Ref. Ares(2022)3534391 - 09/05/2022

D1.5

Third release of MaX software: Final report on restructuring, exascale readiness and inter-code libraries

Stefano Baroni, Ivan Carnimeo, Augustin Degomme, Pietro Delugas, Stefano de Gironcoli, Andrea Marini, Davide Sangalli, Daniele Varsano, Fabrizio Ferrari Ruffino, Andrea Ferretti, Alberto Garcia, Luigi Genovese, Paolo Giannozzi, Anton Kozhevnikov, Ivan Marri, Nicola Spallanzani, and Daniel Wortmann

Due date of deliverable	31/01/2022 (**month 38**)
Actual submission date	09/05/2022
Final version	09/05/2022
Lead beneficiary	SISSA (participant number 2)
Dissemination level	PU - Public

Document information

Project acronym	MAX
Project full title	Materials Design at the Exascale
Research Action Project type	European Centre of Excellence in materials modeling, simulations and design
EC Grant agreement no.	824143
Project starting/end date	01/12/2018 (month 1) / 31/05/2022 (month 42)
Website	www.max-centre.eu
Deliverable no.	D1.5
Authors	Stefano Baroni, Ivan Carnimeo, Augustin Degomme, Pietro Delugas, Stefano de Gironcoli, Andrea Marini, Davide Sangalli, Daniele Varsano, Fabrizio Ferrari Ruffino, Andrea Ferretti, Alberto Garcia, Luigi Genovese, Paolo Giannozzi, Anton Kozhevnikov, Ivan Marri, Nicola Spallanzani, and Daniel Wortmann.
To be cited as	Baroni et al. (2022): Third release of MAX software: Final report on restructuring, exascale readiness and inter-code libraries. Deliverable D1.5 of the H2020 CoE MaX (final version as of 09/05/2022). EC grant agreement no: 824143, SISSA, Trieste, Italy.

Disclaimer

This document's contents are not intended to replace consultation of any applicable legal sources or the necessary advice of a legal expert, where appropriate. All information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user, therefore, uses the information at its sole risk and liability. For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the authors' view.

Contents

1 Introduction **5**

2 Work on the different codes **6**
- 2.1 QUANTUM ESPRESSO . . . 6
  - 2.1.1 Release Summary . . . 6
  - 2.1.2 Restructuring in MAX-phase2 . . . 7
  - 2.1.3 Exascale readiness . . . 9
- 2.2 YAMBO . . . 10
  - 2.2.1 YAMBO v5.1: Release Summary . . . 10
  - 2.2.2 Code refactoring and modularisation in MAX phase2 . . . 13
  - 2.2.3 Software engineering procedures . . . 15
  - 2.2.4 Exascale Readiness . . . 16
- 2.3 Siesta . . . 18
  - 2.3.1 Global achievements in MAX-phase2 . . . 18
- 2.4 BigDFT . . . 21
- 2.5 FLEUR . . . 24
  - 2.5.1 Hybrid functionals in LAPW (LapwLIB) . . . 24
  - 2.5.2 Build process in view of external dependencies . . . 24
- 2.6 CP2K . . . 25

3 Libraries **26**
- 3.1 SpFFT . . . 26
- 3.2 SPLA . . . 29
- 3.3 SIRIUS . . . 29
- 3.4 FUTILE library . . . 31
- 3.5 PSolver . . . 32
- 3.6 LAXlib . . . 32
- 3.7 FFTXlib . . . 32
- 3.8 KS_Solvers . . . 32
- 3.9 XClib . . . 33
- 3.10 UPFlib . . . 33
- 3.11 xsdtool, qe_h5, and UtilXlib . . . 34
- 3.12 omm-bundle . . . 34
- 3.13 xmlf90 . . . 34
- 3.14 LibFDF . . . 35
- 3.15 libPSML . . . 35
- 3.16 libGridXC . . . 36

4 Conclusions and ongoing work **37**

Acronyms **37**

References **39**

Executive Summary

This report summarizes the restructuring tasks carried out in WP1 in order to prepare MAX codes for the forthcoming pre- and exa-scale HPC platforms.

The first milestone of this endeavour was reached at the preparation of the Software Development Plan [1], where we identified the code functionalities that had to be modularised, the data structures to be encapsulated within each module, and the APIs needed for accessing these data and functionalities. Leveraging this intra-code design work, we were also able to identify modules that were feasible and profitable to recast as standalone libraries, eventually redesigning them for an autonomous extra-code reuse.

The work on modularization was practically accomplished by the second year of the project, when the flagship codes had already acquired their current structure, and most of the planned libraries had reached the production or final testing phase. While we had to operate further adjustments in the code structures, D1.3 [2] and D1.4 [3] can be considered the second milestone of this Work Package.

The last milestone of WP1 deals with making the software architecture of MAX flagship codes robust and resilient against a hardware evolution scenario where multiple and diverse HPC hardware are expected emerge and be available. The technical details of the developments –together with the new features in latest releases– are reported in the code-specific sections of this document. These sections show that, during the last year, there has been a significant effort for improving the support of heterogeneous computing, working at the offload of kernels and data-structures on GPGPUs, with a renovated attention to avoiding or removing the usage of instructions sets that were too specific to the CUDA programming model. Thanks to the previous restructuring work, it was possible to target and localise most of these activities to the computationally relevant kernels. For some actions it was instead necessary to introduce offloading instructions in the science-specific layers of the codes. In these cases, directive-based programming models (such as `openACC` or `openMP5`) are progressively replacing platform or compiler specific solutions (e.g. CUDA-Fortran). To this purpose, the `DevXlib` library, designed and developed within MAX , provides an API and macros abstraction layer to manage different programming models.

While most of the earlier development on this side was tested only on `CUDA` cards, in this last year MAX code developers have started the experimentation of the other accelerator cards, that will be used in the forthcoming pre- and (possibly) exascale machines. The work on this side progresses rapidly, again thanks to the already achieved internal reorganisation of the codes. In conclusion, this further confirms that the restructuring of the MAX flagship codes has successfully prepared them for a fast adaptation to the forthcoming pre-exascale, exascale, and even post-exascale HPC technologies.

1 Introduction

One of the main targets of MAX-phase-2 is to provide the community with electronic structures codes that can be promptly ported on new HPC systems, and whose components can be easily reassembled to implement new algorithms.

In view of this, WP1 has planned and coordinated the modularisation of the MAX flagship codes. It has also identified a set of components that could be refactored in completely standalone libraries, that MAX will eventually distribute as a library bundle. This restructuring activity is integrated with the other code actions included in WP2 and WP3. The work in these three WPs is mainly carried through by the same code developer teams, in close interaction with the HPC specialists involved in WP4. At several stages, significant feedback and requests come from the other field scientists active in WP5 and WP6.

The first step in the work of WP1 was the preparation of the code restructuring plan (software development plan, SDP) and the identification of the libraries that should be extracted [1]. We documented the progress in the code restructuring in two previous release reports [4, 3]. At the end of M24 all the codes had almost reached the desired structure, fulfilling most of the objectives of the plan. In the last year the development work has been oriented toward consolidating the modularisation, improving and extending the code portability, and introducing the algorithmic improvements developed in WP3 into the production versions.

For what concerns the preparation of the libraries, their development started just after their identification in the SDP. The progress and steps in these developments depended from their starting status. In all cases by M18 we were able to reach at least a *proof of concept* version where the implementation code was collected in self-contained units and provided with a first API prototype. These earlier versions were then progressively improved by completing the data encapsulation and the interfaces, passing a first phase as experimental standalone versions (*beta* stage), ideally followed by the release of the *production* versions.

Many of the identified libraries are related to the most computationally intensive kernels, together with basic and low level functionalities. In some cases they directly implement the kernels, while in others they provide a common transparent interface to the (domain specific or general purpose) low level libraries that perform these functionalities. The MAX libraries have been one of the key instruments during these three years for the improvement of the performance and portability of our codes.

Together with the use of domain specific libraries, in particular for the plane-wave codes, the GPU porting made it necessary to include offloading specific code also in the highest code layers (those closer to the scientific developers). This has caused a few issues in the organisation of the codes and their data structures. For instance, for some codes GPU ready versions initially used a separate code base, which was then merged with the main code base only after mitigating and solving software engineering clashes. The adoption of the low-level `DevXlib` library has been instrumental in overcoming these issues, e.g., in YAMBO and QUANTUM ESPRESSO.

In the following Sections we will first present the specific restructuring of each flagship code. Then, in Section 3 we will present the status of the libraries of the MAX bundle. We will summarise the status of the flagship codes, their pre- and exascale readiness, and the ongoing and future work in the conclusive section.

2 Work on the different codes

2.1 QUANTUM ESPRESSO

In the M24-M36 period, two stable versions, `qe-6.8` and `qe-7.0`, have been released. They contain most of the refactoring and modularization actions outlined for WP1 [1] and some of the algorithmic improvements produced by the work in WP3 [2]. In practice, most of the work done in this year of activity focused on the code portability to GPGPU-based heterogeneous architectures (also as a result of expected deployment of peta and pre-exascale machines of EuroHPC). The CUDA-Fortran code QUANTUM ESPRESSO was merged into the main repository in `qe-6.8`. In `qe-7.0`, many of high-level code sections have been refactored with an `openACC` management of the accelerated code regions. This represent a first milestone in the transition of the science-specific code layers to `openACC` and `openMP-5` (in the next releases) for offloading management.

The adoption of the directive-based offloading avoids code and variables duplication, improves the clarity and maintainability of the code and reduces the work needed for porting to GPUs further applications of the QUANTUM ESPRESSO suite. This new approach has already been extensively used in `qe-7.0` for the finalisation of the GPU version of `cp.x` and in the development version for the ongoing acceleration of the linear-response codes.

The possibility to adopt different offloading strategies for the different code regions is one great advantage provided by the work on modularisation done in previous releases. In the following we will first briefly present the main changes introduced in the last two QUANTUM ESPRESSO releases, then summarise the work done for the modularization and data encapsulation in the last three years, and conclude the section with an overview of the ongoing and perspective work.

2.1.1 Release Summary

The QUANTUM ESPRESSO release contain numerous contributions from a world-wide community of scientists and developers.

