D1.1

First report on MAX software architecture and implementation planning

Pietro Delugas, Fabio Affinito, Laura Bellentani, Oscar Baseggio, Claudia Cardoso, Ivan Carnimeo, Pietro Delugas, Fabrizio Ferrari Ruffino, Andrea Ferretti, Alberto Garcia, Luigi Genovese, Gregor Michalicek, Nicola Spallanzani, Davide Sangalli, Daniel Wortmann, and Stefano Baroni

Due date of deliverable	30/06/2023 (**month 6**)
Actual submission date	30/06/2023
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	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.	D1.1

Authors	Pietro Delugas, Fabio Affinito, Laura Bellentani, Oscar Baseggio, Claudia Cardoso, Ivan Carnimeo, Pietro Delugas, Fabrizio Ferrari Ruffino, Andrea Ferretti, Alberto Garcia, Luigi Genovese, Gregor Michalicek, Nicola Spallanzani, Davide Sangalli, Daniel Wortmann, and Stefano Baroni
To be cited as	Delugas et al. (2023): First report on MAX software architecture and implementation planning. Deliverable D1.1 of the HORIZON-EUROHPC-JU-2021-COE-01 project MAX (final version as of 30/06/2023). EC grant agreement no: 101093374, SISSA, Trieste, Italy.

Disclaimer

Contents

Executive Summary

This report describes the main developments that will be performed on the MAX codes –BIGDFT, FLEUR, QUANTUM ESPRESSO, SIESTA, and YAMBO– to extend their portability, improve their efficiency and scalability, streamline their deployment and maintenance, enable selected new features and a common MAX interoperability layer. The final objective of this work is to make the codes able to operate at full efficiency on pre-exascale and exascale machines and to pave the way for the realisation of the MAX Lighthouse applications that will be deployed on the EuroHPC clusters. We have structured the four year work-plan into three main stages:

- The initial phase will address all preliminary work needed to ensure the availability of stable and efficient production versions of the codes and develop a software engineering framework to streamline the development activities. We will reach these objectives with the M12 release. We will realise proofs-of-concept, testing, and early deployment solutions for specific machines and architectures. The output of this work will serve as a base for preparing the D1.3 deliverable at M18, which will update this Software Development Plan and conclude the initial phase.
- In the second stage, we will implement the core part of our plan, which, at least as test/beta features, will be part of the M30 release, representing the second milestone of our work plan.
- In the final stage, we will work on the finalisation of our plan, improving code robustness and resilience, and inserting the improvements and technical solutions indicated by the other work packages: WP2 for what concerns the Lighthouse applications and the interoperability, and WP3 and WP4 for fine-tuning and optimisation of our codes and libraries in the different platforms of interest. These outcomes and the Lighthouse application apparel will be delivered with the M48 release.

The activities in the plan have been collected into four Tasks with the following goals:

T1.1 **Extend portability and enhance single-node performance.** This task will extend the GPU support to all devices, toolchains, and architectures of interest for the EuroHPC program (including NVIDIA, AMD, and Intel hardware). Different code groups will follow different paths consistent with the specific code features, converging towards a unified implementation model at the end of the project. QUANTUM ESPRESSO and FLEUR will directly implement the offloading using either openACC or openMP APIs, keeping the two implementations as close as possible. The YAMBO group will work on enhancing and extending the DeviceXlib APIs (jointly with OpenACC and OpenMP) and will use the extended APIs to realise an architecture-agnostic implementation. The SIESTA and BIGDFT groups will focus on the computational libraries, experimenting, integrating, and in some cases developing computational libraries able to support ROCm, HIP and oneAPI back-ends. In particular, about this last point, BIGDFT will work directly at transitioning its computational CUDA libraries to the SYCL programming model using Intel's DPC++ toolchain.

We will also work on other aspects concerning portability and performance portability. We will complete the acceleration of all the code parts of our quantum engines and add GPU support to all the property calculators and post-processing utilities.

T1.2 **Improve the parallel efficiency.** The MAX codes already feature effective MPI parallelisation schemes that, in non-accelerated architecture, may scale up to several thousands of ranks and that, thanks to the accelerators, obtain significantly better performance in heterogeneous nodes with a much smaller number of ranks. Within MAX phase-3, one of our main goals is to increase the parallel efficiency of our Lighthouse applications to several thousand GPUs. Different measures have been devised to achieve this goal, depending on the code. A broad categorisation of these measures includes:

- refactoring and optimising the communications and synchronisation in the lower parallelisation levels;
- reducing the memory footprint on the device memory to enable extensive usage of the auxiliary parallelisation levels;
- implementing dynamical work distribution to remove work imbalances.

In many cases, appropriate parallel libraries will be crucial to achieve the desired parallel performance. As a situation common to all MAX codes, this will be done in particular for the case of GPU-aware parallel linear algebra.

T1.3 **Enhance maintenance, sustainability, and deployment.** This task implements the necessary instruments for integrating and deploying the outcomes of our development activity into the EuroHPC ecosystem. The main points of this action will address:

- Automated build systems based on `autotools` or `CMake` to support all new platforms and features
- Meta-programming, templating and auto-formatting tools and procedures to maintain a unified code base for CPU and GPU accelerated versions of the codes.
- Recipes for package managers for all MAX codes, with support for at least `Spack` and/or `EasyBuild`.
- A testing apparatus to ensure continuous integration and to monitor possible regressions.
- Environment managers, containers, and other automatised deployment tools.

T1.4 **Implement interoperability hooks and new property calculators for the Lighthouse applications.** In this task, we have collected all those implementations more directly connected with realising the Lighthouse applications. These actions will be of two main kinds. One more general and common to all the codes consists of implementing the interoperability hooks designed in WP2 to streamline the data exchange and the interaction of the codes during the execution of the scientific workflows. These hooks will permit the workflow manager to check the status

of each of its components, recover possible check-pointing data, and move large amounts of data between the different applications. In the first stage, the codes group will start some pilot implementations, such as the one of the data exchange between the `phonon` code of QUANTUM ESPRESSO and YAMBO. These pilot implementations will serve as guidelines for the design of the interoperability layer. The details on these interoperability hooks and further interoperability features will then be delivered, within WP2, in the project's second year and delivered with the M30 releases.

The other type of action is more specific. It consists of implementing the new property calculators that will be needed in order to perform the specific tasks of each code inside the scientific workflows.

The plan summarised above will be a starting map for orienting the first stage of the implementation and facilitating interaction with other WPs. More precise details on several points of the implementation plan will be decided on top of the output of the first exploratory phase and leveraging that of WP2 for what concerns the data exchange and the interoperability hooks, that of WP3 for what concerns the technical solutions to adopt for the single and parallel performance and for implementing the software engineering framework of Task 3. Interaction with WP4 will help to explore new and alternative HPC architectures that have not been considered in the first version of our plan.

1 Introduction

The main goal of the third phase of the MAX CoE is to deploy several applications, based on the CoE flagship codes, on the pre- and exascale machines of the EuroHPC program. Such applications will exploit the computing power of these clusters to solve outstanding scientific problems, reaching, thanks to the newly available computing power, scales and accuracy that have been unfeasible up to now.

Since these applications aim at identifying and exploiting the potentialities of the exascale machines and at devising and implementing methods, techniques, and best practices, we have followed the wording of the EuroHPC call referring to them as *Lighthouse applications* in the rest of the text. Some will be directly derived from the current MAX codes and work as a monolithic distributed application. Other Lighthouse applications will instead combine several exascale blocks that will have to be executed concurrently to perform complex workflows whose combined execution has been barren out by the computational cost of some of its components.

The MAX flagship codes are QUANTUM ESPRESSO, SIESTA, FLEUR, BIGDFT, and YAMBO, which were already targeted by previous phases of the MAX CoE. For each code, a summary box is reported at the end of this document describing the main scientific methods and target systems, some technical aspects, and the performance in the HPC environment.

This report describes the necessary measures to be implemented in the MAX codes that WP1 carries out to accomplish the above goals. Apart from realising the Lighthouse applications, these measures also aim at enabling the MAX codes to efficiently use all computing architectures of interest for the EuroHPC ecosystem. In the same way, the part of the work planned to support the exascale workflows will be done in concert with WP2 to improve interoperability with workflow managers.

Work-Package 1 is organised into 4 Tasks, which aim at:

- Ensuring optimal portability and performance on all architectures of interest.
- Improving and, where necessary, refactoring the distribution of data and computation over many nodes to optimise the parallel efficiency, particularly in the case of GPU-based heterogeneous machines achieving a many-node performance comparable to that obtained on single nodes.
- Adopting the necessary software-engineering measures to ensure sustainable deployment, maintenance and development on different systems.
- Implementing new features, communications hooks and robustness measures necessary for the codes to perform the Lighthouse applications and exascale workflows in general.

This document is organised as follows: in Section 1.1 we define the milestones of our program and the timelines for the specific actions to be performed on the codes. The following four Sections analyse the main problems related to each of the four Tasks described above and the solutions we intend to implement; Section 6 discusses the questions this first report leaves open and in an exploratory phase. As will be apparent by the content of these sections, many decisions and details of our developments will eventually depend on the outcomes of the other work packages; for example, all the preparatory

work for the implementation of the scientific workflow presented in Section 5.1 will be fine-tuned thanks to the earlier experimentation done in WP2. The technical challenges about portability, performance, and scalability discussed in Sections 2 and 3 are currently also analysed in WP3 and WP4 in close collaboration with this work package. The results of these interactions and further necessary updates will be constantly reported in the project's continuous reporting documents and delivered in the following scheduled reports and, in particular, in the report of M18 that should represent the final update of this plan.

1.1 Milestones

The whole WP1 development plan spans four years of work that will be carried out within the five distinct code packages corresponding to the MAX flagship codes, building to an overall code base of many hundred thousand lines of Fortran. Another significant part of the work will be aimed at integrating workflows, queue managers and external libraries into the final Lighthouse applications. Given the extent of the plans presented in this document, the timeline provided here is, for many aspects, still only indicative, especially concerning the actions scheduled after M18-24. The timeline will be updated and integrated with the successive reports of the WP1 group. In particular, an update and critical revision of the software development plan will be presented at M18 with a dedicated deliverable.

The three major releases scheduled as deliverables of the MAX project constitute the main milestones in our road map. While we will be able to include in the community releases only those implementations that are production-ready, other intermediate objectives will be associated with the milestones and released via development or beta branches.

The first release, scheduled at M12, will mainly focus on performance and performance portability, interoperability, and the continuous integration and deployment of the MAX codes in the EuroHPC ecosystem. By that time, a software engineering framework enabling the sustainable maintenance of our codes will be in place and constantly upgraded and improved with the project's progress. These latest achievements will be instrumental for the next set of developments, mainly focused on parallel efficiency and implementing new features, bringing to the second major release at M30.

At this stage, we will have completed all the necessary measures to allow our codes to operate efficiently on exascale machines, performing monolithic calculations and operating concurrently within the target scientific workflows. To reach this milestone, it will be necessary to complete the planned parallel efficiency/joint-throughput improvements and implement the necessary property calculators and interoperability hooks. In the programme's last phase, the effort will mainly concentrate on the final realisation of the Lighthouse applications and their deployment and demonstration on the targeted exascale clusters. In this period, we will also work on integrating the technical solutions for inter-node communications and extreme data exchange that will be designed and implemented in WP2 and WP3.

2 Task 1: Portability and single node performance

This task aims at realising a starting base of the MAX codes that can run efficiently on all platforms of interest for the EuroHPC network. As such, it gathers the work done by the MAX CoE to adapt its flagship codes to heterogeneous architectures and implement a portable, architecture-agnostic performance model. The main measures adopted up to now to achieve this target have been (i) the complete encapsulation of GPU offloading thanks to architecture-specific libraries, and (ii) the direct transposition on GPU of the basic MPI parallelisation and data distribution schemes.

