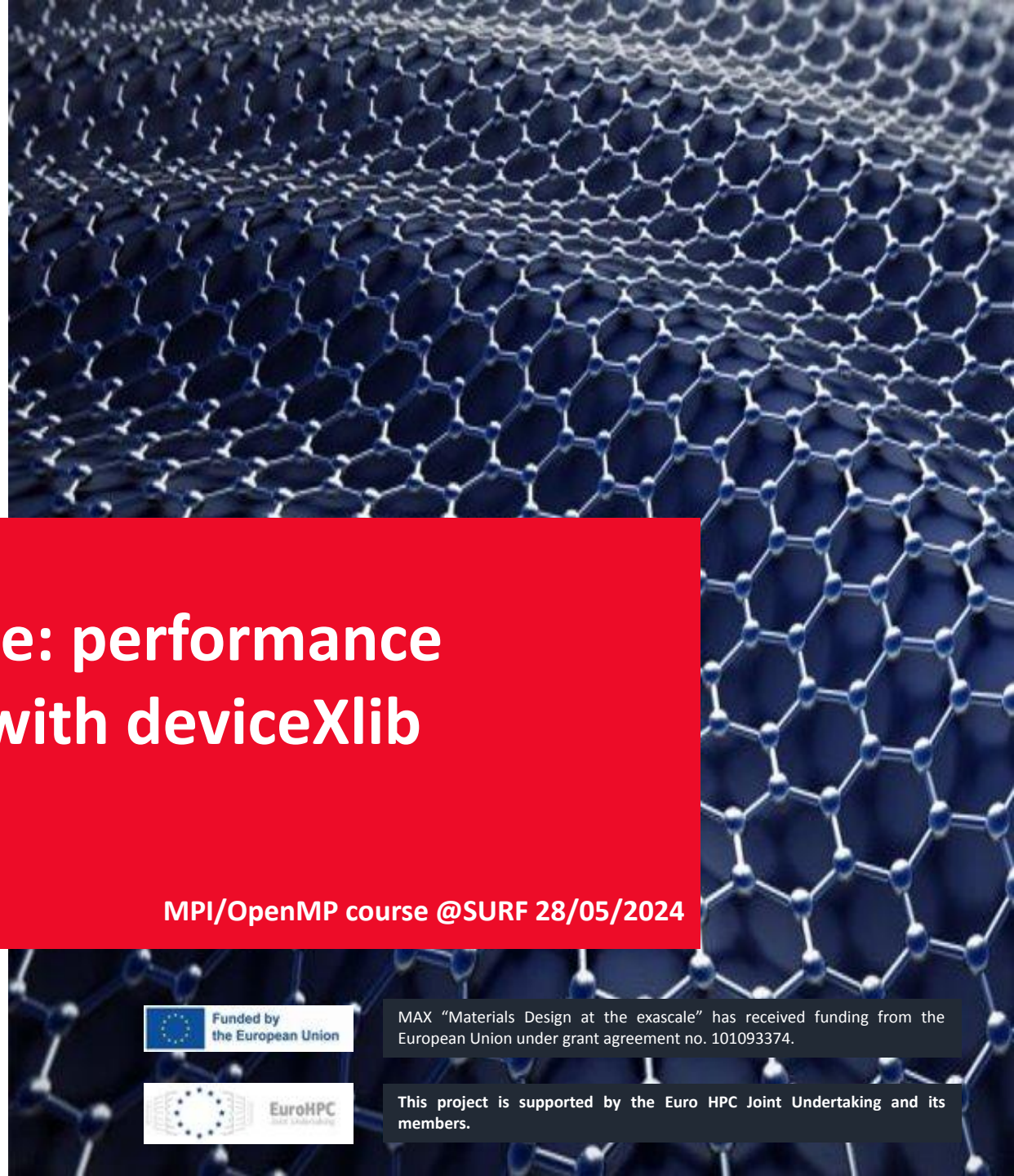
MAX DRIVING THE EXASCALE TRANSITION



Materials design at the eXascale: performance portability of the Yambo code with deviceXlib

Nicola Spallanzani
S3 Centre, Istituto Nanoscienze CNR, Modena - Italy

MPI/OpenMP course @SURF 28/05/2024

Funded by the European Union

MAX "Materials Design at the exascale" has received funding from the European Union under grant agreement no. 101093374.

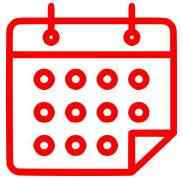
EuroHPC

This project is supported by the Euro HPC Joint Undertaking and its members.

MaX: «Materials design at the eXascale European Centre of Excellence»

Main Goal

Enable open source community codes in electronic structure (materials science) and their workflows on exascale machines.



Our story

1st phase 2015 – 2018

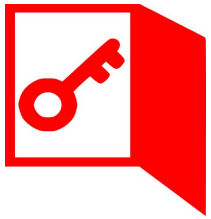
2nd phase 2018 – 2022

3rd phase:

48 Months Project (Jan 2023 – Dec 2026)

4 Key-Actions

1) Restructure MAX flagship codes towards exascale and extreme scaling performance.



2) Design, development and implementation of the architecture and orchestration of the exascale workflows.

