D1.2

First release of MAX software: report on performed and planned refactoring

Stefano Baroni, Uliana Alekseeva, Augustin Degomme, Pietro Delugas, Stefano de Gironcoli, Andrea Ferretti, Alberto Garcia, Luigi Genovese, Paolo Giannozzi, Anton Kozhevnikov, Ivan Marri, and Daniel Wortmann

Due date of deliverable	30/11/2019 (**month 12**)
Actual submission date	29/11/2019
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 modelling, simulations and design
EC Grant agreement no.	824143
Project starting/end date	01/12/2018 (month 1) / 30/11/2021 (month 36)
Website	http://www.max-centre.eu
Deliverable no.	D1.2
Authors	Stefano Baroni, Uliana Alekseeva, Augustin Degomme, Pietro Delugas, Stefano de Gironcoli, Andrea Ferretti, Alberto Garcia, Luigi Genovese, Paolo Giannozzi, Anton Kozhevnikov, Ivan Marri, and Daniel Wortmann
To be cited as	Baroni et al. (2019): 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.

Disclaimer

Contents

1 Executive Summary

This report describes the M12 release of the MAX flagship codes, featuring their first production-ready accelerated versions running on specific architectures, and their extended re-factorisation into stand-alone libraries to be distributed independently of their parent codes.

Overall, all MAX flagship codes are now able to run on accelerated systems. In particular, QUANTUM ESPRESSO (`pw.x`), YAMBO and FLEUR can run on NVIDIA GPU's; SIRIUS can work on both NVIDIA and AMD GPU's. Siesta has achieved a good degree of acceleration using the external ELPA [1] and PEXSI [2] libraries. CP2K relies on the high performance of DBCSR [3] and other back-end libraries. It is expected that these code versions will be used by a large number of end users, thus providing us with a wide test base and valuable feedback from the community.

The development of the libraries is proceeding as scheduled in the MAX Software Development Plan [4]. At this stage, essential libraries both addressing general-purpose tasks and designed to enhance performance portability are at least in a beta version. As far as performance portability is concerned, FFTXlib from QUANTUM ESPRESSO and SpFFT from the SIRIUS platform address 3D Fourier transforms; LAXlib from QUANTUM ESPRESSO addresses ubiquitous numerically intensive large-scale linear-algebra tasks; PSolver from BigDFT is a real-space Poisson equation solver. All these libraries have allowed us to perform the first extensive set of performance portability tests during the Trieste MAX Hackathon held on November 25-29, 2019. This topic is extensively presented in the D2.1 deliverable [5].

At this stage, the accelerated versions of the MAX flagship codes are still based on specific standard software platforms (*e.g.* CUDA-Fortran for QUANTUM ESPRESSO, YAMBO, and FLEUR and CUDA-C for SIRIUS), which are architecture-specific and proprietary. This feat does not yet completely accomplish the goal to make our software ready for multiple exascale HPC environments, which will likely be based on diverse technologies. The present status of development has nonetheless allowed us to locate and isolate the kernels and the most recurrent constructs used to access and synchronise the data to be offloaded to generic accelerators. These preliminary efforts are instrumental to the achievement of the next milestone, presently in the works, that consists in transitioning these accelerated versions to standardised and architecture-agnostic programming models. Moreover, the broad use of directives will make it possible in the near future to easily port our codes to the new OpenMP v.5 specifications, which will facilitate a straightforward port to all future accelerated architectures.

2 Introduction

The MAX flagship codes and their main features and diffusion are described in the MAX website [1]. As outlined in the MAX Software Development Plan (SDP) [4], the activity of WP1 has focused on the reorganisation of the codes and on the preparation of the standalone libraries distributed through the MAX repository. These WP1 actions are finely intermingled with those performed by WP2, WP3 and WP4. For the sake of conciseness, this report will mostly focus on the code reorganisation issues and on the libraries extracted from the codes while details on performance portability and algorithmic enhancements are reported concomitantly on deliverables D2.1 [5] and D3.2 [6].

Releases and new HPC architectures. The interwoven port to accelerated architectures and re-factorisation into stand-alone libraries of our codes have been inspired by slightly different philosophies for the codes and platforms using plane waves (QUANTUM ESPRESSO, YAMBO, FLEUR, SIRIUS) and those based on localised basis sets (SIESTA, CP2K, BIGDFT). In the former case, numerically intensive tasks are rather homogeneously dispersed over a broad code basis and it would be impossible to efficaciously focus our efforts onto a small subset of it. For this reason, we have decided to proceed in two steps, the first of which consists of two parallel tasks. The first such task has been the development of a fully functional version of the codes running on heterogeneous architectures based on NVIDIA GPUs and the CUDA-FORTRAN programming model. While this effort is limited to a specific architecture, it has allowed us to identify the bottlenecks of the communication between CPUs and accelerators, as well as the most recurrent constructs to be abstracted in view of porting our codes to multiple architectures. In parallel, the CPU versions of the codes have been extensively refactored into modular libraries, which will allow an easy interface with low-level communication libraries where architecture-specific operations will be encapsulated and concealed, only exposing architecture-agnostic APIs. These two tasks are constantly integrated into the main code base. The abstraction of the CUDA-FORTRAN types and constructs will make the both highly modular and adapted (or easy to adapt) to multiple heterogeneous architectures, as sketched in the left panel of Fig. 1. Codes based on localised basis sets, instead, are characterised by a much higher concentration of numerically intensive tasks into more limited portions of the code, thus making it possible to proceed sequentially to refactor the code first and port then a limited fraction of it to accelerated architectures, as depicted in the right panel of Fig. 1.