The first concept was instrumental for all codes and has been adopted extensively. This has been particularly fruitful for offloading large linear algebra problems and, for some codes (e.g. SIESTA, BigDFT), sufficient to achieve significant performance on heterogeneous nodes. Instead, the second concept has been crucial for those codes in which the data and computation distribution is more pervasive. However, as we will see in the next Sections, it has been found that the simple direct transposition of the basal MPI parallelisation schemes, albeit successful, may limit the scalability over many thousands of ranks due to communication and synchronisation overheads. Measures to mitigate and overcome such limitations have been collected in Task 2.

As for this task, we are working on several distinct lines:

- **Expanding the support** to all devices, tool-chains, and architectures of interest for the EuroHPC ecosystem.
- General improvement of the offloading strategy based on **library encapsulation**, by optimising the initialisation, interfaces, and data transfer and adding support for alternative libraries.
- Improvement and **reorganisation of the offloading** in the high-level parts of the codes by removing the duplication of functions and variables, phasing out CUDA-Fortran and developing and further adopting the `DeviceXlib` abstraction layer.
- **Acceleration of the residual computational kernels** whose offloading was postponed because of their relatively low computational cost.

Most of the effort in the Task will likely be concentrated on the first point. The production versions of the MAX flagship codes currently support NVIDIA GPU cards using specific libraries and, when needed, explicit offloading via CUDA-Fortran and/or openACC programming models. The work to support other cards and architectures while following the same path will first require some adaptation to the different vendor-specific software stacks and, in a second phase, a work of abstraction from vendor-specific details to keep our high-level code source architecture agnostic.

In this respect, we will work on siding the CUDA-library back-ends with those for `HIP` and `ROCm` libraries for AMD cards and `OneMKL` for Intel GPUs. In some cases, Fortran support for such libraries and runtime APIs is still not mature enough, and additional effort is needed to bind directly to C APIs via the native C-Fortran compatibility layer. At a lower level, supporting different architectures involves, instead, either the testing and adoption of the analogous appropriate computation libraries for such systems, the development of such libraries with portable C++ programming models such as SYCL, or the addition of the openMP offloading model to the Fortran library code.

The code porting and abstraction refactoring have been scheduled differently in the MAX flagship codes. For example, the QUANTUM ESPRESSO group has prioritised the openMP offloading, working in a separate *earlier-deployment* branch where the openMP directives are introduced beside the openACC ones. This work proceeds concurrently with work done on the main branch, where the CUDA-Fortran is gradually replaced with openACC. Instead, the YAMBO group works at the `DeviceXlib` abstract APIs, progressively adopting them and directly realising a platform-agnostic offloading.

The work of these two groups is planned to converge towards a common programming model that, leveraging the `DeviceXlib` APIs, will pave the way to implementing a common strategy for unifying the code base with openMP, openACC, and CUDA-Fortran support. Such convergence has been made possible by the joint selection of the GPU offloading operations performed at a high level and by defining the functionally and formally similar constructs used to perform them. `DeviceXlib` APIs will abstract these operations, reducing the code verbosity and simplifying the structure of the pre-processor flags.

One of the first outputs of this programme is expected to be the `pw.x` quantum-engine of QUANTUM ESPRESSO, with a production-ready version to be deployed on LUMI-G by M12. Nevertheless, the final definition of the overall strategy and its adoption plan will be presented in more detail in the D1.3 report at M18. The final goal of this effort is to deploy MAX codes on heterogeneous nodes equipped with AMD and Intel GPU devices and the possibility to use different toolchains in all accelerated architectures.

Regarding the library encapsulation, the actions related to the interfacing with accelerated solvers, matrix manipulation, utilities and GPU-aware MPI libraries will involve all the codes. In many cases, numerical libraries act as the back-end of computational intensive kernels, and optimised interfaces and efficient data offloading – done with CUDA-Fortran, openACC, openMP, runtime APIs or with a general abstraction layer such as the one provided by `DeviceXlib` – are instrumental in streamlining the host-device data movement and reducing the library initialisation overhead. An example is the work planned for QUANTUM ESPRESSO (see Sec. 2.4) for generalising and adapting the 3D-FFT interfaces. Other activities planned for this point include developing or adopting new libraries: for instance, the new SYCL-based Poisson solver in BIGDFT and the accelerated FFT kernel in YAMBO.

The third point involves mainly those high-level code parts where it has been necessary to access and operate on data allocated on the accelerator memory. A significant effort is planned by the QUANTUM ESPRESSO group to remove all CUDA-Fortran constructs at the high-level part of the suite and replace them with less invasive openACC offloading directives. The YAMBO team will possibly follow a similar path. As we have seen above, such action is necessary in order to add the openMP offloading and to adopt the new `DeviceXlib` APIs.

The fourth action point of this Task is instead dedicated to the extension of the offloading within the MAX codes to reduce the non-ported parts of the codes further, thereby reducing the occurrence of possible bottlenecks. For example, the SIESTA group has planned the re-implementation of some parts of the code that currently use sparse-data algorithms, trading lower space complexity for new acceleration possibilities and the adoption of accelerated kernels for the Poisson solver and the exchange-

correlation calculations. Similarly, the QUANTUM ESPRESSO group plans to enable GPU usage in all the linear response parts of the suite. The openACC offloading support should be carried out for the M12 milestone, while full-fledged multiplatform/multimodel support is scheduled for the M30 milestone.

The rest of the section is organised into subsections where we first discuss the work on `DeviceXlib`, and then the porting of each of the Lighthouse codes in more detail.

2.1 An abstraction layer for portability: DeviceXlib

In this section, we sketch the development plan and objectives for `DeviceXlib`, as a tool to hide and abstract GPU-oriented instructions. At present, the library supports multiple programming models (CUDA-Fortran, OpenACC, and OpenMP-GPU), combined with different linear algebra libraries (Blas/Lpapack, cuBLAS, cuSOLVER, MKL, RocBLAS), eventually aimed at different GPU hardware and brands (covering at least NVIDIA, INTEL, and AMD GPUs).

- Further **integration of different programming models**, runtime, libraries. We will implement a common abstraction layer and APIs designed to be as independent of the underlying layers as possible. Work is planned in particular for the CUDA-F, OpenACC, and OpenMP-GPU support currently at different maturity levels, and that needs to be made more homogeneous and mature to be interfaced with the new APIs.

- **Extended linear algebra support**. `DeviceXlib` provides wrappers for common linear algebra (LA) functionality on GPUs (such as BLAS and LAPACK routines). The coverage is limited to a few operations largely used within MaX codes. The goal is to support more LA operations and subroutines without being extensive, providing an abstract set of wrappers.

- **FFT on GPUs**. Wrappers for FFTs with GPU awareness have been identified as a relevant extension to be included in `DeviceXlib`. FFT is pervasive in MaX codes and requires the support of multiple libraries (cuFFT, MKL, rocFFT, at least) when aiming at different GPU brands. Therefore, FFTs represent a clear extension target for `DeviceXlib`.

- **Extended support of custom kernels**. By design, `DeviceXlib` provides abstract APIs for several custom kernels not covered by standard LA libraries, with examples ranging from remapping vector indexes to evaluating specific matrix elements. `DeviceXlib` does not aim at covering all possible kernels encountered in MAX codes, which would be not possible and not meaningful, but rather to provide a set of common kernels that could be called via a subroutine (with automatic support of multiple GPU accelerators) rather than having it explicit in the main code base. Extending the set of covered kernels goes along with developing MAX codes and is a planned activity within MAX phase-3.

- Support for **integration in MAX codes**. While `DeviceXlib` is mainly co-developed within the YAMBO and QUANTUM ESPRESSO communities with the help of HPC centres and HW vendors, the effort will be put in place to support the needs of and ease the integration with other MAX codes. One of the main

advantages of having an in-house developed portability library is the flexibility in accommodating needs and steering developments according to the community needs. In this respect, MAX has a critical mass large enough to support and steer `DeviceXlib` development and integration. A few extra electronic structure codes external to MAX have already started discussions about possibly adopting `DeviceXlib`, which will be supported compatibly with the available PM effort.

2.2 BIGDFT

As a massively parallelised DFT code, BigDFT has been GPU enabled since 2009, utilising the NVIDIA CUDA language. One year later, this functionality has been extended to the usage of the OpenCL standard, developed by the Khronos group, to enable the multi-platform application of the wavelets convolution kernels used by the code. Such a GPU acceleration has then been employed in the acceleration of the Fock operator calculations with the intensive usage of `cuFFT` calls in the *I*nterpolating Scaling Function Poisson Solver kernel. In particular, the expensive evaluation of the exact exchange operator required in the cubic-scaling PBE0 approximation can be offloaded to GPUs to decrease the time-to-solution of hybrid functionals and, consequently, to achieve computing times which are competitive to the less accurate PBE approximation. Clearly, CUDA only allows for offloading to NVIDIA GPUs, which is no longer sufficient considering the usage of other companies' GPGPUs (e.g. AMD's Instinct series and Intel's Max Series) in several of the fastest supercomputers in the world. [1]

There are several cross-platform alternatives to CUDA for the development of heterogeneous codes. For example, OpenMP (with offload capabilities as defined in the OMP 5 standard), OpenACC, OpenCL, and SYCL. The latter is an open standard developed by the Khronos group, released in 2014 to enable cross-platform code development for heterogeneous processors in C++. There are several implementations of SYCL, most notably Intel's DPC++ (part of Intel's OneAPI suite), Codeplay's `computecpp` and Open SYCL. The present developments around the BigDFT code focus on Intel's DPC++ implementation of SYCL for accelerating the Fock operator on multi-platform architectures.

SYCL has several advantages compared to the above-mentioned alternatives. First, SYCL code can be used to run computations in parallel on the CPU, as well as to offload the computations to accelerators (most notable GPUs from AMD, Intel, and NVIDIA), which may result in fewer code paths, thereby reducing the amount of code which has to be maintained. Second, SYCL is a high-level single-source language following the C++ standard minimising the development effort compared to e.g. OpenCL. Third, the performance of SYCL implementations is, in general, highly competitive.

Finally, due to the similarities between CUDA and SYCL (cf. Figure 1), the migration from existing CUDA code is straightforward. In particular, Figure 1 was automatically generated by Intel's DPC++ compatibility tool, which allows for a quick migration of CUDA code to SYCL code, although with the downside that an additional DPCT interface layer is often added, which may result in sub-optimal code. We plan to extend the SYCL implementation of BigDFT based on the automated translation from the DPCT tool which was manually cleaned and optimised to achieve the best performance.

The future guidelines of BigDFT developer groups will therefore be based on such

[1] See e.g. https://www.top500.org/lists/top500/2023/06/

```cpp
//CUDA
__global__ void post_computation_kernel(int nx, int ny, int nz, double *rho, double *data1, int shift1, double *data2, int shift2, double hfac)
{
    int tj = threadIdx.x;
    int td = blockDim.x;
    int blockData = (nx*ny*nz) / (gridDim.x*gridDim.y);
    int jj = (blockIdx.y*gridDim.x + blockIdx.x)*blockData;

    for (int k=0; k<blockData/td; k++) {
        int idx = jj + tj + k*td;
        data1[idx+shift1] = data1[idx+shift1] + hfac*rho[idx]*data2[idx+shift2];
    }
}


//SYCL
void post_computation_kernel(int nx, int ny, int nz, double *rho, double *data1, int shift1, double *data2, int shift2, double hfac, const sycl::nd_item<3> &item)
{
    int tj = item.get_local_id(2);
    int td = item.get_local_range(2);
    int blockData = (nx*ny*nz) / (item.get_group_range(2)*item.get_group_range(1));
    int jj = (item.get_group(1)*item.get_group_range(2) + item.get_group(2))*blockData;

    for (int k=0; k<blockData/td; k++) {
        int idx = jj + tj + k*td;
        data1[idx+shift1] = data1[idx+shift1] + hfac*rho[idx]*data2[idx+shift2];
    }
}
```

Figure 1: Example of one of the BigDFT CUDA kernels compared to the SYCL equivalent. The SYCL code was automatically generated using Intel ® DPC++ Compatibility Tool version 2023.1.0. Note that there is a simpler version of the above SYCL code.

a blueprint, where we will inspect the optimal sources of optimisation in view of (*i*) extending the FUTILE library API to include in the same structures CUDA and SYCL calls to accelerated kernels, thereby enhancing the readability of the host code, (*ii*) measuring the performance figures on the basis of the percentage of the total bandwidth of the node, (*iii*) providing a fully portable programming paradigm which has the ambition to be executed on the present-day and emerging technologies, with limited intrusivity for the developer and minimal guidelines for the user. We plan to use the Just-In-Time programming paradigm (like an underlying OpenCL layer) to explore the portability of the approach.