3) Co-design and energy efficiency for HPC architectures

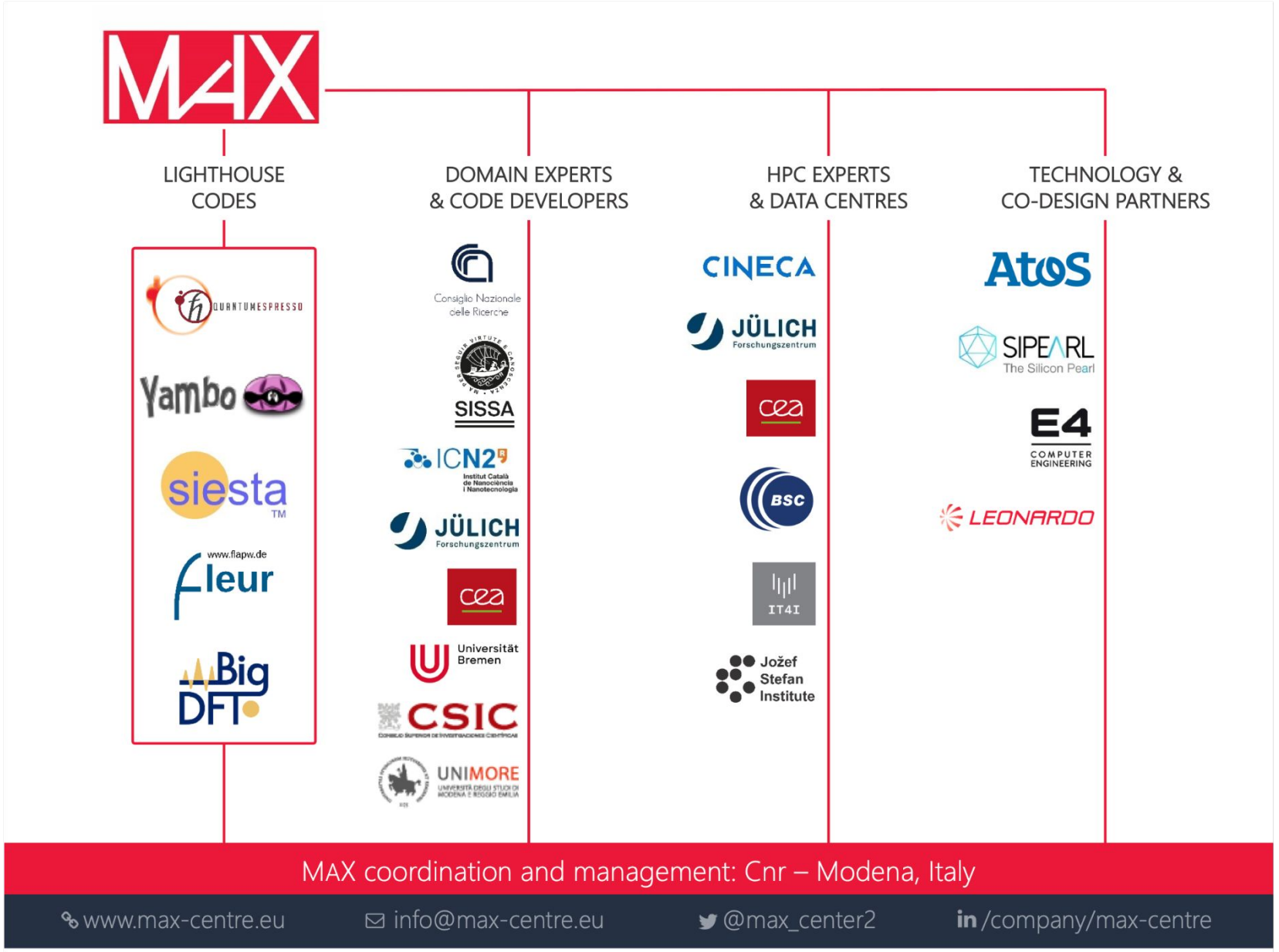
4) Widen the access to codes and foster transfer of know-how to user communities.

16 Partners with unique expertise

- Materials Science
- Software Development
- Code Validation
- System & Data Management
- Communication & Outreach

MAX DRIVING THE EXASCALE TRANSITION

A partnership with the required skills



MaX: Lighthouse Codes

Focus: ‘first principles’ materials science codes at the exascale and beyond

The prime open-source (set of) code(s) for quantum materials modelling using the planewave pseudopotential method.

A code family for calculating groundstate as well as excited-state properties of solids within the context of density functional theory.

A density-functional code able to perform efficient electronic structure calculations and ab initio molecular dynamics simulations of molecules and solids.

An electronic structure pseudopotential code that employs Daubechies wavelets as a computational basis, designed for usage on massively parallel architectures.

A code that implements ground-state as well as excited-state properties in an ab initio context.

A Python materials’ informatics framework to manage, store, share, and disseminate the workload of high-throughput computational efforts.

MaX: Goal

First exascale supercomputer:

Frontier (@ORNL)

- 1100 PFlops
- 37888 GPUs (AMD MI250X)

We need to turn MaX lighthouse codes into exascale-enabled applications:

- **large scale MPI parallelism** (order of 10000 tasks)
- combined with **GPU awareness**

Single-node optimisation

- make sure MaX codes can exploit accelerated nodes featuring multiple GPU brands

Multi-node parallel efficiency

- make the codes scalable in the presence of GPUs

Scientific software engineering

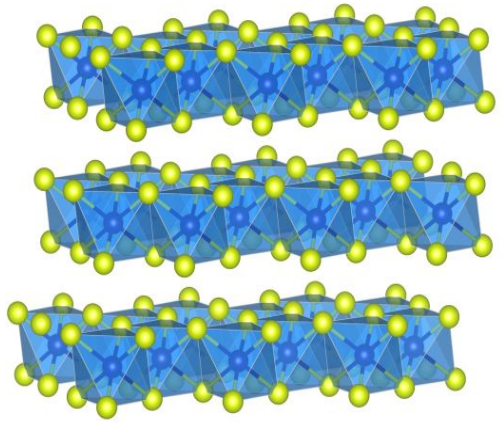
- support long-term maintainability and community contributions

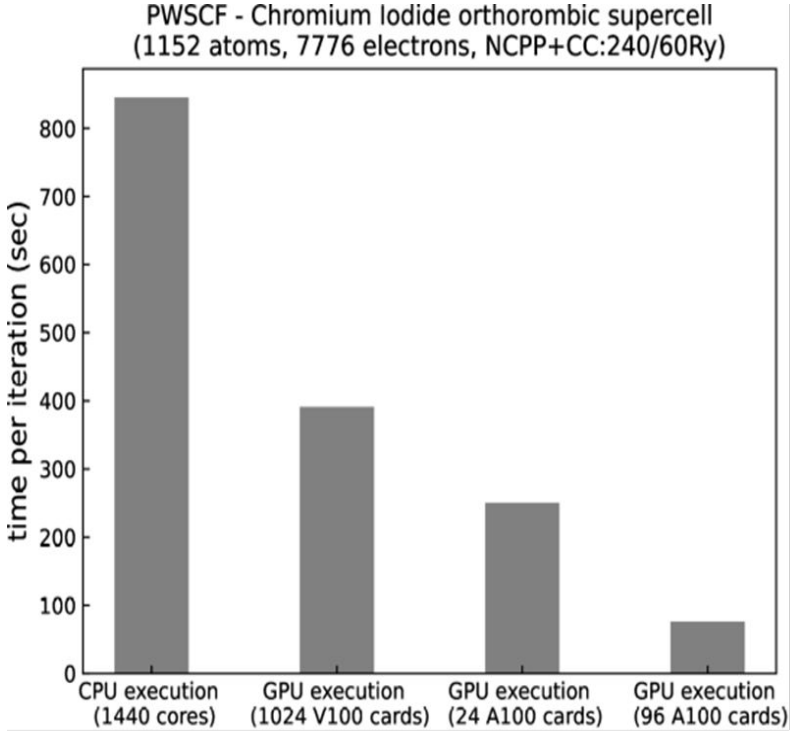
New Scientific features

Quantum ESPRESSO: on pre-exascale

- Currently running on **NVIDIA, AMD, and INTEL accelerated machines**
- Using **CUDA-Fortran, OpenACC, and OpenMP** programming models
- DATA: courtesy of I. Carnimeo & F. Ferrari Ruffino

