D1.2

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

Daniel Wortmann, Fabio Affinito, Stefano Baroni, Laura Bellentani, Oscar Baseggio, Ivan Carnimeo, Pietro Delugas, Fabrizio Ferrari Ruffino, Andrea Ferretti, Alberto Garcia, Paolo Giannozzi, Luigi Genovese, Sergio Orlandini, Davide Sangalli, Nicola Spallanzani, Daniele Varsano

Due date of deliverable	31/12/2023 (**month 12**)
Actual submission date	27/12/2023
Final Version	27/12/2023
Lead beneficiary	FZJ (participant number 4)
Dissemination level	PU - Public

Document information

Project acronym	MaX
Project full title	Materials Design at the Exascale
Research Action Project type	Centres of Excellence for HPC applications
EuroHPC Grant agreement no.	101093374
Project starting/end date	01/01/2023 (month 1) / 31/12/2026 (month 48)
Website	http://www.max-centre.eu
Deliverable no.	D1.2
Authors	Daniel Wortmann, Fabio Affinito, Stefano Baroni, Laura Bellentani, Oscar Baseggio, Ivan Carnimeo, Pietro Delugas, Fabrizio Ferrari Ruffino, Andrea Ferretti, Alberto Garcia, Paolo Giannozzi, Luigi Genovese, Sergio Orlandini, Davide Sangalli, Nicola Spallanzani, Daniele Varsano
To be cited as	Wortmann et al. (2023): First release of MaX software: report on performed and planned refactoring. Deliverable D1.2 of the HORIZON-EUROHPC-JU-2021-COE-01 project MAX (final version as of 27/12/2023). EC grant agreement no: 101093374, FZJ, Juelich, Germany.

Contents

Executive Summary

This report complements and accompanies the M12 releases of the MAX lighthouse codes. The releases represent a consolidation of the results achieved in phase 2 of MAX regarding performance, portability, and reliability and also contain the more mature results of the development work that is going on in phase 3 of the CoE. We complete this report by presenting and discussing the progress of the MAX codes for what concerns the more advanced and non-production-ready developments that have been going on in MAX codes during this first year.

The most mature achievements for portability (T1.1) have been obtained for CUDA architectures and are, for the most part, already integrated into the production releases. Other significant results have been achieved for different types of heterogeneous architectures, such as the support of the AMD accelerators and, more in general, considerable work for supporting OpenMP offloading (e.g. QUANTUM ESPRESSO, YAMBO, `DeviceXlib` to name a few) or using `SYCL` based platforms (e.g. BIGDFT)

Moreover, parallel performance has been largely consolidated, eliminating known issues and inefficiencies. The code base at this point also represents a starting point for the more advanced and exascale-oriented parallelization strategies that we are experimenting with. The effort to expand and streamline the continuous integration and deployment of these advanced developments is crucial for monitoring the project and for achieving an effective integration within the EuroHPC environment. For this reason, we have worked with WP3 and WP4 to increase the code coverage with CI tests, and to experiment with the different deployment tools (CD) within the collaboration with CASTIEL-2 regarding CI/CD on EuroHPC machines.

We also report about the development activities aimed at enabling the implementation of exascale workflows and generally targeted to deploying the MAX lighthouse applications. These include the necessary interoperability traits planned in T1.4 and the introduction of new features and property calculators in the codes.

1 Introduction

This report accompanies the first release of the MAX lighthouse codes in the third phase of the CoE. While the previous phases of MAX focused mainly on the structure and organisation of the codes to prepare them for the transition to the exascale, in this phase, we aim instead at exploiting this refactoring to create some *lighthouse applications* capable of using the exascale computing power to perform outstanding and unprecedented scientific calculations. This work is mainly contained in the WP2 of MAX . The plans, ambitions and strategies with which the CoE intends to pursue them are provided in the WP2 D2.1 report, where it is clearly visible that significant enhancements in performance, interoperability, and new property calculators are assumed on the part of the codes. This crucial task of the project is in charge of WP1. Thus, this document presents the work done, released, and in progress, as integrated into the broader scope of the MAX targets, connecting, exploiting and helping the work of the other WPs.

These ambitious targets demand the introduction of many new technical solutions that will be worked out in WP3 and gradually introduced and tested by WP1, again in collaboration with WP3. A broad overview of this work on performance assessment and profiling is provided in the WP3 report (D3.1), jointly with a discussion of possible alternative strategies. Many of the changes proposed involve significant refactoring and long development work. In parallel with this primary focus, we keep striving to make MAX codes and applications exascale-ready and efficient, facilitating and enhancing their deployment for all architectures and tool-chains in the current HPC landscape, with particular attention to the clusters belonging to the EuroHPC initiative.

For this reason, these first code releases are intended to provide a stable, efficient, and maintainable production version of the codes that can be easily integrated and efficiently employed in state-of-the-art HPC clusters, including only a few reliable innovations among the many solutions that are currently under development for the realisation of the lighthouse applications. This starting plateau will provide the codes' maintainers and users with a safe base for reliable and stable production codes in the following years while the core developers will be able to concentrate on the major, somewhat evolutive changes that will involve the codes, for example, for what concerns the parallelization schemes, the offloading strategies, and the interoperability layers. This stable production base will also serve the researchers and developers working on the other key targets of WP1, including the implementation of new property calculators, the expansion of the CI/CD codes' apparel and its integration into the CASTIEL/EuroHPC ecosystem, and the experimentation on data-exchange and interoperability.

Within the above strategy, several innovations have been included in this production version, bringing significant progress in terms of new features:

- BIGDFT release provides a new building `Python` package for workflow management and new features for excited state calculations.
- `FLEUR` has expanded its implementation of the LDA+U method, introducing intersite corrections (LDA+U+V) and a new DFPT module that allows for the calculation of phonon modes and frequencies with linear response.
- QUANTUM ESPRESSO has expanded the range of its GPU-ready applications that now include the `PHonon` code and all the `turbo_TDDFPT` packages for

excited states calculation; they also report progress concerning OpenMP offloading and an accelerated version of `pw.x` for the AMD toolchain will be provided in parallel with the official release.

- SIESTA has implemented a new CMake-based build system to manage dependencies and streamline deployment, has integrated new features (DFT-D3, DFT+U with Spin-orbit coupling, on-the-fly wannierization) and has improved efficiency through the removal of bottlenecks.

- YAMBO has implemented and released the MPA scheme (multi-pole approximation [1, 2]) for full frequency GW with GPU-porting and extension to metals, as well as relevant performance improvements concerning BSE and dipoles runlevels. Moreover, significant progress has been achieved in terms of the overall GPU-porting, with the code able to support three different programming models (CUDA-Fortran, OpenACC, and OpenMP-offload) and running the whole GW runlevel on both Nvidia and AMD GPU-accelerated machines (notably including Leonardo, LUMI, and most of EuroHPC machines).

The rest of the document is organised in code-specific sections, where each development team describes in detail the work done within WP1 during the first year of MAX. A conclusive section will provide other general comments on the document's content and discuss the perspective and subsequent actions of WP1.

System	N_{atoms}	N_{SFs}	**H** NNZ	**S** NNZ	**K** NNZ
1CRN	642	1623	22.09	9.40	37.20
Pentacene	6876	19482	2.89	1.04	5.70
1L2Y	1942	4045	5.90	2.13	11.57
Water	1719	3438	9.43	3.43	18.30

Table 1: Matrix properties of four example systems. NNZ refers to the percentage of non-zero elements in the Hamiltonian (**H**), overlap (**S**), and density kernel (**K**).

2 BIGDFT

2.1 Background and Current Status

The current release of the BIGDFT code is 1.9.4. In recent months, the modularity of the BIGDFT-suite has been further enhanced, and the code has been split into a *client* and a *core* package. The client package contains the library necessary to produce and analyse a workflow that could be run remotely on a supercomputer, where the core software suite is installed. The latter represents the collection of libraries that perform the HPC part of the calculations related to the code execution. As already done for the suite as a whole, compiling the two suites relies on splitting the code components into modules, which are compiled by the `bundler` package. This package is defined from a fork of the `Jhbuild` program conceived in the context of the GNOME developers consortium. We have recently extended the code distribution to `Conda` (full suite and core) and `pip` (client) package managers.

Furthermore, the wavelet-based approach has enabled the implementation of an algorithm for DFT calculations of large systems containing many thousands of atoms, with a computational effort which scales linearly with the number of atoms [3, 4]. As mentioned in the description of the lighthouse applications (see report D2.1), this enables the extraction of first-principles derived quantities, which can characterise the electronic structure of systems which were impractical to simulate even very recently. The formalism of BIGDFT facilitates a linear scaling approach while offering numerous benefits for bottlenecks related to linear algebra. The use of strictly localised, quasi-orthogonal basis functions further ensures that the matrices are sparse and well conditioned; the overlap matrix has a low spectral width, and the ratio of the band gap to the spectral width of the Hamiltonian is relatively high. In Tab. 1, we report the matrix dimensions and sparsity for four different systems: a 1CRN protein in the gas phase, a pentacene cluster, a 1L2Y protein in solution, and a cluster of water molecules.

These properties lead to substantial efficiency gains when using the diagonalisation-free methods implemented in our CheSS [5] and NTPoly libraries [6]. BIGDFT can be routinely applied to large systems. For example, calculating a 12,000-atom protein system requires about 1.2 hours of wall time on 16 nodes of the Irene-ROME supercomputer. In the previous deliverable, we presented a study of parallel efficiency with shared code memory.