2.3 FLEUR

Currently, `FLEUR` is mainly optimised for two different computing architectures: the standard Intel/AMD x86 CPU architecture and the NVIDIA GPUs. This was achieved using OpenMP for multithreading on the CPU and OpenACC for GPU programming. We plan to extend this by:

- **Adaption of further programming models for GPUs.** As OpenACC is in practice only supported by a few compilers to the extent we need, in fact, we so far only found the NVIDIA HPC compilers aiming at NVIDIA GPUs able to deal with our code, we plan to extend our code by using, e.g. OpenMP and its offloading features. In this process, we also foresee the option to utilise the features implemented in the `DeviceXlib` and to create an interface between `FLEUR` and this library.
- **Extended use of standard linear algebra.** In the previous years of MAX , we have already successfully adjusted the algorithms used in several computational critical kernels to use standard linear algebra operations, which can be used from highly optimised libraries. We plan to extend this strategy to further parts of the code, particularly the local orbitals.
- **Exploration of calculations with reduced precision.** So far, `FLEUR` uses double-precision arithmetic throughout the full code base. We will explore if this is required for all kernels and all properties to be calculated. We expect that reducing the precision can be tolerated for some workloads and thus lead to increased single-node performance.

2.4 QUANTUM ESPRESSO

The current production version of QUANTUM ESPRESSO features extensive support for heterogeneous nodes based on NVIDIA cards and CUDA-based architectures. This last version, also thanks to the completion of the first actions of this plan, features a stable and efficient offloading implementation for all the main applications of the suite: Quantum engines –`pw.x` and `cp.x`–, `Phonon`, and `TDDFPT` applications. We have started refactoring the CUDA support, removing most CUDA-Fortran parts and replacing them with less invasive OpenACC APIs and directives. The acceleration of residual kernels, the post-processing tools and new features will be strictly worked out directly in OpenACC.

With a much smaller impact on the source code, the OpenACC offloading approach presents a structure almost directly reproducible with openMP, the other model that we will support for offloading. For this reason, this refactoring is a crucial preparatory step for reaching the combined support of these two offloading models that will enable us to use all three main types of GPUs currently on the scene (CUDA, AMD, and Intel) and keep us ready for prompt adaptation to possible future architecture such as RISC-V.

After this initial phase, the work for implementing the support of openMP will take a large part of the effort in this Task. Such porting is currently distributed in a separate branch, whose content will be progressively merged with the openACC stable source. This will happen together with the adoption of the new `DeviceXlib` APIs. We organise the rest of the Section in small paragraphs describing this task converging lines of work for QUANTUM ESPRESSO.

Refactoring and extension of CUDA offloading. The offloading to NVIDIA accelerators is based on three main points. First, the support and usage of several tailored, vendor-specific numerical libraries by our computational libraries. For example, `cuFFT` in `FFTXlib`, `cuSOLVER` in `LAXlib`, the more ubiquitous `cuBLAS` are accessed by the APIs in `UtiliXlib` and, for the future, via `DeviceXlib` new interfaces. As these libraries work on device data, the second component of the implementation is the host-device data mapping and the device data management with capabilities for slicing and transposing the data. This second component is also necessary for high-level code outside the numerical libraries. The third component is using advanced APIs to convert the compute-intensive parallelisable loops into accelerated kernels.

In the first versions of the QUANTUM ESPRESSO suite, all these GPU-related operations were implemented using the `CUDA Fortran` language extension. This choice, albeit efficient, strongly increased the maintenance burden because the `CUDA-Fortran` extension requires explicit name-space duplication for the mapped data, thus duplicating many routines. As the first part of this program, we have gradually moved to the less invasive usage of `OpenACC` directives and APIs. The `OpenACC` directives also have the other strong advantage of being conceptually similar to the `OpenMP5` directives, thus leading to similar final code structures. This simplifies the future porting of the QUANTUM ESPRESSO suite to AMD and Intel GPUs, as described in the following paragraphs.

We are keeping the `CUDA Fortran` APIs in those cases where the name-space duplication brings clear advantages. For example, in the case of Fortran interfaces, arguments with the `device` attribute can be used to resolve at compile time the CPU or GPU implementation of the same interface. In this sense, `OpenACC/CUDA Fortran` interoperability has been extensively exploited for handling the device data structures in the respective parts of the code.

As the refactoring proceeds, the QUANTUM ESPRESSO code is becoming increasingly being structured in a three-layered scheme:

1. An upper layer of code where the GPU managing is based on `OpenACC` directives, where it is possible to develop new code with a very limited effort spent in handling GPU directives;

2. A middle layer of code, mostly hidden in modules and interfaces, where `OpenACC`

and `CUDA-Fortran` interoperability is used, whose main function is to prepare data for the most intensive computational steps;

3. A deep layer of code, mostly based on vendor-specific numerical libraries, strongly depending on the specific hardware.

Noticeably, this structure – inspired by the MAX separation of concerns philosophy– is very flexible and constitutes an ideal background for the future integration of `OpenMP` directives. Furthermore, we expect that the use of the `DeviceXLib` library will further simplify the upper and middle layers of the code, especially when the branches with NVIDIA and AMD/Intel offloading will be merged, and `OpenACC` and `OpenMP` directives will overlap.

Implentation of OpenMP offloading. Together with the `openACC` APIs' adoption, we have also planned the support of OpenMP target APIs so to ensure the portability on GPU architectures (such as AMD and Intel), whose support for `OpenACC` is less mature or even absent. The plans for these developments leverage former collaborations with Intel experts, such as the hackathon organised by Intel in May 2022 to integrate an OpenMP accelerated version of `FFTXlib` into `pw.x`. To avoid conflicts, the openMP porting developments are collected on a separate branch that proceeds in parallel with the OpenACC-CUDA-Fortran refactoring.

In contrast with the `openACC` + CUDA-Fortran approach, when using OpenMP we cannot resolve the GPU-specific implementation leveraging the *device* type for arguments. We plan to implement a fully consistent compile-time resolution mechanism based on the openMP directives `variant` and `dispatch`. This mechanism will be crucial for an effectively agnostic adoption of the different vendor-specific numerical libraries, also because these will be different for AMD –with the `ROCm` and `HIP` back-end– and Intel –with the `oneAPI` back-end. At variance with the `openACC` branch, where we operate the incremental refactoring on a fully effective implementation, the progress on the `openMP` branch is characterised by the gradual addition of accelerated code parts and a consequent increase in code performance.

The first achievement of this effort has been offloading the main computational core for both norm-conserving and ultra-soft pseudo-potentials, thereby covering a wide range of systems for production. Efficient communications for AMD and Intel GPU architectures have been integrated by enabling and testing GPU-aware MPI protocols. Profiling capabilities and code instrumentation with AMD tools (`rocprof` and `roctracer`) have been added to the internal QE timers.

These achievements will be instrumental for the work up to the next milestone at (M12) when we plan to deploy a production-ready version of `pw.x` on the LUMI-G partition. To this end, the offload will be extended to the initialisation and finalisation of PWSCF, and the locality of global variables on the GPU will be increased with unstructured data mapping. The plan further envisages the implementation of an accelerated asynchronous FFTs algorithm in the OpenMP version and the offload of LDA+U calculations for strongly-correlated systems.

After having prioritised the earlier deployment for AMD and Intel architecture, the next steps in the openMP porting will target the unification of the code base, preparing the merge of the `openMP` version into the main development branch. At the M30 mile-

stone, QUANTUM ESPRESSO will feature a well-established and field-tested unique code-base available for deployment for all GPU architectures that support the openMP offloading.

Low level libraries. The performance portability of QUANTUM ESPRESSO is rooted in the lower-level computational libraries (first and foremost `FFTXlib` and `LAXlib`), and a consistent part of the portability work of this plan is targeted at ensuring their support to all necessary back-ends, and at enhancing and streamlining their interfaces. For what concerns `FFTXlib`, the support for NVIDIA cards is provided by the CUDA-Fortran implementation. The openMP offloaded version supports instead the AMD and Intel cards. This version has been developed in collaboration with Intel and AMD developers.

The first target of the actions concerning `FFTXlib` is the performance optimisation of the `openMP` version. One crucial point will be the implementation of the batched mode execution on many wave functions to exploit the throughput capability of the cards better and reduce the data movement overhead by overlapping calculation and communication. This will be done in openMP using task parallelism to match the performance obtained by the NVIDIA counterpart based on asynchronous streams. Another planned effort concerns the refactoring of the library interface with the twofold aim of streamlining the usage of the library by wrapping up the necessary data transposition and removing any reference of the specific back-end from the high-level code base. Such abstraction will also be instrumental in implementing the automatic choice of specific optimisation parameters on different machines and architectures.

The porting of the `LAXlib` library for the linear algebra will rely on rocSolver and oneMKL (for AMD and Intel architectures, respectively) as backend libraries for serial accelerated diagonalisation, in analogy with cuSolver for NVIDIA. In the second stage, the incorporation of a distributed linear algebra library will be studied to deal with large matrices (10k$\times$10k at most) and to optimise their memory distribution. The main `LAXlib` calls will be encapsulated by means of wrappers similar to those of `FFTXlib`.

The OpenMP offloading of other low-level libraries, such as `upflib` and `XClib`, will strictly follow the scheme adopted for the OpenACC model, thereby allowing for both internal and external offloading of the input/output data, depending on the developer choice. The external library `LibXC` will be supported in all cases, but with no offloading while waiting for future developments on its portability.

2.5 SIESTA

The acceleration of the solver stage (the most time-consuming section in a typical execution of SIESTA) is achieved through the use of properly accelerated external libraries. For diagonalisation, we use ELPA, both with a native interface and through the ELSI library-of-solvers interface (which implements further DFT-specific sections of the solver stage). The ELPA library is a strategic asset of a number of high-profile projects (including among the CoEs not only MAX but also Nomad), and is in very active development to support new architectures. It has recently added support for AMD GPUs and SVE vector instructions, and kernels for the upcoming Intel GPUs are in development.

We have recently refactored the original linear-scaling solver to use the DBCSR library as a back-end. This library also offers GPU support. The `PEXSI` solver (see

Sect. 3.4) does not currently have a released GPU-enabled version, but their developers have done some work in that direction.

In SIESTA we plan to:

- Continuously monitor the porting of solver libraries to new architectures.
- Integrate interfaces to support the new architectures.
- Extend the internal use of ELPA to other sections of the code, particularly those related to analysis tools (band structures, density of states, projections, etc).

In SIESTA, the non-solver parts of the code (like the setup of the Hamiltonian) are mostly linear-scaling, as they involve sparse data structures thanks to the finite support of the basis orbitals. But this very sparsity is a problem for acceleration: the typical idiom to walk through the data structures involves indirection in the form of index arrays, destroying the opportunities for streaming in accelerators. Depending on problem size and machine characteristics, it might pay to select a code path in which the underlying algorithm has been changed from sparse to dense. The balance between the low memory use and small operation count of the sparse algorithm and the high-throughput of the dense one can tip in the latter's favour in some circumstances (in particular, when time-to-solution is paramount).