The results obtained in term of reorganisation of the codes are indeed significant in both cases: all our codes can be kept aligned to the main code bases with little effort. The M12 releases at month 12 represent thus a significant progress towards the sustainable programming paradigm for the development and maintenance of our software in the forthcoming HPC architectures.

Libraries. The preparation of a collection of standalone libraries, meant to extend the benefits of the CoE efforts to the broader code developers community, is proceeding as planned. Although most of the computation-intensive libraries are still at a beta stage, they are available to a wide community of beta testers world wide. They are also of course

[1] http://www.max-centre.eu/software/codes

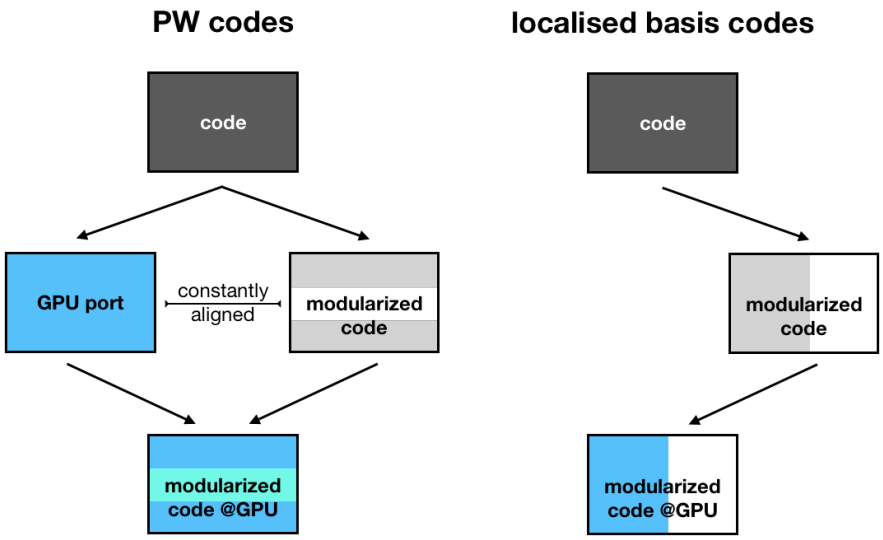

Figure 1: Strategies for modularisation and offloading to accelerators. Codes with local basis sets have efficiently encapsulated their compute intensive regions leveraging specific domain specific libraries (such as ELPA, PEXSI, libDBCSR) without losing accessibility. Codes with plane waves access the offloaded data through all the execution, and to preserve their usability it has been necessary to introduce the explicit management of offloaded data using CUDA-FORTRAN. These offloaded regions are now being encapsulated and abstracted so as to produce modularised and portable codes.

available within MAX for the performance portability planned in WP2. In addition, the SIRIUS developers have delivered a new library for 3D FFT operations (`SpFFT`) which provides a first working implementation of the 3D FFT general API presented in the WP1 SDP and that will be adopted also by FFTXlib in its production version. We would like to mention that the first of the MAX Hackathons was held at ICTP in Trieste on November 25 to 29. The Hackathon has seen the participation of many developers of WP1, WP2, WP3 and WP4. The developers have worked at experimenting the status of the libraries and assessing the maturity readiness of their APIs.

The rest of this document is divided into two main sections. The first section is dedicated to the MAX code release and is organised into code-specific subsections, where the reorganisation of the codes and the ongoing actions will be presented and discussed. The second section will describe in specific subsections the MAX libraries, both production-ready ones and those in beta stage.

3 Releases

3.1 BigDFT

BigDFT is an electronic structure pseudopotential code that employs Daubechies wavelets as a computational basis, designed for usage on massively parallel architectures. It features high-precision cubic-scaling density functional theory (DFT) functionalities enabling treatment of molecular, slab-like as well as extended systems, and efficiently sup-

ports hardware accelerators such as GPUs since 2009. Also, it features a linear-scaling algorithm that employs adaptive Support Functions (generalised Wannier orbitals) enabling the treatment of system of many thousand atoms. The code is developed and released as a software suite made of independent, interoperable components, some of which have already been linked and distributed in other DFT codes.