- New Features in `qe-7.0` and `qe-6.8`
  - `cp.x` version for GPU completed and made more efficient and performing.
  - The new solver added to `KS_Solver` library based on the RMM-DIIS algorithm [5] improves performance and scalability in systems with a large number of bands.
  - Interface with the `TRIQS` package, for charge self-consistent DFT+DMFT calculations via `Wannier90`, added.
  - `projwfc.x` can be used to compute the PDOS in a local basis.
  - `DFT-D3` dispersion correction parallelized with MPI and openACC.
  - Many-Body Dispersion (MBD) correction library.
  - New `turboMagnon` code added to the `TDDFPT` module.
  - New Bethe-Salpeter iterative solver using Direct Screened Interaction method [6]
  - Calculation of DORI and of ELF for spin-polarized systems.

- Grand-Canonical SCF [7, 8] for constant chemical potential.
- Calculation of spin-current matrix elements [9] for spin Hall conductivity using Wannier interpolation, in `pw2wannier.x`

• Important refactoring in `qe-7.0` and `qe-6.8`

- Merger of CUDA-Fortran code into the main trunk.
- Extension of the modularization and encapsulation of pseudopotential related functionalities into the `upflib` library.
- `openACC` offloading enabled in `qe-7.0` and completion of the first step of the transition of `PW` and `CPV` code bases to the directive-based model.
- Offloading of the exchange-correlation routines in `XClib` completely refactored from `CUDA-Fortran` to `openACC`

2.1.2 Restructuring in MAX-phase2

The restructuring of QUANTUM ESPRESSO in these three years was outlined in the Software Development plan [1]. The goal was to refactor its building blocks in a modular way, providing them with well-encapsulated data structures and complete APIs sets. The plan also identified [1] a set of functionalities that could be cast as stand-alone libraries to be included in the MAX library bundle.

We started the modularisation with the two mathematical libraries `LAXlib` and `FFTXlib` that execute the most compute-intensive parts of the calculations and also manage the MPI distribution of the largest data structures: 3D-FFT grids (`FFTXlib`), and large dense matrices (`LAXlib`). We also created a lower-level library called `UtilXlib` that collects the interfaces and data structures for the MPI management, error handling, timing and profiling. The lower-level utility-layer was completed later with the adoption of the `DevXlib` library for the management of accelerator devices. This lower layer is instrumental to the complete disentanglement of the libraries and to their portability to different architectures.

A second part of the work involved the electronic structure field-specific layers that implemented the algorithms used in quantum-engines and property calculators. Following up the work already done in previous years we completed the encapsulation of a lower layer of electronic structure domain specific functionalities of common use to QUANTUM ESPRESSO applications. These include

• `KS_Solvers`, that contains the implementation of several iterative diagonalization algorithms;

• `Modules`, that contains general interfaces and data-structures used in most of the applications of the suite;

• `LR_Modules`, that contains interfaces and data structures specific of DFPT applications.

Out of these three layers, only `KS_Solver` has been refactored as a standalone library and included in the MAX library bundle. The other two layers instead provide the largest part of the common code used by all the quantum-engines and property calculators

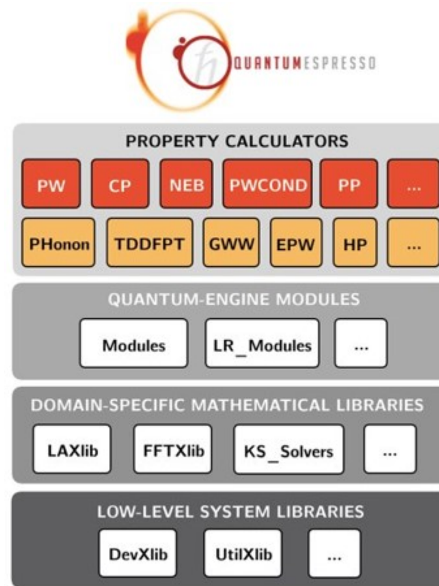

Figure 1: Code layers in QUANTUM ESPRESSO. The **high-level code** is constituted by property calculators and quantum engines. The **quantum-engine modules** provide all those electronic structure specific functionalities frequently used in the applications. The **mathematical libraries** provide access to compute-intensive functionalities. **Low-level utility libraries** provide APIs for the basic management of data, parallelism, timing and errors.

of the suite. We have chosen to target their development and maintenance towards the general usage within the QUANTUM ESPRESSO platform.

Some parts of `Modules` of more general interest, have though, been extracted and refactored as a standalone libraries. The pseudo-potential related functionalities have been moved to the `upflib` library developed in collaboration with the YAMBO group. Also, the routines that compute the exchange-correlation functionals and their derivatives have been moved to the new `XC_lib` library.

The organisation of the code has thus reached its final aspect at the end of the second year. Its layered structure has been already described in [3]. We review here that description for completeness.

Code structure. We have 4 different layers in the code as depicted in Fig. 1. The application layer is constituted by partially independent "property calculators":

- `PW`, containing the `pw.x` quantum engine for self consistency, plus a set of related property calculators;
- `CPV`, containing the `cp.x` quantum engine for performing Car-Parrinello ab initio molecular dynamics;
- `PP`, containing a wide range of post-processing applications;
- `atomic`, providing an all-electron solver for atomic problems, plus utilities for generating and testing pseudopotentials and PAW datasets;

- PHonon, providing applications for vibrational and dielectric properties of solids using DFPT;
- TDDFPT, for computation of electronic excitation spectra using TDDFPT;
- HP, computing on-site and inter-site Hubbard correction terms to DFT using DFPT.
- EPW, computing electron-phonon interactions coefficients and related properties.

The application layer depends upon lower-level libraries and interfaces layers:

- A few domain-specific algorithmic modules:
  - Modules, a general interface module containing many common functionality calls;
  - LR_modules, containing common functionalities for linear-response codes;
  - UPFlib, described below in section 3.
  - XC_lib, described below in section 3.
- Computation-intensive mathematical Kernels KS_solvers, FFTXlib, LAXlib.
- UtilXlib: a low-level general utility library for environment initialization, timing, logging and error handling.

The automated build system may be configured either using autoconf or the CMake platform. The usage of external libraries has been particularly streamlined with the use of git submodules.

2.1.3 Exascale readiness

The version qe-7.0 has been tested in many of the relevant platforms.

For what concerns homogeneous nodes, the code relies on the efficiency of the software stack. Apart from the Fortran compiler few other libraries have in fact significant impact on the performance. The MPI, FFT and BLAS libraries plus Scalapack or ELPA. In the last version the impact of the latter ones has been mitigated by the introduction of the RMM-DIIS iterative solver.

Among heterogeneous CPU + GPGPU systems, the QUANTUM ESPRESSO production version supports those equipped with NVIDIA GPU cards. The support of the other cards (AMD or Intel) is still at a prototype levels. We expect that the support of the directive-based offloading will be soon improved by all cards and compiler vendors. This will solve the portability issues for the higher level layers of the code. For what concerns the mathematical libraries, several projects are currently ongoing. In the currently running (Jan, 24 to Feb 20) MAX virtual hackathon on AMD cards we are testing the use of SPfft and a LAXlib interface for the ROCSolver and MAGMA libraries.

The overwhelming memory footprint of calculations with very large number of bands constitutes a major general issue for the exascale readiness of QUANTUM ESPRESSO. More work on this side is needed to improve the MPI band parallelization, implementing the data distribution among the band groups.

2.2 YAMBO

During the M24-M36 period (Dec 2020 – Nov 2021), YAMBO v5.0 was released (2nd of February 2021) and subsequently stabilised with a few updates (from v5.0.1 to v5.0.4). Most of the changes introduced with version 5.0 were discussed in the previous reports. Since then, the development of the code mainly followed two directions.

(i) A new release, YAMBO 5.1, has been developed and prepared (including new features, further refining the code structure and modularisation, and improving performance and memory footprint). It will be officially made public after the MAX -YAMBO school of April 2022.

(ii) In parallel, a great part of the work has been focused on further developing the YAMBO support of GPGPU-based heterogeneous architectures (extending and optimising the support of NVIDIA GPUs, but also addressing AMD and INTEL cards), in view of the deployment of peta, pre-exascale, and exascale machines of EuroHPC. The GPU porting has involved a significant restructuring of the code, and has been done exploiting the `deviceXlib` library and sharing the experience and success with other MAX codes such as QUANTUM ESPRESSO. It is, at present, in a dedicated branch (`devel-gpu`) not yet ready to enter the YAMBO 5.1 release. We plan to have it integrated in the v5.2 release (end of 2022 / beginning 2023).

In the following we will first briefly present the main actions and achievements of the YAMBO code, accomplished during the M24-M36 period (Dec 2020 – Nov 2021) and report in details the release notes of YAMBO 5.1 (see Sec. 2.2.1). Then we will shortly summarise the work done in the last three years (MAX phase-2). YAMBO was born and developed for a long time as a "tool for research". While the MAX phase-1 project has significantly improved performances and scalability of the code, mostly focusing on many-core homogeneous machines, MAX phase-2 has forced YAMBO to shift its developing model (for what concerns both internal procedures and software engineering aspects), turning more and more YAMBO into a "massively parallel community code for materials science applications". All of this while facing important technical developments such as, e.g., the introduction of a comprehensive support for GPUs.

To explain these changes, we fist summarise the work done for the modularisation and refactoring of the code (see Sec. 2.2.2). Then we also present some procedures adopted and needed to make this transition possible (see Sec. 2.2.3). Finally we will discuss the developments present in the `devel-gpu` branch, also including ongoing and future developments, and sharing our perspective on the co-design activities carried out in MAX phase 2 (see Sec. 2.2.4).

2.2.1 YAMBO v5.1: Release Summary

The main developments (new features, improvements, optimisations) to be released with YAMBO v5.1 are shortly summarised in the following, for different code run-levels.

Coulomb cutoff, screening and dipoles

- Added new Coulomb cutoff technique for for lower dimensional materials.

- Improved handling of anisotropy. For the $\mathbf{q} \to 0$ limit, now an average over three Cartesian direction can be selected in input.
- YAMBO can now compute screening without SOC and later use it in a calculation with SOC.
- Added option to project spin dipoles in valence and conduction band channel only, to study independently the spin dynamics of electrons and holes.
- New scheme for dipoles via k-space derivatives, alternative to the already implemented covariant dipoles.

TDDFT

- Added possibility to lower cutoff on f_{xc} also in eh space. Useful for comparison with G-space simulations.
- Defined `F_xc_mat` for magnons.
- Added support to hybrid functionals.
- Implemented computation of f_{xc} via finite differences functional derivatives of v_{xc}. This approach makes possible to construct f_{xc} within GGA.