We will explore these alternative algorithms, beginning with the easiest targets (e.g., building of the density matrix from eigenvectors; computation of the (projected) density of states), and progressing to more complex ones (e.g. the computation of the charge density on the grid). The expectation is that the dense algorithms can be cast in the form of calls to accelerated dense linear-algebra routines, and that they can be deployed on different architectures using techniques for transparent offloading, as described in the `DeviceXlib` section of this report. The `TranSiesta` module can also benefit from this kind of algorithm re-implementation to accelerate some of its kernels.

2.6 YAMBO

In the following, we list the main development actions planned for the YAMBO code within the scope of T1.1.

- **Full integration of DeviceXlib**. While `DeviceXlib`is already in use in YAMBO, the library has undergone a large refactoring, with the further abstraction of the exposed APIs and the addition of new functionality, together with the newly available support of OpenACC and OpenMP. In this respect, part of the effort in exploiting the OpenACC and OpenMP programming models within YAMBO (see below) is also devoted to supporting the refactored version of `DeviceXlib`, with full integration of the library within the YAMBO code base.
- **Full support of OpenACC and OpenMP-GPU** programming models. The CUDA-Fortran porting of YAMBO is almost complete for the main parts of the code (e.g. screening, GW, and BSE). It may need further refinement, and porting for most advanced features (e.g. real-time, electron-phonon, etc ...). The main task to be accomplished is the completion and optimisation of the OpenACC (already ongoing) and OpenMP-GPU porting, which requires both integration with `DeviceXlib`

as well as explicit support of OpenACC and OpenMP kernels with the main codebase. Importantly, this has to be done without duplicating the source (a strict requirement) and by keeping the source code as readable as possible to keep the community engaged.

- **Benchmarking and profiling**. Different porting solutions will be implemented and field tested, especially on GPU-accelerated architectures and targeting EuroHPC machines. Importantly, these actions also include comparing different programming models, such as CUDA-Fortran versus OpenACC (but also against OpenMP-GPU, when ready), to clarify whether all models are really needed (performance and portability-wise). For instance, CUDA-Fortran will be removed when equally functional but less disruptive OpenACC constructs are available for NVIDIA GPUs.

- **Testing and optimisations aimed at many-core architectures**. Activities (e.g. loop rearrangement for vectorisation) will be targeted with particular emphasis on ARM-SVE, in view of the release of SiPearl Rhea, AWS Graviton 3 and deployment on EUPEX & The EU pilot for exascale cluster prototypes. Regarding schedule, these activities will have a larger weight in the second half of MAX phase-3.

3 Task 2: Parallel efficiency

This task collects all those coding activities aimed at stepping up the scalability of the MAX codes in order to make them able to exploit efficiently up to several thousand GPUs. MAX codes are, in fact, efficient parallel codes with a demonstrated capability to scale on homogeneous nodes up to several thousand MPI ranks by adopting multiple levels of parallelism.

As seen in Section 2, transposing the lower parallelism level into the accelerators is a straightforward strategy that allows for a quick and efficient adaptation to heterogeneous nodes. As a result, the times to solution are systematically shorter than those obtained with homogeneous nodes. The basal parallel groups reach the best performance with a significantly smaller number of nodes, and the consequent reduction of communication makes it easier to increase the auxiliary parallelisation throughput.

This task will cope with the technical difficulties such approaches present when extended up to several thousand MPI ranks and GPUs. One of the main technical issues is the impact on the GPU-node efficiency of communication and synchronisation. To overcome or mitigate this issue, we will work with WP3 experts to devise and experiment with different technical solutions, such as the adoption of point-to-point communications between GPUs, the use of architecture-specific communication libraries, or the implementation of fine-tuned batching of computations in the accelerators to minimise the impact of synchronisation on the GPU throughput. The plan also includes the implementation of alternative workload distributions devised to avoid splitting the individual tasks into an exceeding number of subtasks.

Another possible significant limitation of the current implementations is the use of one-to-one replicas of the host data structures. This causes an unnecessarily large memory footprint which may pose a bound on the size of systems that can be simulated. In this respect, forcing the distribution of the data structure over a larger number of GPUs

may worsen the communication issues and hinder the usage of the auxiliary parallelisation levels. To mitigate these issues avoiding significant performance impairments, we have planned actions for increasing the granularity of the offloaded data structures and accurately fine-tuning the host-device data copies.

For all of these measures, a first period of analysis and experimentation is scheduled for the first year of activity. The results of this analysis will be reported in WP3 deliverables and for what concerns WP1 actions in the M18 update of the software development plan.

Another crucial issue for the parallel performance of all the MAX codes is the capability to operate in joint throughput on very large distributed matrices with the full range of operations available and well-performing. Being a very general need of HPC applications, many domain-specific libraries are available. This task will mainly work on maintaining and expanding the support for the various libraries available. The rest of this section is organised into specific subsections that will discuss the work on each code in detail.

3.1 BIGDFT

BigDFT is a wavelet-based density functional theory (DFT) code with two main operation modes: an approach that scales cubically with the number of atoms and a second one that is linear scaling (LS). In the LS approach, the total work, divided among MPI tasks, grows in proportion to the number of atoms, so large core counts are more naturally accessible to LS-BigDFT than cubic scaling approaches. The overall aims of the forthcoming work about BigDFT will address the limitations identified in the current MPI communication scheme it relies on and will improve the performance of its lower-level OpenMP parallelisation.

In the context of the MAX CoE, the starting point for successfully exploiting the parallelisation of BigDFT is the weak-scaling limit of the LS algorithm of the code. This would enable us to identify the resources needed for the calculations in various platforms and system categories on a per-atom basis. As an illustration, starting with a crystal of 2CzPN containing 1000 molecules (54000 atoms), we extract subsets of various sizes and launch jobs on the target machine with the desired number of nodes and threads.

For example, we show the performance of BigDFT on the HPE Apollo2000 Gen10 Plus supercomputer located at the Research Center for Computational Science in Okazaki, Japan. Each node has two AMD EPYC 7763 processes with 128 cores. We used 8 nodes for each calculation, with 16 MPI tasks and 8 OpenMP threads. The CPU time per atom is reported in Figure 2. We observe that we quickly reach the linear scaling regime even with around 1000 atoms.

LS-BigDFT was designed to exploit HPC from the outset, with a long-standing hybrid MPI/OpenMP approach. However, it has been several years since the performance of the low-level OpenMP parallelisation has been assessed, with many of the routines having been designed or tested only on low thread counts. Currently, the OpenMP speedup typically saturates at around 4 threads, leaving scope for improvement for runs using 8-16 threads/MPI. This performance, though, strongly depends on several simulation parameters. We plan, therefore, to perform a detailed OpenMP profiling of a set of typical BigDFT runs, resulting in a short list of performance-limiting routines, where there is no existing OpenMP, or the current OpenMP implementation is inefficient. The OpenMP

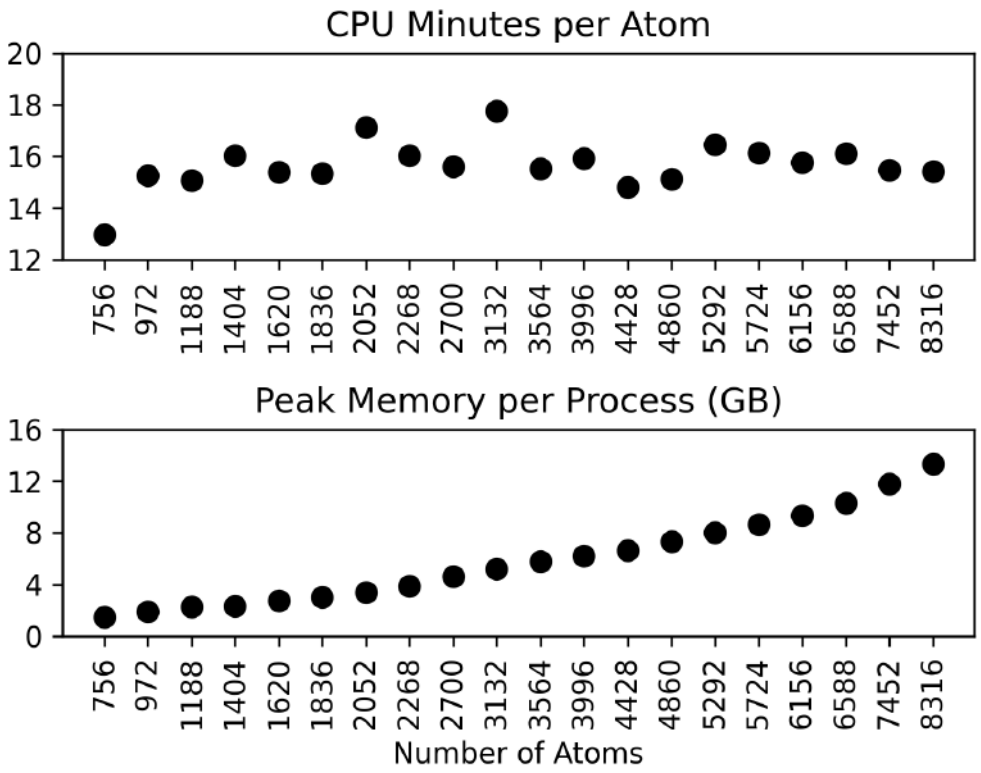


Figure 2: Calculation time and memory consumption (per process) of calculations of 2CzPN clusters of increasing size.

OpenMP Threads	1	2	4	8	16	1	2	4	8	16
Initialisation	1.2	0.9	0.6	0.4	0.4	1.0	1.4	2.1	2.8	3.3
Wavefunction Opt.	16.8	11.8	7.1	4.8	3.8	1.0	1.4	2.4	3.5	4.4
Total	18.1	12.7	7.7	5.3	4.2	1.0	1.4	2.3	3.4	4.3

Table 1: Internal profiling (left) of times and speedups (right) for 10 CBP molecules (620 atoms) on ARCHER2 for LS-BigDFT, using 128 MPI tasks and varying the total number of cores.

parallelisation for these routines will then be rewritten.

Since the choice of the number of nodes to be used for a given run is often dictated by memory limitations, not just parallel efficiency, these improvements aim to better exploit the computing power of new architectures while also increasing the upper limit in the number of cores which can efficiently be used for a given problem size. The OpenMP performance depends on several factors, and different routines are affected differently. For instance, the initialisation routines show worse speedups than wavefunction optimisation (see Table 1) and take proportionally more time for fragment calculations, influencing overall performance. It is, therefore, essential to perform benchmarks for different setups and system sizes, including LS and fragment, free and periodic boundary conditions.

Based on the results, the routines with the worst OpenMP performance that take a noticeable portion of the runtime will be targeted for improvement. The exact details of changes will depend on the identified routines and may include routines which are specific to the LS code, as well as those which are shared with cubic scaling BigDFT. However, changes are expected to include adding OpenMP to routines where it does not already exist, and changing the scheme used, e.g. replacing uses of `OMP` section, which are restricted to 2 threads only, or testing different approaches to thread scheduling.