The version 1.9.0 of the BIGDFT code has been released and published during November, 2019. This version includes a number of features that have been enabled by MAX activities on performance portability (see [5]) and algorithmic improvements and functionalities(see [6]). This is the first version of BIGDFT suite which is versioned by `git` version control system. This versioning, together with the usage of `git subrepo` utilities, enables us to perform *at the same time* the versioning of the BIGDFT code and all the independent libraries that are distributed for MAX from the BIGDFT consortium.

3.2 CP2K

CP2K is a quantum chemistry and solid state physics software package that can perform atomistic and electronic simulations of solid state, liquid, molecular, periodic, material, crystal, and biological systems. CP2K provides a general framework for different modelling methods such as DFT using the mixed Gaussian and plane waves approaches GPW and GAPW. Supported theory levels include –among the many— DFTB, LDA, RPA, semi-empirical methods and classical force fields.

The CP2K code relies heavily on the high-performance implementation of the back-end libraries, in particular DBCSR [3] – sparse matrix-matrix multiplication library. In the past 12 months the following developments have been completed:

- added automatic generation and just-in-time (JIT) compilation of optimised matrix-matrix multiplication kernels in DBCSR library;
- added ROCm backed for the DBCSR library;
- `pdgemm` wrapper for the COSMA library was developed to optimise the large matrix-matrix multiplications in cRPA type of calculation;
- added ROCm backed for the COSMA library;
- added interface to SIRIUS library.

3.3 Fleur

Fleur is a code family for calculating ground state as well as excited-state properties of solids within the context of DFT. A key difference with respect to the other flagship codes –and indeed most other DFT codes— lies in the treatment of all electrons on the same footing. Thereby we can also calculate the core states and investigate effects in which these states change.

The MAX 4.0 release of Fleur is focused on several measures to increase the performance on (pre-)exascale architectures and to increase the modularity of the code. This implies the introduction of new algorithms and functionalities as well as a code reorganisation to encapsulate certain code parts into libraries.

The main performance-related improvements include (see [5]):

- a new algorithm for the setup of the Hamiltonian was employed. Making use of special matrix properties, it increases performance by using matrix rank-k updates instead of matrix-matrix multiplications;
- a Kerker preconditioner for the charge density mixing was made available to the user. This functionality reduces the number of self-consistency iterations required for calculations on large unit cells;
- using CUDA-FORTRAN the most time consuming code parts of Fleur calculations have been ported to the GPU.

The code modularization and encapsulation were performed with specific libraries in mind:

- juDFT-Library: the juDFT-Library is now fully separated from Fleur and compilable on its own. It encompasses basic functionalities. These include among others the interfacing to IO Libraries like libxml2 and HDF5 as well as timing routines;
- FLEUR-LA: to separate the functionalities of the FLEUR-LA library several linear algebra (LA) relevant datatypes have been identified, encapsulated, and made available for distributed and nondistributed data processing and storage. Wrappers around basic LA operations have been implemented allowing for a transparent usage of specific LA libraries;
- LAPWlib: for the development of the LAPWlib the main computational kernels and data types required for the usage of hybrid functionals have been identified. The refactoring of these code parts into a stand-alone library is partly already available in the main Fleur code branch and partly in a separate branch. Other functionalities already make use of and benefit from this refactoring, e.g., the reduced density matrix functional theory (RDMFT) extension which is under development;
- IO-t: to realise the IO-t library the setup data types have been encapsulated together with IO functionalities. This has been made available in a separate code branch soon to be integrated into the main branch.

3.4 QUANTUM ESPRESSO

QUANTUM ESPRESSO is the major open-source code suite for quantum materials modelling using the plane-wave pseudopotential method. It implements DFT, but it also includes more advanced levels of theory: DFT+U, hybrid functionals, various functionals for van der Waals forces, many-body perturbation theory, adiabatic-connection fluctuation-dissipation theory.

The M12 release of QUANTUM ESPRESSO presents important internal changes introduced to rationalise and simplify the usage of stand-alone libraries and to make the usage of OpenMP parallelism as well as the offloading of computations to accelerator devices easier. The development of the stand-alone libraries has been moved to external repositories, making easier for third party developers to used them. The current production release will be the last one to use the internal version of the libraries.

The most important novelty of this release is the official support of the NVIDIA accelerated architecture within a production-ready release. Besides allowing QUANTUM ESPRESSO users to exploit these HPC architectures of increasing diffusion, this version is an important starting point for alternative and future hybrid-architectures. The organisation of this version as well as the development path to future heterogeneous systems is the topic of a dedicated subsection.

Another important experimental feature, such as the usage of mixed precision for the FFT operations is not included in the release, but is available in a branch of the (public) QUANTUM ESPRESSO `git` repository. Although experimental, the mixed-precision branch has already been profitably used in production runs by expert users, also leveraging the high-level support provided by the MAX CoE.