GW

- `W-average`: full implementation of the W-average method for 2D system added in the claculation of the GW correlation self-energy. This method, based on a physically-driven interpolation of W joint with a MonteCarlo sampling, allows for a significant convergence acceleration wrt **k**- and **q**-points.
- `GW-MPA` (multi-pole approximation) treatment of the frequency convolution needed to evaluate the self-energy at the GW level.

BSE

- Reorganisation of the BSE subroutines: `K_Transitions_setup` split into two subroutines, `K_dipoles` and `K_IP_sort` created, created `K_restart` file to handle restart, `K_multiply_by_V` and `K_driver` split into to subroutines.
- Inversion of `qindx_B` indexes (performance), distribution in memory and parallel I/O (see also section on IO).
- Added support for double grid with Haydock solver.
- `W-average`: a tentative implementation of the W-average method (at the moment for 2D systems) has been added to the BSE kernel construction.

Self-consistent and real-time modules: `yambo_sc`, `yambo_rt` & `yambo_nl`.

- Two independent chemical potentials can be added to model the non-equilibrium excitonic insulator via `yambo_sc`.
- Added possibility to perform simulations with two external fields in `yambo_nl` to model transient absorption experiments.

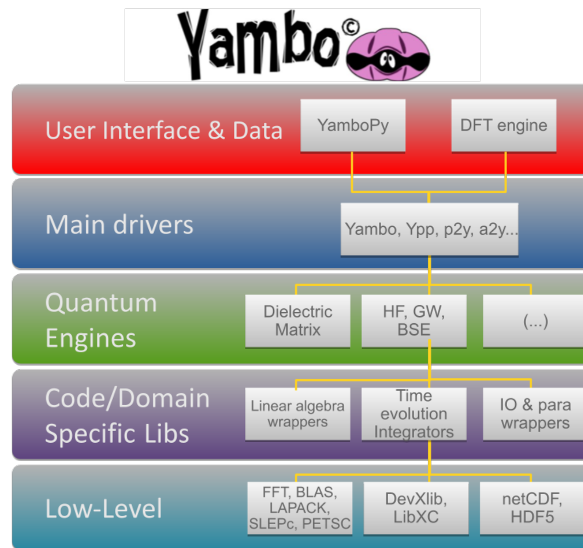

Figure 2: Code layers in YAMBO. The code can be driven directly or by using the `yambopy` python layer. The high-level code is constituted by main drives that use the `Ydriver` library. These drivers call the second–level drivers that are quantum engines that perform property calculations. These interact via the modular structure described later with the domain specific and low–level libraries.

- Ehrenfest dynamics. First experimental implementation available via `qe_pseudo` library.
- Coded calculation of ARPES spectral function starting from GKBA reconstruction of $G^{<}(t,t')$ from density matrix via `ypp_rt`.
- Added calculation of field envelop and extraction of Rabi coupling in `yambo_rt`.
- Transient Absorption via `ypp_rt` restored and greatly improved.
- Improved handling of phenomenological dephasing in degenerate subspaces in `yambo_rt`.

Electron-phonon in `yambo_ph`

- Double grid for exciton-phonon self-energy.
- Electron-phonon self-energy now works with irreducible and expanded electron phonon matrix elements $g_{kk'}$.
- Possibility to plot diagonal elements of the $g_{kk'}$.
- Coded ph-el self-energy and added calculation of equilibrium phonon linewidths using double grids, including support for reading bare el-ph matrix elements.
- Added calculation of phonon spectral functions via frequency-dependent ph-el self-energy.

2.2.2 Code refactoring and modularisation in MAX phase2

Overall, during the phase 2 of MAX , the modularization activity has moved along the following lines: code refactoring, extraction of code procedures as libraries, replacement of internal procedure with external, optimised libraries. As already stressed, modalularization plays also a crucial role in the optimisation of the code for parallel environment, and in the support of GPU accelerators in particular. Indeed as many layers of GPU-aware programming models (CUDA, OpenACC, OpenMP) are added to the source code, a clean, ordered, and minimal structure avoids bugs, allows for readability of the sources, and, in turn, enables to the scientific community to stay engaged with the development.

From a software point of view, the current structure of YAMBO, emerging after the development activities of MAX phase 2, is described in Fig. 2. Notably, there is an outer layer meant to interface with other codes (e.g. QUANTUM ESPRESSO) or for post-processing and automation (YamboPy). Next one finds the highest level drivers and quantum engines focusing on the scientific property calculators (e.g. dielectric function, self-energies, BSE). These layers are followed by internal tools and domain specific libraries on the one side, and by mathematical and performance libraries on the other. Hardware-specific implementations, if any, are kept at the lowest levels, and possibly hidden in libraries (see e.g. multiple GPU-aware programming models wrapped by deviceXlib). Remarkably, this is an actual implementation of the separation of concerns concept, where one wants to expose scientific developers with as few HW-specific aspects as possible.

The main software-architecture actions (modularization, internal and external libraries, interfaces to DFT codes, YAMBO python-layer) undertaken since M24 (Oct 2020) are summarised in the following.

Modularization.

- **Modularization of the I/O subroutines**. The `io_control` and `io_connect` subroutines have been extracted from the global module and the same strategy has been applied to the `mod_IO_interfaces` module.

- YAMBO adopts several **internal tables** to store the indices and maps for later reuse. However, the price to pay is that the size of the tables can become very large in some cases, such as in the presence of many **k**-points. In this release, we worked out the case of the `qindx_B` variable, that can be now distributed in memory taking advantage of `HDF5` parallel I/O. While computing the table, the `HDF5` file is used as a buffer to check the values to be computed.

- **Modularisation of the Bethe–Salpeter subroutines**. Several long routines have been split in thematic subroutines, re-written, and cleaned. A list includes: `K_Transitions_setup`, `K_multiply_by_V`, and `K_driver_split`.

- **Work variables**. In order to modularise the large number of work variables used in specific sections of the code we have introduced specific headers that are in common to different sections of the code:

```
DEFINE_BSK_COMMON_INDEXES
FILL_BSK_COMMON_INDEXES
FILL_BSK_KERNEL_INDEXES
FILL_BSK_CORR_INDEXES
```

- **I/O compression**. The size of the excitonic Hamiltonian can become exceedingly large when dealing with nanostructures and/or surfaces. In the latest YAMBO release we have implemented the HDF5 compression of the excitonic matrix database.
- **Parallel I/O**. A parallel I/O support has been coded for the Hartee plus Exchange–Correlation integrals databases used in real–time simulations.
- **New I/O interfaces**. The use of new I/O interfaces has been extended in different sections of the code (def_variable and io_variable).

Internal libraries. At M24, stable, well-tested and portable sections of the code have been extracted as standalone libraries. In order to simplify the user access to these libraries we have moved all of them to a static part of the dedicated YAMBO git repository. At the same time we have activated git large-file-support (LFS) on the repository to get a fixed link for the files. This has lifted the need for users to clone the libraries source.

External libraries. YAMBO depends on several external libraries. Each of these libraries is paralleled by a set of dedicated internal structures (modules, interfaces) of the code. Concerning the most recent developments:

- The devicexlib support has been upgraded to deal with an external library contained either in a git repository or in a archive file.
- Finally YAMBO is now able to read and re-construct all the information about pseudo-potentials via the QE_pseudo library.
- The libxc library support has been updated from version 2.2.3 to version 5.1.5. All new libxc interfaces have been adopted and now YAMBO is able to link recent external libxc libraries.
- For testing purposes an interface to the parallel netcdf library has been added that can be detected by the internal configure script.
- Internally downloaded Slepc and Petsc libraries updated to version 3.14.2 and 3.14.6, with adoption of python3 for automatic configuration.
- Automatic configuration of MKL Intel libraries coded.

Interfaces with ground–state codes: p2y. The p2y interface has been further developed to keep in sync with the latest releases of QUANTUM ESPRESSO (QE). In particular, YAMBO is properly interfaced with qe-7.0, while keeping back compatibility with previous versions. Additional data such as atomic projections (via the QE xml file atomic_proj.xml) have also been included in the interface support and maintained.

Yambopy. Yambopy is a python layer providing additional functionalities for both QUANTUM ESPRESSO and YAMBO. In particular, it provides:

- Easy pre- and post-processing of the simulation data, including hard-to-get, database-encoded xml and netCDF data.

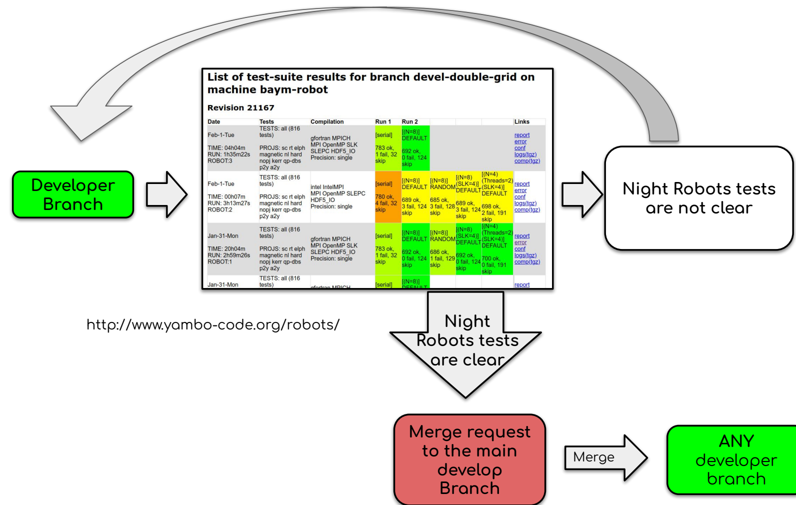

Figure 3: YAMBO merging procedure: the proposed developer branch has to give no errors in night runs of the of the `test-suite` in several workstations before being eligible to be merged. In case of failures fixes and amendments are required before repeating the testing procedure.

- Many visualization and plotting options.
- Input file parsing and generation.
- Abilty to run QE and YAMBO executables both on local machines and HPC facilities.
- Command-line options for simple tasks.
- Simple automatization workflows (e.g., convergence tests).
- Hackability: user-specific functionalities can easily be added into the code.