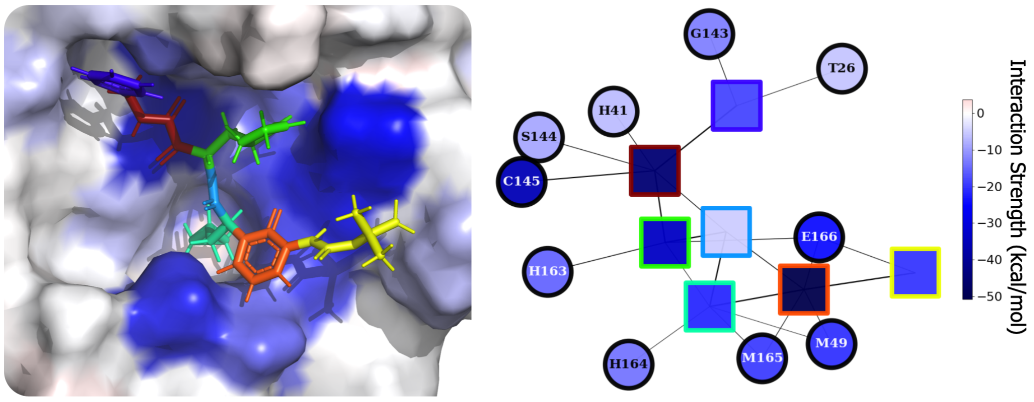


Figure 1: Interaction graph of a ketamide inhibitor (squares: molecular fragments) with the main protease of SARS-CoV-2 (circles: amino acids). The complexity reduction framework measures the interaction strength (kcal/mol) of each inhibitor fragment with the amino acids of the protease.

2.2 Development Priorities

While BigDFT can handle systems consisting of thousands of atoms, these computations remain computationally intensive. Therefore, considering the expense of fully sampling a system's configuration space, expecting DFT calculations to replace commonly used force field methods is unrealistic and possibly unnecessary [7]. In this respect, it is interesting to focus on three critical areas of development: (1) creating analysis techniques that utilise large-scale DFT calculations to provide new insights into emergent properties; (2) developing methodologies to investigate excited state properties inaccessible with force fields; and (3) designing tools for constructing complex, multi-scale workflows [8]. In the following, we provide practical examples of these kinds of development.

2.3 Complexity Reduction study of Enzyme inhibitors

The necessity for analysis techniques for large systems motivates the creation of our *complexity reduction framework* [9, 10]. This framework utilises the converged density matrix and Hamiltonian to decompose systems into coarse-grained fragments based on two metrics: the purity indicator, measuring the quality of a fragment, and the fragment bond order, quantifying inter-fragment interactions. Combined with energy decomposition analysis, these metrics allow for automatic system partitioning, designing embedding environments for QM/MM type approaches, and generating graph-like representations of system interactions.

The complexity reduction framework finds applications in biochemistry, particularly in proposing systematic strategies for modulating the stability of protein/ligand and protein/protein assemblies. BIGDFT's ability to handle large systems has been successfully applied in drug design for inhibitors of the main protease of SARS-CoV-2 [11]. The framework has been instrumental in representing simulation results as interaction graphs, providing detailed insights into the magnitudes and details of interactions at the residue scale (see Fig. 1).

2.4 Excited States

Another priority for BigDFT is the development of excited state approaches, focusing on methods seamlessly integrated with the fragment-based framework. Constrained DFT (CDFT) [12, 13] has been implemented to associate a charge constraint with a given fragment [14], enabling calculations of charge transport parameters, such as those in organic light-emitting diodes (OLEDs) [15]. A recent variant, T-CDFT, incorporates an off-diagonal transition-based constraint, allowing for the treatment of local and charge-transfer excitations at a lower computational cost than TDDFT, with compatibility with the fragment-based framework. This approach could be valuable in considering explicit environmental effects in excited state calculations for large and potentially disordered systems, such as OLED morphologies [16]. We are presently planning to make robust workflows associated with those calculations. To do that, we need to employ robust and reliable workflow orchestrators.

2.5 Workflow Management

To facilitate driving thousands of calculations of large systems on HPC machines, we have been developing a Python package named `remotemanager`, which will be illustrated in the D2.1 document. Such a package automatically serialises data and arbitrary Python closures, which are executed through a supercomputer's queuing system. A remote function might perform a BIGDFT calculation, run other chemistry codes, or perform resource-heavy pre/post-processing steps. Calculations are performed with a *l*azy modality so that the first time a workflow is run, the computationally demanding calculations are asynchronously submitted remotely, and subsequent workflow runs skip the calculation steps. This makes it easy to build analysis routines on top of the data generated from large-scale DFT calculations.

As presented in a previous deliverable, in recent months, the Poisson Solver of the BIGDFT suite has also been ported on Intel GPU architectures to accelerate exact exchange calculations on architectures similar to the just deployed Aurora system. We plan to enlarge the system sizes to be simulated with this architecture as soon as we access a portion of the machine.

3 FLEUR

The current status of FLEUR is reported concerning the main tasks of WP1. We report the programming effort and the currently implemented algorithms and concepts to achieve high single-node performance on various platforms and the different levels of parallelism to enable efficient simulations on multiple nodes. In addition, we report the recent changes in the configuration and build process to ease the deployment on various machines. Finally, new features relevant to future exascale simulations are summarised.

All the described features are part of the first release of FLEUR in this project phase called MaX-R7.0, available on the FLEUR webpage: https://www.flapw.de .

3.1 Single node performance

The FLEUR's single-node performance is determined by the availability of optimised kernels for the particular architectures. On the CPU side, we rely heavily on either explicitly parallelised loops with the OpenMP multithreading paradigm or vendor-provided optimised multithreaded math libraries. For GPUs, we employ the OpenACC programming environment.

In particular, when it comes to running on NVIDIA GPUs, the CPU time of many kernels running on the GPU is significantly reduced compared to the CPU version. This leads to a situation in which parts of the code that were previously less relevant and thus not yet ported become bottlenecks. Currently, the following main kernels are ported to the GPU and show satisfying performance:

Kernel	Description
diagonalisation	Solution of generalised Eigenvalue problem, external libraries
hsmt_sph	Spherical contributions to Hamiltonian and Overlap matrices
hsmt_nonsph	Non-spherical contribution to Hamiltonian
abcoef	Wavefunction coefficients used for charge density
wavefproducts_INT	Projections of products of wavefunctions onto the mixed-product basis used in the hybrid functional code-path (Interstitial contribution)
wavefproducts_MT	Projections of products of wavefunctions onto the mixed-product basis used in the hybrid functional code-path (Muffin tin contribution)
sparse_matmul	Product of projections with Coulomb matrix for hybrid potentials

In addition, several kernels have been ported to GPUs but currently do not show satisfactory performance (though possibly helping, e.g., in reducing data movement from host to device memory). The most relevant of those are:

Kernel	Description
hsmt_lo	Local orbit contribution to the Hamiltonian and overlap. This kernel often does not expose enough parallelism to yield good performance.
hsmt_soc_offdiag	Spin-orbit matrix elements. Not optimised yet.

Finally, some parts of the code have not been ported to GPUs because they usually require little computational effort. However, with increasing coverage of the GPU code and on machines where only a tiny fraction of the computational capabilities is provided by the CPU, these parts also become a target for GPU porting. Here, we have to list in particular:

Kernel	**Description**
vgen_coulomb	The Hartree potential generator. This code actually contains many different computationally relevant routines, making porting quite difficult.
vgen_xc	The exchange-correlation part of the potential, in principle, exposes many low-level parallelisms suitable for GPU porting. However, it also features deep call-trees requiring a lot of coding.
pwden	The charge density generation in the INT region relies on FFTs. Here, the memory handling for GPU would have to be added.
mix	The mixing routines can become a bottleneck for long mixing histories.

Work on extending our `OpenACC` GPU version for these additional kernels has been started and will continue during the project.

The `FLEUR` team also explored the possibility of replacing or augmenting the `OpenACC` implementation with an `OpenMP` offloading version. While we did this for some kernels, so far, our experience with corresponding compiler support is rather disappointing. We could successfully run some `OpenMP` code on NVIDIA GPUs using the NVHPC compilers, but of course, for this combination, the `OpenACC` version already provides a good solution. We plan to continue monitoring the situation and revisit the `OpenMP` programming paradigm as soon as the compiler support for Fortran improves.

3.2 Parallelisation strategies

`FLEUR` contains several levels of parallelism. Besides the already mentioned intra-node multithreading, we use several different strategies to distribute the calculation using MPI on multiple independent tasks on other computing nodes. This MPI parallelism is also used to exploit multiple GPUs. For a standard DFT simulation, we have two main levels of MPI parallelism: (i) the **k**-point loop, which leads to primarily independent problems that, therefore, can be distributed without much communication and thus very good scaling, and (ii) the eigenvector parallelism, which basically relies on distributing the Hamiltonian matrix over many nodes and therefore requires a distributed matrix diagonalisation. While the second level of parallelism of the eigenvalue problem scales less optimal due to the communication requirements, the solvers used for such a distributed eigenproblem made significant progress and now also usually scale up to hundreds of nodes. For the hybrid functional code, the parallelisation over the **k**-points is mostly replaced by a parallelisation over **kq**-pairs. This code scales quadratically with the number of **k**-points, which can lead to a significant additional performance benefit.