The QE-GPU version and Offloaded Data Management. The QE-GPU version has evolved from an early prototype based on v6.1 of QUANTUM ESPRESSO. The prototype used specific CUDA-FORTRAN extensions provided by the PGI Compiler:

- Specification to define variables and allocate space in the device (GPU) memory and statements to copy and read data to/from the device memory;
- `cuf kernel` directives to compile loops as vectorised instructions to be executed by the device. Such directives are analogous to OpenACC and OpenMP parallel/kernel regions and may eventually be moved to any of these more general specifications with a minor effort;
- the possibility, similarly to CUDA-C, to write kernels in Fortran language. Such kernels may be defined as `global` and be called by the CPU (host) and executed by the GPU device, or as `device` and be called and executed in the device. The kernels allow in many cases a more efficient usage of the GPUs, reducing the number of interactions with the CPUs. Kernels are a specific features of CUDA FORTRAN and are not easily portable to other accelerated architectures.

Starting from this early prototype we have produced a fully functional production version. This version is currently stored in a specific `git` repository and kept aligned by the maintainers to the main code base. In preparing this first release version, we have:

- privileged the use of `cuf kernel` directives, restricting the usage of explicit kernels to cases where they are strictly necessary;
- defined a synchronised data structure for all most important allocatable arrays used in the program; these data have to be concomitantly allocated in the host as well as in the device. Currently these global arrays are protected only at a programming policy level, checking the synchronisation before reading the arrays and toggling their status when the device or the host copy of the data are modified. This policy warrants that CPU and GPU always access an updated copy of the data, while avoiding redundant data transfers;
- implemented a utility library to manage memory workspace in the device in order to reduce the number of allocations and deallocations during the execution.

This setup allows us to keep the QE-GPU version aligned with the main code base with a sustainable effort.

As a further strategic development, all most recurrent constructs are being collected into an external library, `deviceXlib`, discussed in section 4.12. This work has being done in collaboration with the YAMBO developers, who have also adopted the CUDA-FORTRAN programming model as starting platform. This new library will provide APIs and data types that will allow developers to move out of the main code base most of the architecture-dependent parts of the implementation.

Mixed Precision. An experimental "mixed-precision" branch is available in the official QUANTUM ESPRESSO gitLab repository. This branch uses single-precision Fast Fourier Transforms (FFT's) and 3D data fields in the calculation of the $V_{loc}\psi$ term of the Hamiltonian (`vloc_psi` routine). This reduces both the computational workload and the memory footprint of the code, with no observable loss of accuracy. Double precision is instead used in the iterative diagonalisation or orthogonalisation phases, to avoid loss of accuracy. This branch is ready to use for expert users, that can also leverage the high-level support provided by MAX for time- and memory-demanding jobs. It has been successfully used for production calculations using `cp.x` with very large disordered systems [7].

Towards abstracting 3D fields. The adoption of general interfaces in many routines was hindered by the usage of different formats for the 3D data fields depending of the different kind on physical model adopted or the different functional used. The main results of this action included in this release are:

- charge and magnetisation densities are stored into a single data type, consisting of a field for the total density, plus an optional field for the scalar magnetisation density of the LSDA case, or three optional fields for the three components of the magnetisation of the noncollinear magnetic case;
- the interfaces of routines that compute the exchange correlation potentials have also been refactored. Such interfaces yield now the potentials in output as an arrays allowing a more effective usage of openMP and accelerator parallelism;
- Functionals from the `libxc` library have been included in a more consistent and general way. It is now possible to choose any LDA, GGA, mGGA functional from `libxc`, also in combination with those of the internal QUANTUM ESPRESSO library. Compatibility checks between libxc and QUANTUM ESPRESSO have been included. The `libxc` functionals are now available for phonon calculations as well.

3.5 Siesta

Siesta is a first-principles materials simulation program based on DFT. It was one of the first codes to enable the treatment of large systems with first-principles electronic-structure methods: its efficiency stems from the use of strictly localized basis sets and from the implementation of linear-scaling algorithms which can be applied to suitable systems. A very important feature of the code is that its accuracy and cost can be tuned in

a wide range, from quick exploratory calculations to highly accurate simulations matching the quality of other approaches, such as plane-wave methods.

The MAX -M12 release for Siesta (see Ref. [8]) includes an expanded ELSI library interface functionality for solvers, PSML support, and the latest changes from BSC for initial bottleneck removal (as described in the D4.2 report). The new ELSI merge 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 will also be 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. The performance enhancements come in three significant fronts (further details and data in the report for the D2.1 deliverable):

- Further levels of parallelisation: A feature common in principle to all solvers is that the SIESTA-ELSI interface is fully parallelised over k-points and spins (no support yet for non-collinear spin). This means that these calculations can use two extra levels of parallelisation (beyond the standard one of parallelisation over orbitals and real-space grid). In addition, the PEXSI solver, beyond a reduced scaling (at most $O(N^2)$ for dense systems, and $O(N)$ for quasi-one-dimensional systems) offers *two* extra levels of parallelisation: over poles, and over trial points for chemical-potential bracketing. It can be used for large systems with very high numbers of processors.