In the MaX context, Yambopy is located in between the YAMBO main executables and `AiiDA`. The `yambopy` project is composed of a gpl repository freely downloadable from the main YAMBO GIT repository, several tutorial sections on the YAMBO wiki webpage and a private developmental version. The long-term goal is to give the users full control over the data produced by the simulations, as well as more flexibility in their workflows. The newly released version 0.2.0 of the code is updated and maintained for the latest versions of QE and YAMBO and is capable to read and manage almost all relevant `netCDF` databases produced by YAMBO, as well as the main `xml` outputs from QE.

2.2.3 Software engineering procedures

Due to the large increase of YAMBO coding branches (currently 114 active branches on the GIT official repository), we have introduced a coordinated validation and merging procedure between the development and the stable/community branches. A key role in the merging procedure is played by the YAMBO `test-suite` that collects about 4000 tests covering most of the code features. The tests contained in the `test-suite` are run every night on several workstations by an automatised procedure and the results obtained from the branch under testing are automatically uploaded on a public web-page.

The test-suite is also hosted on a dedicated git repository and is equipped with several branches. The merging procedure is therefore based on the following rules:

- The develop branch is protected against direct writing, meaning that the only option to modify it is via merge requests. By default the develop branch is test–suite green (error free), as is the target of a merge request from a branch that has successfully passed all the tests of the test-suite.
- Each development branch (here named devel-feature) that aims at being merged with the develop branch has to create a dedicated test-suite branch.
- The workstations (robots) run night instances of the test-suite on the combination devel-feature branch of YAMBO on the tests of the devel-feature branch of the test-suite. When the branch is green, i.e. gives no errors on all reference workstations, then the devel-feature branch can be considered a merge candidate.
- The developer opens a **pull request** and the source is reviewed by other expert developers. When the review becomes positive (perhaps after a few recommendations are met) the devel-feature code and test-suite are merged with the main references.

The above procedure is schematically represented in Fig. 3.

2.2.4 Exascale Readiness

As already mentioned in the introduction part of Sec. 2.2, the phase-2 of MAX sees the YAMBO developers mainly focused on restructuring the code with new development strategies, mostly and notably focused on GPU-programming models. In fact, one of the most important aspects in terms of the exascale readiness and portability is the one related to the support for GPU acceleration (not limited to NVIDIA cards, but also addressing AMD and INTEL GPUs).

For this purpose, we first worked on a proof-of-concept (that we named gpu-multi) that helped us understand how to have multiple GPU-oriented programming models to living together in the same code without making is too complicated, disrupted, or extensively replicated. Briefly, the reason for this work is the need to face the lack of a standard and the consequent birth of more programming models for GPU offloading. In the deliverable D4.6 we have thoroughly investigated both the motivation and the implementation of this proof of concept.

The experience gained in the development of gpu-multi was then brought to the YAMBO code within the devel-gpu branch. The first programming model ported is the directive-based OpenACC, which goes alongside the already present CUDA-Fortran. To avoid code duplication we made an intense use of preprocessor macros that activate the language chosen at compile time. A detailed explanation of the adopted implementation is present in the deliverable D2.3.

With the purpose of continuing the porting of the second GPU-aware backend (OpenACC) and with the idea of favouring the support of even alternative programming models (such as OpenMP GPU), we decided to collect all the most common routines involving GPU offloading and handling in a library named deviceXlib. This idea, shared

with the QUANTUM ESPRESSO development team, was a natural step forward considering the software-architecture work done during MaX phase-2, largely focused on code modularization and wrapping. The library is developed in a collaboration between the QUANTUM ESPRESSO and YAMBO developers. YAMBO already makes use of some routines of the library in the release that is going to be released and a complete support to the library is expected from the release 5.2. The details of the implementation of `deviceXlib` (notably already targetting CUDA-Fortran, OpenACC, OpenMP programming models, for NVIDIA, AMD, and INTEL GPUs) are discussed thoroughly in the deliverable D2.3.

2.3 Siesta

In the past year, the Siesta program has seen substantial enhancements. Work on modularisation, and on interface refactoring to enable further modularisation, has continued since M24, and a number of significant new functionalities have been implemented. We can summarise the enhancements in the following list:

- Implementation of a basis-contraction scheme. (See more details in WP3 report.)
- A first implementation of support for exact exchange. (See more details in WP3 report.)
- Improvements to the automatic basis-set generation capabilities.

  This feature, together with the use of PSML pseudopotentials, facilitates the use of Siesta in high-throughput scenarios.
- Further refactoring of the OMM (Orbital Minimisation Method) linear-scaling capability, using the sparse matrix-matrix multiplication library DBCSR and the general matrix handling layer library `MatrixSwitch`. This is one of the actions featured in WP3, and can have significant impact on the demonstrators of WP6 that deal with systems with an energy gap in the electronic structure.
- Addition of the DFT-D3 scheme for approximate dispersion corrections.
- Further enhancements to the resilience and robustness of the code.

2.3.1 Global achievements in MAX-phase2

Modularisation. As discussed in the software-development plan, Siesta has benefit from a significant modularisation effort (see Fig. 4).

New key functionalities.

- The ELSI library brings in a new collection of solvers and significant performance enhancements.

  Siesta was already able to use the ELPA, OMM, and PEXSI solvers with its own ad-hoc interfaces. Now these are available through ELSI *with a common interface*, and the code is also able to exploit the $O(N)$ density-matrix purification algorithms supported by the NTPoly solver, the new EIGENEXA solver, and a few more. With the common interface in place, any additions and enhancements to the supported solvers can be used with almost no code changes. This has been particularly important for the performance enhancement of the code, including GPU acceleration, as detailed in the reports for WP2 and WP4.
- Support for modern multi-projector norm-conserving pseudopotentials through the PSML format and library. This allows users of the code to employ curated pseudopotentials from databases such as Pseudo-Dojo (https://www.pseudo-dojo.org), and is particularly important in the context of calculations with spin-orbit coupling.
- Incorporation of the PSolver library. This provides the important capability of performing simulations without imposing periodic boundary conditions.

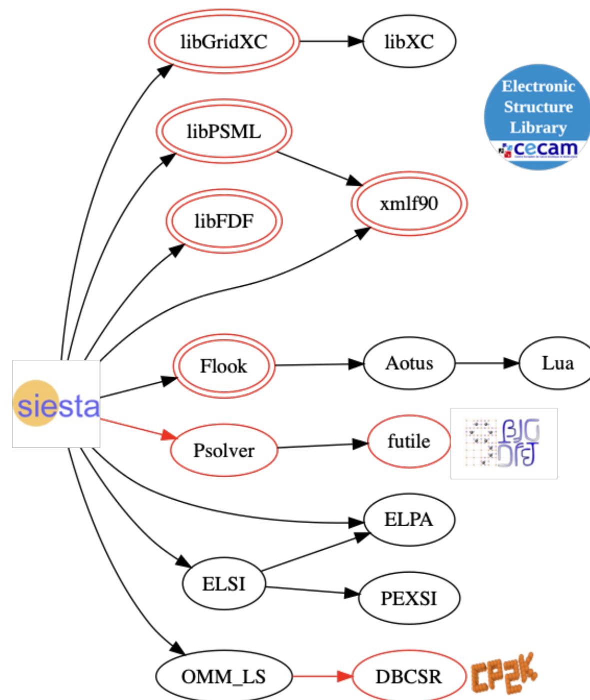

Figure 4: Sketch of the modularisation of the Siesta program. Each bubble represents a library. Dependencies are indicated by arrows, and logos at the end of a path show the MAX code in which the libraries originate. Bubbles with double contours mark those libraries originating in Siesta, and made available also through the Electronic Structure Library

Building system. The now very large number of potential dependencies of the code calls for enhancing the flexibility and robustness of the building and deployment system. This is a very demanding endeavour, but we are making progress by decoupling the tasks of configuration management, the settings of options for compilation, and the compilation and deployment, and handing them over to different tools. In particular, we are using `git submodules` for the first task, and use modular scripts for the second one. We have also a Cmake framework which is undergoing field tests to improve its generality and portability to a variety of architectures.

Contribution and re-use of libraries. Apart from using external libraries for functionality and performance, the Siesta project has contributed a number of libraries. They are now at the production level, and are already available to the wider community as part of the ESL (Electronic Structure Library, https://esl.cecam.org) initiative. They are described further in the Libraries section below.

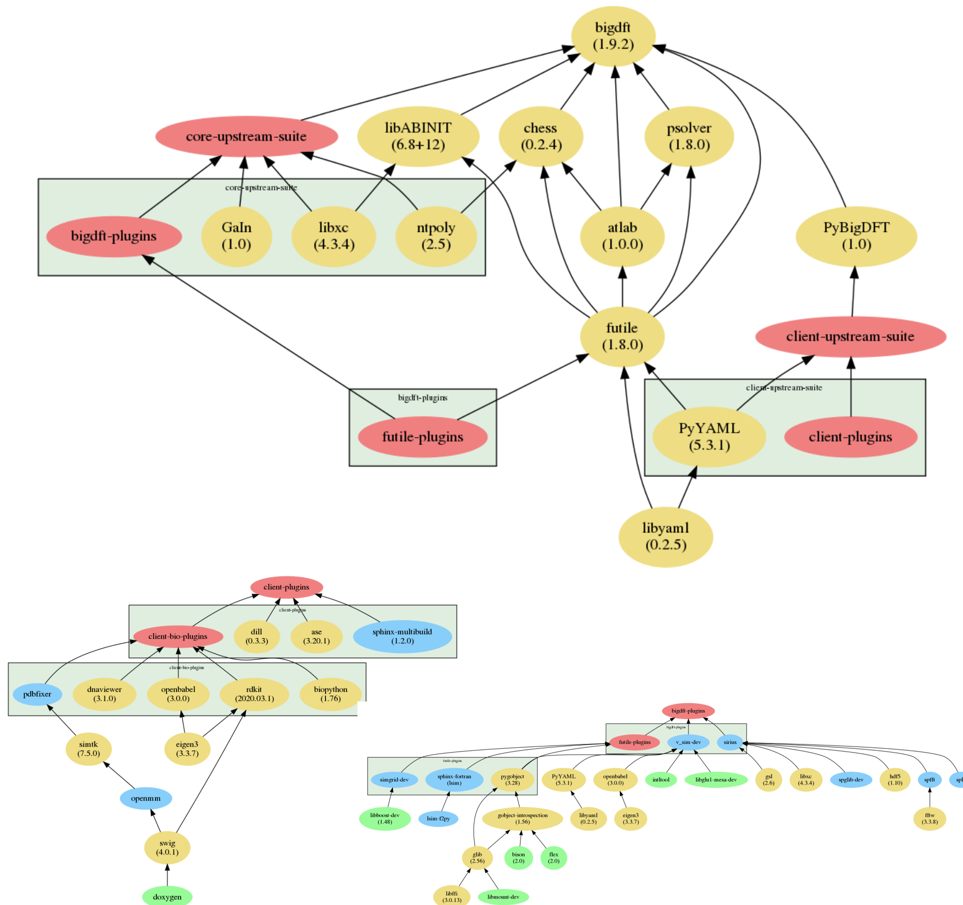

Figure 5: The compilation dependencies of the libraries in BigDFT suite. Upstream packages are now organized in *plugins*, and their compilation is optional, for domain-specific and system libraries, respectively. The packages of the consortium are depicted in yellow.

