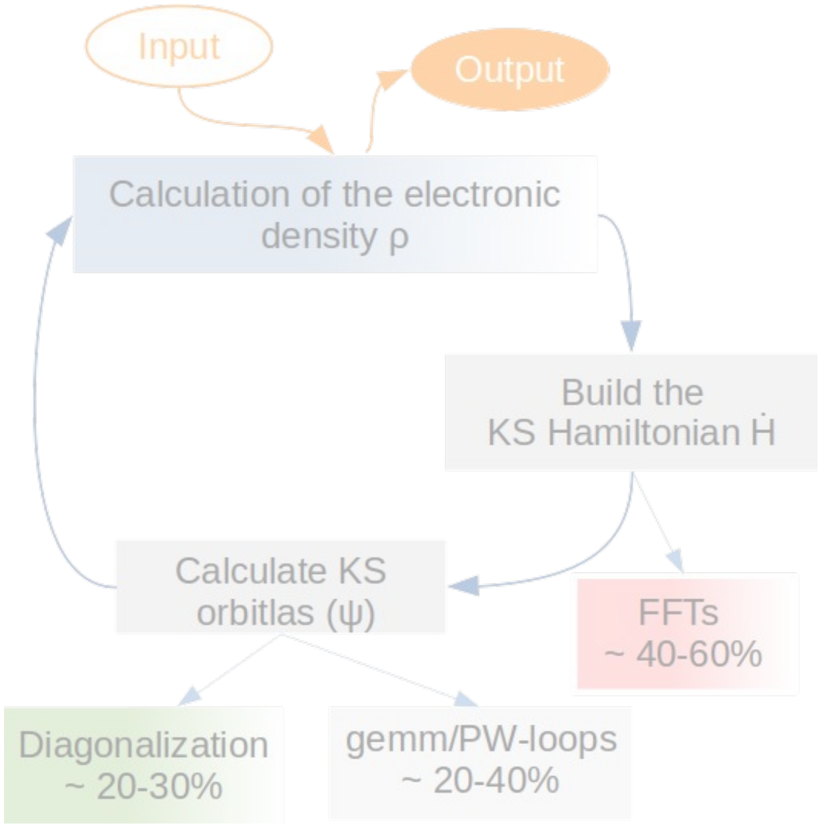
$$\psi_{ik}(\mathbf{r}) = \sum_{\mathbf{G}}^{N_{PW}} C_{\mathbf{G},ik} \frac{e^{i((\mathbf{G}+\mathbf{k})\cdot\mathbf{r})}}{\sqrt{\Omega}}$$