- Mixed-precision support: The ELPA solver can be invoked in single-precision mode, which can speed up the initial steps of the electronic self-consistent-field (scf) cycle. In fact, it has been shown that in Siesta one just needs to perform the final scf step in double precision to maintain the standard level of precision. This leads to substantial CPU-time savings.

- Accelerator offloading: The ELPA library now offers GPU support in some kernels, and further work in the ELSI project is expanding it to more kernels. It also offers an interface to the accelerator-enabled MAGMA library. Finally, the PEXSI developers (also participants in ELSI) are working on adding GPU support.

  Recall that in Siesta most cpu-time is spent in the solver stage (solution of the generalised eigenvalue problem $H\Phi = \epsilon S\Phi$), and the setup of H is typically much lighter weight. Besides, the setup of H makes use of heavy data indirection for sparse matrices, which hinders the porting to GPUs. Having GPU-enabled solvers is the best option for us.

The PSML merge will mark the first MAX -style refactoring in Siesta: a number of modules/libraries have been made stand-alone and are now called from the program:

- libfdf
- libgridxc
- xmlf90

- libpsml

These are already advanced beta-or-production quality, and are already available to the community as part of the ESL (Electronic Structure Library, https://esl.cecam.org) initiative. They are described further in the Libraries section below. We note that several MAX members (Luigi Genovese, Stefano de Gironcoli, and Alberto Garcia) are collaborators in the ELSI and ESL projects.

3.6 SIRIUS DSSDP

SIRIUS domain-specific software development platform is a collection of C++ classes, GPU kernels and C/Fortran/Python bindings to perform DFT calculations with plane-wave pseudopotential and linearised augmented wave full potential methods. SIRIUS is already interfaced with QUANTUM ESPRESSO code and used in production for some projects at CSCS. Recently SIRIUS was interfaced with CP2K code extending its capabilities with plane-wave and LAPW functionality and a prototype of i-PI with SIRIUS Python backed was also developed.

The following developments and refactoring actions in SIRIUS have been completed in the past 12 months:

- memory pool implementation to minimise the allocation of host, host pinned and device memory;
- optimisation of GPU memory consumption in various parts of the library;
- conservative usage of GPU memory when major arrays stay in host memory and `cuBlasXt` is used for matrix operations;
- added support of ROCm (for AMD GPU cards);
- LDA+U implementation was refactored and made more efficient;
- mixer interface was completely rewritten using variadic templates; now a host code (SIRIUS is this case) can pass arbitrary objects for mixing provided that the functions to computed inner product $\langle f|g\rangle$ and linear operation $ax + y$ are supplied;
- FFT subsystem of SIRIUS was entirely removed in favour of the newly developed SpFFT library;
- interface to `libvdwxc` was added to enable Van der Waals corrections;
- split header files into .cpp and .hpp in order to improve compilation time;
- finish the prototype of direct wave-function optimisers in Python; this required adding necessary Python bindings and API functions.

In total 127 pull requests were merged into main development branch of SIRIUS during the past 12 months.

3.7 YAMBO

YAMBO is an open-source code that implements Many-Body Perturbation Theory (MBPT) methods (such as GW and BSE), and allows for accurate prediction of fundamental properties as band gaps of semiconductors, band alignments, defect quasi-particle energies, optics and out-of-equilibrium properties of materials, including nano-structured systems.

The M12 release of the YAMBO code contains several improvements and extensions compared to the last release done at the end of the MAX (2015-2018) project (YAMBO 4.3). A first pre-release [9] was made available in September 2019 (YAMBO 4.4): its main developments include parallel IO and restarting features, interfaces with external codes and new capability and parallel improvements of the real time module. A new MAX release (YAMBO 4.5) has been made available by November 2019: its main additional development is to enable the NVIDIA GPU support of YAMBO through a CUDA-FORTRAN implementation. Moreover, some run levels of the codes have been further modularised, the interface with QUANTUM ESPRESSO is improved, and new features have been added to the post-processing utility.
More specifically, the new developments and refactoring in YAMBO 4.4 include:

- Parallel I/O and BSE restart: the BSE matrix can now be written in a single file if parallel I/O is activated at configure time (`-enable-hdf5-par-io`). In this way, the calculation of the BSE matrix can be restarted also after a crash or excess of the wall-time limit. This feature turns out to be essential when YAMBO is used in automated series of calculations and workflows (e.g. using AiiDA).

- Interfaces with external codes: a new interface (`a2y`) with the Abinit code [10] has been developed. This interface is based on NetCDF libraries allowing to read directly the NetCDF wavefunctions (`filename.WFK`) files, avoiding to rely on intermediate formats as KSS or to be linked to the ETSF/IO library as it was needed in the past. In the new interface, the sphere of G-vectors is automatically expanded to the higher cutoff needed for the density.