Chromium Iodide
- 7776 electrons
- 1152 atoms

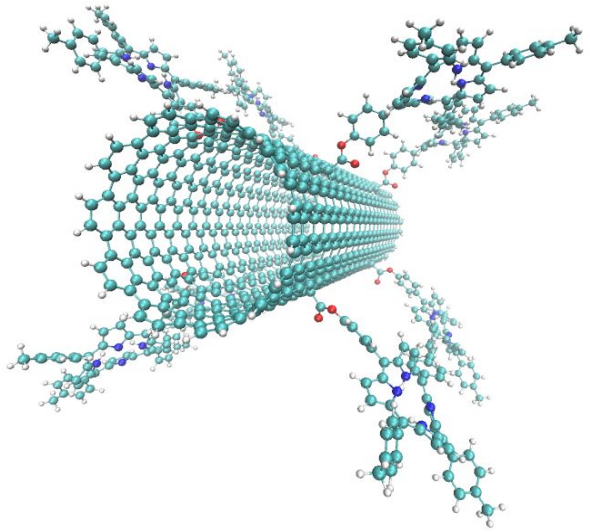


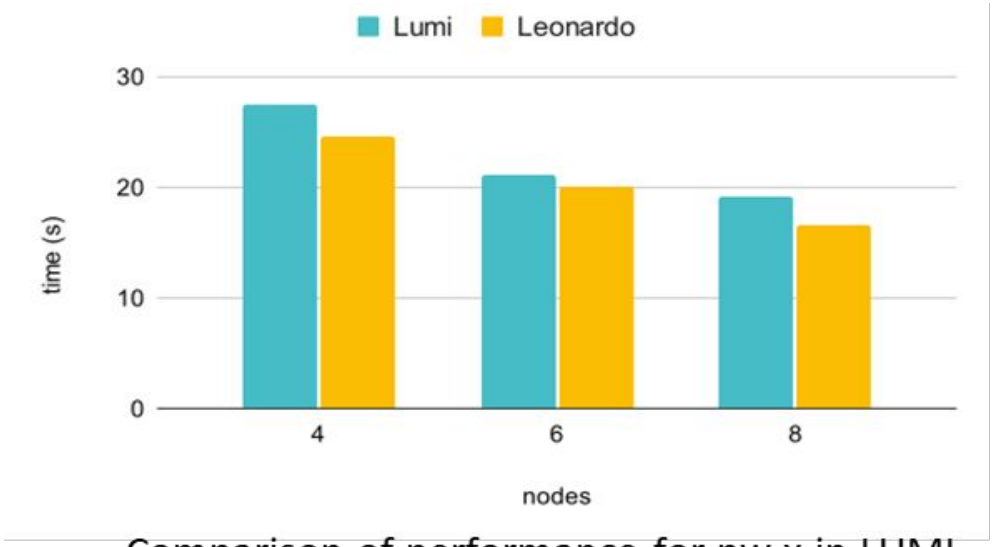


Quantum ESPRESSO: one further step towards the exascale, I. Carnimeo et al., JCTC, **19**, 20, 6992-7006 (2023)

Quantum ESPRESSO: on pre-exascale

- Currently running on **NVIDIA, AMD, and INTEL accelerated machines**
- Using **CUDA-Fortran, OpenACC, and OpenMP** programming models
- DATA: courtesy of I. Carnimeo & F. Ferrari Ruffino





Comparison of performance for pw.x in LUMI-G and LEONARDO-booster for CNTPOR10 test case.

YAMBO: on pre-exascale

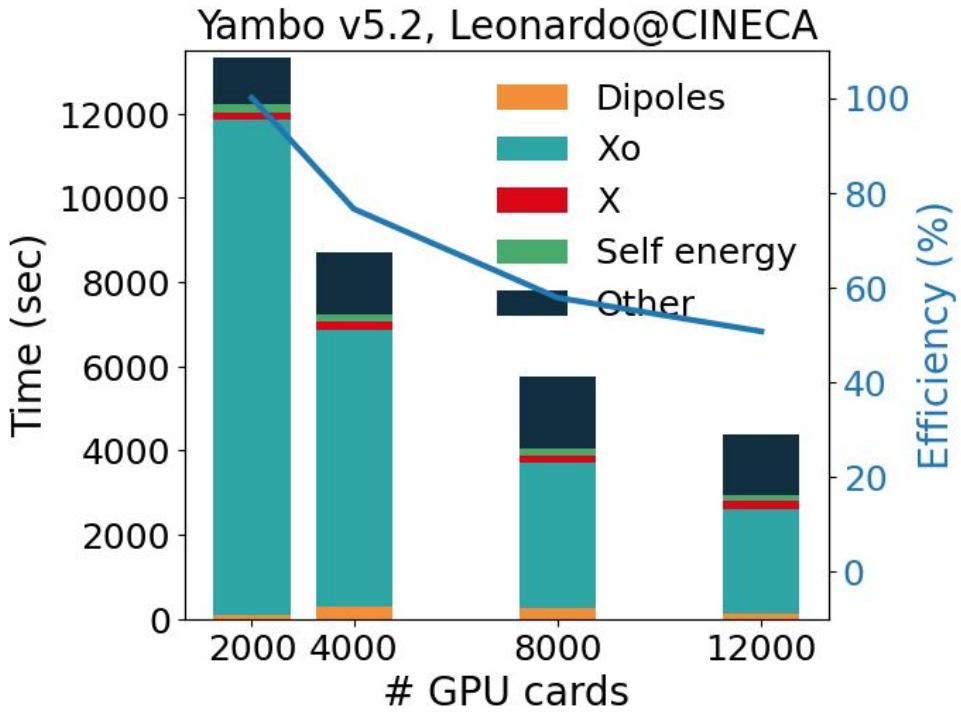
- Currently running on **NVIDIA, AMD, and INTEL accelerated machines**
- Using **CUDA-Fortran, OpenACC, and OpenMP** programming models
- Integration of deviceXlib, a MaX component for wrapping device-oriented routines and utilities