The parallel scheme of PWscf

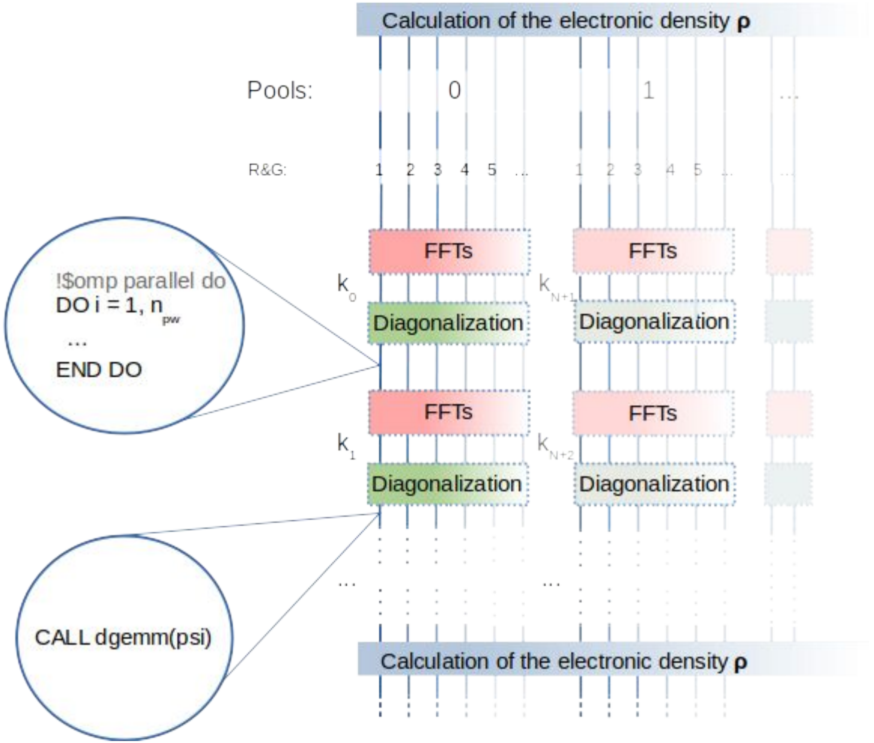




The parallel scheme of PWscf



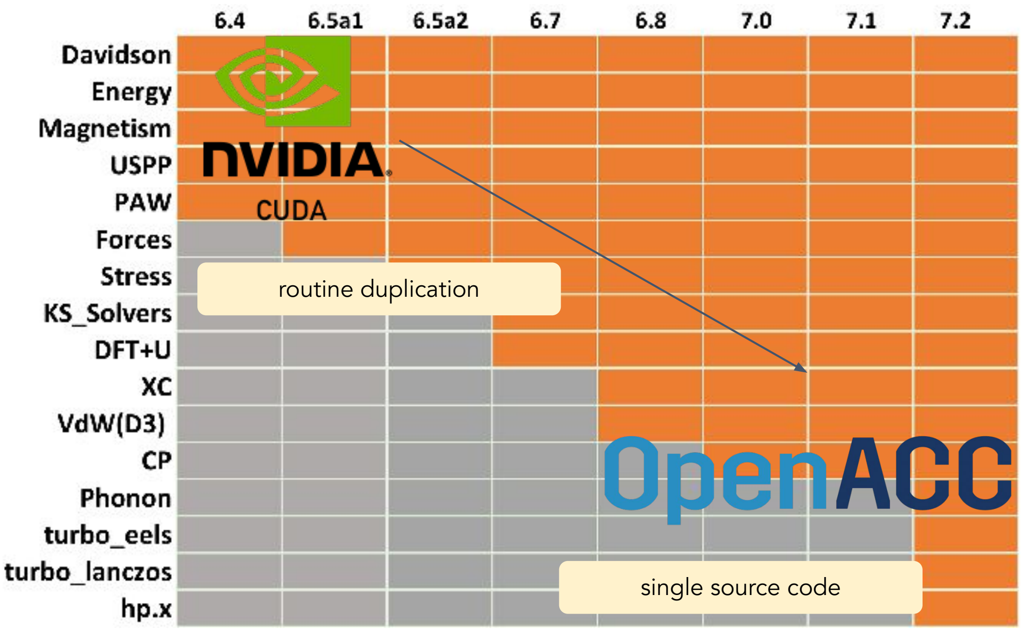
MPI + OpenMP



Porting strategy

Towards a portable GPU version

The transition from CUDA to Openacc



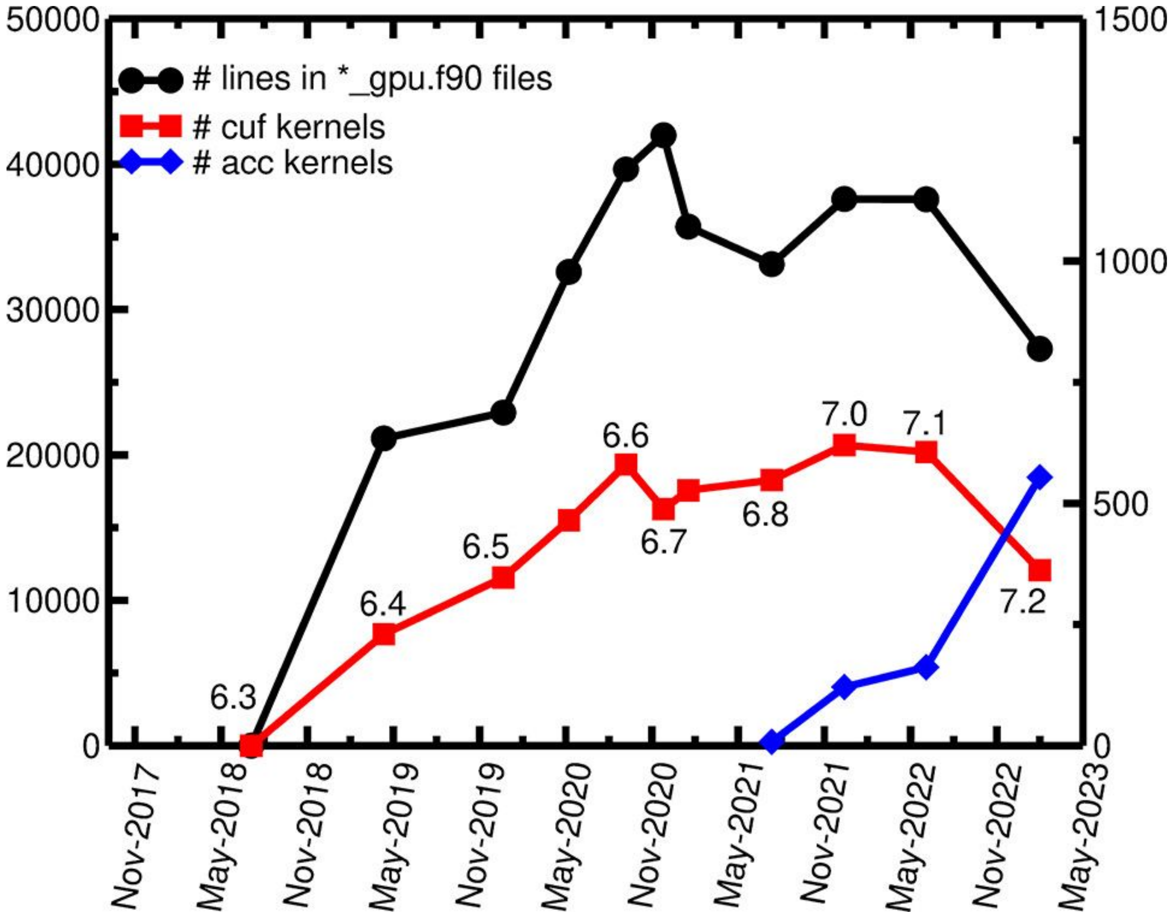
DIRECTIVE-BASED PROGRAMMING MODELS

MAINTAINABLE

PORTABLE

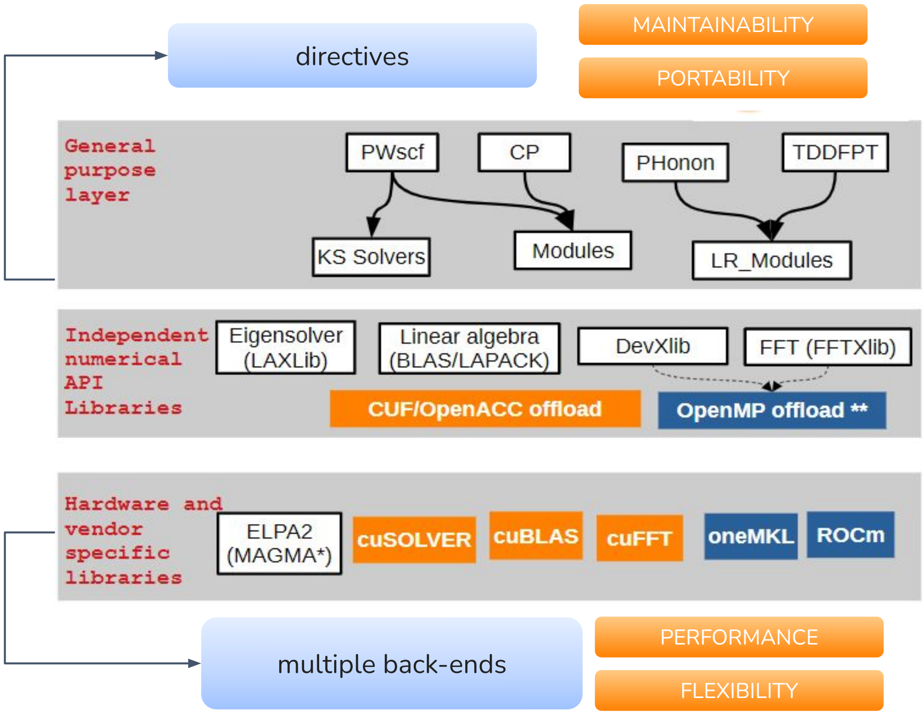
SINGLE SOURCE CODE

Towards a portable GPU version



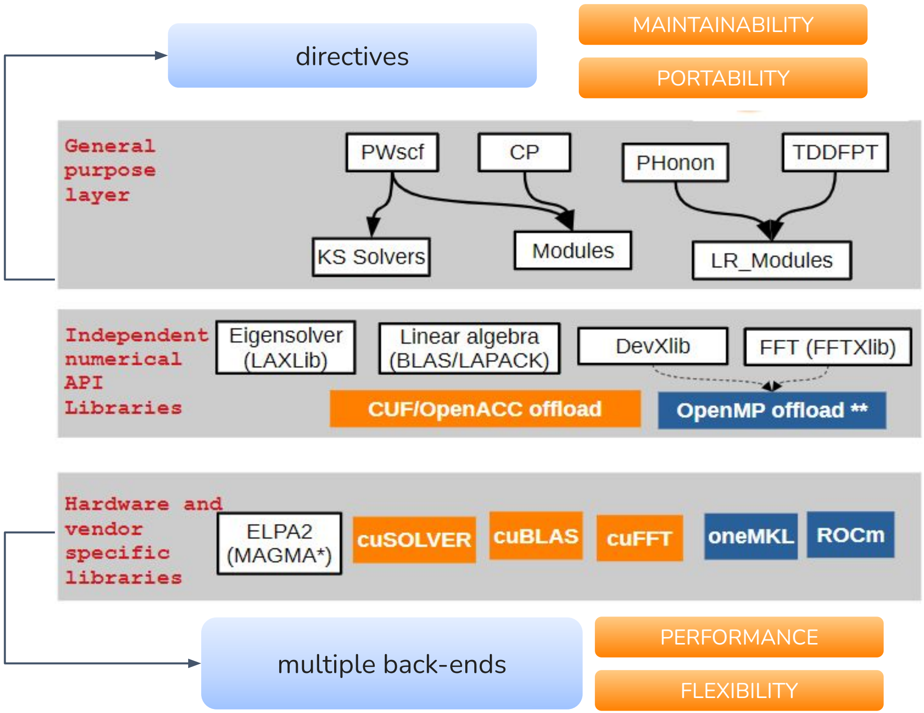
Towards a portable GPU version

Modularity supports interoperability and new programming models



Towards a portable GPU version

Modularity supports interoperability and new programming models

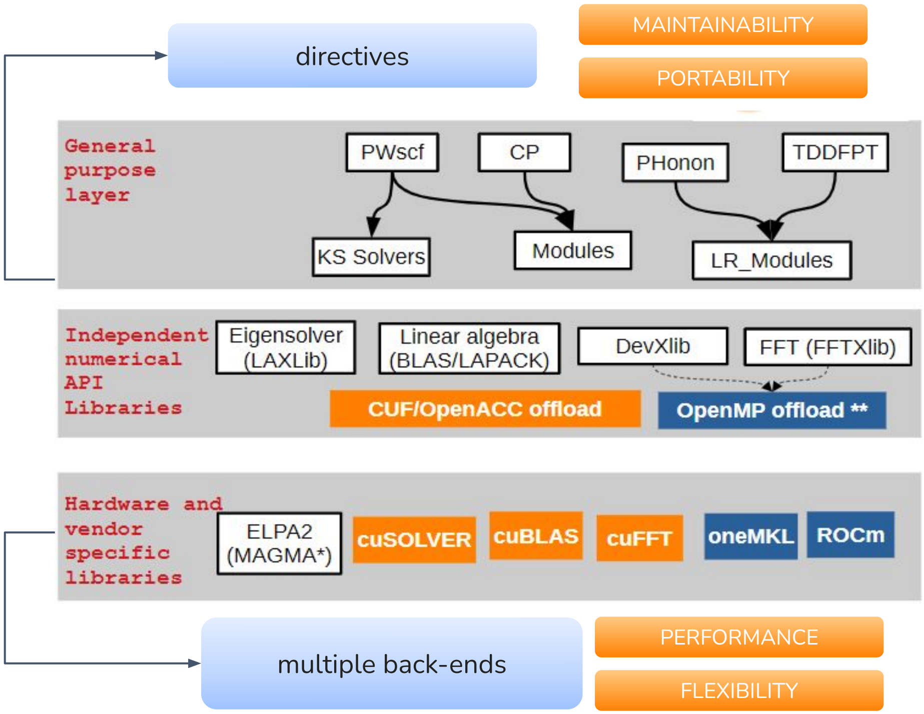


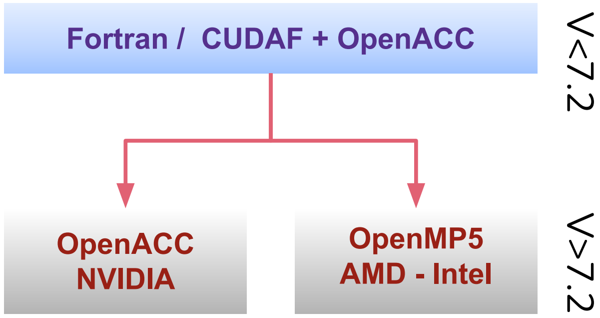
Fortran / CUDAF + OpenACC

V<7.2

Towards a portable GPU version

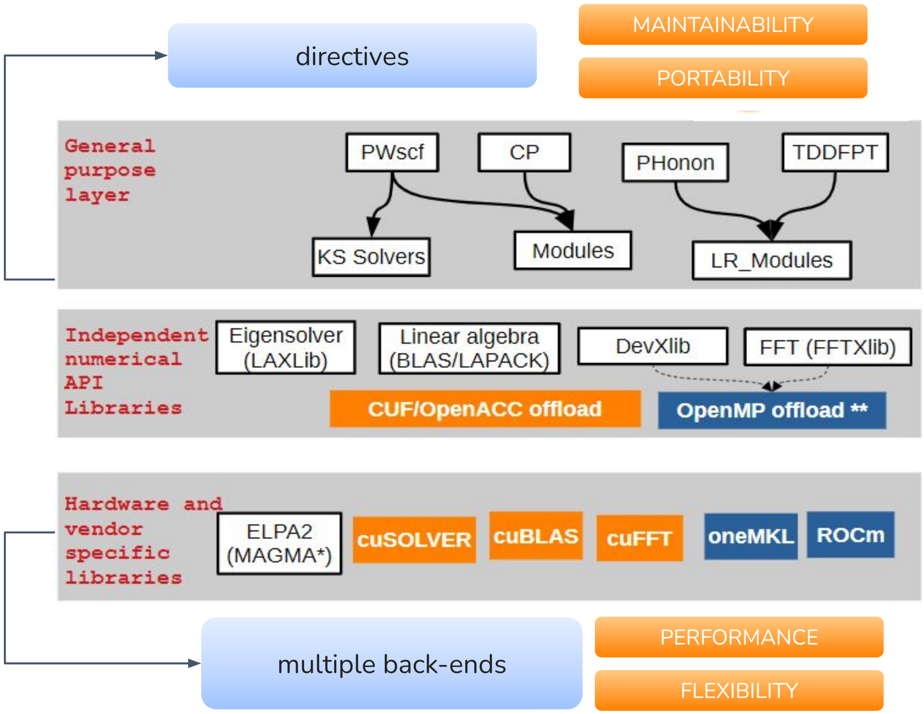
Modularity supports interoperability and new programming models

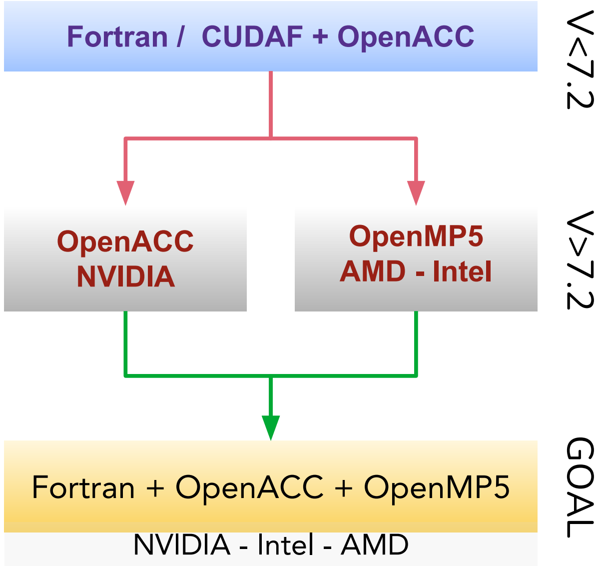




Towards a portable GPU version

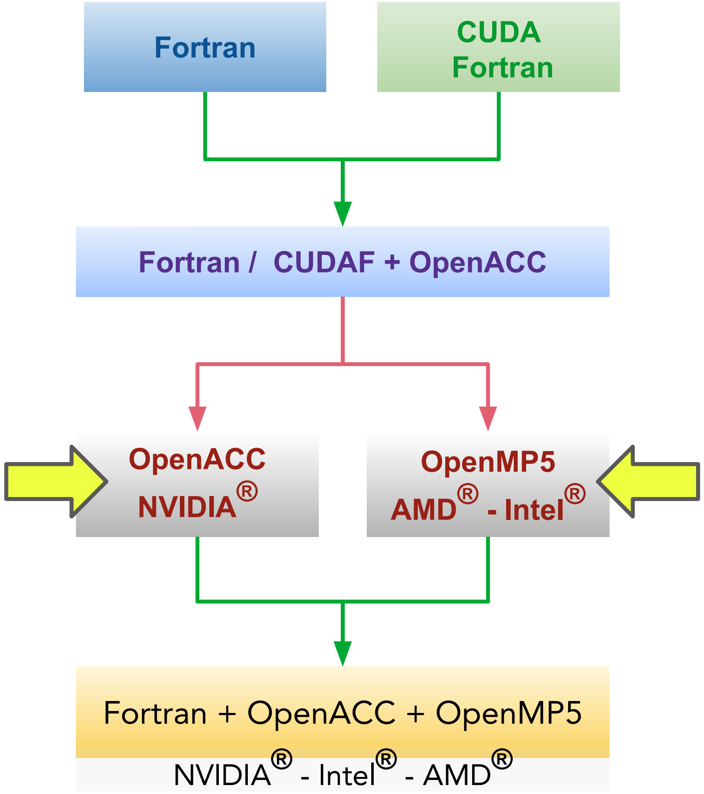
Modularity supports interoperability and new programming models





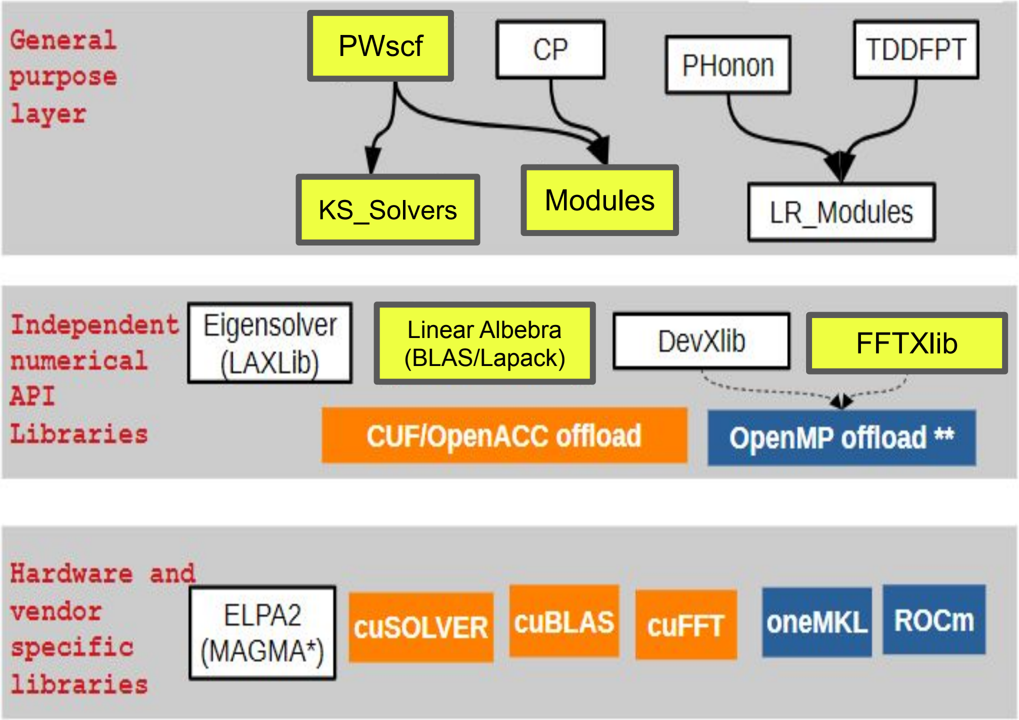
On the porting roadmap

- 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.

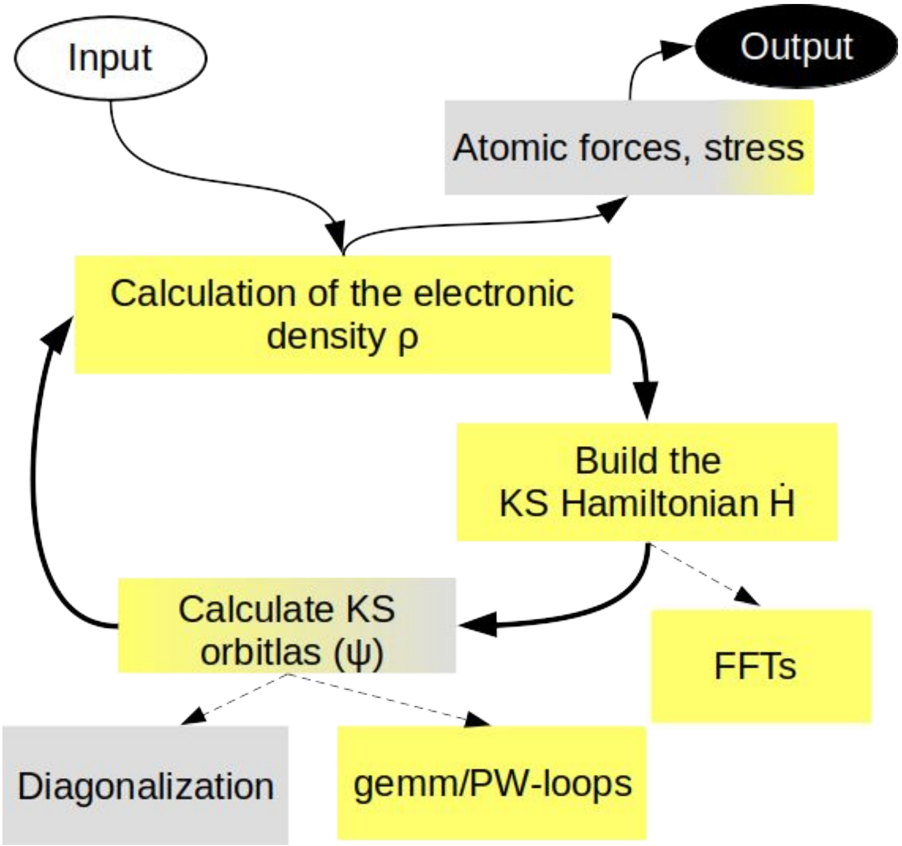
OMP porting of QE



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 of QE



Ported:

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

To be ported:

- diagonalization (zhegv);
- **batched FFTs**;
- **Hubbard**, forces, stress;
- codes other than PW.