  The interface with the QUANTUM ESPRESSO code (`p2y`) has been further developed and remodularised. The current version of `p2y` correctly works also with a set of hybrid functionals and it is compatible with QE up to version 6.4.1.

- YAMBO real time module: the module for running real time simulation has been significantly extended. It is now possible to select the desired level of approximation in the input file from TD-IP to TD-SEX. Importantly, the parallelisation over k-points has been significantly improved.

- New Interpolation driver: the interpolation routines in the YAMBO post processing utility (`ypp`) have been restructured and modularised. In the new version, two algorithms based on nearest neighbour and Fourier interpolation have been implemented.

The new developments and refactoring in YAMBO 4.5 include:

- Dipole driver. All the subroutines computing the dipoles have been moved inside a single folder with the creation of a driver independent from all other parts of the code and with its own parallel structure. This means YAMBO can now be asked

to just compute the dipoles. The new reorganisation is well suited for extracting a library.

- Porting of YAMBO on GPUs: the main routines of YAMBO (dipoles, non-interacting response function, reducible response function, exchange and correlation part of the self-energy, and BSE) have been ported on GPUs using the CUDA-FORTRAN programming model, and in particular by exploiting CUDA-FORTRAN cuf-kernel directives and the cublas, cufft, and cusolver CUDA libraries.
- The interface with the QUANTUM ESPRESSO code (`p2y`) has been extended to read the projection on the atomic wavefunctions coefficients;
- The YAMBO post processing tool (`ypp`) has been extended to deal with calculations of exciton spin properties.
- Modularisation, including:
  - The whole procedure for the Bethe–Salpeter matrix computation has been modularised in different sections consisting in a) build-up of the screened interaction (`K_screened_interaction.F`), and b) computation of the correlation part of the kernel (`K_correlation_kernel.F`).
  - The input file generation and parsing (`INIT.F`) has been decomposed in three separated procedures: (1) input file I/O (`INIT_input_file.F`), (2) command line parsing (`INIT_read_command_line.F`), (3) database check (`INIT_check_databases.F`).

4 Libraries

4.1 FUTILE library

The FUTILE project (Fortran Utilities for the Treatment of Innermost Level of Executables) is a set of modules and wrapper that encapsulate the most common low-level operations of a Fortran code. It provides wrappers and controls for (log)file dumping, string handling, input file parsing, dynamic memory allocations, profiling, error handling as well as MPI interfaces and Linear algebra wrappers. It also implements advanced data storage objects like linked lists and trees, and provides their bindings to python dictionaries as well as iterators. This package is meant to simplify the work of Fortran code developers as its APIs are inspired from Fortran approach. Particular attention is paid in not downgrading the performance of the upper level subprograms. The FUTILE library has entering its production stage and it is available in the repository The API of FUTILE project is defined and almost stabilised at the time of the writing of the present deliverable. Its documentation is now in its stabilisation phase and it can be found at the URL in Ref. [11].

4.2 PSolver

This package features a real-space based solver employing interpolating scaling functions (ISF) for the solution of the Poisson Equation in vacuum and in continuum solvents.

ISF provide features of flexibility and precision that make this package well suited for a integration in various Electronic Structure codes. It also provides GPU acceleration and parallelisation scheme that make its usage suitable for calculations of the action of the Fock Operator, that is of great interest in the context of Hybrid functionals calculations. The Psolver library is also released in production stage and can be downloaded from the URL in Ref. [12].

4.3 atlab

This library deals with common operations which have to be performed on the atoms in the context of a electronic structure code. It abstracts the representation of simulation domain, iterator on real-space points, atomic structure and related concepts (I/O of volumetric files, handling of atomic basis functions, symmetry operations, just to name a few), and it is also meant to provide a software development platform to define ionic movement operations *prior* to the actual specification details of the Quantum Engine. The API of `atlab` is under stabilization in these months. Work has been done to insert `cmake` build system in the library and to compare this with autotools.

4.4 libconv

The `libconv` library is under testing and it is planned to insert it in the BigDFT code during the first months of 2020. Presently its automatic generation has been performed in traditional Intel architecture as well as ARM.

4.5 PyBigDFT

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 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.
- The implementation of the fragment analysis and handling for large scale DFT calculations is inserted in the API of PyBigDFT. This is associated to the Fragment approach of BigDFT that has been released in the context of WP3.
- For each new production result of the BigDFT consortium, the calculations are triggered and pre-postprocessed from PyBigDFT. The corresponding notebooks are then made available to the community as supplementary materials. See Refs. [13] and [14] for examples.
- The PyBigDFT API has been carefully checked with respect to the compatibility with Python3. Flake8 execution scripts are inserted in the continuous integration of the library.