Leonardo-Booster (@CINECA):

4 Nvidia A100(64GB) / node

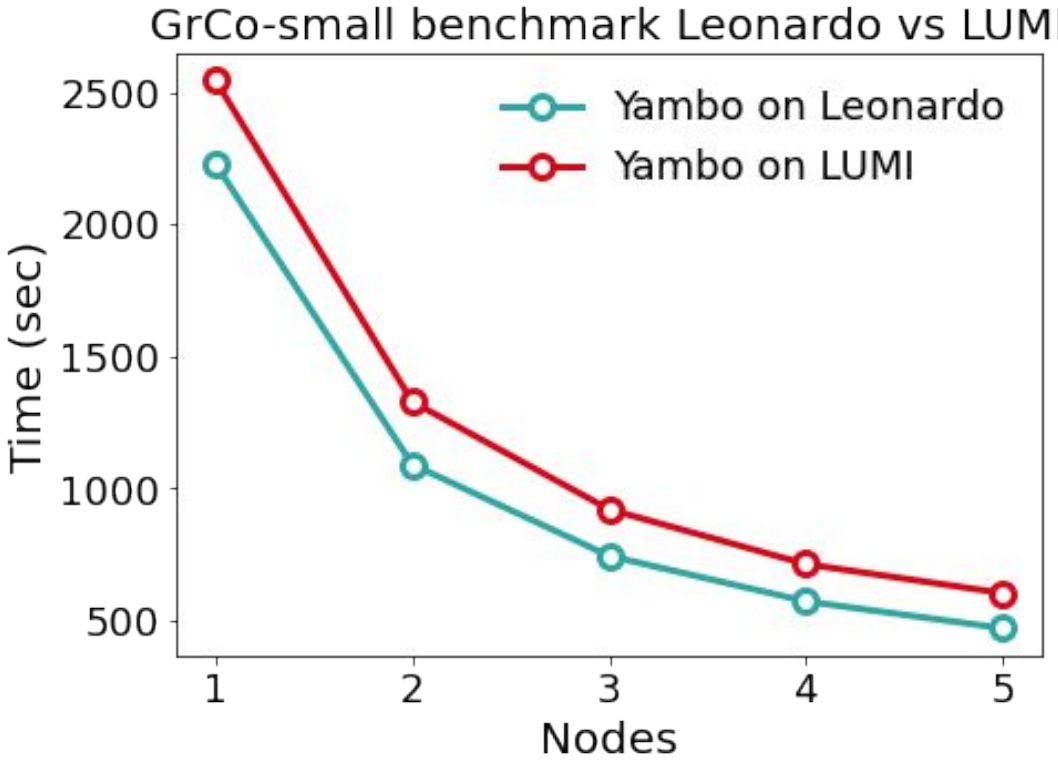
Runs: up to 3000 nodes almost 90% of the whole machine (3496 nodes)

Scalability test obtained thanks to the Leonardo Early Access Program.



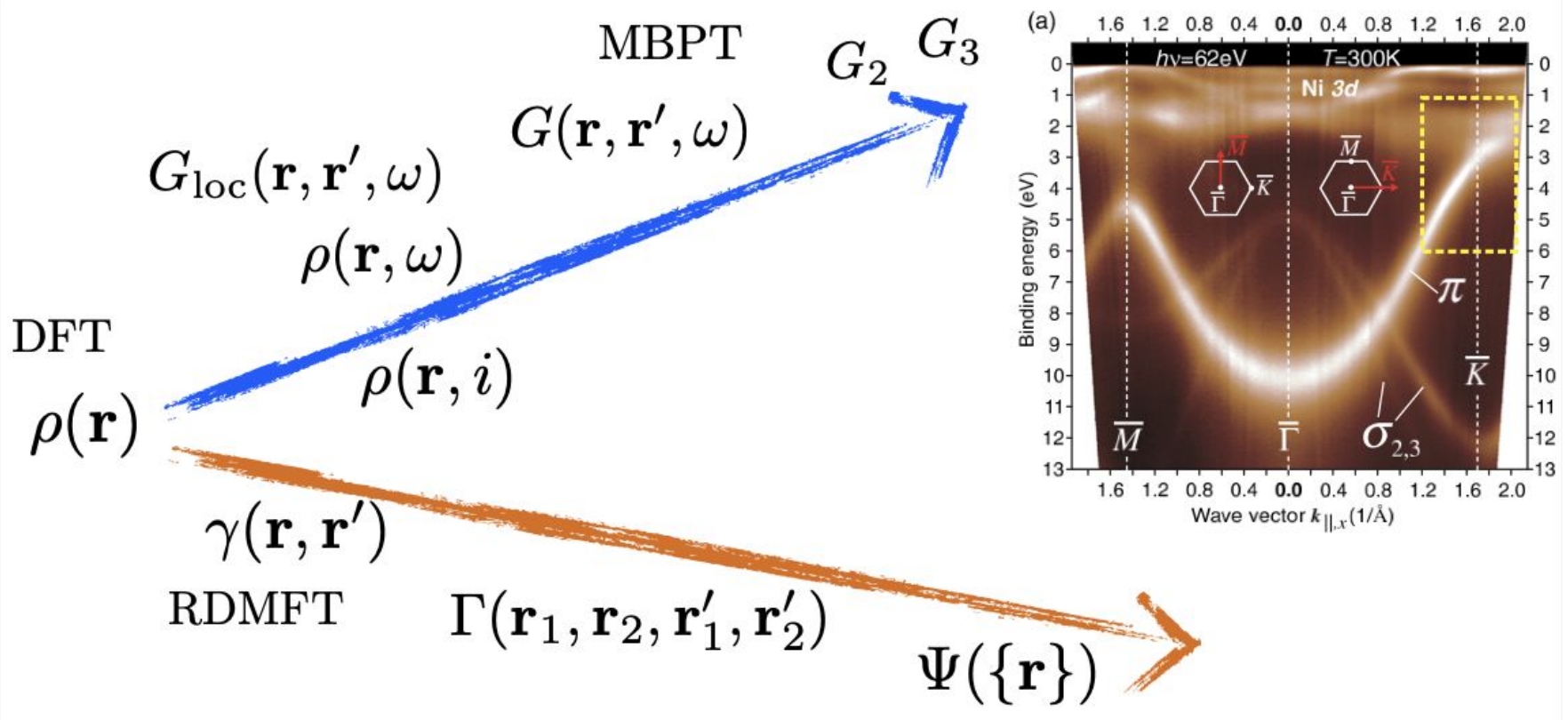
YAMBO: on pre-exascale

- Currently running on **NVIDIA, AMD, and INTEL accelerated machines**
- Using **CUDA-Fortran, OpenACC, and OpenMP** programming models
- Integration of deviceXlib, a MaX component for wrapping device-oriented routines and utilities