While these parallelisation schemes are well established in `FLEUR`, well tested, and have demonstrated their effectiveness, we experience increasing difficulties in the level of usability. Basically, the system to simulate (particularly the number of **k**-points used, the basis size and the treatment of magnetism) defines the memory footprint and the computational effort. To map this onto the available resources is not always a trivial task. Finding the correct balance between intra- and inter-node parallelism, between the levels of MPI and OpenMP parallelism, is often beyond the typical expertise of our users. This is even more problematic if GPUs are used, as these introduce new, additional parallelisation options and new constraints, particularly with respect to memory available on the GPUs. We are currently working on a twofold strategy to ease this burden for the users:

- **More flexible MPI parallelism**. So far, FLEUR always uses optimally load-balanced parallelisation. E.g. if we use 6 MPI tasks on a system with 27 **k**-points, we use a 3×2 scheme in which 3-points are calculated in parallel by two tasks simultaneously (using the parallelism of the eigenproblem) and we have to loop this 9 times to obtain results for all **k**-points. As the **k**-point parallelization is very efficient, the option to use a 6×1 scheme might be faster, even if we use only half the nodes in the last loop.

- We created a tool to provide a **suggestion for a suitable parallelization** scheme given the system to simulate. While this already tries to take the computation load into account, we still have to incorporate the memory footprint of the problem and the constraints imposed by these. This is of particular importance for GPUs, as here too small problems also lead to significant performance degradation.

3.3 Deployment

As a general trend common to other scientific software projects, the increasing dependency on external libraries makes it more difficult to install FLEUR on modern HPC machines. While we also have a Spack-based installation path, in our experience, the philosophy of building the whole software stack from scratch, as done by default by Spack, often also fails. Hence, we are extending our cmake-based build system to enable the simultaneous automatic build of some external dependencies and the FLEUR code itself. While this significantly increases build time, it also usually ensures a compatible build of libraries. Currently, we use the submodule feature of git to download the following dependencies automatically and compile them using cmake targets:

HDF5	IO-library
ELSI	Diagonalization
SCALAPACK	Parallel linear algebra
Libxc	physics library

Of course, this is only done if no compatible version of these libraries are found on the system. With this feature, FLEUR can currently be compiled and installed on machines on which at least the following software is available:

Fortran, C and C++ compiler	must be interoperable (Intel, NVHPC, GCC tested)
LAPACK/BLAS	should be optimized to the hardware
libxml2	standard open-source XML library.

In addition, an MPI implementation and the CUDA development environment for GPUs should be available.

3.4 New functionalities

We recently implemented into `FLEUR` two new scientific features that will be relevant for future simulations:

LDA+U+V

We implemented a corresponding non-local V-term in an extension [17] to the well-known LDA+U method, in which a local Coulomb interaction term (the U-term) is added to the DFT Hamiltonian. This method can include more subtle effects due to the Coulomb interaction at neighbouring sites and thus can be used to correct deficiencies of standard DFT functionals. Of particular interest is the possibility of mapping relatively expensive hybrid functional calculations onto this much simpler model, thus enabling the simulation of situations in which the hybrid functionals are either too expensive or not applicable because they should not be applied to the full system under investigation. In combination with appropriate workflows in which such a mapping of hybrid functionals results is performed, this new feature can enable the simulation of large systems with complex electronic structures.

Phonons in DFPT

Using density-function perturbation theory (DFPT) [18], we implemented a linear response treatment to simulate phonons. The corresponding theory and first results have recently been submitted for publication [19]. While phonons are a very basic property of solids, and their simulation using DFT is generally well established, this is not true for an all-electron treatment of the electronic structure as provided by `FLEUR`. Hence, we hope our new implementation will pave the way for interesting applications, ranging from phonons in complex (magnetic) materials to heavy elements or phonons as a driving force for superconductivity. We plan to implement the electron-phonon coupling matrix elements next and are already extending the current code to use the film mode of `FLEUR` to simulate phonons in 2D materials.

4 QUANTUM ESPRESSO

This section presents the status of the QUANTUM ESPRESSO (QE) suite at the moment of the `7.3` release, which is the first delivered in this third phase of the MAX project. Together with the released code, we also discuss the ongoing work for improving the portability, performance portability, extensions and refactoring necessary for the exascale workflow applications, and enhancement of the configuration and build system. The latter is particularly relevant to streamlining the deployment of the QE suite on all EuroHPC clusters, thereby ensuring a continuous integration/deployment of the new developments in all these machines.

During the last year, QUANTUM ESPRESSO had two releases `7.2` and `7.3`. On the general ground, the most significant change introduced in these two versions has been the thorough refactoring of the offloading implementation for CUDA-based heterogeneous machines, with a progressive phase-out of the pristine CUDA-Fortran implementation in favour of the more streamlined `OpenACC` approach. Such refactoring has been, in many ways, instrumental to the ongoing work for the implementation of OpenMP-based offloading that is publicly available in a separate branch of the QUANTUM ESPRESSO repository. Importantly, the work done in the branch aims to be effective in terms of functionality and performance on AMD heterogeneous parallel clusters (notably including the LUMI cluster). It has been benchmarked in part, and some of the results of this work will be reported in the MAX D3.1 report. The final milestone of the `OpenMP` offloading implementation will be the unification of the source code, which the QUANTUM ESPRESSO developers will target on top of the `7.3` tag.

Together with the refactoring, and in many cases leveraging on those changes, the acceleration of other code parts and applications of the QE suite has significantly progressed this year with the `7.2` release, the first featuring the CUDA accelerated version of the `PHonon` code together with those of other linear-response related applications. These accelerated versions have been optimised and released in the `7.3` version of QE. The rest of the section is organised into subsections. The first subsection will summarise the structure and features of the suite; then, we will describe the work done for what concerns the portability and the performance portability; we conclude the section with a description of the ongoing developments.

4.1 Structure and organisation.

QUANTUM ESPRESSO [20, 21, 22] is a software suite for performing electronic structure calculations using Density Functional Theory. It includes advanced theory levels and uses plane waves and pseudopotential representation.

During previous MAX projects, we have organised the suite into layers. The base includes libraries and modules that provide access to core functionalities (`UtilXlib`) and compute-intensive kernels (`FFTXlib`, `LaXlib`, and `KS_Solvers`). Two high-Fortran module collections (`Modules` and `LR_Modules`) provide field-specific electronic structure functionalities as shared building blocks for all suite applications.

- Core general functionalities: `UtilXlib`, `DeviceXlib`
- Compute-intensive kernels: `FFTXlib`, `LAXlib`, `KS_Solvers`

- Field-specific electronic structure functionalities: `Modules`, `LR_Modules`, `UPF_lib`, `XC_lib`

The encapsulation, optimisation, GPU porting and adaptation of these parts of the code have been the object of the work done in previous phases of MAX . On top of these libraries, the application layer is made by the quantum engines, property calculators, and post-processing utilities:

- Quantum-engines:
  - `pw.x`, performing self-consistent total energy and force calculations;
  - `cp.x`, performing Car-Parrinello molecular dynamics simulations;
- `PHonon` computes electronic linear response to ionic displacements, electric fields, phonon dispersion, dynamical charges, and dielectric constants;
- `EPW` computes electron-phonon interactions, combining the output of `ph.x` with maximally localized Wannier functions;
- `TDDFpT` package contains several applications based on time-dependent density functional perturbation theory for computing various types of excitations:
  - `turbo_Lanczos` and `turbo_Davidson` for computing optical properties;
  - `turbo_eels` for computing electron energy loss spectra;
  - `turbo_magnons` for computing magnetic excitations;
- `QeHeat` for computing the electronic part of thermal transport;
- `hp.x` for computing Hubbard correction parameters with linear response;
- `KCW` for performing self-interaction-corrected band structure calculations;
- `GWL` for performing many-body perturbation theory calculations.

The suite's build and testing systems integrate the overall code structure within each package, which can be compiled and tested separately according to the dependency hierarchy. The present release provides two fully functional build systems. The original build apparatus based on `autotools` has been recently sided by a `CMAKE` based one. Both apparels are fully functional, support most toolchains, and cover all known cases. Both systems support both the production version and the development one. For the latter case, both are fully integrated with the `git` versioning system, allowing for flexible usage of external libraries and packages, as within the use of submodules. The `CMake` build system is also integrated into two of the most used automated deployment platforms: `spack` and `conda`.

The QUANTUM ESPRESSO community publicly distributes and integrates new developments via the `develop` branch of the `GitLab` repository[1]. Production-ready tags corresponding to releases are distributed via the `master` branch of the same repository.

[1] https://gitlab.com/QEF/q-e

The official code releases with the standalone source code, compilable without relying on `git`, and integrated with documentation and other assets, are distributed via the QUANTUM ESPRESSO web site[2].

The continuous integration occurs in two phases. At the level of the git repository, the CI tests that the changes to the `develop` branch do not break the build systems for all the supported toolchains. In the second stage, the `develop` nightly builds are thoroughly tested in the QE-testfarm[3] with more than 400 functional tests.