2.4 BigDFT

The build system of the code has been updated and further restructured in the recent version. We have separated the compilation of the code in *core* and a *client* group. We present in Fig. 5 an example of such compilation. In the following paragraphs we will illustrate the status of the development of the libraries that are internal to the BigDFT developers consortium.

The version 1.9.2 of the BIGDFT code has been released in December, 2021. Version 1.9.3 is under preparation, and in beta stage. In regular meetings of the code developers, the next actions were defined to restructure the inner part of the SCF loop in such a way that the graph of the dependencies among the various code sections will be manipulated by the end-user.

FUTILE and PSolver

See Secs. 3.4 and 3.5 in Libraries.

atlab

The library has been also released with a cmake build system, which at present complements the already available *autotools* module. Both approaches are therefore available. Like the FUTILE library, we are presently working at a python module that should abstract some of the operations on input positions and provide that to higher-level libraries like PyBigDFT.

libconv

The libconv library is now stabilised and ready for integration in the code. The code generation has also been tested on architectures like the Fugaku supercomputer.

PyBigDFT

As already presented in the M12 deliverable, this package is a collection of Python Modules that are conceived for pre- and post- processing of BigDFT input and output files. Such modules are supposed to enhance the BigDFT experience by an high-level approach. Also, calculators and workflows are supposed to be created and inspected with modules of the PyBigDFT package. This package is conceived as a set of Python modules to manipulate complex simulation setups in a HPC framework. Recent advances in PyBigDFT have enabled the implementation and usage of new functionalities of BigDFT.

- The AiiDA BigDFT plugin has been inserted in PyBigDFT. It enables the remote, asynchronous execution of a PyBigDFT workflow from a Jupyter notebook. For each new production result of the BigDFT consortium, the calculations are triggered and pre-postprocessed from PyBigDFT.
- The PyBigDFT API has been carefully checked with respect to the compatibility with Python3 and Python2 (the Python2 support will be soon declared obsolescent). Flake8 execution scripts are inserted in the continuous integration of the library.
- Validation and verification techniques as per WP5 are now triggered entirely from PyBigDFT.
- The PyBigDFT tools analysis has been coupled with established packages for the simulation of biological systems that enable the analysis of production results, like the complexity reduction framework used in the WP6 BigDFT demonstrator.

In Figs. 6, 7 and 8 we present some snippets showing how to employ the APIs of PyBigDFT in conjunction with AiiDA calculators.

We have implemented the "traditional" flavour of AiiDA plugin. Insallation proceeds via: `pip install aiida-bigdft`

bundler

The Bundler package has now been merged with the jhbuild upstream version and unified to the ESL package. Such package is defined from a fork of the Jhbuild package,[1] that

[1] https://developer.gnome.org/jhbuild/

```
from BigDFT import Calculators as C, Inputfiles as I
single_point=C.SystemCalculator()
inp = I.Inputfile()
inp.set_xc('LDA')
inp.write_orbitals_on_disk()
log=single_point.run(input=inp,posinp='mol.xyz')
print (log.energy)
```

Figure 6: SystemCalculator: the Aiida CalcJob equivalent

```
from BigDFT import Datasets as D
hgrid_cv=D.Dataset('h_set')
for h in [0.5,0.45,0.4,0.35,0.3]:
    inp.set_hgrid(h)
    hgrid_cv.append_run(id={'h':h},input=inp,runner=single_point)
results=hgrid_cv.run()
energs=hgrid_cv.fetch_results(attribute='energy')
```

Figure 7: Dataset: a small equivalent of a Aiida WorkChain

```
from BigDFT import AiidaCalculator as A
study=A.AiidaCalculator(code="bigdft@localhost",
         num_machines=1,mpiprocs_per_machine=1,
         omp=1,walltime=3600)
%load_ext jupyternotify
%notify
hgrid_cv.wait()
>>> '0 processes still running'
```

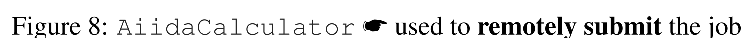

Figure 8: AiidaCalculator ☛ used to **remotely submit** the job

has been conceived in the context of GNOME developers consortium. We are considering the possiblity of submitting a Merge Request to the jhbuild developers to include our needs in the GNOME development process. This package can now used as basis to develop a common infrastructure to compile and link together libraries for electronic structure codes. We are foreseeing some additional modifications to enable an easier utilization by the end user.

sphinx-fortran

The project has been made compatible with python3 and it is now used in the Continuous Integration to build the documentation of the corresponding packages documented in the sources.

2.5 FLEUR

Work in FLEUR focused on several tasks. On the one hand, our efforts to modularize the code further with a specific focus on the functionality required for hybrid functionals has been continued leading to the separation of code into corresponding modules. While the increased modularization and the stronger integration of external libraries into the code has led to significant performance, functionality and design improvements, it also poses a significant challenge when considering the build process. All code relevant for this work is included in the final release MAX -R6.0 of the FLEUR code available at the FLEUR webpage (https://www.flapw.de).

2.5.1 Hybrid functionals in LAPW (LapwLIB)

Hybrid functionals are an extension to the usual exchange-correlation treatment in DFT that are very relevant for many complex oxide materials. At the same time, the evaluation of the non-local potential used in these approximations are computationally very expensive and thus a very obvious target of the refactoring and performance improvements we aim at within MaX. In the full-potential linearized augmented plane wave method (LAPW) employed in FLEUR several specific aspects of the hybrid functional are to be considered. Most relevant is the fact that the hybrid functionals require the evaluation of the Hartree operator for products of wavefunctions. Since these wavefunctions are expanded into LAPW basis functions and products of LAPW basis functions are not LAPW basis functions itself, an additional set of numerical basis functions is required for this evaluation. The refactoring of this code had two different goals. On the one hand, we aimed at the identification of basic operations suitable for encapsulation to generate a set of modules with a clearly defined functionality (LapwLIB). On the other hand, we also had the requirements of refactoring for performance and portability in mind. The resulting code now enables the evaluation of hybrid functionals with an unscreened Coulomb interaction within the MaX version of FLEUR. The performance of this newly structured code is reported in the corresponding section of WP2.

2.5.2 Build process in view of external dependencies

The build of FLEUR relies on an increasing number of libraries, both external to the project and generated by refactoring the original code: therefore it becomes increasingly

complex to establish a reliable build process. In particular, the fact that Fortran modules have to be compiled with the same compiler often restricts the simple use of pre-installed libraries and requires a consistent build of FLEUR together with such external libraries. To enable this we utilize the cmake (www.cmake.org) build system in which we define a series of scripts to determine the required dependencies, download them where appropriate and build the software stack needed by FLEUR. In the future, we foresee the need to improve this process further, e.g. by providing a better integration into package management solutions like Spack (spack.io) for which simple FLEUR recipes already exist. The need to use a rather involved configuration and build tool is of particular relevance in the case of GPU-aware deployment of FLEUR, in which we often experienced challenges in the software stack for Fortran development.

2.6 CP2K

CP2K code has modular approach to the software development and relies on many external libraries. Build system of CP2K needs around 30 external dependencies for the full-featured version of CP2K. To address T1.1.2 (Code refactoring) of this work package, the following steps were taken:

Introduce CMake In order to modernise the build system of CP2K a pull-request for CMake has been opened (https://github.com/cp2k/cp2k/pull/1259). The work is in progress and will be continued.

Improve Spack support The recipe for CP2K in Spack has been updated to the latest version (8.2) of the code and enhanced to support the recent Cray programming environments.

Grid submodule The operations on the real-space regular grid (density expansion in Gaussian basis functions and density integration) were encapsulated in the independent submodule of CP2K (https://github.com/cp2k/cp2k/tree/master/src/grid) and the Fortran API was created. The grid submodule is independent of the host CP2K code.

3 Libraries

Together with the flagship codes, the MAX libraries bundle represent a major outcome of WP1 development effort. Most of the libraries were identified and planned in the Software Development Plan [1], with few new libraries that were added at M18 [2]. At variance with the codes and the development platforms –such as SIRIUS– that provide functional ecosystems into which one can integrate new applications, the libraries have been designed, developed, and tested in view of the standalone use by generic third party codes. The consequent externalisation of such functionalities in the original codes has been also advantageous, allowing a full separation of concerns and, in particular for performance critical kernels, enhancing the adoption and development of architecture-specific implementations.

The implemented functionalities include: (i) low level system utilities; (ii) libraries for the formatted and hierarchical I/O; (iii) computational kernels. The use of completely disentangled libraries for accessing the latter ones has also been effective in enabling collaborations with other groups and vendors for the realisation of specific implementations of these performance-crucial functionalities.

The development of the libraries between M18 and M36 has followed the schedule indicated in the updated plan [2]. We report here the schedule in Table 1 for completeness. Further work is still ongoing for the maintentance and improvement of the interfaces.