3.2 FLEUR

The performance of `FLEUR` relies on several different levels of parallelisation. While OpenMP/OpenACC is used on a single node, MPI distributes the calculations over several nodes. The MPI parallelisation for a standard SCF calculation distributes the computational load by using two different schemes: (a) the different $\mathbf{k}$-points are calculated on disjunct sets of MPI tasks, and (b) the individual eigenvalue problems, the matrix setup and the charge density generation can also be distributed over several MPI tasks. Most of the work is devoted to this second level of parallelism since $\mathbf{k}$-point parallelism is relatively trivial to implement with high efficiency. In contrast, special workloads like, e.g. calculations using hybrid functionals can profit from further levels of MPI parallelisation like distribution of the work done for so-called $\mathbf{kq}$-pairs. To further boost the parallel efficiency of `FLEUR`, we plan to address the following actions:

- **Exploration of iterative solvers.** Iterative solvers for the diagonalisation of the Hamiltonian are usually not employed in the case of dense matrices, as we have to deal with in `FLEUR`. However, these solvers can often be parallelised more efficiently, and hence the slightly higher computational cost could be overcompensated by better scalability. While we have some experience with an older imple-

mentation of the Chase solver, we plan to build on this experience and to extend the implementation.

- **Streamlining of algorithmic dependencies.** In some cases, the calculation algorithm could be simplified by moving code and data structures. This could eliminate the need to store all eigenvectors and the corresponding communication needed to distribute this data. Hence, for larger simulation setups, this could decrease the memory footprint and lead to better scalability.

- **Additional levels of parallelism.** Ultimately, as the diagonalisation of the eigenproblem is the most computationally relevant part of a standard SCF calculation, the scalability of the code is impacted by the limited scalability of the corresponding algorithms. To overcome this, we explore different ideas to expose further parallelism inherent in typical workflows. These could include different strategies to achieve self-consistency and the overlay of different tasks in a calculation workflow. While we have some ideas to explore, these must be tested and concretised during the project.

3.3 QUANTUM ESPRESSO

The developments on QUANTUM ESPRESSO for this task will be organised into 4 main lines of action aimed at improving the parallel efficiency and the scalability of the suite codes at various levels.

- Basal **R&G parallelism**. The parallelisation at this level is made by distributing the 3D data sets that describe the densities, the potentials, and the wave functions together with all the operations performed on these objects. In this case, the speedup is obtained by the general reduction of the latency times of such operations and, in particular, of the time taken to perform the 3D Fourier Transforms on the wave functions. This approach is, though, communication greedy and relies on many synchronisation points that may impair the efficiency of the joint-throughput operations of the GPUs. In collaboration with WP3, we have planned several measures to mitigate the impact of communication and synchronisation and to fine-tune the wave-functions FFT batching to optimise the overlap between the communication and calculation within the accelerators. The main actions planned are:

  - Preliminary profiling of `FFTXlib` on GPUs, in collaboration with WP3, for the individuation and definition of the critical bottleneck for communication and synchronisation. Determination of the communication overhead trend in relation to the number of batched FFTs, and the optimal overlap between the communication and computation. (M12)
  - Replacement of collective all-to-all communications in `FFTXlib` with non-blocking point-to-point communications. (M30)
  - support and adoption of alternative communication libraries, e.g. NVIDIA NCCL and other specific technical solutions indicated by WP3. (M48)

- **Band parallelism**. Complementary to the work planned at the point above to reduce the latency times for processing wave-function data sets, we have also planned

actions for scaling up the calculations by increasing the throughput of the operations on the wave functions. This will be done by distributing distinct blocks of wave functions among different MPI groups. In principle, this strategy can be pushed up the complete removal of the `FFTXlib` communication overhead by reducing the band group size to a single rank using a single GPU or significantly mitigate it by restraining the band groups to a single node, or in case limiting the number of MPI ranks.

To implement this strategy effectively, we have planned a full refactoring of the band parallelisation in QUANTUM ESPRESSO. The first objective of these changes is the complete distribution of the wave-function datasets. This will require the refactoring of many high-level kernels so that they can operate on the wave function manifold in blocked mode instead of the monolithic access implemented so far. This more granular data structure will also allow us to reduce the memory footprint of the wave functions on the GPU memory. The steps of the program for this point are:

- Pilot refactoring of the Gram-Schmidt orthogonalisation kernel to make it work in blocked mode with distributed data and asynchronous device-host memory copies. (M12)
- Implementation of blocked operation with distributed data for the main iterative solvers in `KS_solvers` and `RMM-DIIS`. (M30)
- Extension of blocked band parallelism to all solvers and with real space projectors. (M48)

- **Auxiliary parallelisation levels.** The *pools* and *images* distributions allow us to execute concurrently the many semi-autonomous blocks in which the calculations can be decomposed. These parallelisation levels increase the general throughput of the calculation. Several measures have been planned to extend and streamline the usage of these distributions with GPUs:
  - For what concerns the *image* parallelism, the current plans involve mainly the `phonon` code and aim at streamlining the joint execution of the images, the improvement of check-pointing and restart, data sharing between the images, and dynamical scheduling of the work. Leveraging the experience gained with this work, further actions for the `TDDFPT` codes will be added in the plan integration. (M30)
  - For what concerns the *pools* parallelism, the planned improvements will be general and extended and will aim at reducing the footprint of this type of parallelism on the device memory and at making more efficient the oversubscription of the single GPUs so to increase the available granularity of the *pool* distribution. Because the impact of these improvements will be mostly for relatively small calculations, requiring a small number of nodes, they will be carried through with a lower priority and planned for completion at M48.

- **Distributed linear algebra.** QUANTUM ESPRESSO engines also feature an autonomous MPI group to distribute and operate on large matrices. `LAXlib` APIs

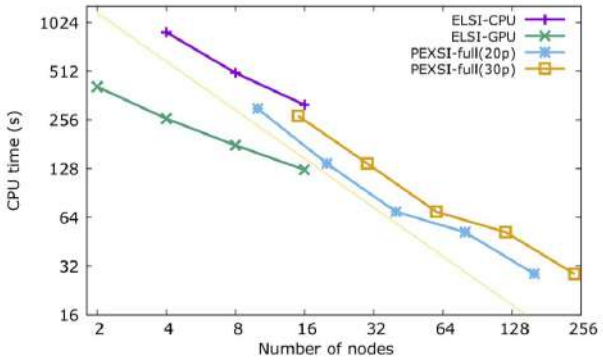


Figure 3: Time to solve the diagonalisation problem corresponding to a piece of sars-cov-2 protein surrounded by water molecules, with approximately 58000 orbitals. Two sets of PEXSI results (for 20 and 30 poles) are shown. The thin line shows the ideal scalability behaviour.

provide the data structures and APIs this parallelisation exploits. The current production version of QUANTUM ESPRESSO doesn't use distributed linear algebra in the presence of GPU-acceleration. All matrices are allocated and operated on a single GPU. Because of calculations with large datasets and to distribute the memory footprint of these matrices, in the third phase of MAX , we have planned several actions to enable and incrementally expand the usage of distributed linear algebra. The planned work and the related timeline are:

- Release of full-fledged support of ELPA GPU solvers in the first MAX release. (M12)
- Add support for other distributed solvers (including, e.g., MAGMA, SLATE, CHASE, NVIDIA cuSolverMG, etc). (M30)

3.4 SIESTA

Siesta's baseline efficiency (regarding resources used for a given system size) is rather high compared with codes that use plane waves. It is seldom needed to employ a very large number of nodes to achieve a reasonable level of performance (measured in terms of time-to-solution and overall cost). Moreover, for very large systems, SIESTA can profitably use the PEXSI solver, not based on diagonalisation but on a pole expansion that has a "scalability reserve" coming from several levels of parallelisation (over orbitals, poles, and values of the chemical potential for interpolation).

A comparison of the (accelerated) diagonalisation solver and PEXSI for a relatively large system was presented in the D4.3 deliverable of the second phase of MAX , and the key figures are reproduced here in Fig. 3 as they point to further developments in the current phase. Next, in Fig. 4 we report total-cost vs time-to-solution, where perfect scalability is represented by horizontal lines, and a non-zero slope measures the marginal cost of improvements in time-to-solution. It is clear that the diagonalisation, particularly the accelerated one, could use some work.

Further work on the scalability of the accelerated diagonalisation solver is, of course, dependent on developments in ELPA, which is our key workhorse for distributed eigenproblems. Still, some actions can be taken independently of those developments, such

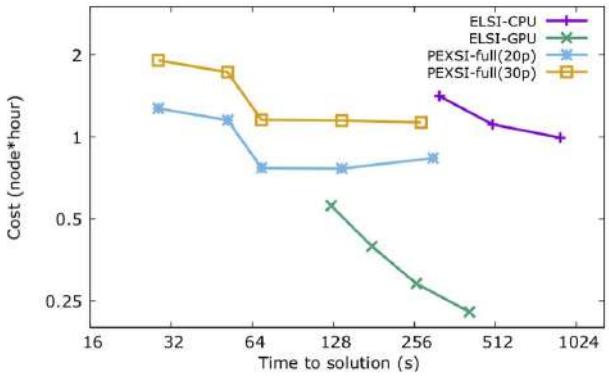


Figure 4: Total cost (per scf step) vs time-to-solution for the virus protein problem. The `PEXSI` lines correspond to different numbers of tasks per pole (from right to left: 8, 16, 32, 64, 128)

as researching the right combinations of numbers of MPI ranks and the number of GPUs used per node. In this task, we will be helped by the analysis of traces for parallel operations to be performed by our BSC colleagues. Further improvements in the parallelisation of the already performant `PEXSI` solver will probably come from the technical advice to be provided by WP3. If, as mentioned in Section 2.5, `PEXSI` is enhanced with GPU support, the same kind of analysis as for diagonalisation will be carried out.

Multiple levels of parallelization are one important route for scalability, as exemplified by the `PEXSI` solver. One section of SIESTA that can benefit from multiple distribution levels is the `TranSiesta` module, which currently is parallelised over the energies used for integration in a complex-plane contour, but not over orbitals.

3.5 YAMBO

YAMBO has proved to scale efficiently both in homogeneous and heterogeneous architectures but some issues remain to be tackled regarding extreme scaling.

- **Memory footprint.** The overall memory footprint is still one of the limiting factors of a typical YAMBO calculation, especially in the presence of GPUs. We intend to address both the memory distribution and the memory buffers by increasing the granularity of data structures. This requires the implementation of alternative distributions of the workload and the handling of the host-device data copies (e.g. device WFs buffering systems on the host, or other non-volatile memory).

- **Load imbalance.** YAMBO makes use of different levels of parallelization. Examples include transferred momenta, number of bands, number of **k**-points, and space variables for the linear response runlevel or transferred momenta, quasi-particles, and number of bands for the calculation of the GW self-energy. Some of the levels can be considered embarrassingly parallel because they are almost independent, but indiscriminate use can lead to unwanted behaviours. For instance, the different symmetry of the **k**-points can result in a very different computational cost of the respective calculation. This leads to an imbalance in the workload and consequently less than optimal scalability and a larger time-to-solution. We intend to

implement schemes to improve the load balance, considering, for example, primary/subordinate or dynamical scheduling programming models or a re-balancing of the tasks.

- **Diagonalisation schemes.** BSE calculations mathematically require solving a large eigenvalue problem, through the diagonalisation of the whole Bethe-Salpeter matrix. Depending on the approximations used, the matrix can be Hermitian (within the Tamm-Dancoff approximation) or not, requiring different numerical approaches. At present, the schemes implemented in YAMBO are: (1) diagonalisation of the full Hamiltonian using the standard (SCA)LAPACK library, (2) subspace iterative Lanczos algorithm which by-pass diagonalisation, and directly gives the dielectric function, with the Haydock approach, and (3) subspace iterative algorithm using the SLEPc library which can extract a limited number of eigenvalues and eigenvectors of the BSE matrix.

  However, a systematic porting on GPU of these diagonalisation solvers has still to be performed. Regarding the first diagonalisation scheme, we intend to implement in YAMBO the GPU interface to non-Hermitian solvers available through the MAGMA library. The remaining schemes will either be ported to GPUs or replaced by alternative approaches. We are also in touch with the developers of the SLEPc library to address this point.