Multiple standards in QE

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
```

Offload

<table>
<tr><th></th><th>CUF only</th><th>CUF interfaces
OpenACC parent code</th></tr>
<tr><td>Host to Device</td><td>if (use_gpu) then
  arg_d = arg
endif</td><td>!$acc update device(arg)</td></tr>
<tr><td>Routine calls</td><td>if (use_gpu) then
  call abc(arg_d)
else
  call abc(arg)
endif</td><td>!$acc host_data use_device(arg)
call abc(arg)
!$acc end host_data</td></tr>
<tr><td>Interfaces</td><td colspan="2">interface abc
  subroutine abc_cpu(v)
  subroutine abc_gpu(v_d)
end interface</td></tr>
</table>

Offload

<table>
<tr><th></th><th>CUF only</th><th>CUF interfaces
OpenACC parent code</th><th>OpenACC only</th></tr>
<tr><td>Host to Device</td><td>if (use_gpu) then
  arg_d = arg
endif</td><td colspan="2">!$acc update device(arg)</td></tr>
<tr><td>Routine calls</td><td>if (use_gpu) then
  call abc(arg_d)
else
  call abc(arg)
endif</td><td>!$acc host_data use_device(arg)
call abc(arg)
!$acc end host_data</td><td>call abc_acc(arg)</td></tr>
<tr><td>Interfaces</td><td colspan="2">interface abc
  subroutine abc_cpu(v)
  subroutine abc_gpu(v_d)
end interface</td><td>subroutine abc_acc(v)</td></tr>
</table>

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**			

Offload

<table>
<tr><th></th><th>CUF only</th><th>CUF interfaces
OpenACC parent code</th><th>OpenACC only</th><th>OpenACC +
OpenMP5</th></tr>
<tr><td>Host to Device</td><td>if (use_gpu) then
arg_d = arg
endif</td><td colspan="2">!$acc update device(arg)</td><td>!$acc update device(arg)
!$omp target update to(arg)</td></tr>
<tr><td>Routine calls</td><td>if (use_gpu) then
call abc(arg_d)
else
call abc(arg)
endif</td><td>!$acc host_data use_device(arg)
call abc(arg)
!$acc end host_data</td><td>call abc_acc(arg)</td><td>#if def __OPENACC
call abc_acc(arg)
#elif def __OPENMP
call abc_omp(arg)
#endif</td></tr>
<tr><td>Interfaces</td><td colspan="2">interface abc
subroutine abc_cpu(v)
subroutine abc_gpu(v_d)
end interface</td><td>subroutine abc_acc(v)</td><td>subroutine abc_acc(v)
subroutine abc_omp(v)</td></tr>
</table>

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	

Courtesy of Ivan Carnimeo (SISSA)

Offload

	OpenACC + OpenMP5		devXlib + OpenACC +OpenMP5
Host to Device	!$acc update device(arg) !$omp target update to(arg)	⇒	**call devXlib**(arg, **offload**)
Routine calls [...]	**call** abc(arg, **offload**)		**call** abc(arg, **offload**)

Interfaces

```
interface abc
  subroutine abc_cpu( v, offload )
  subroutine abc_acc( v, offload )
  subroutine abc_omp( v, offload )
end interface
```

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)

Courtesy of Ivan Carnimeo (SISSA)

Wrappers instead of interfaces

```fortran
  !
  !-----------------------------------------------------------------------
  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
    !
```

```fortran
    !
    !$acc host_data use_device(f_in)
    IF (PRESENT(howmany_set)) THEN
      IF(omp_offload) THEN
#if defined (__OPENMP_GPU)
        IF(omp_map) THEN
          !$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

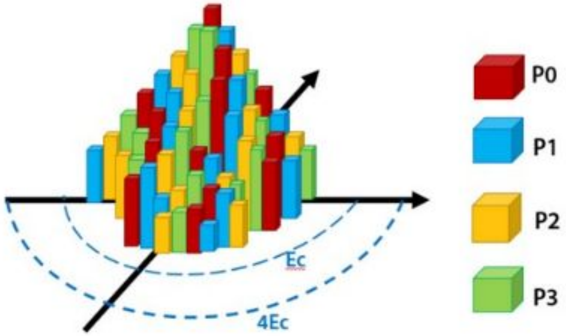
- We need **both offloaded and non-offloaded** low level routines (e.g. FFTXlib, LAXlib) at the same time;
- we use **wrappers with offloading switch** to sort CPU and GPU low level library calls;
- **duplication** of **low level routines** still necessary (avoidable? In the future?);
- ***omp target*** for GPU **protected** from openACC and from CPU *omp.*

Backends

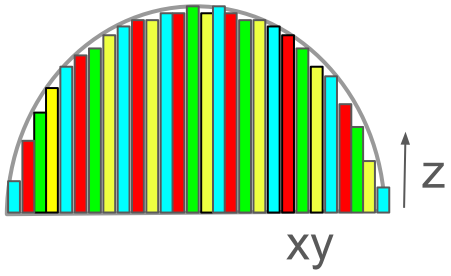
	Explicit streams	gpu/cpu interface	No need c_bind
Linear Algebra			
cuBlas	✓	✓	✓
rocBlas	✓	✗	✓
oneMKL	✗	✗	✓
Fourier transforms			
cuFFT	✓	✓	✓
hipFFT	✓	✗	✓
oneMKL	✗	✗	✓

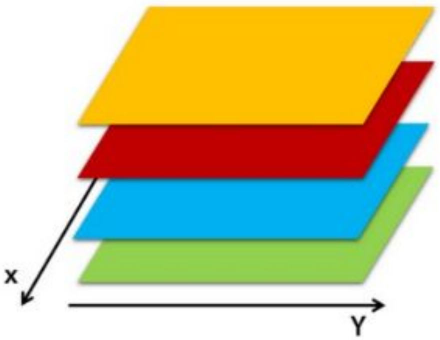
```
SUBROUTINE MYDGEMM2( TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, &
                     BETA, C, LDC, OMP_OFFLOAD )
#if defined(__CUDA)
    use cudafor
    use cublas
#elif defined(__OPENMP_GPU)
#if defined(__ONEMKL)
    use onemkl_blas_gpu
#endif
#if defined(__ROCBLAS)
    use rocblas_utils
#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(__ROCBLAS)
    IF (OMP_OFFLOAD) CALL rocblas_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
```

FFTXlib: slab decomposition

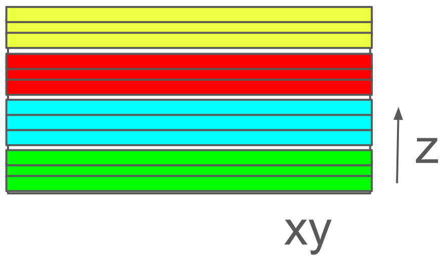


G-space (sticks)

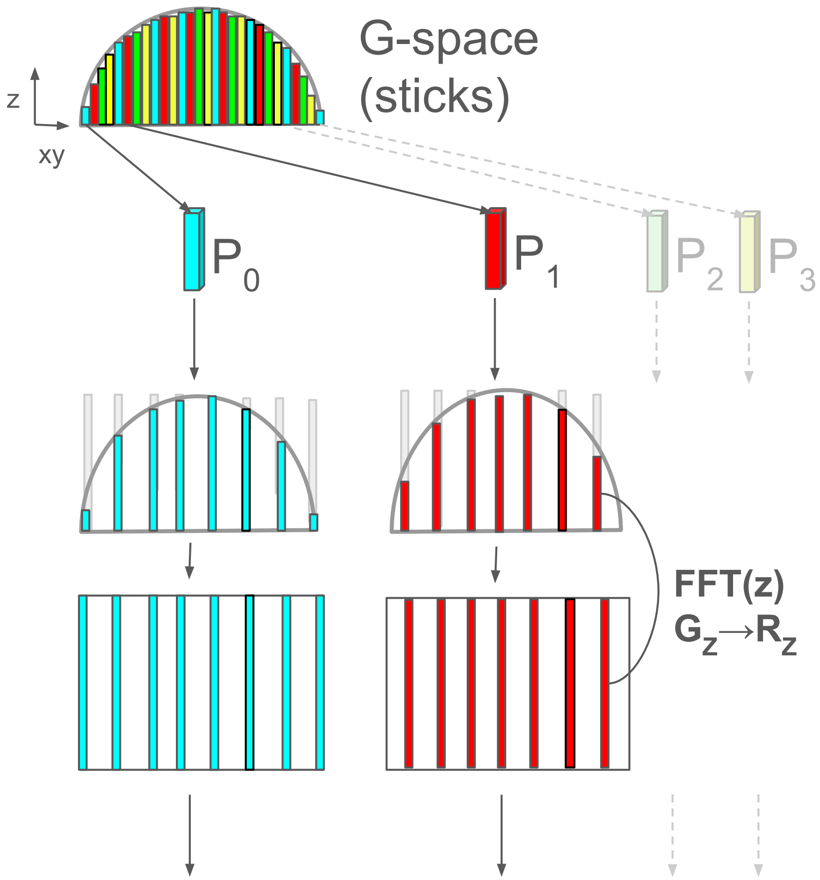


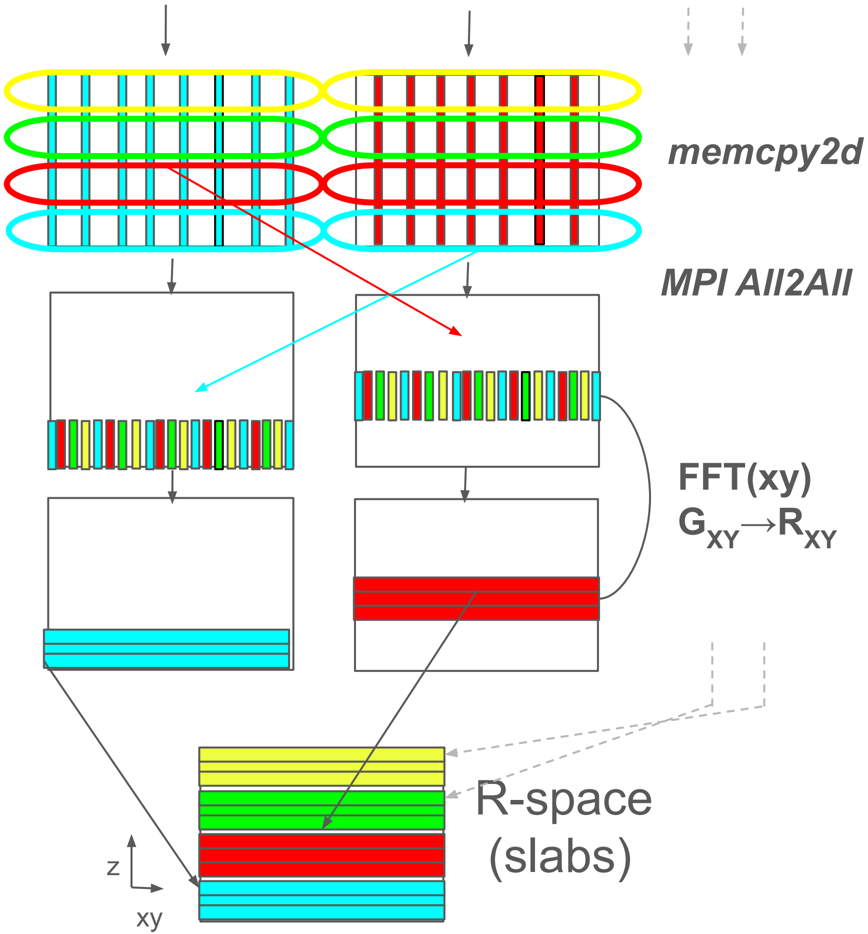


R-space (slabs)