4.2 Portability and performance

This last year, we have focused the developments and actions on ensuring the portability and performance of QE on parallel heterogeneous machines. The results in terms of performance have been reported in a recent paper [22] and a MAX deliverable [23]. These measures, included in tasks T1.1 and T1.2 of the WP1 development plan, are necessary to implement the remaining part of the program. Within this effort, we have performed the following key activities:

- We have refactored our current CUDA-based offloading system by incrementally **replacing CUDA-Fortran with OpenACC**. This has led to removing most of the duplicated code parts, variables, and modules. The high-level code parts, such as applications and modules, have gradually phased out CUDA-Fortran. However, it is still being used for the mathematical libraries. We have modified and updated their APIs as necessary so they can be called transparently from the higher-level software layers.

  This refactoring has streamlined the maintenance and re-usage of our code base and has made integrating new developments with immediate support for heterogeneous machines more accessible. Furthermore, since this implementation is portable to any toolchain supporting `OpenACC`, it can be used on different platforms.

  Overall, this implementation is conceptually very close to `OpenMP` offloading and represents a necessary first step towards supporting multiple platforms and toolchains.

- We have **extended the GPU acceleration** to most of the QE **suite applications**. The `OpenACC` directive approach has made this effort significantly easier. Initially, the acceleration of the `PHonon` program was addressed. Due to modularisation, a significant part of the porting was done internally at the level of `LR_modules` and directly reused in accelerating all other linear-response applications of the suite. In passing, we note that the GPU-enabled version of all applications of the `TDDFpT` package and of `hp.x` is a crucial feature for the development of some of the exascale workflows planned in WP2 (see report D2.1 [24]).

- We are currently working on extending our support to heterogeneous parallel machines based on non-CUDA GPUs. Although `OpenACC` is a universal programming model that can be applied to all accelerated architectures, well-performing support is currently only available for NVidia devices and compilers. To address

[2] https://www.quantum-espresso.org/download-page/
[3] http://test-farm.quantum-espresso.org:8010/

this issue, we have started working on a more versatile implementation of offloading, with a code base that can eventually be compiled using either `OpenACC` or `OpenMP` offloading directives. The first stage of this extension involves preparing a separate branch where `OpenMP target` directives mirror the `OpenACC` functionalities. This branch (labelled `develop_omp5`[4]) is currently focused on obtaining optimal performance with AMD (with Cray Fortran compiler) and Intel accelerators (with the Intel OneAPI toolchain) for `pw.x`, which is the main quantum engine of the suite. Upon the release of QE `7.3`, a LUMI-G-specific tag of `pw.x` will be prepared and made available to the users.

4.3 Development

We have prioritised the work on portability, completing the support of CUDA accelerators and putting special effort into achieving good performance on AMD-based heterogeneous platforms. These developments, already available for deployment and production, practically complete the first stage of Task T1.1 for what concerns QE. They are an integral part of the official release or the LUMP-G-specific version of `pw.x`, and we discussed them in the previous subsection. In this subsection, we discuss all the longer-term developments done for tasks T1.2-4, as outlined in D1.1 [25]. In the next months, these changes will be incorporated into the primary development branches of our repository, fully integrated into the build and deployment systems, and available for testing. The main ongoing developments are:

- **Parallel efficiency (T1.2):**
  - **Band parallelism refactoring:** Band group parallelisation is one of the crucial features for enhancing scalability and reducing the device-memory footprint of QE applications. The current implementation is inefficient and unfit to exploit the joint throughput of heterogeneous parallel machines (see D3.1 [23] for a more detailed analysis). For this reason, we have started working on a thorough refactoring that aims at implementing three main features:
    1. the effective distribution of wave-function data structure to reduce the device memory footprint and streamline inter-band-group communication;
    2. the removal of collective MPI communication calls in favour of point-to-point data exchange between paired band-groups that can be overlapped with the computation within each band-group;
    3. the reorganisation of the kernels that compute $\langle \phi_i \mid \hat{O} \mid \phi_j \rangle$ matrix elements and perform Gram-Schmidt orthogonalisation to enable working in blocked mode.

    As an initial proof of concept for this development, we have been working on a blocked version of the routine that performs Gram-Schmidt orthogonalisation among wave functions. The new routine only allocates the block of bands assigned to its group on the device memory, along with a buffer where

[4] `https://gitlab.com/QEF/q-e/-/tree/develop_omp5`

the bands of other groups are alternately copied to perform the orthogonalisation. In conjunction with WP3, we will analyse and fine-tune the operation of this proof of concept to optimise the synchronisation between data movement and computation.

- **Distributed eigensolver:** The distributed eigensolver is currently used only in homogeneous parallel machines where it is crucial to obtain acceptable performance for all systems where the number of bands is of the order of the thousands, while on GPUs the accelerated solver can (in principle) operate efficiently up to many thousands of bands. The main motivation for enabling the use of distributed eigensolver on GPUs is the necessity of distributing the matrices in cases where their size becomes comparable with the memory capability of the single device. One other temporary reason for this work is given by the fact that the current accelerated eigensolver for AMD-GPUs (`ROCSolver`) is currently inefficient. The diagonalisation of large matrices on LUMI-G is currently better performing when executed with `ELPA` or ScaLAPACK libraries on CPUs. For this reason, the first version of this development will be included in the `pw.x` version for the LUMI-G cluster.

• **Enhanced testing and deployment (T1.3):**

  – Integration of **OpenMP offload in the build systems and source base**. Together with the work to unify the two alternative offload implementations in one code base, it is also necessary to work on integrating the `OpenMP` offloading work on the two build systems. Both build systems have partial support for the OneAPI and Cray/AMD toolchain only. Extending the support to other toolchains is made difficult by the fact that the `OpenMP target` model is, at the moment, supported at different levels by the other toolchains. It will be thus necessary to test and identify the compilers and systems that support it.

  – **Mini-apps and build-up of functional testing apparel**. We are developing lightweight applications that isolate the calls to single significant kernels of QUANTUM ESPRESSO. These mini-apps will serve various purposes, such as profiling and testing in WP3, processor emulation test codes in WP4, and extending the code coverage of the continuous integration utilities in WP1. We have already created a mini-app that wraps the `FFTXlib` operations, and we are currently working on two more applications that address the `KS_Solvers` and `LAXlib`.

• **Features and traits for lighthouse applications (T1.4):**

  – **Development of excited states gradient calculator**. We have begun implementing the calculation of excited state energy gradients in the `turbo_Davidson` application. This feature is crucial for realising one of the exascale scientific workflows of WP2.

  – **Development of formatted error and warning logging**. We are developing a YAML-formatted logging system for the applications of the QUANTUM ESPRESSO suite. The new log file format includes various types and levels of error messages and warnings.

5 SIESTA

5.1 Background

SIESTA is a DFT code based on pseudopotentials and using strictly localised pseudo-atomic orbitals as basis set. The latter feature brings large efficiency gains (e.g. in diagonalisation, due to the much lower cardinality of the basis set when compared to plane-wave codes), and also opportunities for the use of low-complexity, sparse-based methods for the construction of the Hamiltonian and the determination of the electronic structure (e.g. the highly scalable `PEXSI` method using pole-expansion and selected inversion). These features enable SIESTA to treat very large systems with modest resources, and also to exploit massively parallel architectures. See Ref. [26] for more details about the program and its wide field of applicability (including non-equilibrium simulations with the `TranSiesta` module).

5.2 New build system to support modularity and ease deployment

In this first SIESTA release of the third phase of MAX we have consolidated the performance improvements achieved by the end of the second phase of the project, as well as a number of new computational features that enable the program to offer new functionalities. The consolidation, as mentioned in the Introduction, is needed to provide a robust baseline with which to embark on the task of deployment of the code on the various EuroHPC systems. In this regard, it is very important to highlight the work done on the building system, which has been thoroughly redesigned using CMake.

The building system brings together the different internal modules in which SIESTA has been segmented during earlier phases of MAX , and provides a very flexible system for dependency handling, based on a cascade of options (use of pre-compiled libraries, use of copies of their code in submodules, or direct download) that suit both developers and end users. A collection of CMake-based custom modules helps to find the relevant pieces in the software stack of the machine, with relative minor (but non-avoidable) hints from the user. Software-stack hand-shaking should be further streamlined through a collection of Spack recipes that we also provide as an overlaid custom repository.

Not directly included in the SIESTA release, but available on the Siesta Project GitLab repository[5] is a collection of libraries used by SIESTA and made available to other codes (both in MAX and in the wider community). These libraries have also seen significant development and improvements in packaging, sometimes serving as testing ground for CMake idioms that then see further use in SIESTA and other projects.

5.3 Performance improvements

Regarding the performance improvements, we have further developed the interface with the `ELPA` library, both in stand-alone mode and through the `ELSI` library of solvers. `ELPA` offers distributed, accelerated diagonalisation routines which have been ported to various architectures, including kernels for NVidia and AMD GPUs (with Intel GPUs also targeted), as well as for vector instructions such as AVX and SVE. Since the solver stage takes the lion's share of the computing time in SIESTA, this strategy enables the

[5] `https://gitlab.com/siesta-project/libraries`

code to benefit from best-practice external help for its acceleration. Further acceleration of some non-solver kernels is also planned, as they come into view as the solver kernels are optimised.

Parallel efficiency has been enhanced by the removal of some bottlenecks, mostly concerning in the communications involved in the handling of the real-space grid, which represents charge densities and potentials, as well as orbital values for the computation of matrix elements. These operations are linear-scaling, but were suffering from some latency problems that have been addressed through the implementation of asynchronous schemes.