Electronic structure Methods

- electronic structure methods compute-intensive
- **GW and MBPT** at the **high-end usage** of computational resources



The YAMBO code

Yambo is an **open-source** code implementing first-principles methods based on Green's function theory to describe **excited-state properties** of realistic materials. These methods include the **GW approximation**, the **Bethe-Salpeter equation**, real-time NEGF, electron and exciton phonon.

$$\left[\frac{-\nabla^2}{2}+v^s(r)\right]\psi_{nk}(r)=\epsilon_{nk}\psi_{nk}(r)$$

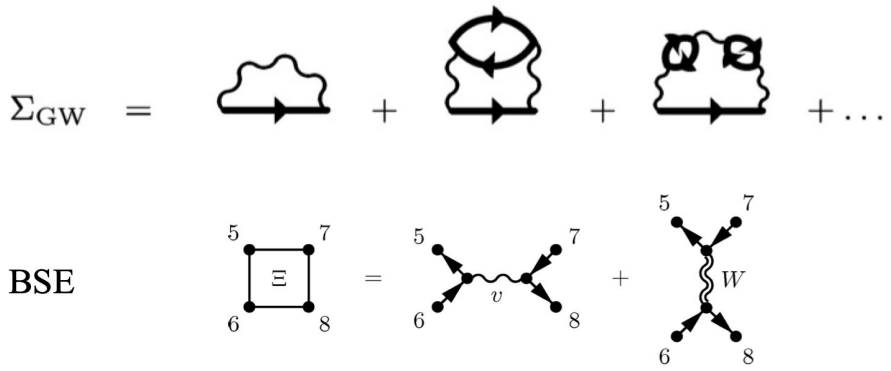
$$G^{(r),KS}(r,r',\omega)=\sum_{nk}\frac{\psi^*_{nk}(r)\psi_{nk}(r')}{\omega-\epsilon^{KS}_{nk}+i\eta}$$

Microscopic screening

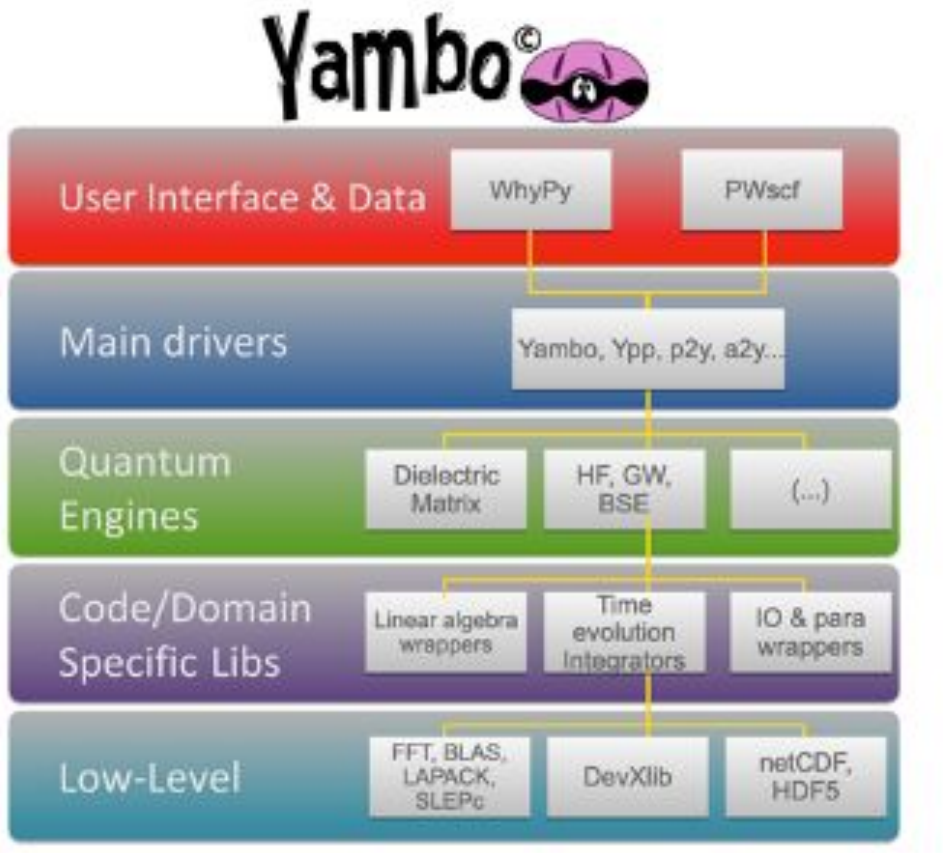
$$\chi_{G,G'}(q,\omega)$$

- **DFT** simulation as **input**
- Interfaced with **Quantum ESPRESSO** and other DFT codes

- **DFT** Green function to define **MBPT** quantities



The YAMBO code: Software architecture



- A. Marini, C. Hogan, M. Gruning, D. Varsano, Comp.Phys.Comm. **180**, 1392 (2009)
- D. Sangalli, et al, J. Phys.: Condens. Matter. **31**, 325902 (2019)

Modularization

- extended use of derived data types
- parallel structure (MPI levels)
- built-in memory tracking
- built-in time profiling

Libraries

- Intense use of wrappers (IO, para, LA, …)
- Internal libraries
- BLAS, LAPACK, FFTW, ScaLAPACK, NetCDF/HDF5, PETSc, SLEPc
- extensive use of deviceXlib

GPU-awareness

- extended CUDA-Fortran support
- single source code
- OpenACC and OpenMP support (beta)

The YAMBO code: Linear Response

$$\chi(\mathbf{q},\omega) = [I - \chi_0(\mathbf{q},\omega)v(\mathbf{q})]^{-1}\,\chi_0(\mathbf{q},\omega)$$

$$\chi^0_{\mathbf{G}\mathbf{G}'}(\mathbf{q},\omega) = 2\sum_{c,v}\int_{BZ}\frac{d\mathbf{k}}{(2\pi)^3}\,\rho^*_{cv\mathbf{k}}(\mathbf{q},\mathbf{G})\rho_{cv\mathbf{k}}(\mathbf{q},\mathbf{G}')\,f_{v\mathbf{k}-\mathbf{q}}(1-f_{c\mathbf{k}})\times$$
$$\times\left[\frac{1}{\omega+\epsilon_{v\mathbf{k}-\mathbf{q}}-\epsilon_{c\mathbf{k}}+i0^+}-\frac{1}{\omega+\epsilon_{c\mathbf{k}}-\epsilon_{v\mathbf{k}-\mathbf{q}}-i0^+}\right]$$