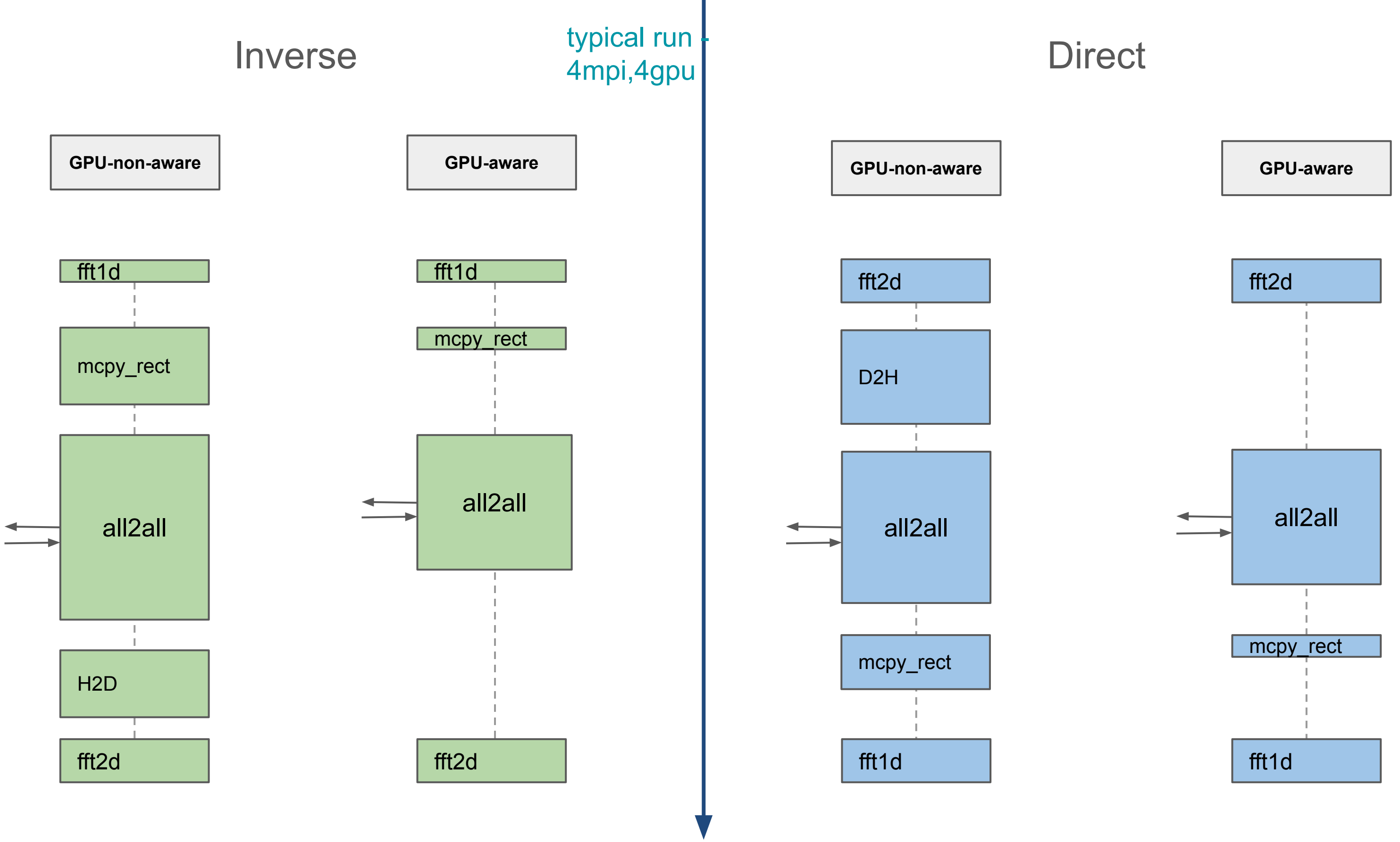


FFTXlib: slab decomposition



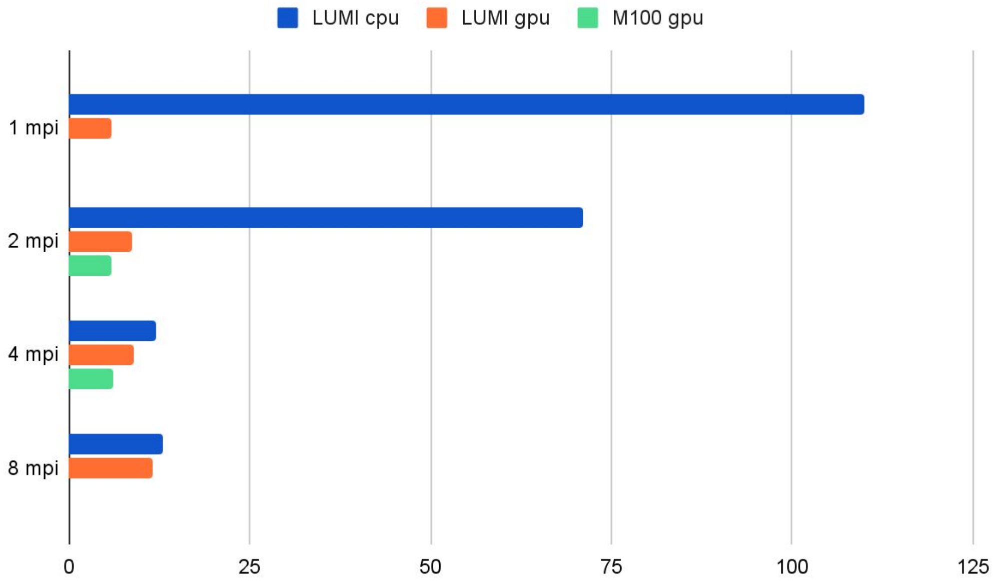


FFTXlib: standard flow



FFTXlib: basic porting

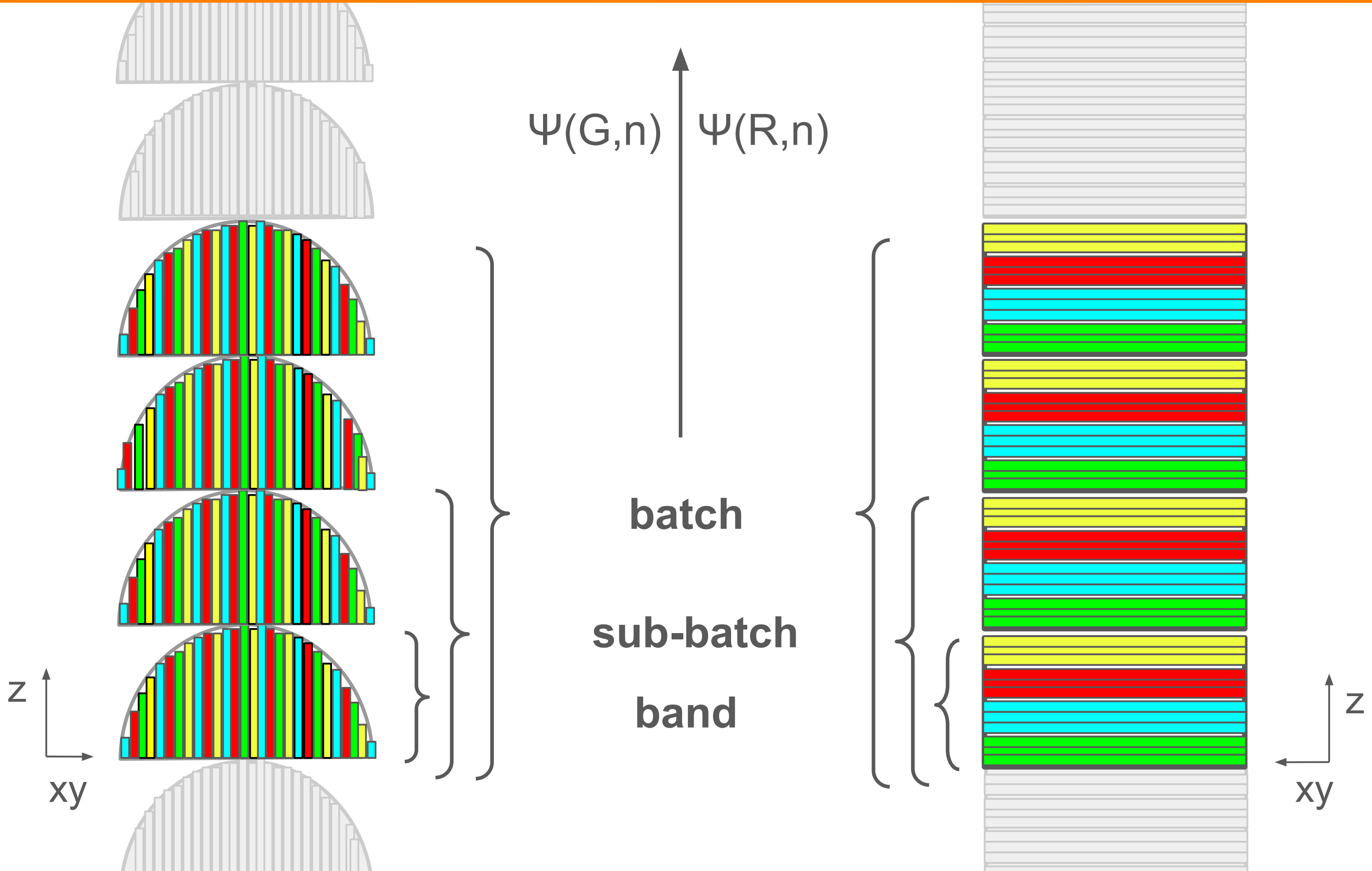
Reference benchmark: **Ausurf 112** (1 scf step). **FFTXlib** calls, **preliminary** results:



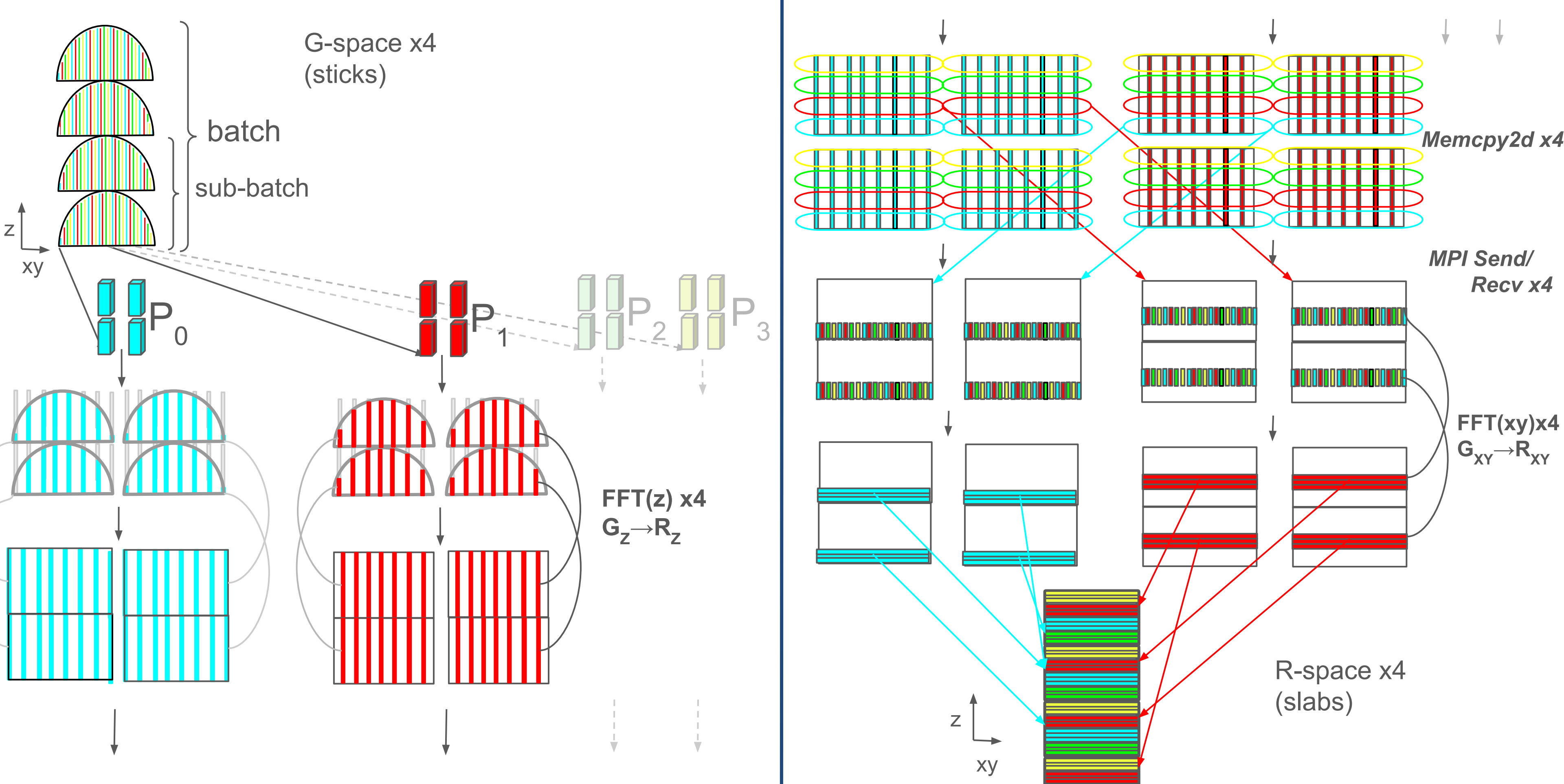
3d FFTs with SLAB decomposition (standard case):

- 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).

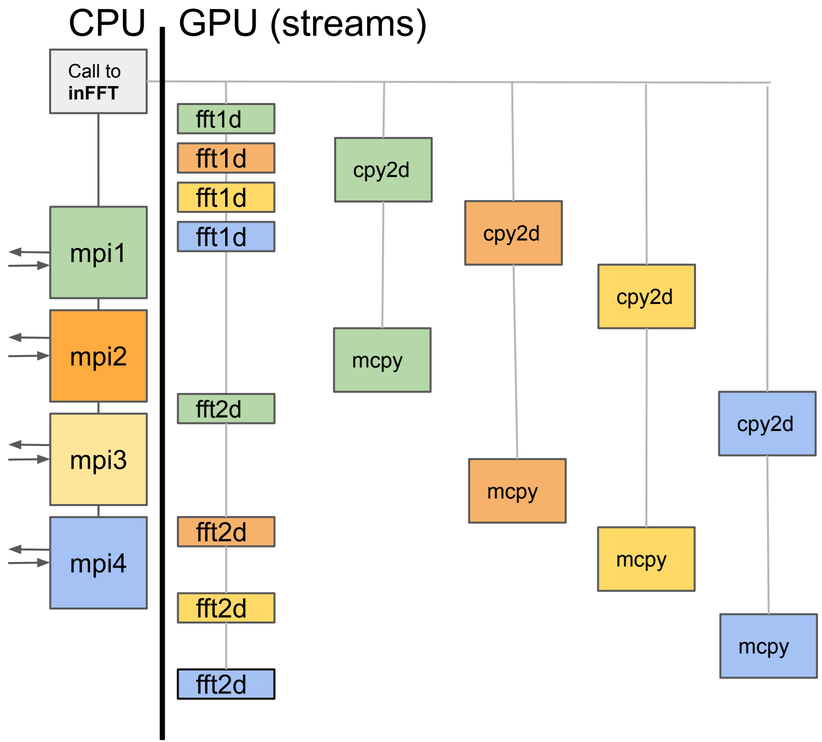
FFTXlib: many bands



FFTXlib: slab decomp. & many bands



Batched FFTs - CUDA/HIP streams

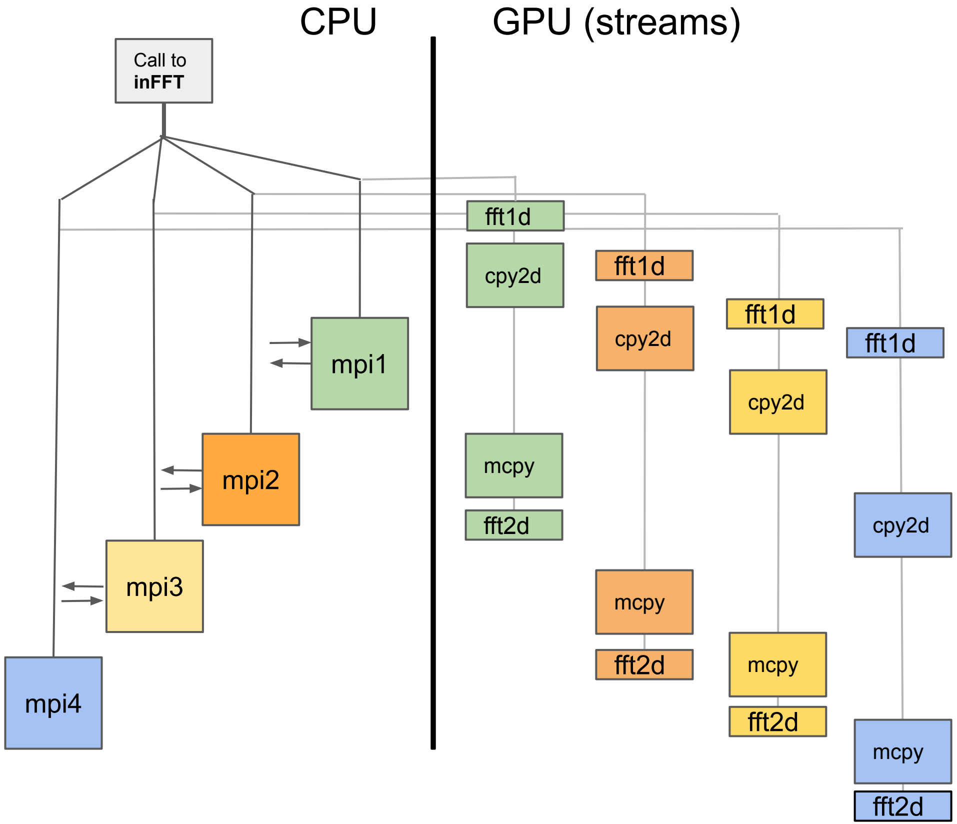


- **Batched** 3d-FFT of the **wave-function**;
- the input array divided in **4 batches** (on bands);
- 1 stream for **FFTs**, 4 streams for **data movements**;
- 4 **async mpi** communications (ISEND, IRECV).