4.6 bundler

Such package is defined from a fork of the `Jhbuild` package,[2] that has been conceived in the context of GNOME developers consortium. When creating a suite code that is made by a collection of various libraries released independently, the problem of linking and compiling together the entire suite might become cumbersome and time-consuming, especially for non-expert users and satellite developers. The `bundler` package provides a set of solutions which try to adress this problem. Like the `jhbuild` package, it is able to compile libraries with different build systems and configuring options.

We have updated the bundler package with respect to the upstream version of jhbuild, that has been recently updated to be compatible with Python3 API. This bundler package has also attracted attention in the context of ESL initiative, and it will be used as a ground basis to develop a common infrastructure to compile and link together libraries for electronic structure codes.

4.7 sphinx-fortran

This project is a fork of the sphinx-fortran package,[3] which has been written in order to use the `sphinx` package to describe package documentations. Such library provides an extension – alternative to other solutions like FORD and Doxygen – which will be of interest for fortran source codes which might be connected to other high-level programming languages like python. With this package fortan API might be exposed (and referenced) together with other programming languages.

We are presently working to understand the compatibility of this library to python 3 and the efforts that should be done to propagate upstream our modifications.

4.8 xmlf90 library

The xmlf90 package is actually 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;

- 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 recently fitted with an autotools-based building system[4].

[2] https://developer.gnome.org/jhbuild/
[3] https://sphinx-fortran.readthedocs.io/en/latest/
[4] https://gitlab.com/siesta-project/libraries/xmlf90

4.9 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.[5].

4.10 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 leads 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) [15, 16] 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 [17] and ATOM [18] pseudopotential generator programs, and are a download-format option in the Pseudo-Dojo database of curated pseudopotentials [19, 20].

The software library libPSML [15, 16] 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.[6].

4.11 libGridXC

The libGridXC library [7] 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,

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

SiestaXC pioneered the implementation of efficient and practical algorithms for support of van der Waals functionals [21]. 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.

LibGridXC,[8] with an automatic build-system, is already available. It is planned to extend its API to offer support for MGGA functionals and for higher derivatives of the exchange-correlation energy density.

4.12 deviceXlib

This is a new library, not initially foreseen in the SDP, whose necessity has emerged while developing the GPU versions of QUANTUM ESPRESSO and YAMBO using the CUDA-FORTRAN programming model. With `deviceXlib` it will provided the abstraction layer needed to hide all architecture specific aspects of the offloading management and simple computational kernel in the main code base.

In order to achieve such goals, the developers have:

- collected most of the needed constructs in dedicated modules which have been added to the main code base;
- used *cuf kernel* directives to offload parallelisable loops to the GPU;
- managed all duplicated data which have to be stored both in the host and in the device, taking care of synchronisation and minimising data transfer;
- defined a device memory buffer manager to provide work memory on the device, limiting the number of `allocate/deallocate` operations.

The new library derives from internal modules of YAMBO and QUANTUM ESPRESSO which currently implement these functionalities in the two codes. The unification and the data-type definition has already started. The development of this library has benefited from the QE-GPU Hackathon organised by CINECA and NVIDIA in Rome on October 7-10, 2019 and from another joint effort during the MAX Hackathon in Trieste on November 25-29, 2019. As soon as the library has a well defined set of APIs, the porting to other programming paradigms will be worked out, starting with CUDA-C which should also easily permit to adopt translation tools such as ROCm to access other hybrid architectures. The development git repository of `deviceXlib` is available on GitLab. [9]

4.13 FFTXlib

This is a high-performance specific library for performing FFT 3D operations than can exploit different kinds of parallelism: pure MPI, hybrid MPI+OpenMP, hybrid MPI + CUDA-GPU. It is currently at the beta stage. Work is ongoing for expanding the portability of the library; implementing mixed-precision support; implementing an API complying with the specifications designed in the SDP.

[8] https://gitlab.com/siesta-project/libraries/libgridxc
[9] https://gitlab.com/max-centre/components/devicexlib

API description. The API is based on a general descriptor type (`fft_type_descriptor`) which carries the information on FFT mesh dimensions and its distribution over MPI tasks. The API provides then interfaces to initialise the grid, query the presence of sticks or planes inside the MPI node and perform FFT operations.

4.14 LAXlib

This library is meant to provide a unique transparent interface for performing parallel linear algebra in different HPC architectures and with different domain specific libraries. It provides the handles to access transparently lower level domain-specific libraries, notably `ScaLAPACK`, `ELPA` and the CUDA-GPU specific libraries for parallel linear algebra.

The library is currently at a beta stage and and can be used on systems based on hybrid MPI+OpenMP parallelism or CUDA-GPU. More work is ongoing to for engineering the APIs in order to provide general, transparent access to parallel liner algebra in different architectures. LAXLib depends upon UtilXLib for error handling and MPI initialisation. UtilXLib is currently included in the LAXlib package and the build system will take care of compiling and linking it as needed. The build system of the library can use either a configure script based on `autoconf` or use `cmake`.