- **GPU-aware distributed linear algebra.** GPU-aware distributed linear algebra libraries are software frameworks that provide optimised implementations of distributed linear algebra operations, taking advantage of both GPU acceleration and distributed computing systems. These libraries are designed to efficiently handle large-scale linear algebra computations by distributing the workload across multiple nodes and utilising the parallel processing capabilities of GPUs.

  In order to exploit these approaches, some considerations need to be taken into account: (i) the library has to adopt efficient communication mechanisms between the GPUs and the computing nodes to exchange data and intermediate results; (ii) splitting the data appropriately across the distributed nodes is essential for efficient GPU usage. By taking these two aspects into consideration we will test YAMBO with libraries that will be available on the market (examples include NVIDIA cusolverMp, ELPA, Slate, etc) in order to implement the inversion of the matrix $\chi^0_{\mathbf{G},\mathbf{G}'}$ and the BSE diagonalisation in distributed fashion.

- **Scalability.** In order to assess the effect of the implementations described in the previous points and the remaining or new bottlenecks of the code, we plan to continuously perform scalability tests in the different architectures available. For this, we will make use of a simple profiling system, consisting of a set of clocks implemented in the YAMBO code, and memory tracking for both host and device. When running in parallel, every process (by default), or a subset (if specified), writes a log file. The analysis of these files will help identify any workload imbalance and non-optimal memory allocation.

4 Task 3: Maintenance, sustainability and deployment

The wide set of activities and developments detailed in this plan will significantly change the internal working and logical flow of the MAX flagship codes. Indeed, these actions will support new features, plugins, and data-exchange methods and the possibility of the codes to be deployed in many diverse architectures. Adopting and implementing software engineering best practices will be crucial to hold together and monitor the correct functioning of all these additions for all the supported architectures. To this end, the present Task is dedicated to the above actions and is mainly aimed at:

- Implementing an **automatic build system** support for all new features in all architectures targeted for the deployment. This will be done for all codes using either `autotools` or `CMake` systems.
- Experimenting with meta-programming, templating, and auto-formatting strategies that can enhance **source abstraction, expressiveness and readability**. The main and critical goal here is to maintain a single code base for CPU and GPU-accelerated sections of the codes.
- Providing **package manager support** for MAX codes deployment using e.g. `Spack`, `EasyBuild`, or `JHBuild`. This should streamline the automatic deployment of the Lighthouse codes on HPC centres, including EuroHPC ones. This is essential since, given a building system, it is still necessary to map the dependencies of the code to appropriately installed libraries, whose location and characteristics might vary from system to system. Ideally, EuroHPC centres could implement a minimally standardised "software stack discovery wrapper", to facilitate this task. For example, even if a centre is using `EasyBuild` to manage the installed software stack, it could still provide a set of `Spack` configuration files that define the appropriate compilers, MPI subsystems, and optimised linear-algebra libraries. The design of this wrapper should be a collaborative task between centres and code owners.
- Building and maintaining a testing apparatus to ensure **continuous integration** (CI) and monitor possible regressions and the performance evolution for all architectures of interest.
- Providing **interfaces for profiling systems** that are abstract enough to work on multiple architectures and system installations. This naturally links to the items on meta-programming and testing, but it is important enough to merit special consideration.
- Working at the integration of the Lighthouse applications in environment managers, containers, and other **automatised deployment tools** (CD).
- Provide regular **software releases** of the MAX flagship codes.

While the general actions listed above will be common to, and implemented by, all MAX flagship codes, specific actions or refinement of the above may also be considered by each code individually. Examples include the streamlining of the IO software stack used by the YAMBO code (aimed at focusing further on the HDF5 library) or the integration of

accelerated solver libraries with all their ported options into the interfaces and the build system to be done by SIESTA.

5 Task 4: Interoperability and exascale workflows

As the proposal states, this Task aims at actions on the codes to successfully deploy exascale workflows, emphasising the application showcases (scientific grand challenges). These actions fall into different classes:

- The **implementation of new property calculators** (e.g., an improved description of thermal or electronic transport, a new spectroscopy, or a new formulation of coupled electron-ion dynamics);
- The **provision of interoperability hooks** in individual codes to allow for their seamless integration into larger workflows when needed (jointly with T2.4);
- The **technical developments** required to treat special classes of systems (e.g., addressing memory footprint for large-scale simulations or treating the presence of dielectric boundaries) since scientific use cases are also going to evolve with the advent of exascale machines;
- The **re-organisation of the I/O of intermediate data** to improve on check-point restarting and code resilience (jointly with T2.4).

Most of these actions strictly depend on developments and design decisions made in other work packages, particularly WP2 and WP3. Hence their insertion in the WP1 software-development plan must be, by necessity, only indicative. Only the development of new property calculators and other capability features can be mapped with some level of confidence at this time, and even there, unforeseen new developments are still possible. The first subsection deals with the latter on a code-by-code basis. The remaining actions are outlined in a separate Section dealing with interoperability and system interfaces and will be detailed in revising this software-development plan at M18.

5.1 New property calculators and capability enhancement

5.1.1 Quantum ESPRESSO

Several new fundamental features will be added to the QUANTUM ESPRESSO suite to pave the way for the implementations of the non-adiabatic molecular dynamics (NAMD) workflow of WP2. The two main features, the excited-state energy gradients and the excited states non-adiabatic couplings will be developed and implemented in the first part of the project and delivered for the production version at month 30. These property calculators will leverage the current `turbo_davidson` code [1], which computes explicitly the electronic excitations (by contrast, the `turbo_lanczos`, although faster, only provides spectral lineshapes) and their response orbitals.

Given that, routine calculations with NAMD algorithms require many evaluations of gradients and non-adiabatic couplings along many trajectories associated with the different photochemical and photophysical decay pathways, it will also be crucial to first have an efficient porting to GPUs of `turbo_davidson`, and enabling the image parallelism

the TDDFPT applications. After these preliminary steps, we will cope with the computation of the derivatives of response orbitals [2], even using a Z-vector approach [3]. This is expected to be the bottleneck of the calculations and will need careful analysis and optimization with a specific implementation for GPUs. The implementation timeline of these features will be:

- Efficient `turbo_davidson` implementation for the CUDA/OpenACC backend. (M12)
- Porting of `turbo_davidnon` to AMD cards with the openMP backend. (M24)
- Enabling *image* parallism in TDDFPT codes. (M30)
- Calculator of excited states energy gradients. (M30)
- Calculator of excited states adiabatic-couplings. (M30)

5.1.2 Siesta

One first group of feature improvements addresses the capabilities related to the ions and electrons dynamics, which is needed for the "sampling" side of workflows. SIESTA is well-positioned for this role due to its intrinsic efficiency for large systems and its implementation (in `TranSiesta`) of non-equilibrium methods for biased systems.

- Implement a new parallelisation level in `TranSiesta` to enable large-scale systems simulations under electrical bias.
- Further development of the TD-DFT subsystem, including non-adiabatic dynamics.
- A more tightly integrated QM/MM capability for large systems.

A further class of features involves high-efficiency analysis tools. Currently, these are mostly implemented as post-processing utilities, working on data on disk produced by a previous execution of the program. Their re-implementation as modules with a pipelining capability will allow their insertion in workflow analysis or screening phases. A singular example of this class of methods is the electronic structure analysis to define fragments in the manner done by the BIGDFT project. SIESTA shares with BIGDFT the use of basis sets with localised support functions and can employ similar methods for the fragment-search task. This will enable interoperability with the workflows based on fragment analysis.

5.1.3 BigDFT

The planned activities of the BIGDFT development group in the current project are mainly related to the extended usage of complex workflows employing linear scaling (LS) calculations. Therefore, the main effort in this context will be related to the serialisation of the descriptors, which can be extracted from the density matrix of large-scale calculations with biological systems. The serialisation of such descriptors into suitable

data structures will enable the efficient usage of workflows at the exascale regime for the characterisation of the interactions of biological systems at the DFT level.

Another relevant point will be extending these calculations to higher levels of theory, like hybrid functionals, which can benefit from accelerated platforms. A proof-of-principle of the method has already been implemented in the context of the previous MAX project and will be used as a ground basis to verify the potential sizes that can be reached with those calculations.

The BIGDFT team is also working on several complementary developments, including `PyBigDFT`'s remote runner approach, which incorporates job submission into `Jupyter` Notebook workflows. This is being tested on a range of supercomputers, and it will be ensured that it works smoothly on EU HPC Architectures.

5.1.4 Fleur

For the `FLEUR` code, we are currently focusing on different types of calculations in which exascale computing could have a significant impact. Significant effort is planned along the following lines.

- **Hybrid calculations.** The newly refactored implementation of the hybrid functionals in `FLEUR` is a natural application case for exascale computing. Having already demonstrated their scalability, their interoperability with several other important features is still to be improved. In particular, combining hybrid functionals with spin-orbit coupling is an important goal to extend the functionality so that exascale workflows using this combination of features become possible.

- **Density functional perturbation theory for phonons.** An implementation of density functional perturbation theory for phonons is currently added to the `FLEUR` code. The numerical challenges are worked on, and the basic algorithm is mostly finished. In the future, we plan to enhance the performance of this feature and use it to calculate further properties, e.g., electron-phonon coupling.

5.1.5 Yambo

This Section presents the topics and features for implementing new property calculators within the YAMBO developer community. In terms of physical features:

- **Coupled electron-ion dynamics.** This covers a large umbrella of approaches dealing with the dynamics of electrons and their excitations with atomic motion. In particular, we will consider going beyond the Born-Oppenheimer approach by implementing a non-adiabatic classical dynamics (e.g. Ehrenfest) of ions coupled to the electronic degrees of freedom. This implementation will then be used in a related scientific demonstrator.

- **Electron and exciton coupling to phonons.** As a complementary approach to describing electron/ion coupling, we plan to directly implement electron-phonon and exciton-phonon coupling within Green's function formalism. These developments are a relevant and interesting example of calculations entering more complex workflows, where combined many-body perturbation theory (MBPT) and density functional perturbation theory (DFPT) calculations are run together.

- **Beyond-GW self-energies**. Self-energy schemes beyond the GW approximation have been extensively studied in the literature and often proposed as viable solutions to address some of the GW shortcomings. Among these, the GW+Cumulant approach [4, 5] to better treat satellites, the second Born or SOSEX self-energies [6, 7, 8] to cure for higher order exchange terms, or TD-HF and BSE vertex corrections [9, 10]. Given the increasing computational power the emerging HPC architecture provides, implementing beyond-GW approaches becomes more and more relevant and timely.

- **GW self-consistency.** One of the critical aspects of state-of-the-art G_0W_0 approaches is the method's lack of self-consistency (or initial state dependence). Here we plan to implement self-consistent schemes aimed at mitigating the issue. In the short term, we plan to implement the so-called QSGW (quasi-particle self-consistent GW) scheme [11]. Alternative and more advanced approaches may be considered in the revised SDP at M30.

Moreover, given their relevance, several technical developments required to treat special classes of systems and properties are also considered:

- **Electrostatic embedding of GW and BSE**. The treatment of the electrostatic environment, including the response of the surrounding medium, is particularly critical when studying the electronic and optical spectroscopic response of materials, including, e.g. layered structures, interfaces, and molecules. To this aim, we plan to implement modelling of the environment for the GW and BSE theoretical schemes based on the polarizable continuum model, i.e. including the environment via its classical dielectric response. The revised SDP may consider further development of the embedding scheme for GW and BSE.

- **Convergence accelerators for materials in different dimensionalities**. MBPT methods can become computationally very expensive because of their algorithmic scaling. Convergence accelerators for these methods, including GW and BSE, are, therefore, quite important and effective. Within this task, we consider developing and implementing advanced schemes to treat materials with different dimensionality (2D layers, 1D systems, etc) with different physical behaviour (metals, semimetals, semiconductors). This activity will enable accurate and computationally less expensive calculations for materials (such as 2D layered systems) of special interest to the scientific community.