Notes:

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

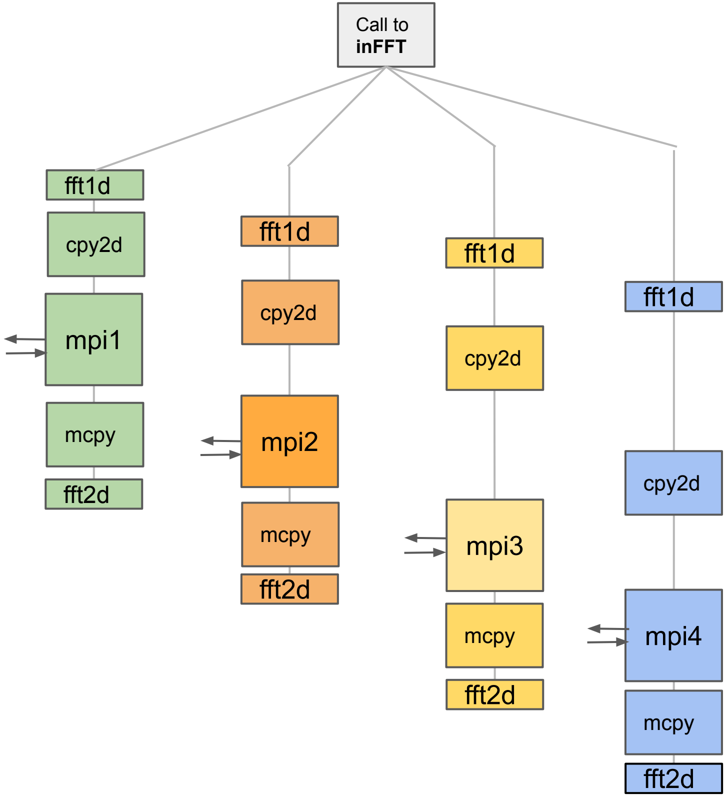
Batched FFTs - OMP tasks + CUDA/HIP streams



- 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;

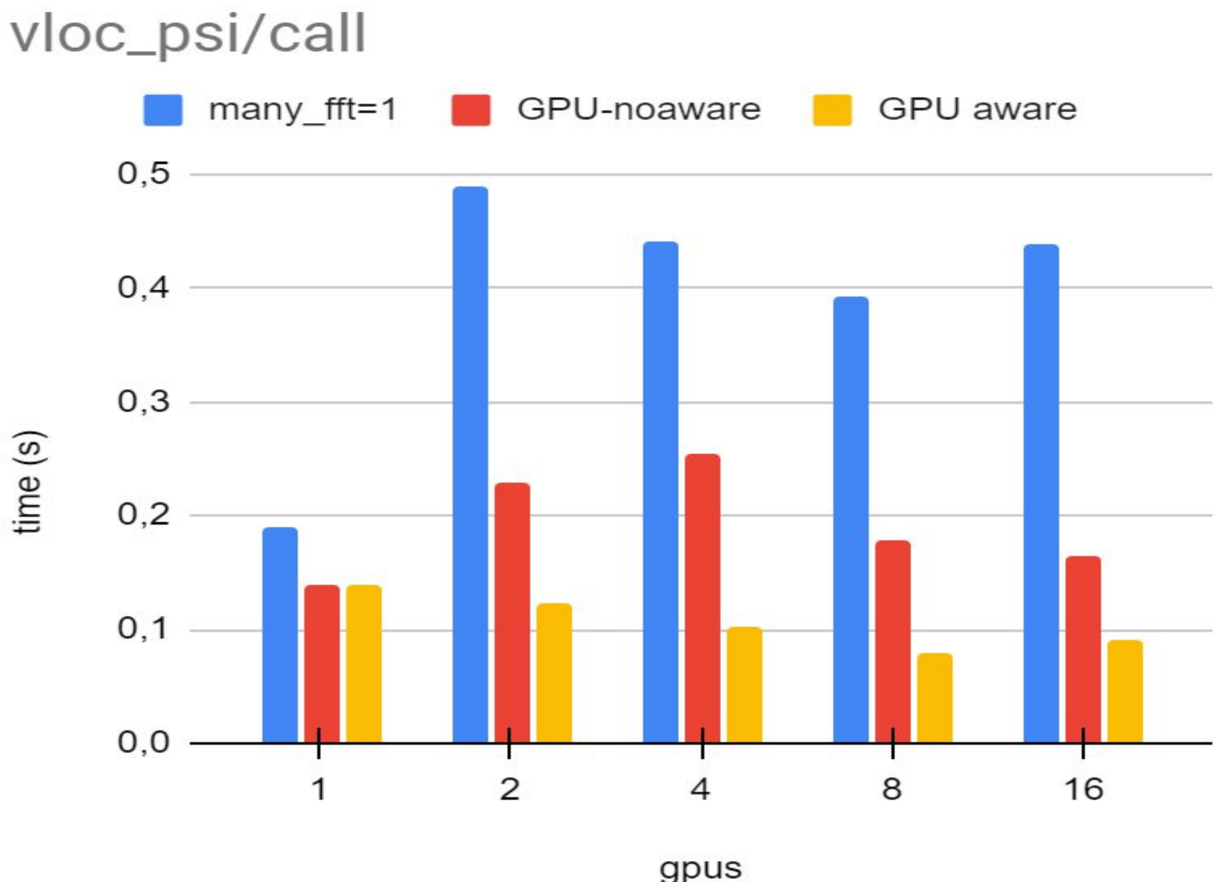
```
nov 28
!$omp parallel
!$omp single

DO j = 0, batchsize-1, dfft%subbatchsize
  currsize = min(dfft%subbatchsize, batchsize - j)
  !
  !$omp task firstprivate(j,currsize) private(i) shared(ptr_callback) detach(event)
  !
  DO i = 0, currsize - 1
   CALL cft_1z_omp( f((j+i)*dfft_nnr + 1:), sticks(me_p), n3, nx3, isgn, &
                  aux(j*dfft_nnr + i*ncpx*nx3 +1:), &
                           stream=dfft%bstreams(j/dfft%subbatchsize+1) )
  ENDDO
  !
  CALL fft_scatter_many_columns_to_planes_store_omp( dfft, aux(j*dfft_nnr+1:), nx3, &
                                                     dfft_nnr, f(j*dfft_nnr+1:), &
                                                     sticks, dfft%nr3p, isgn, currsize, &
                                                     j/dfft%subbatchsize+1 )
  !
  iadc = hipStreamAddCallback(dfft%bstreams(j/dfft%subbatchsize+1),ptr_callback ,c_loc(event), 0)
  !
  !$omp end task
  !
ENDDO
!
```

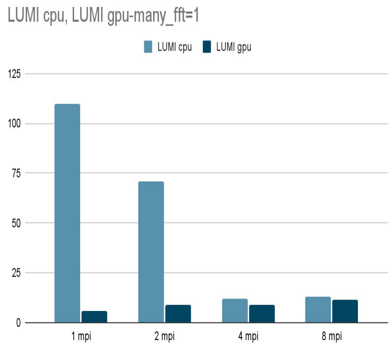
Batched FFTs - OMP tasks + dep.



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

Batched FFTs - performance



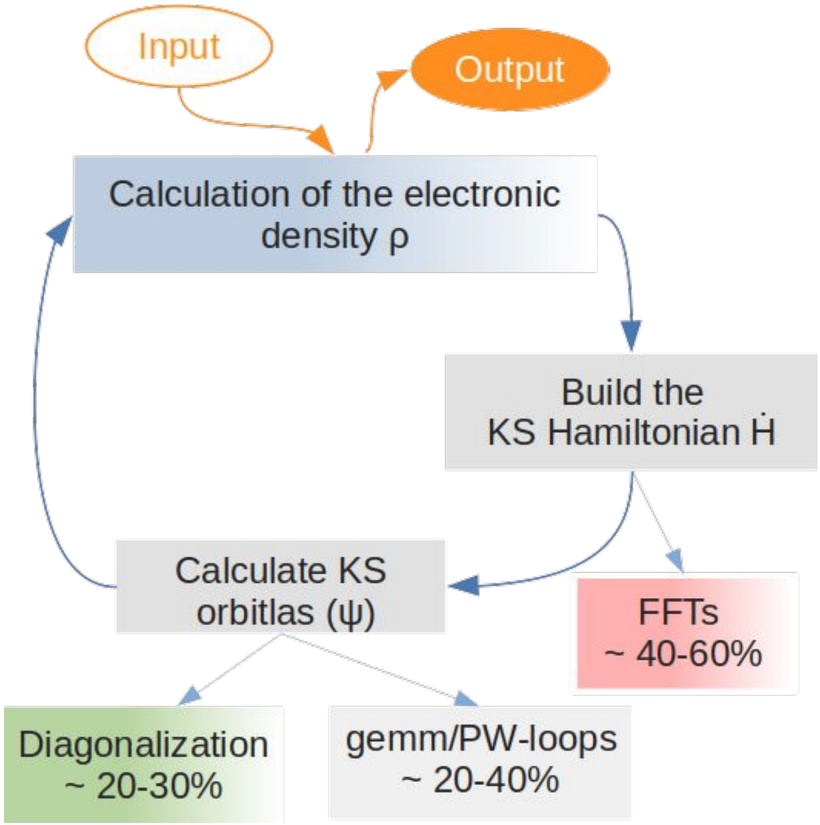


- Gold surface;
- 112 atoms;
- ~1600 electrons.

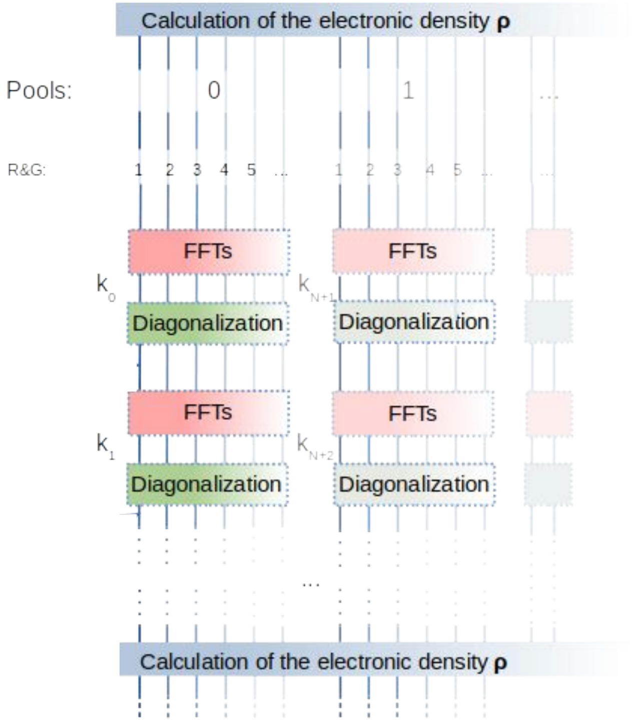
- vloc_psi only.