API description. A descriptor carries all the internal information about the parallelism, the global and local data distribution. The descriptor is currently defined as a Fortran data-type (`la_descriptor`). Further work is ongoing to make the usage of such descriptor more opaque and to make explicit access to its inner variables unnecessary. The following interfaces are provided:

- the `mp_start_diag` routine to initialise the LAXlib MPI environment, the `descla_init` routine to initialise the descriptor;
- the `parallel_toolkit` module which provides interfaces for distributing and collecting replicated matrices, according to the geometry described in the `la_descriptor` type;
- the modules `zhpev_module` and `dspev_module`, providing general interfaces for the diagonalisation of square matrices;
- the `cdiahg` and `rdiaghg` interfaces for the solution of the generalised eigenvalue problem $A\mathbf{v}_i + S\mathbf{v}_i = \epsilon_i \mathbf{v}_i$.

Various improvements of the interfaces are ongoing and will be completed before the release of the production version.

4.15 xmltool

This python tool automatically generates the modules needed for XML I/O. The instances of the library will correspond to the specified XML format; this latter is the only input needed and must be provided using the XSD schema [22] language specifications.

Currently the production version tool produces instances that must be linked to the FoX library. Any instance will provide structured types for each of the XML types defined in the schema, general interfaces for the initialisation, reading, writing and broadcasting each of the structured data-types. A collection of templates for using `libxml` and `gdome` is under development.

4.16 qe_h5lib

This small library provides a minimal, easy to use API to write, read, and modify HDF5 files. The APIs support read and write access to groups and datasets and their attributes. It allows the use of all the main Fortran datatypes. The support for the COMPLEX type is implemented just doubling the dimension in the first direction of the datasets and reading/writing them as real. `COMPLEX` and `REAL` types are currently assumed to be double precision. Strings are written as fixed length but the library is able to read also variable-length strings, which guarantees compatibility with files written by other codes.

The API provides descriptors for handling files, groups and datasets. Passing these descriptors the interfaces allow: opening and closing files and dataset; writing and reading attributes; writing and reading datasets; reading and writing hyperslabs.

Ongoing development work A few improvements to the library are under way. We want to support the following HDF5 features:

- single-precision datatypes;
- compressed datasets;
- parallel I/O.

We are also working to avoid the usage of the HDF5 Fortran modules, binding directly the C routines in order to make linking to the precompiled library easier.

5 Conclusions and ongoing work

The developments and code releases presented here are in line with the general strategy and schedule of the MAX Software Development Plan [4], based on a systematic refactorization of codes enabling their sustainable evolution to (pre-)exascale architectures.

Importantly, the present flagship code releases are already able to run efficiently on heterogeneous architectures based on accelerator devices. This is achieved either by directly leveraging domain specific libraries, or by using the CUDA-FORTRAN programming model. In the latter case, the codes are organised so as to be kept easily aligned to the main code base and provide a solid ground for the forthcoming development work towards a completely architecture-agnostic code base.

References

[1] Eigenvalue solvers for petaflop-applications (elpa). URL https://elpa.mpcdf.mpg.de/.

[2] Welcome to pexsi's documentation! URL https://pexsi.readthedocs.io/en/latest/#.

[3] libdbcsr - a sparse matrix library. URL https://www.cp2k.org/dbcsr.

[4] 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).

[5] Wortmann, D. *et al.* First release of MAX software: report on the performance portability. Deliverable D2.1 of the H2020 CoE MaX (final version as of 30/11/2019). EC grant agreement no: 824143, JUELICH, Germany. (2019).

[6] Genovese, L. *et al.* First release of MAX software: report on the identified actions, update of the software development plan, and software release. Deliverable D3.2 of the H2020 CoE MaX (final version as of 30/11/2019). EC grant agreement no: 824143, CEA, France. (2019).

[7] Cavazzoni, C. private communication.

[8] Releases page for Siesta. URL https://gitlab.com/siesta-project/siesta/-/releases.

[9] Sangalli, D. *et al.* Many-body perturbation theory calculations using the yambo code. *Journal of Physics: Condensed Matter* **31**, 325902 (2019).

[10] Gonze, X. *et al.* Recent developments in the ABINIT software package. *Comput. Phys. Commun.* **205**, 106–131 (2016).

[11] Repository for the FUTILE library. URL https://l_sim.gitlab.io/futile/.

[12] Repository for the PSolver library. URL https://l_sim.gitlab.io/psolver/.

[13] Notebooks for handling the Pseudo-Fragment approach in BigDFT code. URL https://gitlab.com/luigigenovese/pfrag-sicnt.

[14] Notebook to handle the Casida TDDFT approach from PyBigDFT. URL https://gitlab.com/luigigenovese/localization-of-excitations.

[15] 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).

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

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

[18] 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.

[19] 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).

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

[21] 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).

[22] What is an XML schema ? . URL https://www.w3schools.com/xml/schema_intro.asp.