3.1 SpFFT

A (distributed) FFT is one of the computationally intensive kernels of many DFT codes. As such, a common FFT library that is specifically designed for the spherical plane-wave cutoffs and pencil G-vector distribution is in a high demand. To address this problem a SpFFT library has been designed and implemented. SpFFT is a 3D FFT library for sparse frequency domain data written in C++ with support for MPI, OpenMP, CUDA and ROCm. SpFFT has the following highlights:

- support for spherical cutoffs
- support for pencil decomposition of G-vector columns
- support for CUDA and ROCm programming models (NVIDIA and AMD)
- 1D-2D domain decomposition to reduce the MPI communication cost and maximise the load of GPU cards
- support of FP64 and FP32 precision
- CMake build system
- support in Spack package manager
- API documentation (available here)
- Fortran90 interface
- examples for C/C++/Fortran90

The development of SpFFT library is finished and the maintenance of the library will be continued at CSCS. The benchmark of the SpFFT library is presented on the Fig. 9-10

MAX libraries expected roadmap (M18-M36)

Library	Group	Month M18	Month M24	Month M36
FUTILE	BIGDFT	Production	Production	Production
PSolver	BIGDFT	Production	Production	Production
atlab	BIGDFT	PoC	PoC	Beta
libconv	BIGDFT	Production	Production	Production
bundler	BIGDFT	Beta	Production	Production
PyBigDFT	BIGDFT	Beta	Production	Production
sphinx-fortran	BIGDFT	Beta	Beta	Production
juDFT	FLEUR	Production	Production	Production
LAPWlib	FLEUR	Beta	Beta	Production
IO-t	FLEUR	Beta	Beta	Production
qe_h5	Q. ESPRESSO	Production	Production	Production
xsdtool	Q. ESPRESSO	Production	Production	Production
UtilXlib	Q. ESPRESSO	Production	Production	Production
FFTXlib	Q. ESPRESSO	Beta	Beta	Production
LaXlib	Q. ESPRESSO	Production	Production	Production
KS_solvers	Q. ESPRESSO	Beta	Production	Production
LRlib	Q. ESPRESSO	PoC	PoC	Beta
UPF_lib	Q. ESPRESSO YAMBO	PoC	Beta	Production
XClib	Q. ESPRESSO YAMBO	PoC	Beta	Production
DevXlib	Q. ESPRESSO YAMBO	Beta	Beta	Production
Driver_Ylib	YAMBO	PoC	Beta	Production
CoulCut_Ylib	YAMBO	PoC	Beta	Production
LA_Ylib	YAMBO	PoC	Beta	Production
IO_Ylib	YAMBO	PoC	Beta	Production
GridXC	SIESTA	Production	Production	Production
libPSML	SIESTA	Production	Production	Production
ELSI-interface	SIESTA	PoC	PoC	PoC
LibNeigh	SIESTA	PoC	Beta	Production
Lua scripting	SIESTA	Production	Production	Production
libFDF	SIESTA	Production	Production	Production
xmlf90	SIESTA	Production	Production	Production
libDBCSR	CP2K	Production	Production	Production

Table 1: Expected Roadmap for the libraries development in the second 18 months of the project. **PoC :** *Proof of concept* version, **BETA:** release candidate, **Production:** interoperable library ready for release.

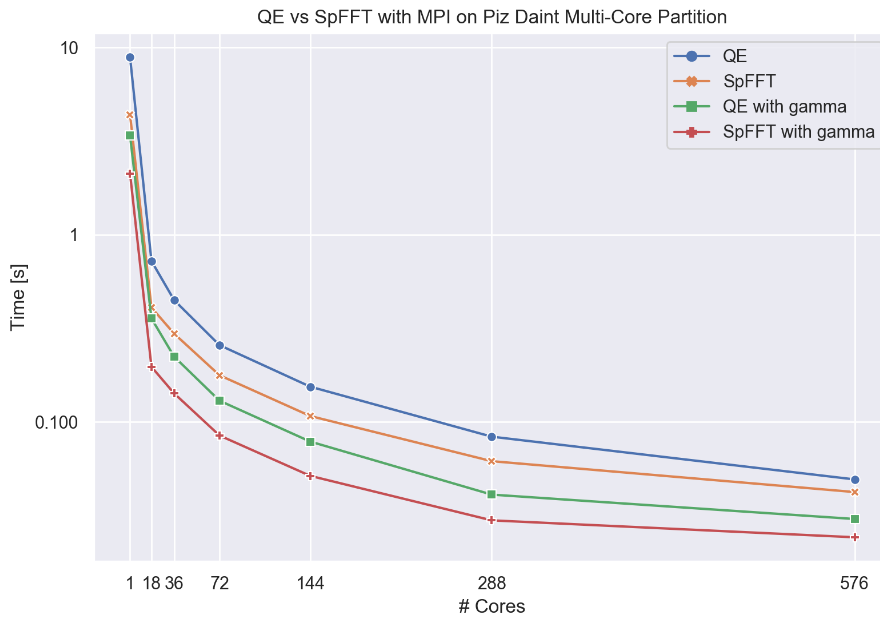

Figure 9: Comparison of the FFTXlib and SpFFT parallel performance using FFTXlib's benchmark mini-app. The FFT grid size is 243 x 384 x 576. This test is also a demonstrator of how to use SpFFT in the scientific application.

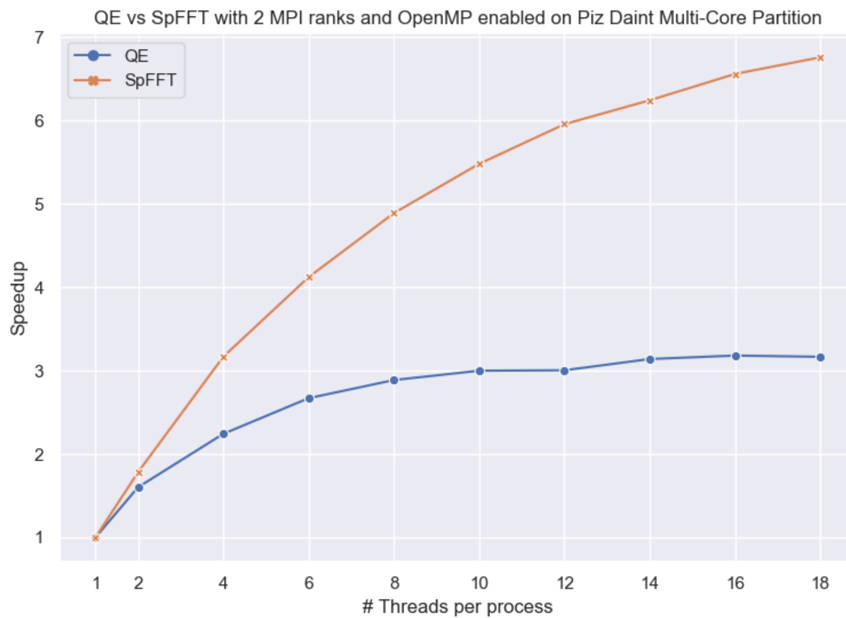

Figure 10: Comparison of the FFTXlib and SpFFT OpenMP threading support. The tests was executed on 1-18 cores of Intel Broadwell processor. The FFT grid size is 243 x 384 x 576.

3.2 SPLA

The analysis of computationally-intensive kernels of plane-wave DFT codes revealed a tall-and-skinny matrix multiplication as one of the bottlenecks. A tall-and-skinny matrix can, for example, be a set of wave-functions distributed over MPI ranks or a set of plane-wave coefficients of the electron-hole pairs in the RPA type of calculations. In both examples a code needs to perform a summation over a long index and create a final (small) matrix usually in ScaLAPACK 2D block-cyclic distribution. To address this problem a SPecialised Linear Algebra (SPLA) was created. Currently, SPLA provides functions for distributed matrix multiplications with specific matrix distributions, which cannot be used directly with a ScaLAPACK interface. All computations can optionally utilize GPUs through CUDA or ROCm, where matrices can be located either in host or device memory. SPLA has the following highlights:

- support for CUDA and ROCm programming models (NVIDIA and AMD)
- support of FP64 and FP32 precision
- support for host and device pointers
- CMake build system
- support in Spack package manager
- API documentation (available here)
- Fortran90 interface
- examples for C/C++/Fortran90

The development of SPLA library is finished and the maintenance of the library will be continued at CSCS. The benchmark of the SPLA library is presented on the Fig. 11.

3.3 SIRIUS

SIRIUS is a domain specific library for electronic structure calculations. It implements pseudopotential plane wave (PP-PW) and full potential linearized augmented plane wave (FP-LAPW) methods and is designed for GPU acceleration of popular community codes such as Exciting, Elk and Quantum ESPRESSO. The following milestones were accomplished during the course of the project and in addition, various bugfixes and performance optimisations were discovered and implemented.

ROCm support The common simple mechanism to wrap CUDA and ROCm kernels was proposed and implemented.

Support for multi-GPU nodes A few changes haad to be tested and implemented in order to run on the multi-GPU nodes (for example, on Marconi100).

Switch to SpFFT and SPLA The custom SIRIUS implementations of distributed FFT3D and linear algebra operations on wave-functions were wiped out and replaced by the newly-developed SpFFT and SPLA libraries.

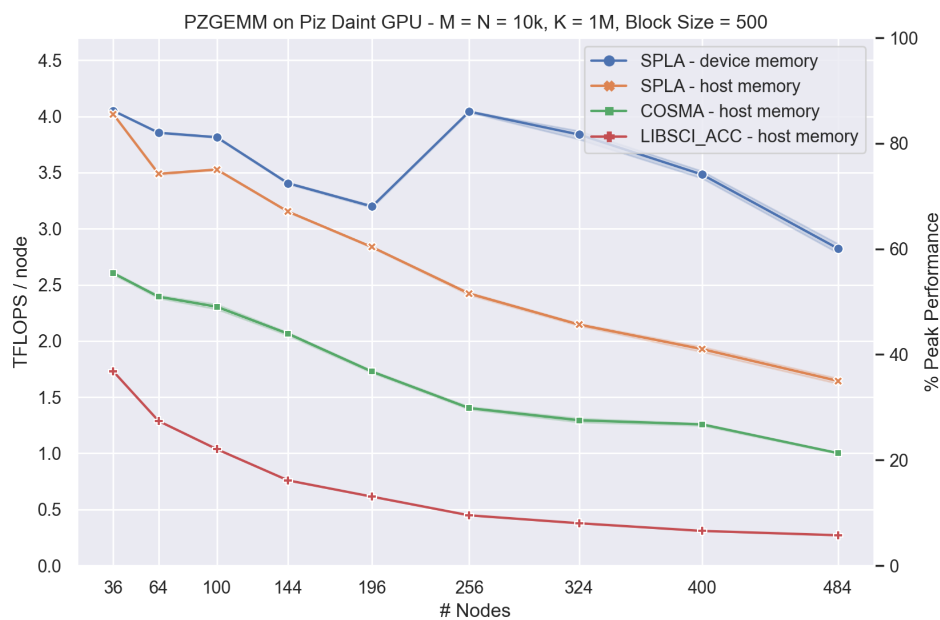

Figure 11: Benchmark of the different matrix-multiplication libraries in the tall-and-skinny multiplication problem with [m,n,k] triplet set to [10'000, 10'000, 1'000'000]. The benchmark was executed on 36-484 nodes of Piz Daint GPU partition. Each node is equipped with 12-core Intel Haswell @2.6 GHz and NVIDIA P100 GPU card. Average performance per node is reported. Lines: blue - SPLA with matrices already allocate on the device; orange - SPLA with host pointers; green - COSMA library with host pointers; red - Cray's proprietary LibSci_acc library with host pointers.