Xo bands (MPI c,v) — **k** momenta (MPI k) — Space variables (MPI g)

```
X_and_IO_CPU= "1 1 2 16 2"
X_and_IO_ROLEs= "q g k c v"
X_and_IO_nCPU_LinAlg_INV= 64
X_Threads= 4
```

- MPI-c,v best memory distribution
- MPI-k as efficient, some memory duplication
- MPI-q may leads to load unbalance and mem duplication
- OpenMP efficient, need extra memory

The YAMBO code: GW (correction) Serf-Energy

$$\Sigma^c_{n\mathbf{k}}(\omega) = \langle n\mathbf{k}|\Sigma^c|n\mathbf{k}\rangle = i\sum_m \int_{BZ} \frac{d\mathbf{q}}{(2\pi)^3} \sum_{\mathbf{G},\mathbf{G}'} \frac{4\pi}{|\mathbf{q}+\mathbf{G}|^2} \rho_{nm}(\mathbf{k},\mathbf{q},\mathbf{G})\rho^*_{nm}(\mathbf{k},\mathbf{q},\mathbf{G}') \times \int d\omega' G^0_{m\mathbf{k}-\mathbf{q}}(\omega-\omega')\epsilon^{-1}_{\mathbf{G}\mathbf{G}'}(\mathbf{q},\omega')$$

QP states (MPI qp)

G bands (MPI b)

q transferred momenta (MPI q)

Space variables (OMP)

```
SE_CPU= "1 4 16"
SE_ROLEs= "q qp b"

SE_Threads= 4
```

- MPI-b best memory distribution
- MPI-qp no communication, memory duplication
- MPI-q may lead to load unbalance and memory duplication
- OpenMP very efficient

The YAMBO code: Linear Response

- X_dielectric_matrix → loop on q
  - X_ALLOC_parallel
    - MATRIX_init
  - X_irredux
    - X_eh_setup →set transitions (k, c, v)
    - X_irredux_residuals → compute residues of X
    - X_GreenF_analytical → compute poles of X
    - PP_redux_wait
  - X_redux → X_par redistribution (BLACS style)

X_par, structure containing:

- X_par%blc
- X_par%blc_d
- …
- X_par%rows
- X_par%cols
  - distributed on g

The YAMBO code: Linear Response, Parallelization Tests

# MPI tasks: 80 **# Thrs / task: 4** # Nodes: 10 # MPI tasks / node: 8 # q: 7 # g: 675 # k: 7 # c: 1956 # v: 44	q	g	k	c	v	**Total time**	**Xo time**
	1	1	1	80	1	09m-01s	440.83s
	1	1	1	40	2	09m-04s	439.98s
	1	1	1	20	4	09m-11s	443.80s
	1	1	1	10	8	10m-50s	535.81s
	1	1	2	40	1	10m-02s	493.27s
	1	1	4	20	1	11m-47s	564.72s
	2	1	1	40	1	08m-51s	431.24s
	4	1	1	20	1	10m-30s	527.58s

GPU-aware programming style

Both **Quantum ESPRESSO** and **YAMBO** are undergoing an important process of cleanup of the source files mostly aimed at:

1. Readability
2. Maintainability

with particular emphasis on the removal of GPU-related code duplication.

Possible solutions could be to:

- use wrappers (deviceXlib)
- use pre-compiler macros
- use GPU-aware external libraries (eg lin-alg)

What is deviceXlib?

- deviceXlib is a library that wraps device-oriented routines and utilities, such as device data allocation, host-device data transfers.
- deviceXlib supports CUDA language, together with OpenACC and OpenMP programming paradigms.
- deviceXlib wraps a subset of functions from Nvidia cuBLAS, Intel oneMKL BLAS and AMD rocBLAS libraries.
- currently working with
  - NVIDIA + CUDA-Fortran
  - NVIDIA + OpenACC
  - AMD + OpenMP-GPU
  - INTEL + OpenMP-GPU

Recent developments

- Repository: https://gitlab.com/max-centre/components/devicexlib
- Recently completely restructured (with the help of G. Rossi, INTEL)
- Tests suite added and bugs fixed
- DeviceXlib version 0.8.3 tagged
- Fully tested on Leonardo and LUMI (among others)

deviceXlib: Structure

```
module devXlib

  use devxlib_malloc
  use devxlib_memcpy
  use devxlib_mapping
  use devxlib_buffers
  use devxlib_linalg
  use devxlib_auxfunc
  use devxlib_async

  implicit none
end module devXlib
```

- devxlib_alloc
- devxlib_free
- devxlib_allocated

- devxlib_memcpy_h2h
- devxlib_memcpy_d2d
- devxlib_memcpy_d2h
- devxlib_memcpy_h2d
- devxlib_memcpy_d2d_p
- devxlib_memcpy_d2h_p
- devxlib_memcpy_h2d_p

- devxlib_map
- devxlib_unmap
- devxlib_mapped

- devxlib_xDOT
- devxlib_xDOT_gpu
- devxlib_xGEMM
- devxlib_xGEMM_gpu

- devxlib_conjg
- devxlib_vec_upd_remap
- devxlib_mat_upd_dMd
- devxlib_mem_addscal

deviceXlib: Pre-compiler Macros

- Pre-compiler macros (DEV_VAR, DEV_SUB, DEV_ATTR) to hide differences across cpu and gpu code, e.g. appending _d or _gpu to variable and subroutine names.

```
#ifdef __DXL_CUDAF
#    define DEV_SUB(x) x##_gpu
#    define DEV_VAR(x) x##_d
#    define DEV_ATTR , device
#elif defined __DXL_OPENACC || defined __DXL_OPENMP_GPU
#    define DEV_SUB(x) x##_gpu
#    define DEV_VAR(x) x
#    define DEV_ATTR
#else
#    define DEV_SUB(x) x
#    define DEV_VAR(x) x
#    define DEV_ATTR
```

include/devxlib_defs.h

deviceXlib: Pre-compiler Macros

```
#if defined __DXL_CUDAF    ||
    defined __DXL_OPENACC  ||
    defined __DXL_OPENMP_GPU
#  define __DXL_HAVE_DEVICE
#endif

/*
! directive sentinels
*/

#if defined __DXL_CUDAF
#  define DEV_CUF $cuf
#else
#  define DEV_CUF !!!!
#endif
```

include/devxlib_macros.h

```
#if defined __DXL_OPENACC
#  define DEV_ACC $acc
#else
#  define DEV_ACC !!!!
#endif


#if defined __DXL_OPENMP_GPU
#  define DEV_OMPGPU $omp
#else
#  define DEV_OMPGPU !!!!
#endif

#if defined __DXL_OPENMP && !defined
(__DXL_HAVE_DEVICE)
#  define DEV_OMP $omp
#else
#  define DEV_OMP !!!!
#endif
```