At the intersection of acceleration and parallel operation, we are currently addressing the issue of the optimal balance of MPI ranks per node in relation to the number of GPUs used. Profiling in CINECA's Leonardo machine has shown that proper optimisation of these numbers could increase performance, but proper account must be taken of those solver operations that are only partially offloaded. It should be noted in this regard that new developments in `ELPA`, such as the complete offload of the Cholesky decomposition step needed for non-orthogonal basis sets, targeted for the November 2023 release of the library, can be immediately exploited by SIESTA with basically a recompilation.

The native interface with the `PEXSI` solver has been updated to exploit newer versions of the library, and the `PEXSI` functionality can also be used through `ELSI`, in a flavour which determines the chemical potential by interpolation rather than with root-finding. In general, the `ELSI` interface provides seamless access to various solvers, and is thus a very important asset for the project. The latest release (2.10, November 2023) offers support for Julich's Cease library. Our current interface to `ELSI` is based on the higher-level "density matrix" API, which internally can use any appropriate solver. Not included in this release is ongoing work on the lower-level "eigenvector" interface which can be used in other parts of the code (e.g., band-structure calculation, (partial) DOS, and wannierization calculations). This will further increase the opportunities for offloading and parallelisation over **k**-points when needed.

5.4 New code capabilities and outlook

Among the new physics features implemented in SIESTA, we can mention:

- Full spin-orbit coupling (SOC), including its compatibility with DFT+U;
- Support for the DFT-D3 scheme for dispersion interactions;
- On-the-fly wannierization, which can be used for interfacing to external codes providing further levels of theory, such as DMFT (Dynamical Mean Field Theory).

Not included in this release, but also being worked on:

- Improvements in the QM/MM interface;
- Full merge of the interface to the `PSolver` library, which needed a reconciliation of the work on asynchronous grid communications with other features for grid parallel distributions;
- Extra level of parallelisation of `TranSiesta`;

- SOC capability in `TranSiesta`;
- An update to the API that enables SIESTA to be "called" like a subroutine. This is part of a more general push towards enabling code interoperability in the context of exascale workflows, as discussed in the D2.1 deliverable from WP2.

We conclude this section by highlighting the need for proper, continuous access to EuroHPC machines for development work. SIESTA deployment is reasonably complete on Leonardo but only partial on LUMI and Vega, as access to those machines was granted early in the project for a limited time.

6 YAMBO

YAMBO is a scientific code implementing first principles electronic structure methods based on the Green's function formalism and naturally oriented to the description of excited state properties. As such, it typically exposes methods that are computationally rather demanding, thereby being oriented to exploit large HPC partitions.

Activities: during the reporting period, YAMBO has undergone a large refactoring and relevant development aimed at supporting GPU hardware from multiple vendors, as well as at including new scientific features and fixes.

Availability: several releases of YAMBO have been published, including: `v5.1.2` (mostly bug fixes against `v5.1.0`), `v5.2.0` (new stable features discussed in this document and released to the community), `v5.2.1` (bug fixes). Moreover, we have created the branch `v5.3` as a beta version for the next stable release, including further developments presented below. Developments that are not yet mature are released via dedicated branches of the public YAMBO Git repository.[6]

6.1 DeviceXlib: development and integration in YAMBO

As already mentioned in D1.1, the `DeviceXlib` library has undergone a large refactoring in the first part of the year and in the last months we worked in stabilising it. Most of the work was done in a recent hackathon with the aim of porting the library and the YAMBO code on the `LUMI-G` supercomputer. We list below the main activities done on the library:

- **Support for `Cray` compilers.** The configure procedure was not able to properly recognise the `Cray` compiler via the `ftn` wrapper. So autotools scripts were modified in order to recognise and to set the appropriate flags. Both the ones used for the compiler itself and also for the OpenMP activation.

- **Support for `ROCm`.** The autotools scripts were also modified in order to locate the `ROCm` library and properly link the `rocBLAS` interfaces.

- **Support for rocBLAS.** In the source code of the library, we updated the rocBLAS interfaces and added the support to use the libraries' routines for a subset of BLAS routines. Now, it is possible to use a devxlib wrapper for those routines that completely hides the backend used.

- **Support for profiling tools.** To verify the offload and improve the profiling capabilities on AMD architectures, we added a module with start and stop APIs, having `ROCTX` ranges as a back-end. These ranges improve the readability of traces. With them, we can label a portion of the source code, and visualise its execution on the timeline of `Perfetto`, the preferred tool to visualise `rocprof` traces. By adding ranges, it is easier to identify from which part of the source code events are triggered. Note that these APIs already support `NVTX` ranges for `NSight` and can be improved to support ranges for more backends, such as `Score-P` and `TAU` ones to profile with open source tools.

[6] `https://github.com/yambo-code/yambo`

Regardless of the ongoing development of the library, its integration into the YAMBO code occurred concurrently with the porting to offloading paradigms such as `OpenACC` and `OpenMP-GPU` (see below). This undertaking resulted in a comprehensive clean-up, enhancing the overall code readability. Currently, the majority of memory management activities, particularly those related to device memory, are handled by the library through specific APIs. The detailed instructions for the back-end employed in offloading are encapsulated within the library routines, ensuring that these specifics remain transparent to the developers.

6.2 Portability and single node performance

In this section, we report about the main activities put in place concerning performance portability and intra-node performance, notably including the exploitation of `DeviceXlib` in YAMBO and the extension of available GPU-aware back-ends from `CUDA-Fortran` alone to also `OpenACC` and `OpenMP-GPU`.

- **Porting strategy.** Before presenting in detail the new implementations, here we report a summary of the general GPU-porting strategy followed by YAMBO:
  - Extensive use of pre-compiler macros (`DEV_VAR`, `DEV_SUB`, `DEV_ATTR`) to hide differences across CPU and GPU code, e.g. appending `_d` or `_gpu` to variable and subroutine names.
  - Data representation in GPU memory is obtained either explicitly (as when using `CUDA-Fortran`) or implicitly via host pointers + memory mapping (as with `OpenACC` or `OpenMP-GPU`).
  - Extensive use of `DeviceXlib` is made, allowing YAMBO developers to handle explicitly, though abstractly, data `memcpy` to/from GPU memory, and to perform basic operations on such data (including linear algebra kernels).
  - When specialised kernels are needed, extensive use of directive loop decoration is made (protecting directive sentinels via the pre-compiler macros `DEV_OMP`, `DEV_CUF`, `DEV_ACC`, `DEV_OMPGPU`, see Tab. 2 and source code Listing 1). This already covers the largest part of computational loops in the code.

- **Full support of OpenACC and OpenMP-GPU.** One of the most significant achievements in the development of YAMBO during the last MAX edition was the porting to GPU. The adopted strategy has been thoroughly described in deliverable D2.3[7] of MAX-2, wherein `CUDA-Fortran` was employed for offloading onto the device. Within the first year of MAX-3, we have introduced the possibility to also use `OpenACC` and `OpenMP-GPU` as programming model for GPU support. These three back-ends are all currently available and working. Ideally, if `OpenACC` on NVIDIA GPUs were found as performing as `CUDA-Fortran`, the latter could be dropped (`CUDA-Fortran` being more invasive into the source base). In details, the following actions have been performed:

[7] http://www.max-centre.eu/project-repository

- Full integration of DeviceXlib (see above): the library integration took place when only the CUDA-Fortran porting was completed to ensure replication of the same results and performance.
- The OpenACC porting of the code has focused almost exclusively on the larger loops already decorated for GPU offloading (marked by the explicit use of CUF kernels). The OpenACC directives have then been added to the decoration (see source code Listing 1), which is activated only through the appropriate compilation flag enable-openacc. The porting has been completed for the GW workflow and tested with success on architectures accelerated with NVIDIA devices.
- During the last AMD hackathon organised for the porting of codes to the LUMI supercomputer,[8] significant development work was undertaken on both the autotools and OpenMP-GPU porting fronts. The configuration tool for YAMBO was unable to recognise the Cray compiler through the ftn wrapper. After addressing this issue, similar to what was done for DeviceXlib, support for ROCm and HIP libraries was introduced (see source code Listing 1). Thanks to the adopted strategy (outlined above), the subsequent OpenMP-GPU porting was confined to adding directives for the main code loops. However, during the porting process, challenges were encountered due to the immaturity of the software stack on both the compiler and library fronts.
- A fully working OpenMP-GPU porting of the GW runlevel on AMD GPU-accelerated machines has been obtained.
- The next step will be to verify the OpenMP-GPU porting on the Intel GPU architecture. While we anticipate that the directives may not require significant adjustments, further intervention on the library side will be necessary to enable support for e.g. oneMKL in both Linear Algebra and FFT operations.

In Summary: Following the strategy for GPU porting developed within MAX and outlined above, YAMBO has been largely factorised by exploiting the DeviceXlib library. Both codes are now capable of compiling and running on accelerated machines with NVIDIA devices, utilising both CUDA-Fortran language and the OpenACC directive paradigm. They also run on accelerated machines with AMD devices using the OpenMP offloading programming model. Additionally, DeviceXlib has been successfully tested on machines accelerated with Intel devices. As for YAMBO, the final porting to this architecture is expected shortly.