QE codes and performance results

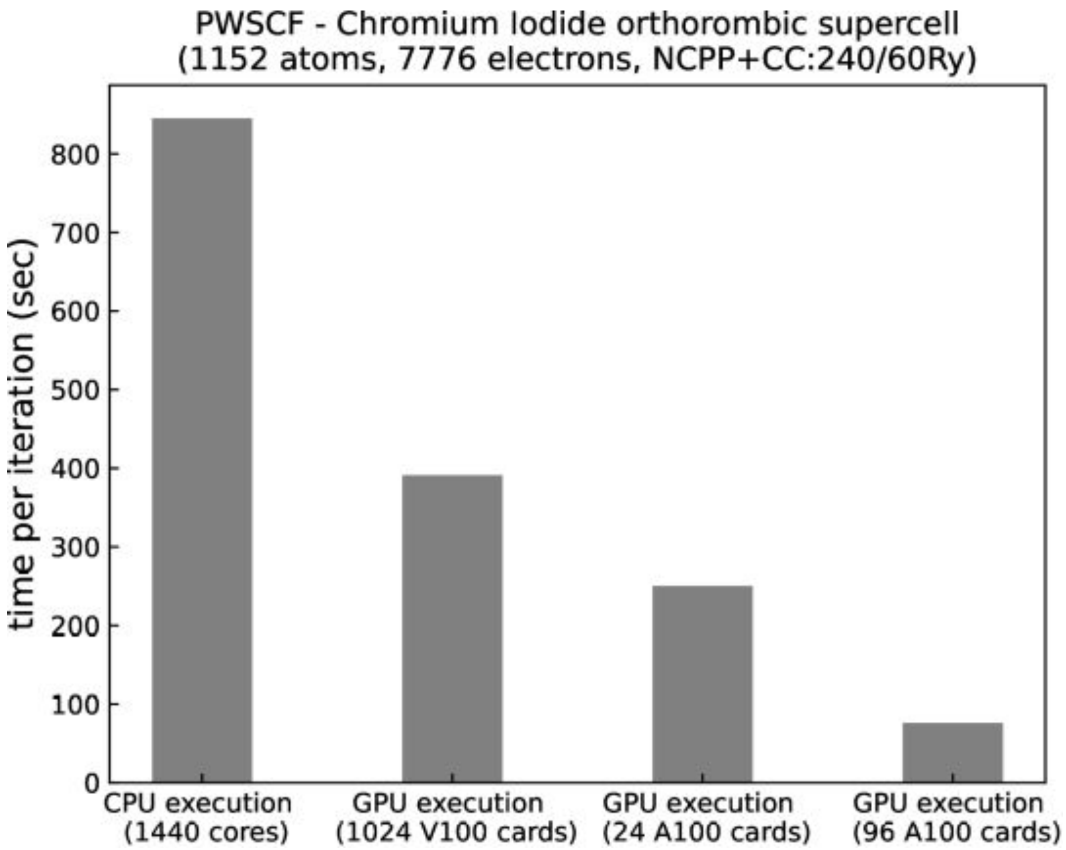
PWscf: flow



MPI + OpenMP

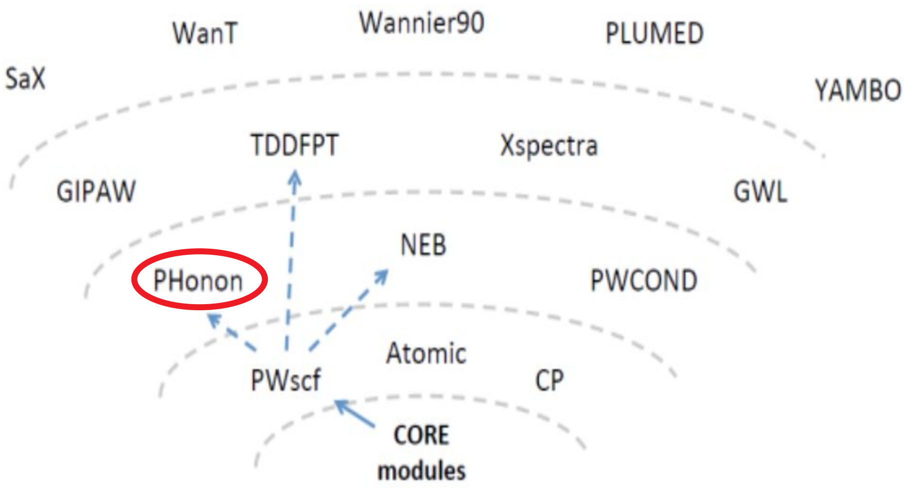


PWscf: performance



- ‘only’ **16 GB** for **V100** (so no pools) but still 1024 GPUs better than optimized run on 1440 cores (Marconi m100);
- **80 GB** for **A100** (Selene): 24 GPUs (no pools) better than 1024 V100;
- **~3x** speed-up with **96 A100** (3 pools) **vs 24 A100** (no pools).

PH: phonon



The **PHonon** code works for a rather wide variety of systems and methods:

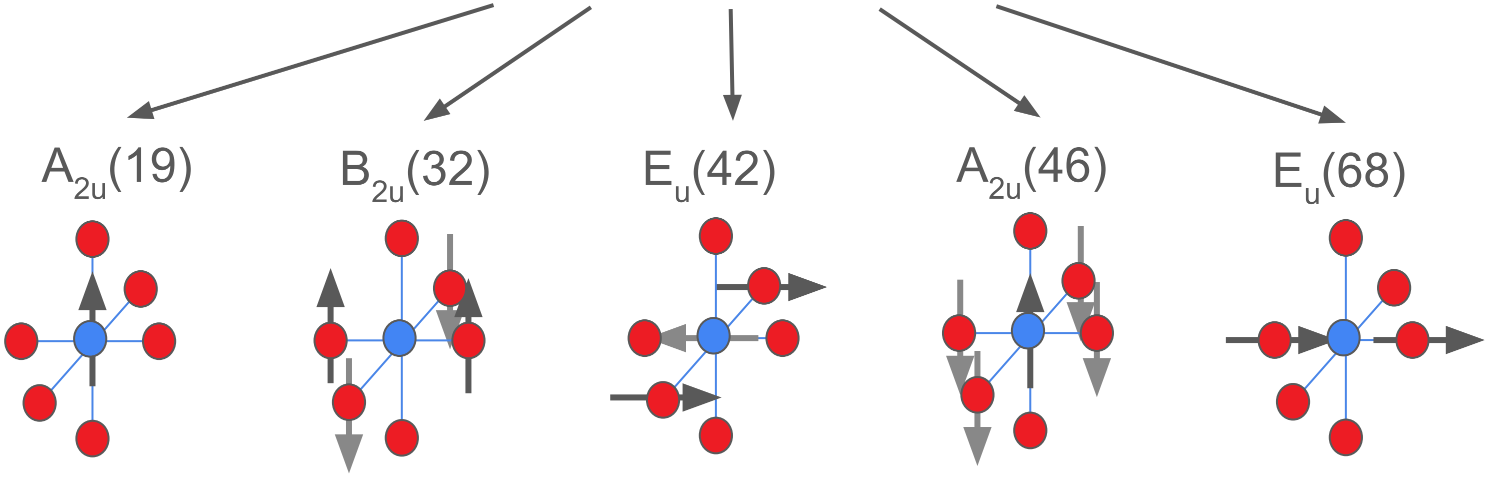
- ✓ **Insulators** (also polar insulators, with LO-TO splitting)
- ✓ **Metals**
- ✓ **Magnetic systems** at the scalar relativistic collinear level
- ✓ Spin-orbit coupling (fully relativistic approach)
- ✓ Electric field calculations: Born effective charges, **dielectric tensor**

PH: phonon

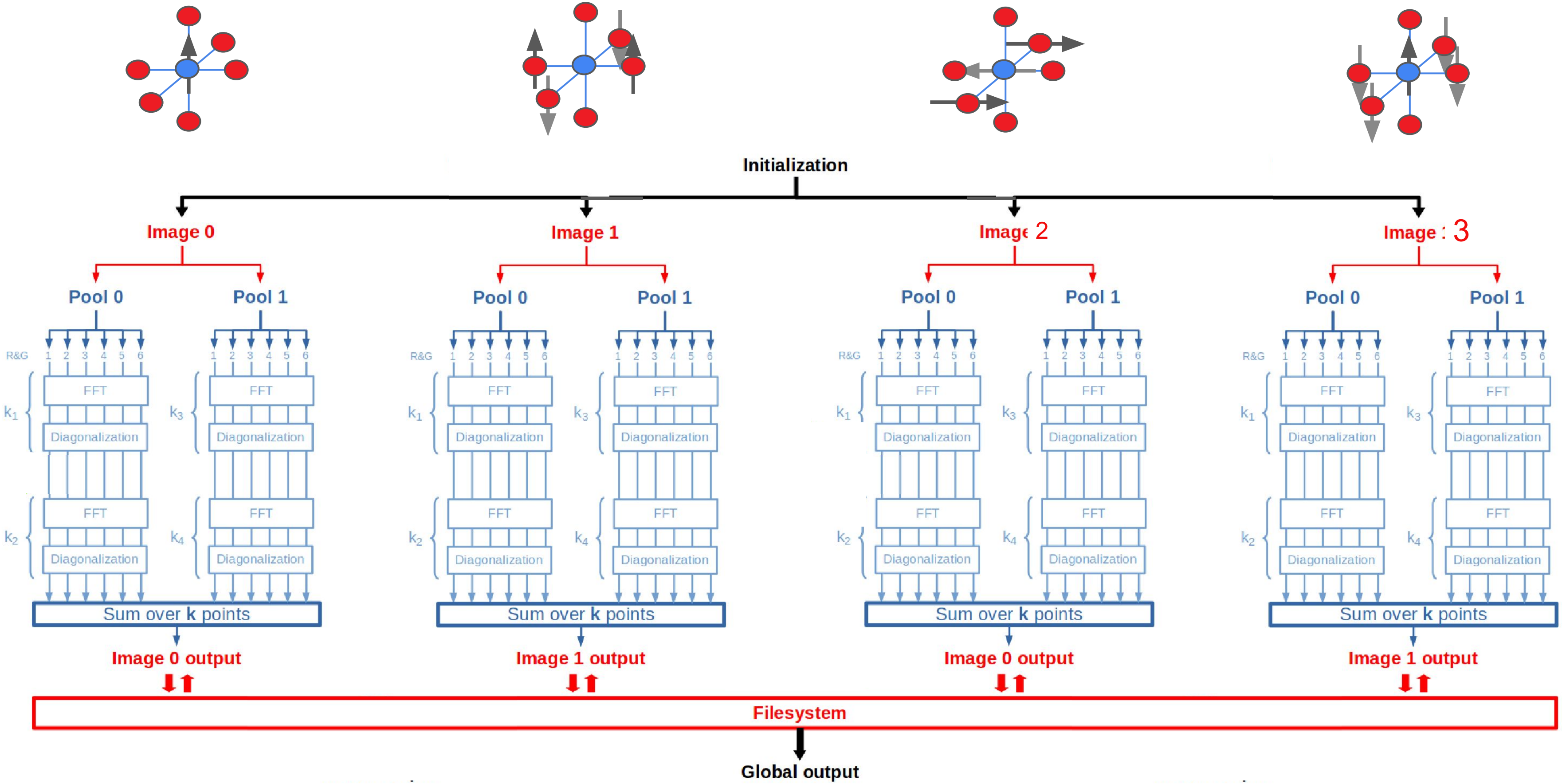
Interatomic Force Constants (IFC)

$$\tilde{D}_{s\alpha,s'\beta}(\mathbf{q}) = \frac{1}{\sqrt{M_s M_{s'}}} \sum_{\mathbf{R},\mathbf{R}'} \frac{\partial^2 E_{tot}}{\partial \mathbf{u}_{s\alpha}(\mathbf{R}) \partial \mathbf{u}_{s'\beta}(\mathbf{R}')} e^{i\mathbf{q}(\mathbf{R}'-\mathbf{R})}$$