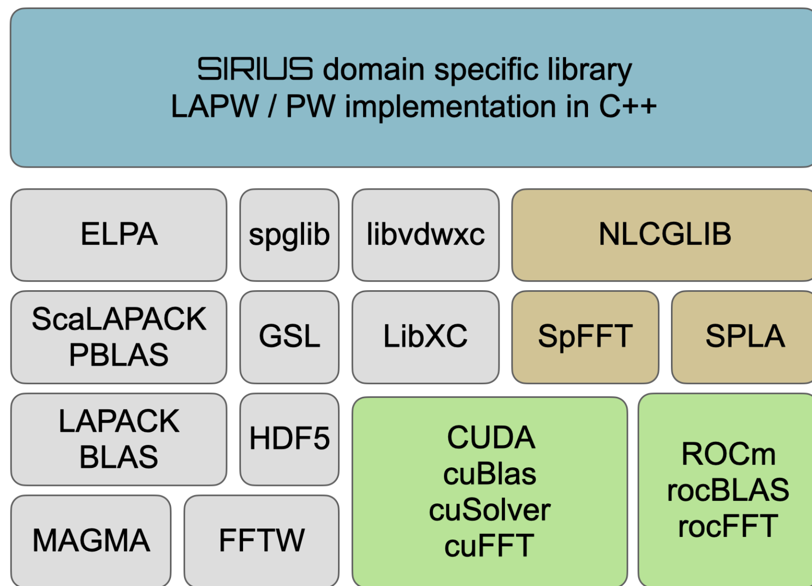

Figure 12: SIRIUS is built on top of the current software stack. Spack package manager is used to build and resolve the dependencies.

Add interface to NLCGLIB An interface to robust wave-function optimisation library NLCGLIB was added to SIRIUS. This allows to achieve the convergence in systems where regular density mixing scheme fails.

Interface SIRIUS with CP2K and BigDFT SIRIUS was interfaced with CP2K and BigDFT to enable plane-wave ground-state functionality in these codes.

Callback mechanism Callback functions to compute radial integrals of local part of pseudopotential, core charge density, beta projectors and augmentation charge from the host code were introduced in order to improve the numerical reproducible.

Total energy contributions SCF correction to total energy and entropy contribution from smearing were added to SIRIUS.

Improved Davidson solver The locking mechanism was added to Davidson solver in order to exclude the converged wave-functions from the subspace diagonalization.

3.4 FUTILE library

This library is in production stage since several months already. It has already been employed in several codes like YAMBO and FLAME. We are presently working on the release of a python package which includes the modules which are associated to the operations performed by futile.

3.5 PSolver

The PSolver library is also released in production stage and can be installed independently.

3.6 LAXlib

This library provides general transparent interfaces for linear algebra operations on large 2D matrices distributed on an MPI group. The library provides a general API which exposes the different available solvers on a unified high level interface such that our codes can easily exploit the library best suited for the used architecture and problem size. For simplicity, we adopted two different data-layouts in which the matrices can be provided. One set of routines that require the full matrices to be provided and thus is suitable for small problems or shared memory setups and routines that deal with distributed matrices stored across nodes in which we adopted the SCALAPACK distribution scheme with the corresponding BLACS descriptors. The production version supports the GPU-aware cublas library. The support for other GPU aware libraries such as oneMKL, ROCsolver, MAGMA, and ELPA are under development. In addition to the computational routines, LAXlib also contains callback routines in order to register code-specific timing as well as routines and error-handlers to allow for a seamless integration into the code-specific infrastructure. The library is constructed with-out explicit Fortran modules but with general include (header) files such that it imposes no compiler specific dependencies in its usage.

3.7 FFTXlib

This is a Fortran library that performs distributed 3D FFTs. Its distributed data structure is designed to fit the needs of plane-wave pseudopotential codes. In the reciprocal space data are distributed among MPI ranks in sticks along the z direction. In real space data are distributed as slabs and slab-slices. The 3D transform is computed as a succession of 1D FFTs on stick, or 2D FFTs on slabs. The 3D data distribution is transposed from the z-stick distribution to the z-slice via a sequence on MPI all-to-all calls. The API has been update to make possible to choose at run time between the "pencil" (1D-1D-1D) and "slab" (1D-2D) decomposition of 3D FFT grids. The library runs in pure CPU mode or in heterogeneous mode. Different strategies are used to optimize the performance (see the WP2 D2.3 deliverable) in the two cases.

3.8 KS_Solvers

This library collects the implementation of various iterative diagonalization algorithms. They are used inside DFT codes to diagonalize the Kohn-Sham Hamiltonian.

The iterative diagonalization algorithms are disentangled from any specific Hamiltonian builder. The Hamiltonian operator may be called by the library as an external routine; in this case the external subroutine must use the same data structure as in KS_Solvers. It is also possible, for some of the schemes, to use the Reverse Communication Interface (RCI); in this case the interface receives ψ and $H\psi$ as arguments, the task of computing and converting them to the format expected by the RCI must be done by the driver.

In the latest version we have introduced the RMM-DIIS [5] scheme and improved the ParO and PPCG schemes (see also WP3 D3.4 deliverable).

3.9 XClib

Starting from release 6.8 the exchange-correlation (XC) DFT management (LDA, GGA, MGGA) has been entirely shifted into an external library, XClib. The library interfaces through a set of XC-wrapper routines, provides by default a wide set of internally implemented functionals. The interface also allows access to those implemented in the external `Libxc` library, if linked. The management of the `Libxc` parameters has been simplified by means of a few wrapper routines that allow the user to set the desired values without the need to dig into technical details. A small program, `xc-infos`, has been included in order to guide the user through the DFT choice and usage by providing information on all the available DFT features, on their usability and on the references of both the internal and external (Libxc) ones. While doing this, the DFT list and its management have been reorganized and put in a single module in order to make it easier to add, remove, or modify functionals and to get easy access to all the available information. Obsolete functionals have been removed and the nomenclature of Libxc-based ones has been modified (now index based) in order to avoid overlaps between names of different DFTs (one of the recurring issues reported from QE users). CUDA support that previously was based on `CUDA-Fortran` has been completely refactored in `openACC`. The new version in more portable (works with gfortran, experimental AMD compilers).

Also the APIs have been refactored for more portability and reusability. The input/output arguments of the xc-wrappers can be provided both on host or device, depending on the value of an optional input variable, `gpu_args`. If this flag is set to true, the input arguments (density and related differentials) are received as device variables, the calculation is performed on GPU and the output is left on device. Otherwise the input is copied on device and the output results moved to the host. A testing program `xc_test` has been included, in order to keep control over the large amount of available DFTs, both internal and external (Libxc), and to check the xc-output matching between different parallelization schemes (MPI, openMP, openACC) and different versions of the library itself. The program calculates the xc energy and potential (and optionally derivatives) over a fixed density grid and stores the results on a XML file, which can be used as a benchmark set of data in a second run. XML data sets for QE v6.8 have been included in XClib so that they can be used by the program as comparison whenever a modification in XClib is added by developers. Based on the XClib GPU porting, the potential calculation in QE has been accelerated too and combined with the CUDA enabled interfaces for FFTs, with very good performance increase (up to 10x with respect to CPU-only for medium size runs).

3.10 UPFlib

This developed library contains all functionalities needed in plane-wave pseudopotential codes for reading pseudopotential data-sets in the `UPF` format [2] and generating potentials and projectors. These functionalities are needed for the development on any post-

[2] http://pseudopotentials.quantum-espresso.org/home/unified-pseudopotential-format

processing application that uses the wavefunctions produced by QUANTUM ESPRESSO. The library contains also a set of functionalities and utilities for the conversion of pseudopotentials from other formats to UPF. The library is GPU ready. Within the MAX consortium the library is used developed and maintained by the YAMBO and QUANTUM ESPRESSO teams.

3.11 xsdtool, qe_h5, and UtilXlib

These three utilities were completed in production phase before M18.

- The `xsdtool` [3] is a python application that, receiving in input an XSD schema, generates the source code necessary for writing, reading and storing the content of the XML files organized according the said schema. The `xsdtool` replaces the `xmltool` proposed in D1.1.
- The `qe_h5` library is constituted by two self-contained modules that provide a set of `Fortran` interfaces for writing and reading integer, real and complex datasets.
- `UtiliXlib` provides a low level utility layer that is used by all parts of QUANTUM ESPRESSO to initialize some basic functionalities such as MPI groups, error handling and timing and profiling utilities. The utility is mostly used in QUANTUM ESPRESSO, but it is very general and can be used in any `Fortran` application.

3.12 omm-bundle

The OMM-bundle [4] comprises a functionality to solve the Kohn-Sham problem by the orbital minimization method (libOMM library), as well as auxiliary libraries for the involved matrix operations. Among the latter, MatrixSwitch is a multi-format matrix storage and operation library.

Within MaX, the libOMM and MatrixSwitch libraries have been extended to link them with Siesta for linear and cubic-scaling calculations. It is noteworthy that most of the code for both operation modes is the same, and specific aspects are handled transparently by MatrixSwitch by calling the appropriate backends: ScalaPack for the cubic-scaling mode, and the DBCSR library (see CP2K section) for the linear-scaling mode.

3.13 xmlf90

The xmlf90 package is a set of libraries to handle XML in modern Fortran. It has two major components:

- a XML parsing library. The parser is designed to be a useful tool in the extraction and analysis of data in the context of scientific computing, and thus the priorities are efficiency and the ability to deal with very large XML files while maintaining a small memory footprint. The most complete programming interface is thus based on the very successful SAX (Simple API for XML) model. For completeness, a partial DOM interface and an experimental XPATH interface are also present;

[3] https://github.com/QEF/xsdtools
[4] https://gitlab.com/ElectronicStructureLibrary/omm-bundle

- a library (xmlf90-wxml) that facilitates the writing of well-formed XML, including such features as automatic start-tag completion, attribute pretty-printing, and element indentation. There are also helper routines to handle the output of numerical arrays.