5.2 Interoperability hooks, system interfaces, and other measures to enable workflows.

The details on the common layer of interoperability hooks and other measures to enable workflows will eventually be decided within WP2. Their implementation plan will be presented at M18 in the update of the present Software Development Plan. The codes will work together to implement some general interoperability measures, such as the enhancement of the formatted I/O, also extended to error logging and the preparation of performance models and predictors that can help workflow managers to manage computational resources efficiently. In this earlier phase, the code development teams have

started several pilot developments concerning data exchange and resource control. The outcomes of these pilot developments will be helpful for the final design and implementation. In this Section we report some examples of these pilot developments.

- **YAMBO** and **QUANTUM ESPRESSO** groups will work together to implement several data-exchange schemes:
  - YAMBO-QUANTUM ESPRESSO interface. Two steps will be followed: (i) the exchange of electron-phonon (el-ph) data will be improved to make it suitable for exciton-phonon calculations; (ii) YAMBO will load the info of the pseudo-potential files in the UPF format using the `upflib` APIs.These data will allow the calculation of the electron-phonon matrix elements and the performing of coupled electron-ion dynamics.
  - YAMBO calls QUANTUM ESPRESSO on the fly during the time-evolution simulations to re-compute the reference DFT electronic structure once ionic positions have been changed during dynamics.
  - QUANTUM ESPRESSO calls YAMBO on the fly to compute GW energy levels or bands and optical properties during molecular dynamics (MD) runs performed by QUANTUM ESPRESSO.
  - Moreover, YAMBO may compute BSE forces (gradients of the BSE excitation energies) to be later used for surface hopping molecular dynamics. This interaction also pertains to the case where QUANTUM ESPRESSO calls YAMBO on the fly.
- The **SIESTA** group will explore aspects of resource control and load-balancing that are important in the context of "producer-consumer" workflows, such as the computation of properties for a series of snapshots in a molecular-dynamics run. Initial experiments with HyperQueue have shown that it might be possible in most cases to abstract the resource handling at a higher level, without requiring modifications to the system's scheduler.
- **BIGDFT** will work on designing and implementing a data structure suited for saving and exchanging the serialised descriptors of the electronic density matrix.
- **FLEUR** will work on designing and implementing an interface for exchanging spin model parameters with external applications.

6 Conclusions

This report represents a first version of the Software development Plan of MAX phase-3. It presents a general outline of the organisation of the codes and discusses the technical changes underlying the optimisation strategies that we intend to adopt. Being drafted at a relatively early stage, the document does not define all the final technical details of our development plan but rather identifies the general lines of action, the objectives, the issues, and the available technical solutions to be explored before actual adoption.

This document represents a necessary reference for WP1 as well as the other technical work packages (WP2, 3 and 4) for identifying the contact points between specific work package plans. These contact points will be crucial for many of the postponed technical decisions. We thus take advantage of this conclusive Section to summarise the main inputs from WP1 for the other work packages and the necessary interactions for the next steps and decisions in WP1.

Regarding WP2, in Section 5.1 this report provides a preliminary list of the property calculators that will be developed or enhanced for the scientific workflows. The work on these implementations and their earlier outputs will be instrumental for the design of the interoperability layer to be done within WP2. The collaborative actions between different code groups will be particularly useful to this end. For example, YAMBO and QUANTUM ESPRESSO developers have already started working on a common layer to exchange wave function and deformation potential data sets between the two codes.

The final design of the interoperability layer, Section 5.2, provides instead a preliminary description of the work that will be done for implementing the data exchange between our applications and the flow control. A complete description for these implementations will leverage the output of WP2 for defining a common interoperability layer and assessing the capabilities of `AiiDA` and `Hyperqueue` in workflow orchestration.

The interaction with WP3 will concern all technical issues requiring general HPC expertise. As input for WP3, we provide a detailed discussion of all the technical questions needed. Most of these points were discussed in Sections 2 and 3 and involved optimising offloading to GPUs, memory management, and the communications between them. Tuning the configuration of work schedulers will be needed to offer a more flexible resource allocation. Within WP3, the work for benchmarking and profiling will be instrumental in choosing the best strategies for refactoring some code parallelisation and automatic tuning of the parallelism. Another general point where this document prompts interaction with the experts in HPC centres is the GPU-aware distributed linear algebra. Here WP3 input will be instrumental in identifying the appropriate libraries for the different problems and machines, and in designing the data distribution for such applications.

In section 2, we have presented our current work plan for portability. It is strongly oriented towards heterogeneous architectures based on GPUs. While this is the prevailing solution and would allow us to support all current EuroHPC clusters, the HPC situation is still subject to sudden scenario changes. The actions needed to remain responsive to such changes will be agreed upon together with WP 3 and 4. Notably, these work packages monitor the evolution of HPC hardware, which is a key action for what concerns our software development plan. The interactions on this point will also make use of our work on unit testing presented in Sec. 4 that could be used to produce mini-apps for the emulation of the single kernels in candidate architecture and for individuating possible bottleneck and portability issues in these architectures.

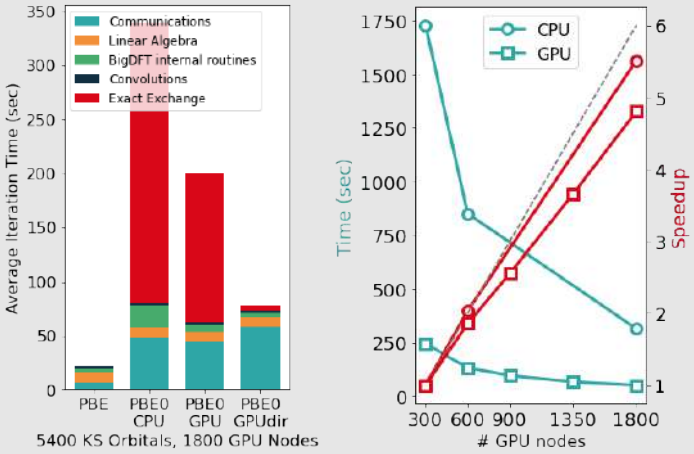
BIGDFT

URL: https://www.bigdft.org

Description and scientific targets. Starting in 2005, the BigDFT EU project aimed to test the advantages of Daubechies wavelets as a basis set for DFT using pseudopotentials. This led to the creation of the BigDFT code, which has optimal features of flexibility, performance and precision. In addition to the traditional cubic-scaling DFT approach, 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. This feature enables electronic structure calculations of systems which were impractical to simulate even very recently. BigDFT has rapidly become a mature and reliable package suite with a wide variety of features, ranging from ground-state quantities up to potential energy surface exploration techniques. BigDFT uses dual space Gaussian type norm-conserving pseudpotentials including those with non-linear core corrections, which have proven to deliver all-electron precision on various ground state quantities. Its flexible poisson solver can handle a number of different boundary conditions including free, wire, surface, and periodic. It is also possible to simulate implicit solvents as well as external electric fields. Such technology is employed for the computations of hybrid functionals and time-dependent (TD) DFT.

Diffusion. BigDFT is free and open source software, made available under the GPL license. The code is developed by few individuals, ranging from 6 up to 15 people. The active code developers are, or have been, located in various groups in the world, including EU, UK, US, and Japan. For these reasons, in addition to production calculations aimed at scientific results, the code has been often employed as a test-bed for numerous case-study in computer science and by hardware/software vendors, to test the behaviour of novel-/prototype computer architectures in realistic runtimes. The compilation of the code suite relies on the splitting of the code components into modules, which are compiled by the `bundler` package. This package lays the groundwork for developing a common infrastructure for compiling and linking together libraries for electronic structure codes, and it is employed as the basis for the ESL bundle.

Performance in HPC environments. BigDFT is a award-winner DFT code, recipient of the first edition (2009) of the Bull-Fourier prize for its "the ground-breaking usage of wavelets and the adaptation to hybrid architectures, combining traditional processors and graphic accelerators, leading the path for new major advancements in the domain of new materials and molecules". It is parallelized using a combination of MPI and OpenMP and has support for GPU acceleration since the early days of GPGPU computing. Such supports involve both CUDA as well as OpenCL computing kernels, and can be routinely applied to large systems. For example, the calculation of a 12,000 atom protein system requires about 1.2 hours of wall-time on 16 nodes of the Irene-ROME supercomputer. This calculation can be further accelerated for systems composed of repeated sub-units using a fragment approach for molecules, or in the case of extended systems, a pseudo-fragment approach, both among the outcomes of MaX2 project.



The Figure above shows the benefits induced by a multi-GPU calculation on a PBE0 calculation (seconds per 2 SCF iterations) of a system made of 5400 KS orbitals on Piz Daint. Such calculations can scale effectively up to the range of thousands of GPUs and compute nodes.

To facilitate driving the calculations of dense workflow graphs involving thousands of simulations of large systems, the code suite includes a python package called PyBigDFT as a framework for managing DFT workflows. PyBigDFT is able to handle building complex systems or reading them from a variety of file types, performing calculations with BigDFT linked with AiiDA package, and analyzing calculation results. This makes it easy to build production analysis, thereby enabling new users' production HPC experiences on top of the data generated from large scale DFT calculations. Currently, BigDFT is being deployed on large HPC machines including Fugaku (RIKEN, JP), Archer2 (Edinburgh, UK), and Irene-ROME (TGCC-CEA, FR).

FLEUR

URL: https://www.flapw.de

Description and scientific targets. FLEUR is an all-electron density functional theory code based on the full-potential linearized augmented plane wave (FLAPW) method. A key difference with respect to the other MAX -codes and indeed most other DFT codes lies in the treatment of all electrons on the same footing. The key component of FLEUR is a versatile DFT code for the ground-state properties of multicomponent one-, two- and three-dimensional solids. A special focus lies on non-collinear magnetism, the determination of exchange parameters, spin-orbit related properties (topological and Chern insulators, Rashba and Dresselhaus effect, magnetic anisotropies, Dzyaloshinskii-Moriya interaction) and magnon dispersion. A link to WANNIER90 enables the calculation of intrinsic and extrinsic transverse transport properties (anomalous-, spin- and inverse spin Hall effect, spin-orbit torque, anomalous Nernst effect, or topological transport properties such as the quantum spin-Hall effect etc.) in linear response theory using the Kubo formula. FLEUR includes LDA+U as well as hybrid-functionals for the accurate description of e.g. oxide materials and by linking against the libxc library, many more functionals are accessible. Using its link to the SPEX code more advanced treatments using the GW method or the GW+T approximation to magnetic excitations are possible starting from FLEUR. The well established FLAPW scheme is usually considered providing the most accurate DFT results and used as a reference for other methods. In addition, several quantities e.g. related to the properties of core-electrons are only available by the use of code not relying on the pseudopotential approximation.

Being applicable to all elements of the periodic table and by including all electrons, the code has its particular strength in the fields of electronically and magnetically complex materials, for example materials involving transition metals, heavy or rare-earth elements and thus is frequently used to calculate magnetic or spin-dependent properties in metals or complex oxide materials. It provides a natural link to other methods via the calculation of parameters for atomistic magnetic simulations or similar multiscale modelling methods.

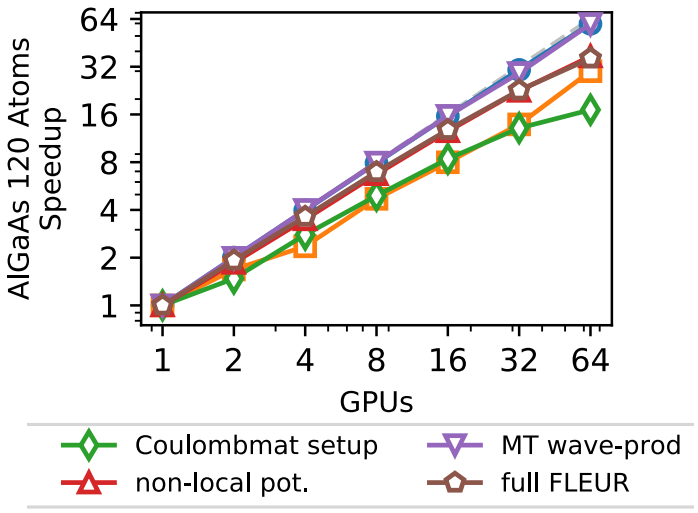
Diffusion. FLEUR is distributed freely under the MIT license and has a growing user community. While about 3000 users registered on the older FLEUR-webpage the current free distribution scheme does no longer allow user tracking.