YAMBO: deviceXlib Wrappers

```
use devxlib, ONLY:devxlib_memcpy_d2d
[...]
do jb=Sx_lower_band,Sx_upper_band
    [...]
    call DEV_SUB(scatter_Bamp)(isc)
    [...]
    if (isc%is(1)/=iscp%is(1)) then
        call DEV_SUB(scatter_Bamp)(iscp)
    else
        ! dev2dev, iscp%rhotw = isc%rhotw
        call devxlib_memcpy_d2d(DEV_VAR(iscp%rhotw),DEV_VAR(isc%rhotw))
    endif

    DP_Sx_l=DEV_SUB(Vstar_dot_VV)(isc%ngrho,DEV_VAR(iscp%rhotw),&
            DEV_VAR(isc%rhotw),DEV_VAR(isc%gamp)(:,1))
    DP_Sx=DP_Sx + DP_Sx_l * const
enddo
```

Macros:
DEV_SUB: append "_gpu" to the name of the subroutine
DEV_VAR: append "_d" to the name of the variable
DEV_ATTR: substitute ", device" when CUDA-Fortran is enabled

YAMBO: single source code

- Single source code
- Simple Fortran data structures to organize cpu and gpu variables

```
subroutine DEV_SUB(scatter_Bamp)(isc)
[...]
complex(SP), pointer DEV_ATTR :: WF_symm_i_p(:,:), WF_symm_o_p(:,:)
complex(SP), pointer DEV_ATTR :: rhotw_p(:)
complex(DP), pointer DEV_ATTR :: rho_tw_rs_p(:)
!
! define pointers to enable CUF kernels
! when compiling using CUDA-Fortran
!
WF_symm_i_p => DEV_VAR(isc%WF_symm_i)
WF_symm_o_p => DEV_VAR(isc%WF_symm_o)
rho_tw_rs_p => DEV_VAR(isc%rho_tw_rs)
rhotw_p => DEV_VAR(isc%rhotw)
[...]
```

Macros:
DEV_SUB: append “_gpu” to the name of the subroutine
DEV_VAR: append “_d” to the name of the variable
DEV_ATTR: substitute “, device” when CUDA-Fortran is enabled

YAMBO: Concurrency of Backends

Macros:
DEV_CUF: substitute “$cuf” when CUDA-Fortran is enabled
DEV_ACC: substitute “$acc” when OpenACC is enabled
DEV_ACC_DEBUG: substitute “$acc” when OpenACC and debug are enabled
DEV_OMPGPU: substitute “$omp” when OpenMP-GPU is enabled
DEV_OMP: substitute “$omp” when OpenMP is enabled

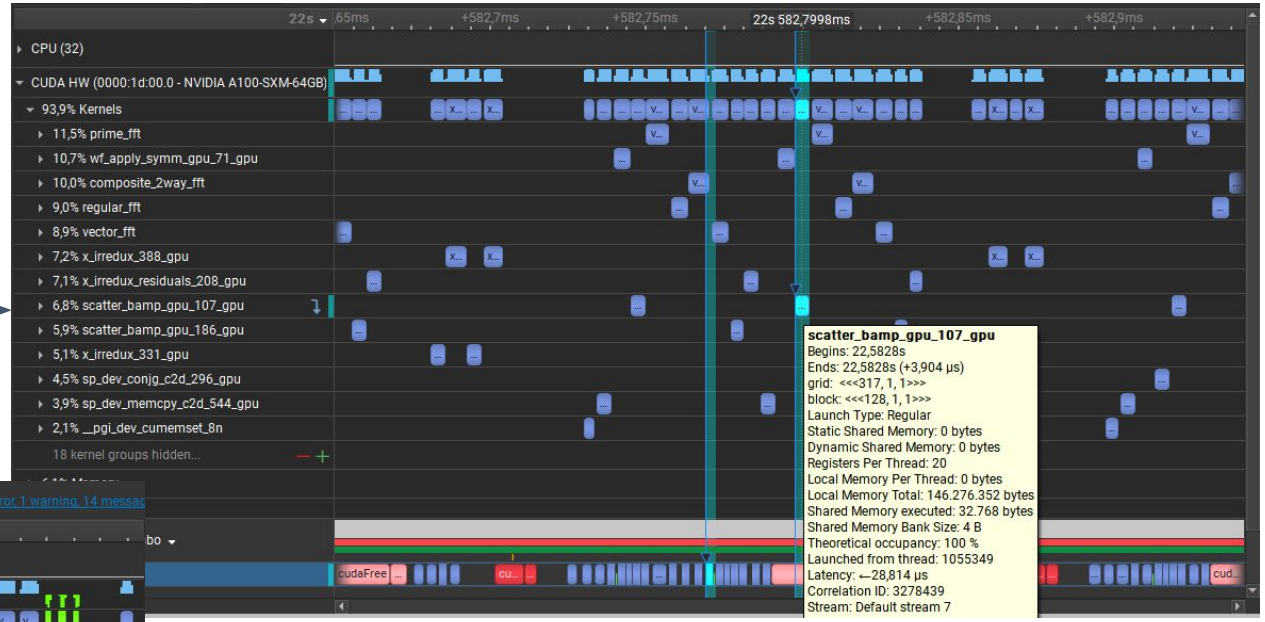
```
complex(SP), pointer DEV_ATTR ::
           WF_symm_i_p(:,:),
           WF_symm_o_p(:,:)
complex(DP), pointer DEV_ATTR ::
           rho_tw_rs_p(:)

WF_symm_i_p =>
           DEV_VAR(isc%WF_symm_i)
WF_symm_o_p =>
           DEV_VAR(isc%WF_symm_o)
rho_tw_rs_p =>
           DEV_VAR(isc%rho_tw_rs)
```