The library is at the production stage, fully documented, and with a set of examples. It has been fitted with an autotools-based building system[5].

3.14 LibFDF

FDF stands for Flexible Data Format and LibFDF is the official implementation of the FDF Specifications for use in client codes. At present the FDF format is used extensively by Siesta, and it has been an inspiration for several other code-specific input formats. The key feature of FDF is that it provides a much needed flexibility in the handling of the input to a program. It is based in a keyword/value paradigm (including units), and is supplemented by a block interface for arbitrarily complex blobs of data. New input options can be implemented very easily. When a keyword is not present in the FDF file the corresponding program variable is assigned a pre-programmed default value. The library is at the production stage and distributed on GitLab.[6].

3.15 libPSML

The common historical pattern in the design of pseudopotential file formats has been that a generator produced data for a single particular simulation code, most likely maintained by the same group. This implied that a number of implicit assumptions, shared by generator and user, have gone into the formats and fossilized there. This led to practical problems, not only of programming, but of interoperability and reproducibility, which depend on spelling out quite a number of details which are not well represented for all codes in existing formats. PSML (for PSeudopotential Markup Language) [10, 11] is a file format for norm-conserving pseudopotential data which is designed to encapsulate as much as possible the abstract concepts in the domain ontology, and to provide appropriate metadata and provenance information. PSML files can be produced by the ONCVPSP [12] and ATOM [13] pseudopotential generator programs, and are a download-format option in the Pseudo-Dojo database of curated pseudopotentials [14, 15].

The software library libPSML [10, 11] can be used by electronic structure codes to transparently extract the information in a PSML file and adapt it to their own data structures, or to create converters for other formats. It is currently used by Siesta and Abinit, making possible a full pseudopotential interoperability and facilitating comparisons of calculation results. Efforts are underway to expand the ecosystem of PSML tools to interoperate with QUANTUM ESPRESSO and YAMBO. The library is at the production stage and distributed on GitLab.[7].

[5] https://gitlab.com/siesta-project/libraries/xmlf90
[6] https://gitlab.com/siesta-project/libraries/libfdf
[7] https://gitlab.com/siesta-project/libraries/libpsml

3.16 libGridXC

The libGridXC library [8] started life as SiestaXC, a collection of modules within Siesta to compute the exchange-correlation energy and potential in DFT calculations for atomic and periodic systems. The "grid" part of the name refers to the discretization for charge density and potential used in those calculations. The original code included a set of low-level routines to compute $\epsilon_{xc}(\mathbf{r})$ and $V_{xc}(\mathbf{r})$ at a point for LDA and GGA functionals (i.e., a subset of the functionality now offered by libxc), and two high-level routines to handle the computations (in parallel) in the whole domain (with radial or 3D-periodic grids), including any needed computations of gradients, integrations, etc. In addition, SiestaXC pioneered the implementation of efficient and practical algorithms for support of van der Waals functionals [16]. The current libGridXC retains and streamlines most of the SiestaXC functionality, and enhances it by offering an interface to libxc that supports a much wider selection of XC functionals. The library can also deal with non-collinear spin densities.

The library is in production stage. It has been recently extended to support libxc V5, which, among other improvements, supports GPU operation. A refactoring of LibGridXC's internal structure, needed to actually exploit this functionality, is in advanced stages of implementation. A second-generation API, to offer support for MGGA functionals and for higher derivatives of the exchange-correlation energy density, is being designed.

[8] https://gitlab.com/siesta-project/libraries/libgridxc

4 Conclusions and ongoing work

The MAX flagship codes and libraries are able to run on many of the currently used HPC architectures. While support for NVIDIA GPUs, even at scale, is consolidated, the main concern for the exascale readiness comes from the yet experimental level of support that we have for AMD and Intel GPGPUs. Most of the ongoing work is thus currently targeted at enhancing the support of AMD and Intel accelerators in our libraries and flagship codes.

As we had prospected in the software development plan (SDP [1]), the major reorganisation of the codes carried through in the years of MAX pahse 2 has been very helpful in this respect. The main achievements of the reorganisation are the thorough modularisation of the code; the removal of most of the global data structures; the adoption of effective APIs for connecting the many functionalities; the refactoring of many code parts in standalone libraries.

In this last year, thanks to these acquired features, we have been able to rapidly enhance the performance and the performance portability of our codes. We have introduced new algorithms (see WP3 report D3.4); developed, tested and adopted specific libraries that in order to run efficiently on different architectures (see WP2 report D2.3).

The complete encapsulation of the different code parts has enabled us to shift most of the effort for exascale readiness from the whole code to specific computationally intensive kernels. Some of these kernels were developed by MAX and included in our library bundle, while others were taken from other platforms as for example ELSI. In all cases the libraries circumscribe the porting task to well defined code parts. This makes it more feasible to tackle the porting issues in collaboration with experts from HPC centres, or software engineers from HW vendors. Collaborative development events have been organised by MAX WP8, as for example the Hackathon held online from January 24-th to February 20-th 2022, targeted at porting and testing MAX code in AMD GPGPUs.

Even if the modularisation has removed most of the porting instructions out the high-level code layers, and the encapsulation enhances the adoption of local data structures, in some cases it is still more efficient to allocate and access to global arrays in the accelerator memory. This choice affects the code structures, causing issues such the duplication of variables and subroutines. The problem has been progressively solved by adopting for the higher code layers a directive-based approach, either using `openACC` or `opemMP5`. For this purpose the MAX bundle contains the `DevXlib` library. This library provides transparent interfaces and precompiler macros for managing and operating on accelerator data.

The work done in the last year and the ongoing activities demonstrate clearly that the restructuring work has successfully prepared our codes for being used on current peta- and pre-exascale HPC machines by EuroHPC.

Acronyms

API Application Programming Interface. 4, 37

CPU Central Processing Unit. 9

DFPT Density Functional Perturbation Theory. 7, 9

ELSI ELectonic Structure Infrastructure [17]. 37

GPGPU General Purpose GPU. 4, 6, 9, 10, 37

SDP Software Development Plan. 5, 37

TDDFPT Time Dependent Density Functional Perturbation Theory [18]. 9

References

[1] Baroni, S. *et al.* First report on software architecture and implementation plan. Deliverable D1.1 of the H2020 CoE MaX (final version as of 30/03/2019). EC grant agreement no: 824143, SISSA, Trieste, Italy. (2019).

[2] Baroni, S. *et al.* Second report on softwarearchitecture and implementation planning. Deliver-able D1.3 of the H2020 CoE MaX (final version asof 31/05/2020). EC grant agreement no: 824143,SISSA, Trieste, Italy (2020). URL http://www.max-centre.eu/sites/default/files/D1.3%20Second%20report%20on%20software%20architecture%20and%20implementation%20planning.pdf.

[3] Baroni, S. *et al.* Second release of MAX software: Report on first common APIs, data structures and domain-specific libraries. Deliverable D1.4 of the H2020 CoE MaX (final version as of 30/11/2020). EC grant agreement no: 824143, SISSA, Trieste, Italy. (2020). URL http://www.max-centre.eu/sites/default/files/D1.4_Second%20release%20of%20MaX%20software_Report%20on%20first%20common%20APIs%2C%20data%20structures%20and%20domain-specific%20libraries.pdf.

[4] Baroni, S. *et al.* First release of MAX software: report on performed and planned refactoring. Deliverable D1.2 of the H2020 CoE MaX (final version as of 29/11/2019). EC grant agreement no: 824143, SISSA, Trieste, Italy. (2020). URL http://www.max-centre.eu/sites/default/files/D1.2%20First%20release%20of%20MAX%20software_report%20on%20performed%20and%20planned.pdf.

[5] Kresse, G. & Furthmüller, J. Efficient iterative schemes for ab initio total-energy calculations using a plane-wave basis set. *Phys. Rev. B* **54**, 11169–11186 (1996).

[6] Elliott, J. D., Colonna, N., Marsili, M., Marzari, N. & Umari, P. Koopmans meets bethe–salpeter: Excitonic optical spectra without gw. *Journal of Chemical Theory and Computation* **15**, 3710–3720 (2019).

[7] Pederzoli, M. & Pittner, J. A new approach to molecular dynamics with non-adiabatic and spin-orbit effects with applications to qm/mm simulations of thiophene and selenophene. *The Journal of Chemical Physics* **146**, 114101 (2017).

[8] Hagiwara, S., Hu, C., Nishihara, S. & Otani, M. Bias-dependent diffusion of a h2o molecule on metal surfaces by the first-principles method under the grand-canonical ensemble. *Physical Review Materials* **5** (2021).

[9] Ryoo, J. H., Park, C.-H. & Souza, I. Computation of intrinsic spin hall conductivities from first principles using maximally localized wannier functions. *Phys. Rev. B* **99**, 235113 (2019).

[10] García, A., Verstraete, M. J., Pouillon, Y. & Junquera, J. The psml format and library for norm-conserving pseudopotential data curation and interoperability. *Computer Physics Communications* **227**, 51 – 71 (2018).

[11] See: https://siesta-project.github.io/psml-docs.

[12] Hamann, D. R. Optimized norm-conserving Vanderbilt pseudopotentials. *Phys. Rev. B* **88**, 085117 (2013).

[13] ATOM code for the generation of norm-conserving pseudopotentials. The version maintained by the SIESTA project can be accessed at http://icmab.es/siesta/Pseudopotentials/index.html. An alternative version is available at http://bohr.inesc-mn.pt/~jlm/pseudo.html.

[14] van Setten, M. J. *et al.* The PSEUDODOJO: Training and grading a 85 element optimized norm-conserving pseudopotential table. *Computer Physics Communications* **226**, 39–54 (2018).

[15] See: http://www.pseudo-dojo.org.

[16] Román-Pérez, G. & Soler, J. M. Efficient implementation of a van der waals density functional: Application to double-wall carbon nanotubes. *Phys. Rev. Lett.* **103**, 096102 (2009).

[17] ELSI . URL http://elsi-interchange.org.

[18] Rocca, D., Gebauer, R., Saad, Y. & Baroni, S. Turbo charging time-dependent density-functional theory with Lanczos chains. *J. Chem. Phys.* **128**, 154105 (2008).