Irreducible vibration modes

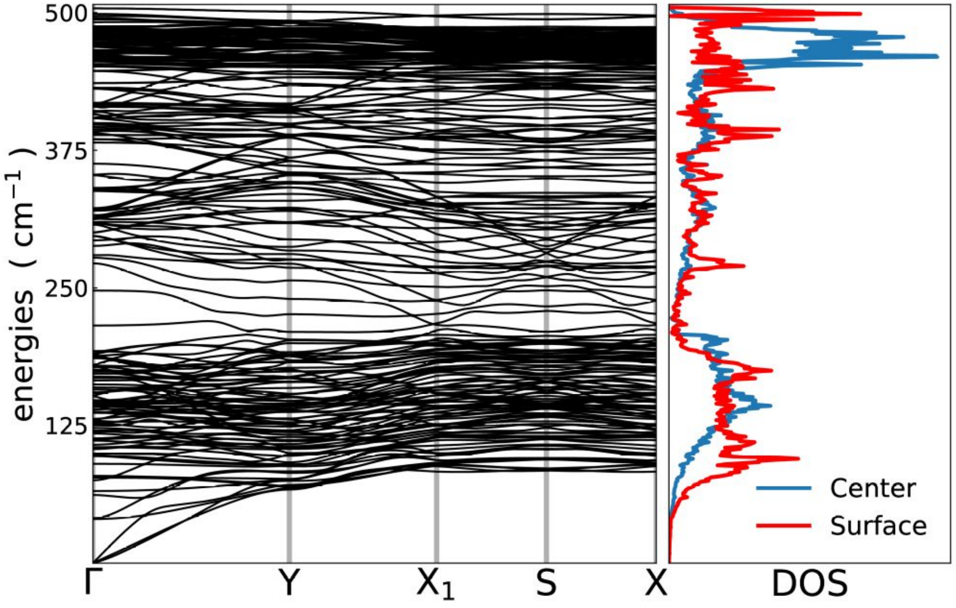


PH: parallel scheme



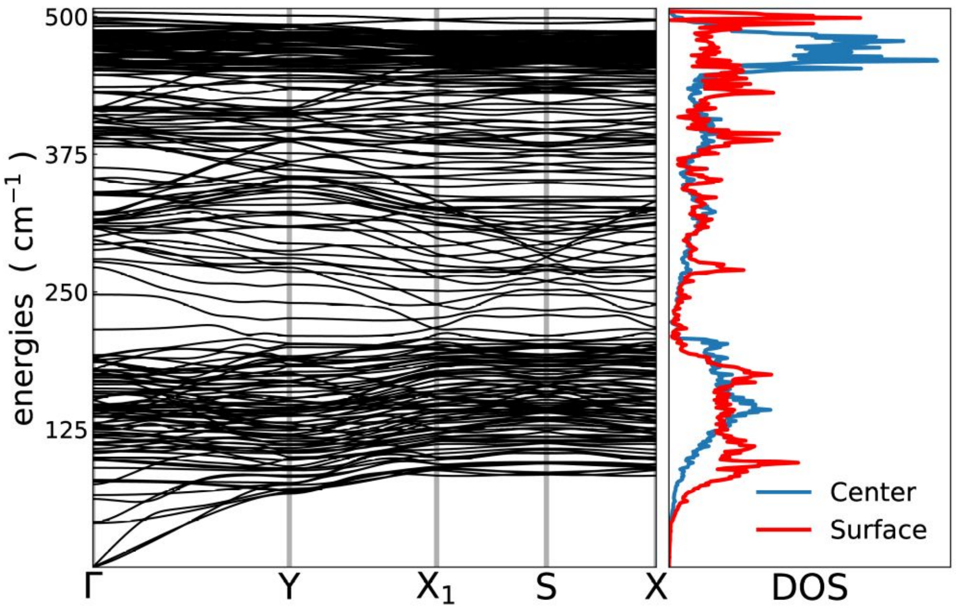
PH: performance

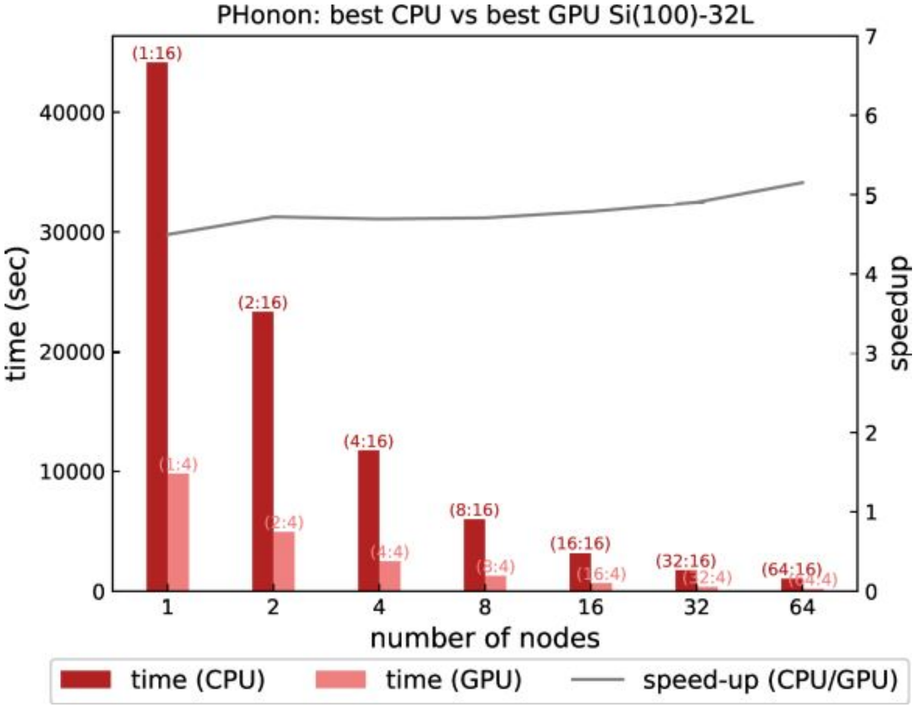
Silicon 100: phonon dispersion



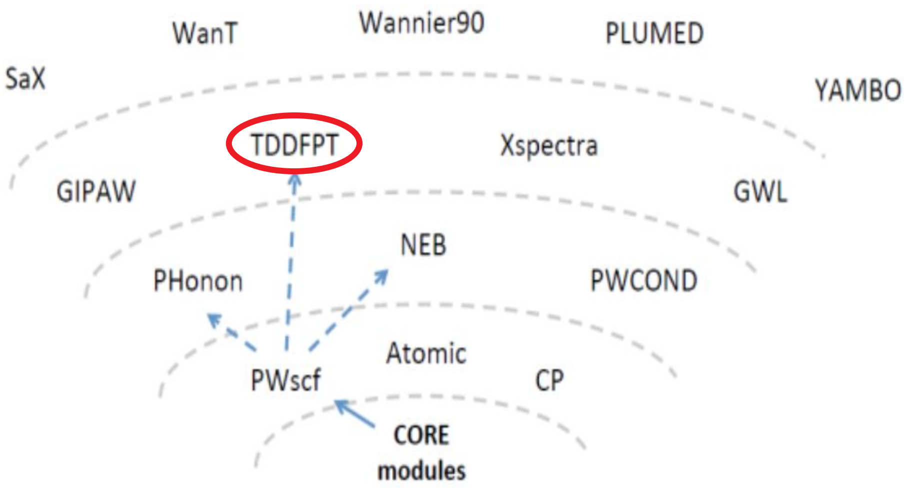
PH: performance

Silicon 100: phonon dispersion





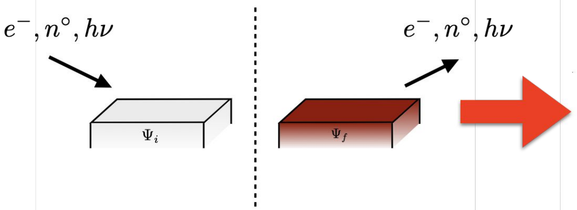
TDDFPT: EELS

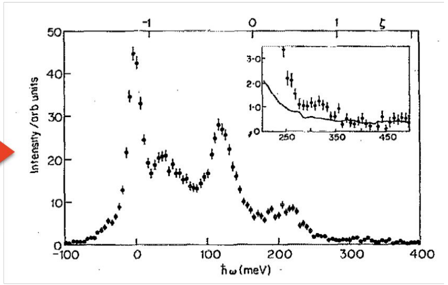


Time-Dependent Density-Functional Perturbation Theory (TDDFpT):

- ✓ optical absorption spectroscopy;
- ✓ Electron energy loss spectroscopy (EELS);
- ✓ Inelastic X-ray scattering (IXS);
- ✓ Inelastic neutron scattering (INS);

TDDFPT: EELS





Dynamical susceptibilities

$$\varphi_{\text{ext}}(t) \longrightarrow A(t) \approx A^\circ + A'(t)$$

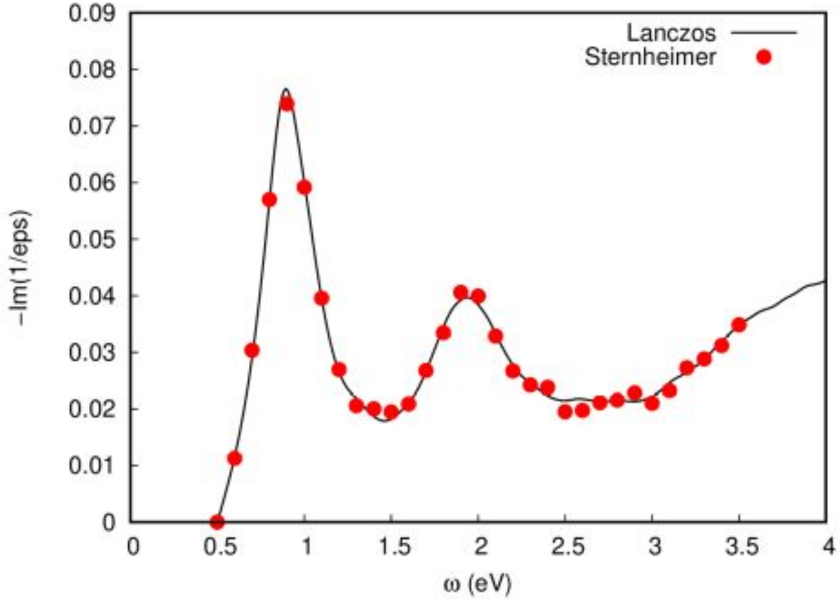
$$A'(t) = \int \mathrm{d}t' \chi(t-t') \varphi_{\text{ext}}(t')$$

Time-Dependent Density-Functional Perturbation Theory (TDDFpT):

- ✓ optical absorption spectroscopy;
- ✓ Electron energy loss spectroscopy (EELS);
- ✓ Inelastic X-ray scattering (IXS);
- ✓ Inelastic neutron scattering (INS);

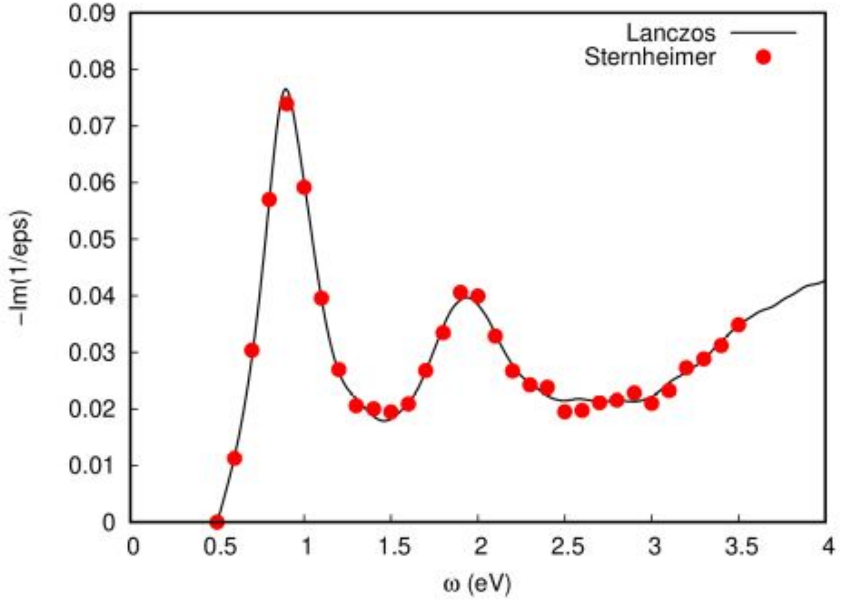
EELS: performance

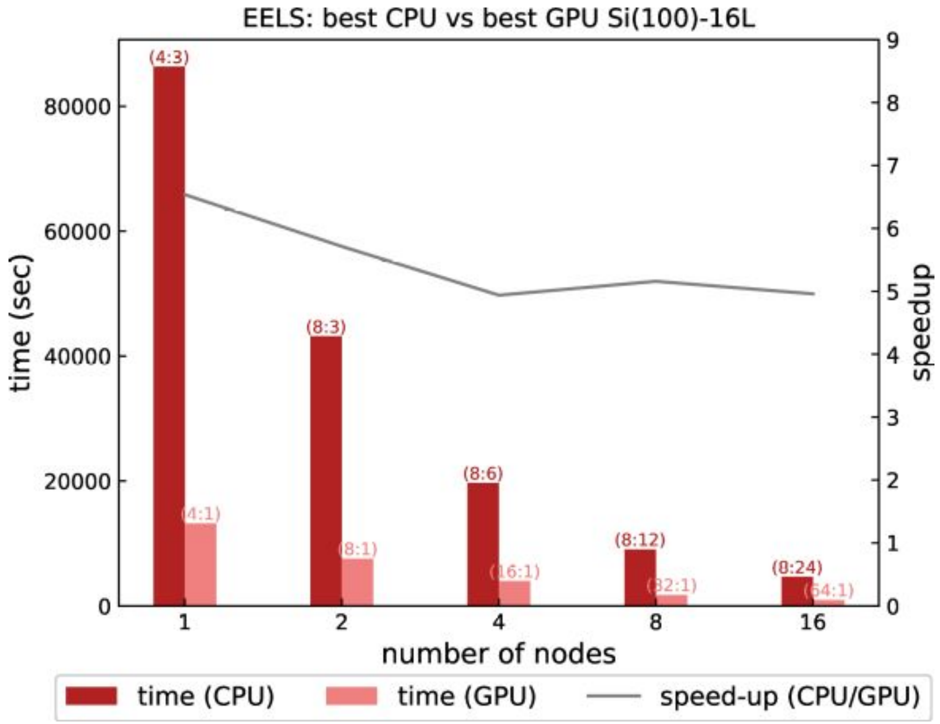
Silicon 100: spectrum of electron energy loss