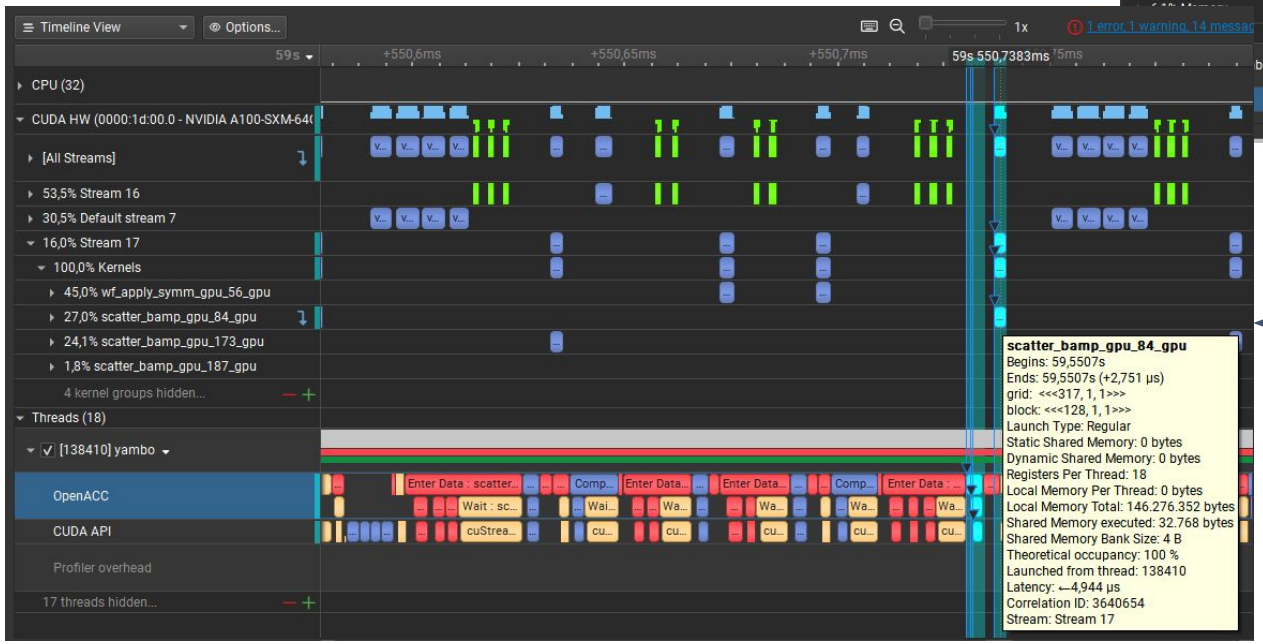
```
!DEV_ACC_DEBUG data present(rho_tw_rs_p,WF_symm_i_p,WF_symm_o_p)
!DEV_ACC parallel loop async
!DEV_CUF kernel do(1) <<<*,*>>>
!DEV_OMPGPU target map(present,alloc:rho_tw_rs_p,WF_symm_i_p,WF_symm_o_p)
!DEV_OMPGPU teams loop
!DEV_OMP parallel default(shared), private(ir)
!DEV_OMP do
do ir = 1, fft_size
  rho_tw_rs_p(ir) = cmplx(conjg(WF_symm_i_p(ir,1))*WF_symm_o_p(ir,1),kind=DP)
enddo
!DEV_OMPGPU end target
!
if (n_spinor==2) then
  !DEV_ACC parallel loop async
  !DEV_CUF kernel do(1) <<<*,*>>>
  !DEV_OMPGPU target map(present,alloc:rho_tw_rs_p,WF_symm_i_p,WF_symm_o_p)
  !DEV_OMPGPU teams loop
  !DEV_OMP do
  do ir = 1, fft_size
    rho_tw_rs_p(ir) = &
        rho_tw_rs_p(ir)+cmplx(conjg(WF_symm_i_p(ir,2))*WF_symm_o_p(ir,2),kind=DP)
  enddo
  !DEV_OMPGPU end target
endif
!DEV_OMP end parallel
!DEV_ACC_DEBUG end data
```

YAMBO: Profiling on Leonardo-Booster

CUDA-Fortran



OpenACC



deviceXlib

	CUDA Fortran	OpenACC	OpenMP

Arrays allocation	`allocate`
	`deallocate`
	`allocated`
	using `device` attribute
		`enter data create`
		`exit data delete`
		`acc_is_present`
		on host memory
			`enter data map`
			`exit data map`
			`omp_target_is_present`
			on host memory

Pointers allocation (GPU only)	`cudaMalloc`
	`cudaFree`
	`associated`
		`acc_malloc`
		`acc_free`
		`associated`
			`omp_target_alloc`
			`omp_target_free`
			`associated`

Memory copies	**Arrays:** direct copy
	Pointers: `cudaMemcpy`
		Arrays: `update device`,
		`update host`,
		`acc parallel loop`
		Pointers:
		`acc_memcpy_to_device`,
		`acc_memcpy_from_device`
		`acc_memcpy_device`
			Arrays: `update to`,
			`update from`,
			`omp target teams loop`

Linear algebra	cuBLAS	cuBLAS	oneMKL rocBLAS

People

Yambo active developers

- Andrea Marini (CNR)
- Myrta Gruening (QUB)
- Daniele Varsano (CNR)
- Conor Hogan (CNR)
- Maurizia Palummo (UNIROMA2)
- Claudio Attaccalite (CNRS, CINAM)
- Davide Sangalli (CNR)
- Elena Cannuccia
- Andrea Ferretti (CNR)
- Alejandro Molina-Sánchez
- Miki Bonacci (PSI)
- Dario Alejandro Leon-Valido
- Fulvio Paleari (CNR)
- Nicola Spallanzani (CNR)
- Ignacio Martin Alliati
- Pino D'Amico (CNR)
- Alberto Guandalini (UNIROMA1)
- Riccardo Reho (UU)
- Giacomo Sesti

deviceXlib developers

- Andrea Ferretti (CNR)
- Giacomo Rossi (Intel)
- Nicola Spallanzani (CNR)
- Davide Sangalli (CNR)

Contributors

- Pietro Bonfà (UNIPR)
- Pietro Delugas (SISSA)
- Laura Bellentani (CINECA)
- Filip Vaverka (IT4I)

Thank you for your attention!

MaX DRIVING THE EXASCALE TRANSITION

JOIN THE COMMUNITY NOW!

Follow us on:

company/max-centre/

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

@max_center2

youtube/channel/MaX Centre eXascale

Funded by the European Union

MaX "*Materials Design at the Exascale*" has received funding from the European Union under grant agreement no. 101093374.

EuroHPC Joint Undertaking

The project is supported by the Euro HPC Joint Undertaking and its members.