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

[8]https://lumi-supercomputer.eu/events/hackathon-nov23/

Macro	Behaviour
DEV_SUB	append "_gpu" to the name of the subroutine when GPU offloading is enabled
DEV_VAR	append "_d" to the name of the variable when GPU offloading via CUDA-Fortran is enabled
DEV_ATTR	substitute as ", device" when CUDA-Fortran is enabled
DEV_CUF	substitute as $cuf when CUDA-Fortran is enabled
DEV_ACC	substitute as $acc when OpenACC is enabled
DEV_OMPGPU	substitute as $omp when OpenMP-GPU is enabled
DEV_OMP	substitute as $omp when OpenMP is enabled

Table 2: Legend of macros used by YAMBO code for managing GPU offloading.

```
 WF_symm_i_p => DEV_VAR(isc%WF_symm_i)
 WF_symm_o_p => DEV_VAR(isc%WF_symm_o)
 rho_tw_rs_p => DEV_VAR(isc%rho_tw_rs)
 rhotw_p     => DEV_VAR(isc%rhotw)
 [...]
 !
 ! ordinary implementation
 !
 !DEV_ACC data present(rho_tw_rs_p,WF_symm_i_p,WF_symm_o_p)
 !DEV_ACC parallel loop async
 !DEV_CUF kernel do(1) <<<*,*>>>
 !DEV_OMPGPU target map(present,alloc:rho_tw_rs_p,WF_symm_i_p,WF_symm_o_p)
 !DEV_OMPGPU teams loop
 !DEV_OMP parallel default(shared), private(ir)
 !DEV_OMP do
 do ir = 1, fft_size
   rho_tw_rs_p(ir) = cmplx(conjg(WF_symm_i_p(ir,1))*WF_symm_o_p(ir,1),kind=DP)
 enddo
  !
 if (n_spinor==2) then
   !DEV_ACC parallel loop async
   !DEV_CUF kernel do(1) <<<*,*>>>
   !DEV_OMPGPU teams loop
   !DEV_OMP do
   do ir = 1, fft_size
     rho_tw_rs_p(ir) = rho_tw_rs_p(ir)+cmplx(conjg(WF_symm_i_p(ir,2))* &
                     & WF_symm_o_p(ir,2),kind=DP)
   enddo
   !
 endif
 !DEV_OMP end parallel
 !DEV_OMPGPU end target
 !DEV_ACC end data
 [...]
 !
 ! perform the actual FFT
 !
#if defined _GPU_LOC
#  if defined _CUDA
 call fft_3d_cuda(rho_tw_rs_p,fft_dim,+1,cufft_plan)
#  elif defined _HIP
 call fft_3d_hip(rho_tw_rs_p,fft_dim,+1,hipfft_plan)
#  endif
#else
[...]
 !
 ! q=0, G=0 case
```

	LAPACK (Sequential)	MAGMA 1 GPU	MAGMA 2 GPUs
Matrix creation	17.4s	17.7s	17.3s
Solver	20340.0s	235.6s	178.4s

Table 3: Performance comparison using LAPACK and MAGMA with one and two GPUs.

```
 !
#ifdef _GPU_LOC
 if (isc%qs(2)==1.and.isc%is(1)==isc%os(1)) &
     & call devxlib_memset_d(rhotw_p, cONE,  range1=[1,1])
 if (isc%qs(2)==1.and.isc%is(1)/=isc%os(1)) &
     & call devxlib_memset_d(rhotw_p, cZERO, range1=[1,1])
#else
 if (isc%qs(2)==1.and.isc%is(1)==isc%os(1)) &
     & call devxlib_memset_h(rhotw_p, cONE,  range1=[1,1])
 if (isc%qs(2)==1.and.isc%is(1)/=isc%os(1)) &
     & call devxlib_memset_h(rhotw_p, cZERO, range1=[1,1])
#endif
 !
end subroutine DEV_SUB(scatter_Bamp)
```

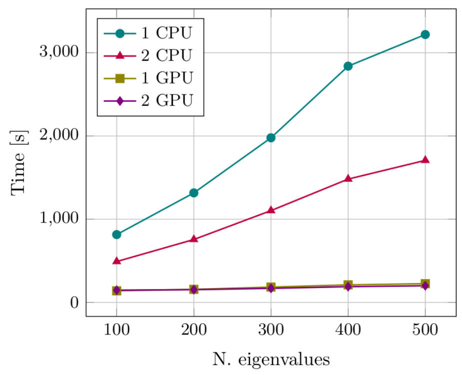
Listing 1: Snippet of source code extracted from the src/wf_and_fft/scatter_Bamp.F routine.

6.3 Parallel efficiency

The MPI communication patter of the GW and BSE run-levels of YAMBO is mainly due to a few reduction operations used to collect and integrate partial contributions to the final result (e.g. the set of quasi-particle corrections). In this scenario, YAMBO is not limited by the communication latency but rather by the load imbalance of the different tasks. A second aspect is then related to the emergence of serial or performance bottlenecks when running at scale. This is the case, e.g., of some linear algebra operations required both in the GW and in the BSE calculations.

We have discussed the load imbalance problem in the report D3.1 of WP3, mostly in connection with the possibility of using workflows with dynamical scheduling of the underlying tasks (and thereby related to the development activities of WP2). In this report we thus present activities focused on the use of external linear algebra libraries, aimed at addressing some of the identified emerging bottlenecks (both in terms of time-to-solution and memory footprint) with special focus on BSE calculations. Finally, we present some benchmarks obtained at scale as a summary of all the developments put in place. In particular, we have performed the following activities.

- **Direct BSE solver (dense diagonalisation).** The direct diagonalization of the BSE matrix is coded via the Lapack and Scalapack libraries, which are not GPU ported. To address this bottleneck, we considered 2 approaches: the MAGMA library, and the cuSOLVER library.
  - The cuSOLVER library (available when working on NVIDIA accelerated machines) is already used by YAMBO to solve the linear system resulting from the Dyson equation for W. In this respect, work is ongoing to use cuSOLVER also for the diagonalization of the BSE kernel. The only limitation



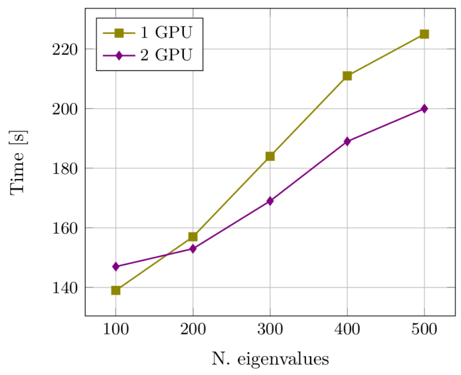


Figure 2: Left plot, comparison of time performance of the Krylov-Schur SLEPc solver with one and two processes CPU vs GPU. Right plot, same comparison with one and two GPUs.

here is related to the lack of a subroutine to diagonalise non-Hermitian matrices (as those resulting from the BSE in the presence of coupling).[9]

– The interface with the MAGMA library was implemented and tested on a machine with 2 Nvidia Quadro RTX 500 cards and 16 Intel Core i9-7960X cores with the Nvidia compiler. Results were compared against a CPU only cluster with 16 Intel Xeon E5-2670 cores per node with the intel compiler. See results in Table 3 from Ref. [27]. Notably, MAGMA also contains a non-Hermitian eigensolver, which can be used to solve BSE problems with coupling (we observed also in this case a remarkable acceleration).

- **Iterative diagonalisation.** Among the solvers of the Bethe-Salpeter equation, YAMBO employs an iterative solver via the SLEPc library to extract few eigenvalues and eigenvectors of the excitonic matrix (the number can be controlled from input). The SLEPc library provides GPU porting and parallelization. As a part of the collaboration with the team of the SLEPc developers, we have been able to use the SLEPc solver on GPUs with very good scaling and minimal changes in the code. The development was tested on the same machines used for the MAGMA library (see Fig. 2, from ref. [27]). There is presently an open pull request to introduce this development into the main YAMBO branch.

- **Advanced diagonalization algorithms via SLEPc.** The SLEPc library (written in C and equipped interfaces for C++ and Fortran) provides a set of perturbative solvers based on Krylov-Schur algorithm. Additionally, it provides an interface with other libraries such as ScaLAPACK (already interfaced with YAMBO) and ELPA. Since the SLEPc library is already linked, minimal changes in the code makes it possible to test different solutions within YAMBO. Also, the handling of the memory distribution over MPI tasks and data exchange is already optimised in the SLEPc library. Working in collaboration with the team of the SLEPc developers, we have further developed the interface to SLEPc in order to remove some

[9]Though these matrices are actually pseudo-Hermitian, YAMBO exploits a general-purpose eigenvalue routine.

bottlenecks on the YAMBO side and to take advantage of some advanced SLEPc features. In particular, we implemented and tested ScaLAPACK and ELPA with this approach. Tests were performed in serial. They did not show any significant difference between ScaLAPACK or ELPA, but highlighted a better handling of the memory compared to the direct interface of YAMBO with ScaLAPACK.

- **Benchmarking and profiling.** This activity has already been thoroughly discussed in deliverable D3.1; therefore, we provide only a summary here. Based on the `JUBE` framework, a script has been developed to launch strong scalability tests for the YAMBO code. Thanks to this script, it is possible to define the system size by tuning relevant input variables. This allows the script to be used for profiling activities as well. Additionally, the ground-state databases required for the benchmark can be calculated through an appropriate workflow or passed directly to the script if already available.