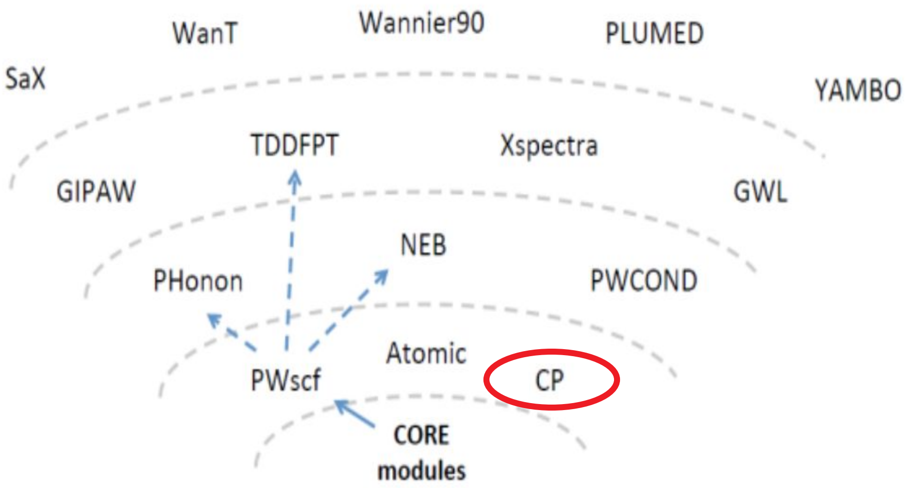
EELS: performance

Silicon 100: spectrum of electron energy loss





Car-Parrinello molecular dynamics: CP

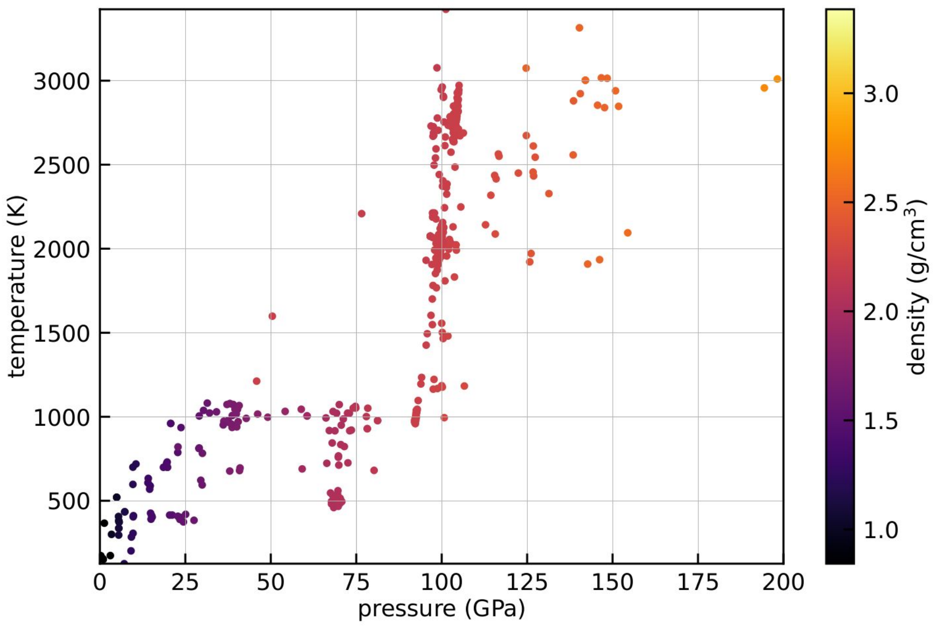


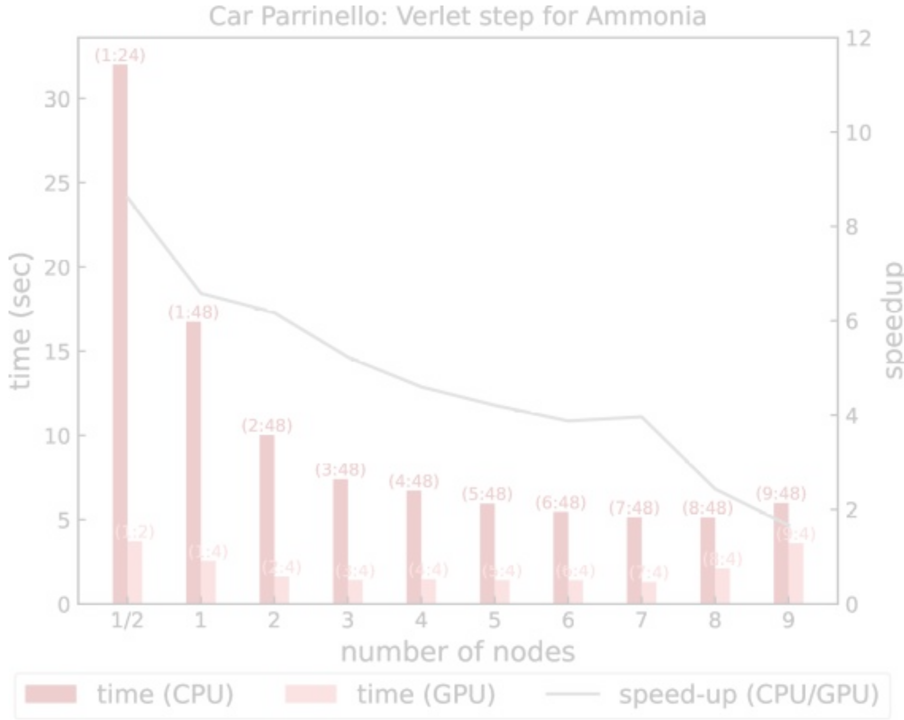
Ab-initio molecular dynamics (**MD**):

- classical molecular dynamics + **QM** electronic structure;
- combines MD with **DFT**;
- accounts for formation or break of **bonds**;
- accounts for complex bindings, e.g. **transition metal ions**

CP: performance

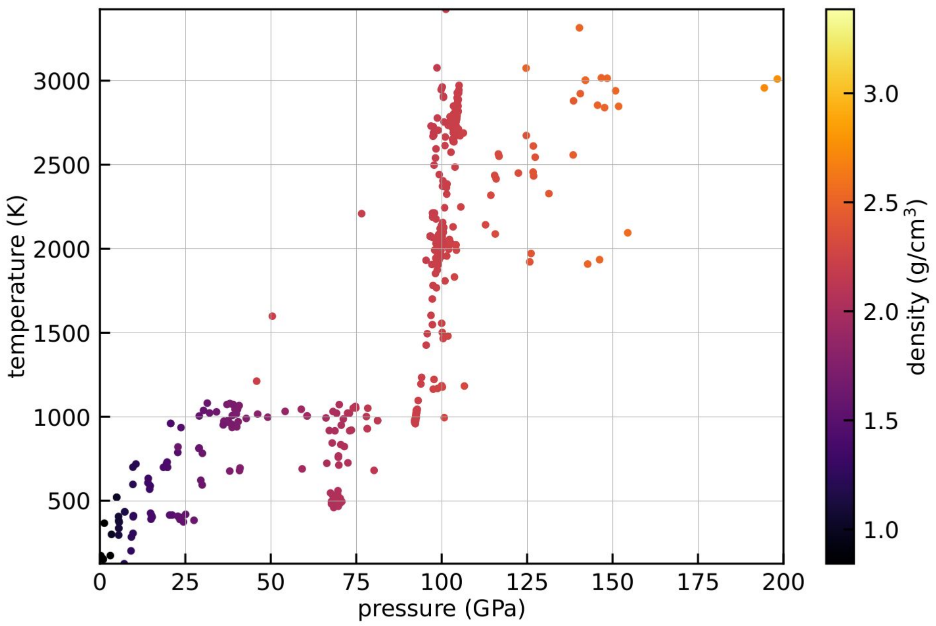
Superionic Ammonia: Eq. of state
144 Nitrogen + 432 Hydrogen (1152 e)

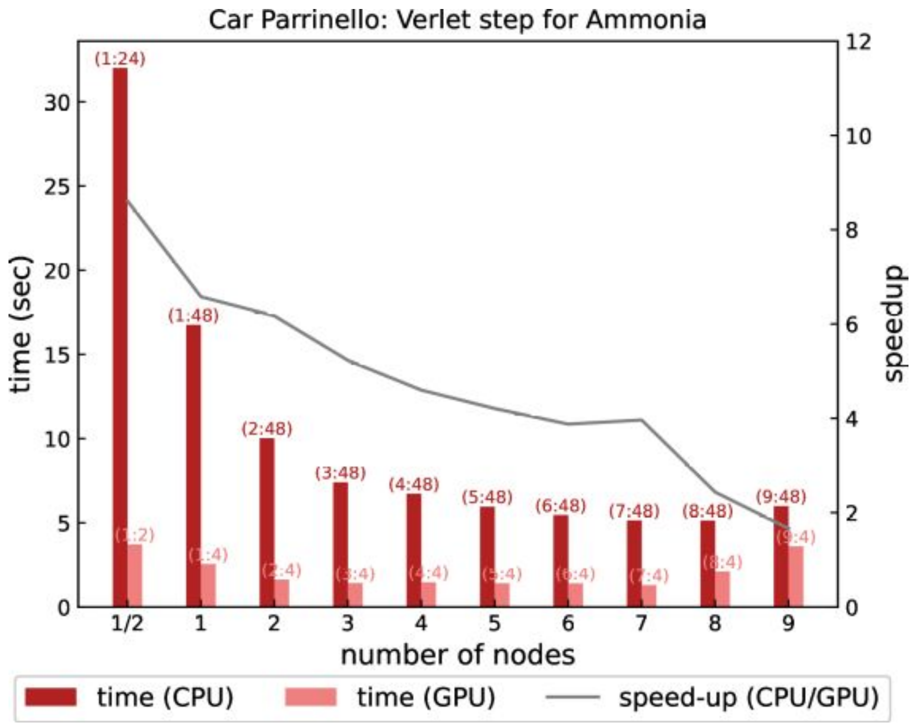




CP: performance

Superionic Ammonia: Eq. of state
144 Nitrogen + 432 Hydrogen (1152 e)





Conclusions

Summary

- **Modularity** of QE;
- **directive** based porting;
- **CPU and GPU** low level routines **at the same time**;
- **multiple standards** with multiple backends;
- full porting on **Nvidia®** side (still transitioning to full openACC);
- ongoing porting on **AMD®/Intel®** side (advanced status on PWscf).

Outlook

- Full port of **PWscf on AMD® and Intel®** by this year likely;
- **merge** Nvidia® standard branch with AMD®/Intel® one;
- port of **codes other than PW** on AMD®/Intel®;
- **extensive benchmarks** on the main the euroHPC machines;
- incorporation of **devXlib**;
- new features….

Acknowledgments

QE DEVELOPERS 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

QE FOUNDATION

- Francesca Garofalo

CINECA

- Laura Bellentani
- Sergio Orlandini
- Fabio Affinito

ARGONNE NL

- Ye Luo

NVIDIA®

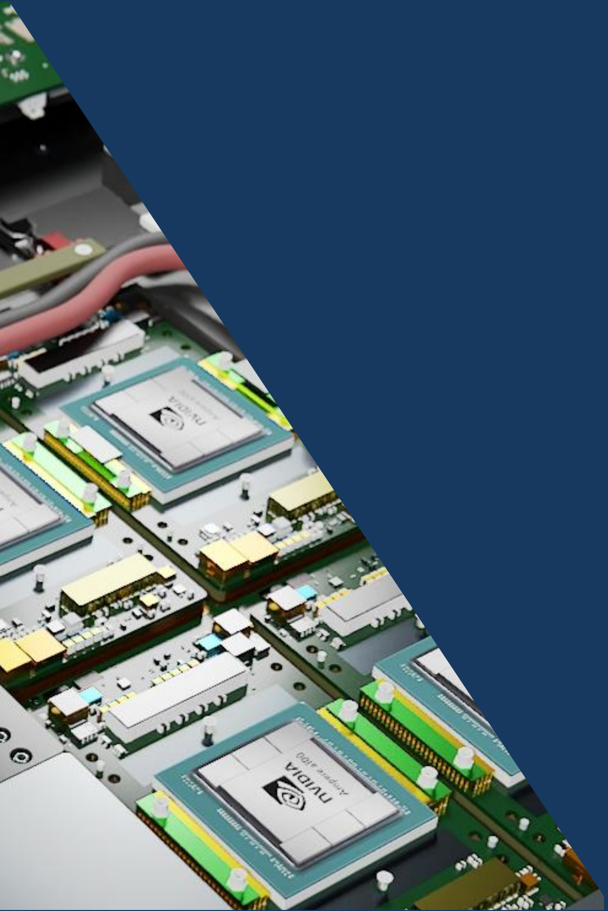
- Filippo Spiga
- Louis Stuber

INTEL®

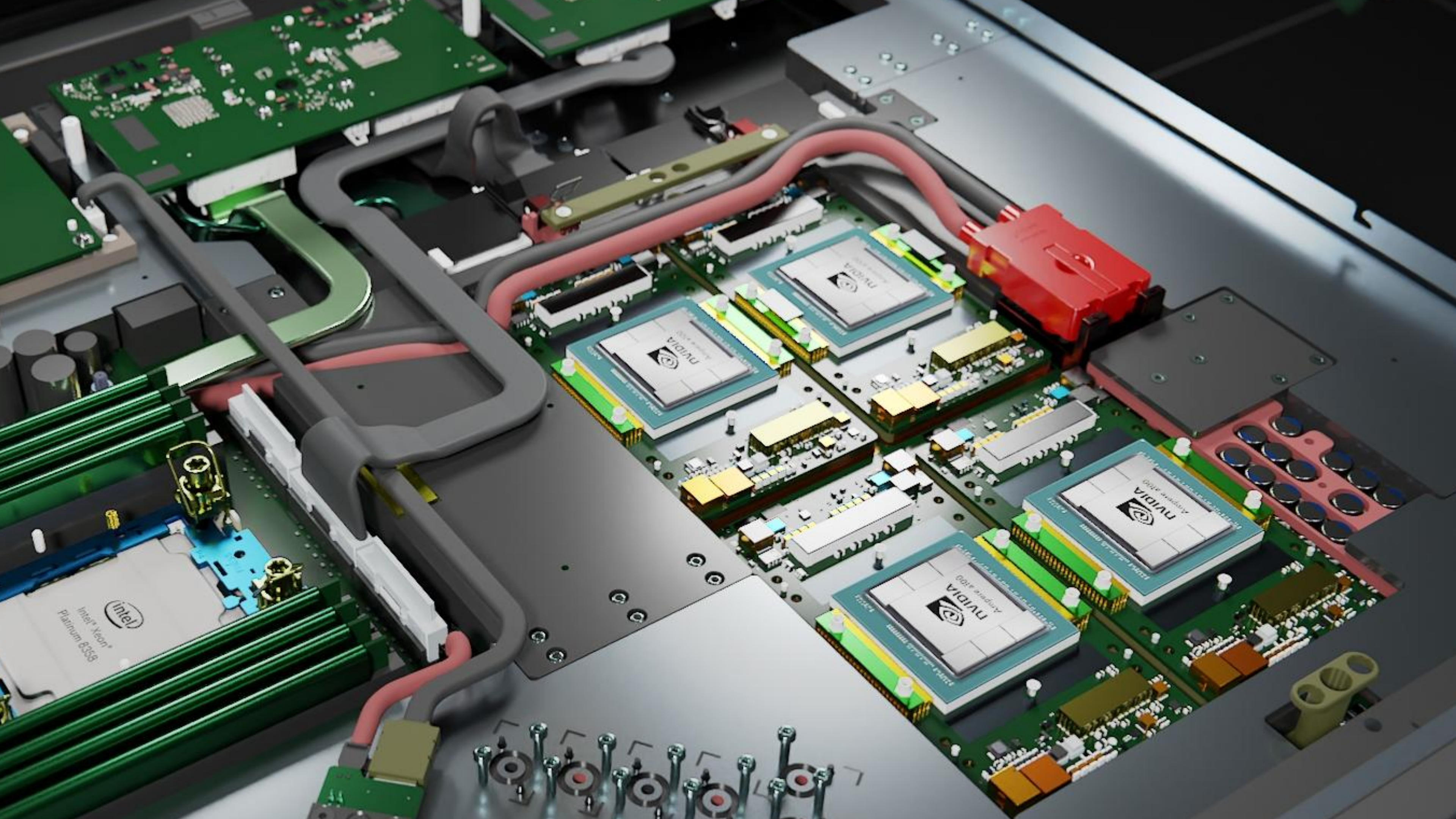
- Giacomo Rossi

AMD®

- Ossian O'Reilly
- Jakub Kurzak



Thank you


intel
Intel® Xeon® Platinum 8358
NVIDIA
Ampere A100