- **Scalability.** As reported in deliverable D1.1 we plan to continuously perform scalability tests making use of a simple profiling system, consisting of a set of clocks implemented in the YAMBO code. The most interesting test that we decided to mention here is shown in Fig. 3. The system studied is a complete GW YAMBO workflow for a Graphene/Cobalt interface (GrCo) composed by a graphene sheet adsorbed on a cobalt slab four layers thick, and a vacuum layer large as the cobalt slab. In the ground-state calculation, performed with QUANTUM ESPRESSO and needed by YAMBO as input data, we set a **k**-point grid $24 \times 24 \times 1$ that led to 61 total **k**-points. In the GW workflow the total number of bands used was 2000, the energy cutoff on the irreducible response function (Xo) set to 20 Ry (both production values) and the multi-pole approximation [1, 2] (MPA) with 10 poles is adopted.

  In this scalability test we were able to scale up to 3000 nodes of Leonardo (87% of the whole Booster partition). We believe this is one of the most important achievements of M12. Being able to utilise almost the entire Booster partition of Leonardo (pre-exascale machine) demonstrates that the YAMBO code is capable of harnessing large portions of an exascale machine, which is one of the main objectives set by our Centre of Excellence. However, concerning scalability, there is still work to be done, particularly to achieve an efficiency of at least 70% at the same level of parallelism achieved in this test. In particular, Fig. 3 shows that the "Other" part of the runs is counter-scaling, and this is the segment of the code that contains the majority of MPI calls.

6.4 Maintenance, sustainability and deployment

In the first year of activity, there was intensive development related to the maintenance, sustainability, and deployment of the YAMBO code within the HPC ecosystem. Some of these activities are already detailed in section 6.2 of this deliverable, so they will only be summarised in the list below:

- **Automatic build system.** Recognition and management of the Cray compiler in the `autotools` system.

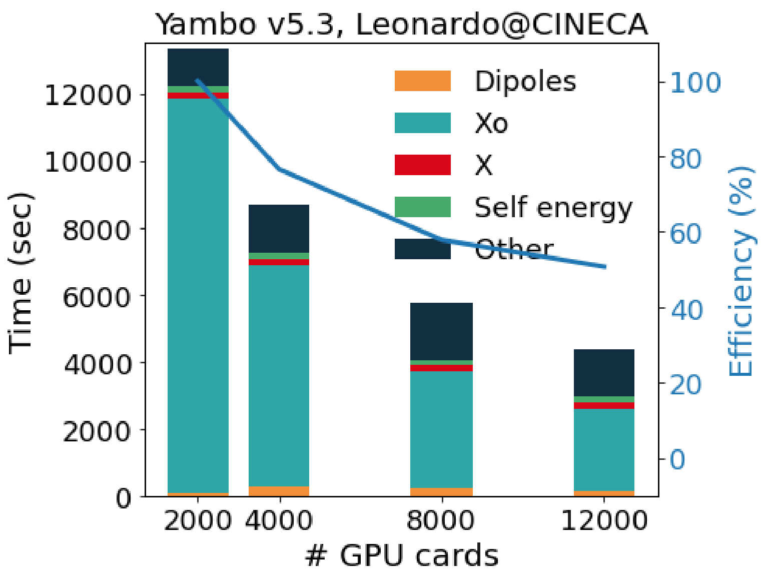


Figure 3: YAMBO strong scalability tests performed on the Leonardo-Booster partition (CINECA). Benchmark use-case: Graphene-Cobalt interface (GrCo) system, including 61 irreducible **k**-points, 2000 bands, 20 Ry cutoff for the irreducible response function, and described at the MPA level using 10 poles.

- **Source abstraction, expressiveness and readability.** In the previous edition of MAX , we embarked on a significant refactoring effort of the YAMBO code with the goal of GPU porting, a topic extensively discussed in past deliverables. However, it is appropriate to summarise the fundamental concepts here. The main objective was to avoid source code duplication for accelerated and non-accelerated versions. To achieve this, we created a preprocessor macro ecosystem that allows us to selectively enable or disable code sections during the compilation phase. Concurrently, a wrapper library (`DeviceXlib`) was developed to encapsulate some of the most frequently performed operations following the GPU porting, specifically the management of memory between the host and the device or within the device itself. This activity experienced significant development during the first year of this MAX edition, as outlined in section 6.2. This development pertained to both the utilisation and expansion of the `DeviceXlib` library and the incorporation of support for two new backends for GPU offloading (`OpenACC` and `OpenMP-GPU`) within the YAMBO code. The expected loss of code readability resulting from maintaining a single source was mitigated and, in some instances, eliminated through the adoption of these two strategies.

- **Package manager support.** A recipe for installing YAMBO has been developed for the HPC software manager `Spack`. Over the past months, this recipe has been refined with a specific focus on compilation for systems accelerated by `NVIDIA` devices. The recipe has been successfully tested, aided by the support of CINECA personnel, on the Booster partition of the Leonardo cluster during the pre-production phase. Another development is scheduled in the near future to integrate support for compilation on accelerated systems with `AMD` and Intel devices.

- **Interfaces for profiling systems.** As previously mentioned, support for profiling ranges has already been added to `DeviceXlib` and is also planned for the YAMBO code in addition to the `NVTX` ranges already supported.

6.5 Interoperability and exascale workflows

New property calculators and capability enhancement

- **Coupled electron-ion dynamics.** The implementation of a scheme dealing with the coupled motion of electrons and ions in non-adiabatic regime (Ehrenfest) has started. The first step has been the implementation of the reconstruction of the non-interacting Hamiltonian (h^0). This implementation has been completed and parallelised (MPI levels). The implementation involved the evaluation of the local and non-local parts of the pseudo-potentials within YAMBO. Besides being a crucial step for the forthcoming development of coupled electron-ion dynamics (see design of exascale workflows, report D2.1), this implementation is also useful by itself. For instance, it allows for the execution of GW calculations on top of any generalised density functional theory (DFT) scheme, including hybrid functionals and DFT+U formulations, which were previously prevented by the need to subtract the double counting of v_{xc}.

- **Beyond-GW self-energies.** Self-energy approximations beyond the GW scheme, as the TD-HF vertex corrections, have been explored [28] in the simple case of spherical atoms. The results have shown that such approximation provides better ionisation energies than the GW approximation. Moreover, they have also shown that new relevant features are introduced in the satellite spectrum, but the experimental spectra are hardly well reproduced due to a missing account of non-linear effects connected to hole relaxation. At this phase, we are still evaluating the possibility to include these developments into the YAMBO code to allow calculation in solid state materials.

- **GW self-consistency.** We have embarked on the implementation of the quasi-particle self-consistent GW (QSGW) method, following the receipt outlined in Ref. [29]. This approach is designed to alleviate the widely recognised issue of initial state dependence inherent in the GW method. The implementation has progressed significantly, relaxing the diagonal approximation for the GW method and leveraging existing modules within the `yambo_sc` project. As the implementation nears completion, our upcoming efforts will be dedicated to rigorous testing and validation over the next few months.

- **Electrostatic embedding of GW and BSE.** Electrostatic embedding has been implemented leveraging the polarisable continuum model (PCM), to account for the effect of the environmental screening on the GW quasi-particle corrections. This implementation has been realised through an interface with the QUANTUM ESPRESSO+ENVIRON library[10]. The new feature is fully parallelised and GPU-ported, and currently, we are validating its accuracy and evaluating its performance on small molecules in various solvents. Preliminary results are shown in Fig. 4.

[10] `https://environ.readthedocs.io/en/latest/`

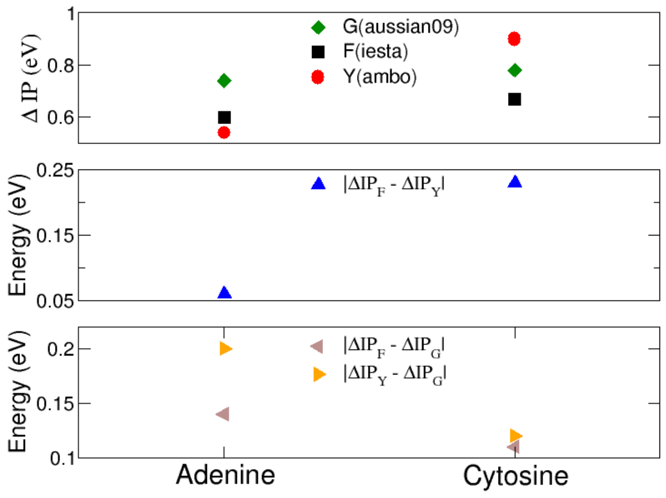


Figure 4: Ionization potential (IP) energy shifts for Adenine and Cytosine immersed in water, obtained with the newly developed embedding formalism. The results obtained with YAMBO are compared with reference data from Ref. [30].

The feature will be made publicly available in upcoming releases. Our immediate focus involves the application to periodic systems, such as nanostructures on surfaces, and the extension of the methodology to the Bethe-Salpeter equation (BSE) scheme.

- **Convergence accelerators for materials in different dimensionalities.** An algorithm based on Monte Carlo integration has been devised to speed up the convergence of **k**-points integration. This algorithm, tailored for 2D semiconductors, is now publicly accessible. Notably, it demonstrates a remarkable acceleration of convergence by orders of magnitude when compared to conventional methods, all while maintaining a consistent level of accuracy. The algorithm performance is extensively detailed in a recently published paper [31]. Building on this success, our current focus involves extending this methodology to address metallic systems. We have successfully applied the algorithm to bulk metals, and work is currently underway to adapt it to treat 2D metallic systems. This ongoing endeavour represents a significant stride toward achieving a comprehensive and versatile approach to **k**-points integration in diverse materials scenarios.

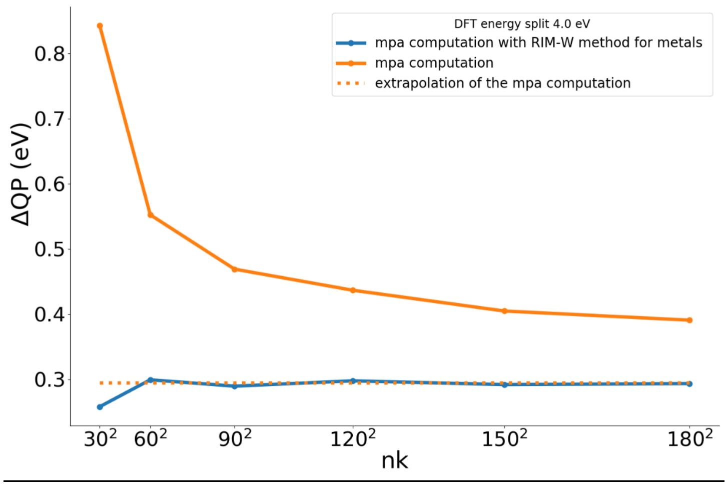


Figure 5: Quasiparticle correction for doped graphene at M point: using the newly developed algorithm, convergence is reached with 60×60 **k**-point grids, while traditional methods fall short in reaching convergence even with grids as large as 180×180.

7 Conclusions

In this document we presented the content of the new MAX code releases and discussed in the introduction the important role that we attribute to these releases as stable productions plateau, both for the HPC users, that can confidently and efficiently use them on most of HPC clusters in Europe, and for the MAX developers that in the next years of work will build on top of them the MAX lighthouse applications and the exascale workflows described in the report D2.1.

The activities where the codes have been mostly active are the search for sustainable support of multiple toolchains, particularly challenging in the case of heterogeneous parallel platforms, and the reorganisation of parallelisation strategies where the presence of GPU accelerators offer new possibilities and pose new technical challenges. In this respect, important achievements have been obtained, with all MAX codes running production-ready on Leonardo, and some of them already deployed on LUMI and ready for Intel-GPU architectures. Moreover, important parallel performance has been demonstrated and documented (notably, YAMBO exploiting up about 90%, 3000 nodes, of the Leonardo booster partition).

The importance and impact of these developments are even more evident in light of the report D3.1 [23] of WP3, where there is a wide description of the various profiling activities carried on for the MAX codes and the many proofs-of-concept that are being worked on by WP3 HPC specialists together with code developers. It is also important that the codes continue the activity of preparing mini-apps and small testing applications that can be used for the profiling and characterisation activities to be done jointly with WP3, and the co-design activities with WP4. Another contact point between this report and D3.1 is the effort, led by CASTIEL, for a sustainable deployment on all EuroHPC machines via the realisation (and exploitation) of a dedicated CI/CD platform. This will be used in the near future to monitor the status of support for all the EuroHPC systems.

Another line of action of the WP1 work in the next months comes from contact points between this document and the D2.1 [24] report, where WP2 outlines the plans and the developments needed for the implementation and deployment of selected scientific workflows. In this context, WP1 and WP2 will work together in the next period to design and implement, within each code, the required interoperability and data-exchange infrastructure (as well to field-test and deploy the workflows). Another important interaction with WP2 will be the implementation of new property calculators necessary for the workflows (already partly descried in the present document).

References

[1] Leon, D. A. *et al.* Frequency dependence in gw made simple using a multipole approximation. *Phys. Rev. B* **104**, 115157 (2021).

[2] Leon, D. A., Ferretti, A., Varsano, D., Molinari, E. & Cardoso, C. Efficient full frequency gw for metals using a multipole approach for the dielectric screening. *Phys. Rev. B* **107**, 155130 (2023).

[3] Mohr, S. *et al.* Daubechies wavelets for linear scaling density functional theory. *J. Chem. Phys.* **140**, 204110 (2014).

[4] Mohr, S. *et al.* Accurate and efficient linear scaling dft calculations with universal applicability. *Phys. Chem. Chem. Phys.* **17**, 31360–31370 (2015).

[5] Mohr, S. *et al.* Efficient computation of sparse matrix functions for large-scale electronic structure calculations: The chess library. *J. Chem. Theory Comput.* **13**, 4684–4698 (2017).

[6] Dawson, W. & Nakajima, T. Massively parallel sparse matrix function calculations with ntpoly. *Comput. Phys. Commun.* **225**, 154–165 (2018).

[7] Ratcliff, L. E. *et al.* Challenges in large scale quantum mechanical calculations. *WIREs Comput. Mol. Sci.* **7**, e1290 (2017).

[8] Dawson, W. *et al.* Density functional theory calculations of large systems: Interplay between fragments, observables, and computational complexity. *WIREs Comput. Mo. Sci.* **n/a**, e1574 (2021).

[9] Mohr, S., Masella, M., Ratcliff, L. E. & Genovese, L. Complexity reduction in large quantum systems: Fragment identification and population analysis via a local optimized minimal basis. *J. Chem. Theory Comput.* **13**, 4079–4088 (2017).

[10] Dawson, W., Mohr, S., Ratcliff, L. E., Nakajima, T. & Genovese, L. Complexity reduction in density functional theory calculations of large systems: System partitioning and fragment embedding. *J. Chem. Theory Comput.* **16**, 2952–2964 (2020).

[11] Genovese, L. *et al.* Protein–ligand interactions from a quantum fragmentation perspective: The case of the SARS-CoV-2 main protease interacting with α-ketoamide inhibitors. *The Journal of Chemical Physics* **158**, 214121 (2023).

[12] Dederichs, P. H., Blügel, S., Zeller, R. & Akai, H. Ground states of constrained systems: Application to cerium impurities. *Phys. Rev. Lett.* **53**, 2512–2515 (1984).

[13] Wu, Q. & Van Voorhis, T. Direct optimization method to study constrained systems within density-functional theory. *Phys. Rev. A* **72**, 024502 (2005).

[14] Ratcliff, L. E., Genovese, L., Mohr, S. & Deutsch, T. Fragment approach to constrained density functional theory calculations using daubechies wavelets. *J. Chem. Phys.* **142**, 234105 (2015).

[15] Ratcliff, L. E. *et al.* Toward fast and accurate evaluation of charge on-site energies and transfer integrals in supramolecular architectures using linear constrained density functional theory (cdft)-based methods. *J. Chem. Theory Comput.* **11**, 2077–2086 (2015).

[16] Stella, M., Thapa, K., Genovese, L. & Ratcliff, L. E. Transition-Based Constrained DFT for the Robust and Reliable Treatment of Excitations in Supramolecular Systems. *arXiv e-prints* arXiv:2106.01142 (2022).

[17] Leiria Campo Jr, V. & Cococcioni, M. Extended dft +u+vmethod with on-site and inter-site electronic interactions. *J. Phys.: Condens. Matter* **22**, 055602 (2010).

[18] Baroni, S., de Gironcoli, S., Dal Corso, A. & Giannozzi, P. Phonons and related crystal properties from density-functional perturbation theory. *Rev. Mod. Phys.* **73**, 515–562 (2001).

[19] Gerhorst, C.-R. *et al.* Phonons from density-functional perturbation theory using the all-electron full-potential linearized augmented plane-wave method fleur. *arXiv* (2023).

[20] Giannozzi, P. *et al.* Quantum espresso: a modular and open-source software project for quantum simulations of materials. *Journal of Physics: Condensed Matter* **21**, 395502 (2009).

[21] Giannozzi, P. *et al.* Advanced capabilities for materials modelling with quantum espresso. *Journal of Physics: Condensed Matter* **29**, 465901 (2017).

[22] Carnimeo, I. *et al.* Quantum espresso: One further step toward the exascale. *Journal of Chemical Theory and Computation* **19**, 6992–7006 (2023).

[23] Affinito, F. *et al.* Interim report on performance analysis of max software (2023).

[24] Garcia, A. *et al.* Exascale workflow design (2023).

[25] Delugas, P. *et al.* First report on max software architecture and implementation planning (2023).

[26] García, A. *et al.* Siesta: Recent developments and applications. *The Journal of Chemical Physics* **152**, 204108 (2020).

[27] Pinto, B. M. Performance improvement of eigenvalue computations in first-principles methods in physics. *Master thesis, Universitat Politècnica de València* (2023).

[28] Vacondio, S. *Assessment of Many-body perturbation theory approximations beyond GW*. Ph.D. thesis, PhD School in Physics and Nanoscience, University of Modena and Reggio Emilia, Italy (2023).

[29] van Schilfgaarde, M., Kotani, T. & Faleev, S. Quasiparticle self-consistent gw theory. *Phys. Rev. Lett.* **96**, 226402 (2006).

[30] Duchemin, I., Jacquemin, D. & Blase, X. Combining the gw formalism with the polarizable continuum model: A state-specific non-equilibrium approach. *J. Chem. Phys.* **144**, 164106 (2016).

[31] Guandalini, A., D'Amico, P., Ferretti, A. & Varsano, D. Efficient gw calculations in two dimensional materials through a stochastic integration of the screened potential. *npj Comput Mater* **9**, 44 (2023).