Performance in Parallel Computing environments. FLEUR is utilising several levels of parallelization to exploit both, intra-node and inter-node distribution of the calculation. On the most coarse level, the calculations can be split over different k-points (and q-points where present) leading to an excellent scaling behaviour. This can be seen in the following table in which we demonstrate excellent weak and good strong scalability on JUWELS booster up to roughly a quarter of the machine with a nominal performance of approximate 15 PFlops.

160 k-points → (x5) → 800 k-points

Nodes	Time	Scaling	Nodes	Time	Scaling
10	434 s	--	50	438 s	--
20	222 s	1.95	100	228 s	1.92
40	121 s	3.59	200	124 s	3.53

In addition to this outer parallelization level, FLEURalso employs more fine-grain parallelism, distributing the calculation associated with the different eigenstates. This parallelization is largely 'orthogonal' to the scaling shown before in which only a single GPU was assigned to this level. This level is strongly dependent on the details of the system as well as the kernel to be used. As an example (without any outer parallelization), we show in the Fig. below the scalability for the calculation using hybrid functionals on the JURECA-DC cluster with 4 NVIDIA A100 cards per node, e.g. up to 16 nodes.



As a production calculation for the system studied here would require several hundred kq-points the combination of the two levels of parallelism discussed here could be scaled up to fill existing supercomputers easily.

QUANTUM ESPRESSO

URL: https://www.quantum-espresso.org

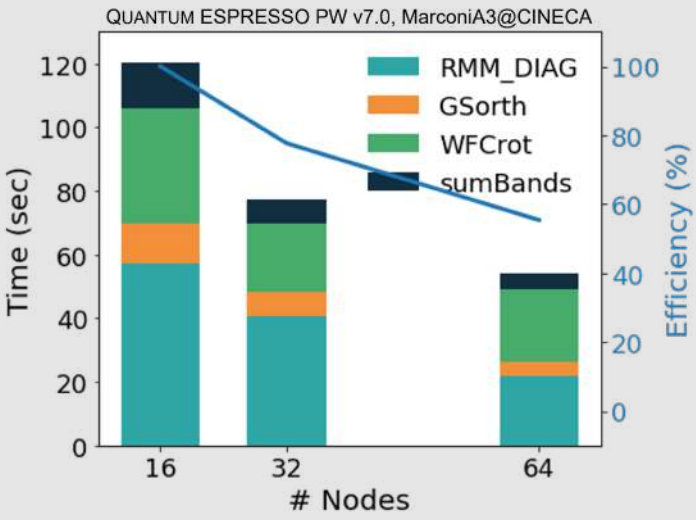
Description and scientific targets. QUANTUM ESPRESSO is a suite of applications for ab-initio electronic structure calculations using plane waves and pseudopotentials. The code is mostly written in modern Fortran. The suite supports standard GGA and many advanced functionals such as non-local vdW-enabled, meta-GGA, Hubbard-corrected, hybrid functionals. It can be linked with `Libxc` and use all functionals implemented there.

The codes of the suite may perform many types of ground-state calculations: self-consistent energies, forces and stresses, structural optimization, molecular dynamics (`PW` and `CP`); search for transition pathways (`NEB`). The suite also contains a rich apparatus of routines and methods for the computation of the linear response to external perturbations. These are used for computing dielectric responses and vibrational spectra (`PHonon`); optical, magnons, and EELS spectra, using time-dependent DFT (`TDDFPT`); electron-phonon coupling coefficients and related properties (`EPW`); self consistent Hubbard correction parameters(`HP`), and more.

QUANTUM ESPRESSO applications are also used as starting point for workflows that implement advanced methodologies such as MBPT, QMC, DMFT and others.

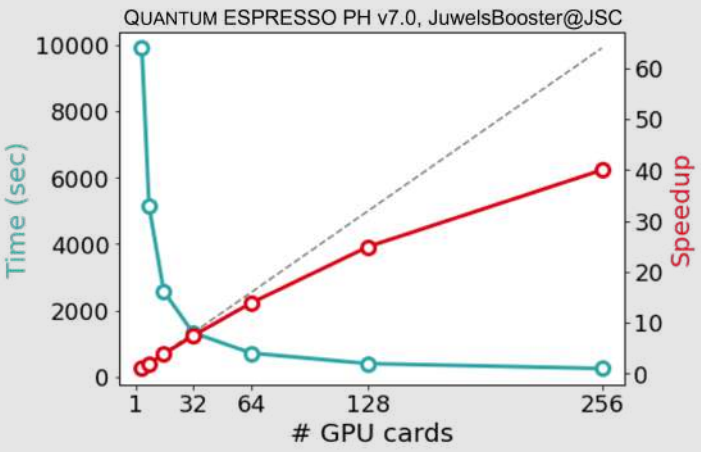
Diffusion QUANTUM ESPRESSO is released under the GPL licence. It has a sizeable basis of users, as apparent from the number of citations of the two documenting papers of 2009 [**?**] (16,000+) and 2017 [**?**] (2,500+), authored by 25,000+ scientists and by the number of messages exchanged on the users' mailing list (2000+/year). In addition to the inner circle of developers (≈20), QUANTUM ESPRESSO often receives contributions from external users and developers. There are currently 107 forks of the main GitLab repository, and 243 of the GitHub mirror. We are aware of at least 12 independent successful software projects that rely entirely on QUANTUM ESPRESSO technology and codebase for further development.

Performance in HPC environments Most of the applications of the suite are designed for the efficient usage of the state-of-the-art HPC machines using multiple parallelization levels. The basal workload distribution can be done using MPI + OpenMP multithreading, or offloading it to GPGPUs, depending on the nodes' architecture. This parallelization level also provides an efficient data distribution among



the MPI ranks. This allows to compute systems with up to $\sim 10^4$ atoms. The offloading or the usage of a growing number of MPI ranks are able to scale down the computational cost of 3D FFTs and other operations on 3D data grids. In the figure above we show the performance analysis for a mid-size case (A Carbon nanotube functionalized with two porphyrine molecules, about 1500 atoms, 8000 bands, 1 k-point) on an HPC homogeneous nodes' cluster. The kernels distribute their workload on the lower parallelization group, except for `WFCrot` whose perfomance relies instead on parallel or accelerated linear algebra specific libraries.

The auxiliary MPI parallelization levels allow to obtain further scaling. The band parallelization level distributes the operations on wave functions of different basal groups. The two upper parallelization levels – pools and images – are very efficient because they distribute the computations in concurrent quasi-independent blocks; as shown in the figure below for a PH calculation on 72 atoms quartz, executed on an heterogeneous nodes' HPC cluster equipped with Ampere GPGPUs.

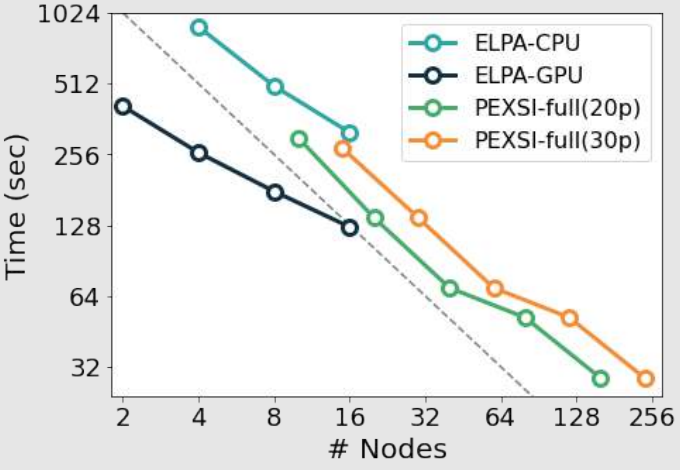


SIESTA URL: https://siesta-project.org

Description and scientific targets. SIESTA is a pseudopotential-based density functional theory software whose strength lies in its use of atomic-like strictly-localised basis sets: the use of a "good first approximation" to the full problem decreases the number of basis functions needed to achieve a given accuracy, and the finite support of the orbitals leads to sparsity in the Hamiltonian and overlap matrices, thus enabling the use of reduced-scaling methods. The functionalities of SIESTA include, amongst others, the calculation of energies and forces, molecular-dynamics simulations, band structures, densities of states, spin-orbit couplings, van der Waals functionals, hybrid functionals, DFT+U for correlated systems, real-time TDDFT, and density-functional perturbation theory. SIESTA contains the transport code TRANSIESTA, which is based on the non-equilibrium Green function formalism and enables open-boundary condition calculations by extending periodic regions with bulk electrodes. TRANSIESTA's features include advanced inversion algorithms, multiple electrodes, thermo-electric calculations, real-space calculations (without k-points), and phonon transport calculations. SIESTA can also be used to provide base calculations for the execution of other electronic structure packages such as BERKELEYGW (optical properties and quasiparticle excitations), GOLLUM (transport), WANNIER90 (maximally localized Wannier functions and advanced electronic properties), or I-PI (nuclear quantum effects in condensed phase systems). SIESTA is known to be used in a wide range of applications, encompassing materials science, nanotechnology, catalysis, biological sciences (including interaction between organic and inorganic materials), geology and materials under high pressure, Martian geochemistry, materials for nuclear reactors, and astrophysical and atmospheric systems.

Diffusion SIESTA is distributed for free under a GPL license. Release tarballs and development versions can be downloaded from its GitLab repository, while tutorials and other learning materials are publicly accessible from a dedicated documentation site. Further hands-on training activities are organised routinely, and SIESTA developers (about 20 of them active at the moment) can be easily reached through the SIESTA mailing list and the usual forums of the electronic structure community. SIESTA has a large use base, as displayed by the number of citations that the reference publication [**?**] receives every year, well over 600 (more than 12,000 citations by early 2022).

Performance in HPC environments SIESTA is written in modern Fortran with both MPI and OpenMP parallelism. For most problems, the most computationally-demanding stage of SIESTA execution is the solver stage (calculation of energy eigenvalues from the above-mentioned sparse objects). On the one hand, SIESTA provides a range of solvers of its own, from cubic-scaling diagonalisation to linear-scaling methods, that exploit existing linear algebra libraries such as SCALAPACK, ELPA, and DBCSR. On the other hand, SIESTA can leverage a number of libraries that implement favourably-scaling solvers, such as CHESS (Fermi Operator Expansion method) and PEXSI (Pole EXpansion and Selected Inversion method). All these libraries are designed for parallel execution, and they are progressively incorporating support for offloading to an increasing breadth of GPU architectures. The figure below exemplifies the scaling of some of the solvers mentioned above in the Marconi 100 supercomputer: starting from the CPU version of the ELPA solver, the same solver with GPU offloading displays a significant speed-up, although with some degradation in its scaling probably due to no longer saturating the GPUs. In comparison, the (CPU-only) PEXSI solver allows for scaling to a significantly larger number of nodes with less degradation. The accuracy of the PEXSI solver improves with the number of poles used in the expansion it performs; it can be seen how the increased accuracy enables scaling to a larger number of nodes, thus not increasing the time to solution.



In the Figure: Time to solve the diagonalization problem corresponding to a piece of SARS-COV-2 protein surrounded by water molecules, with approximately 58,000 orbitals, in Marconi 100). The dashed line shows the ideal scalability behaviour.

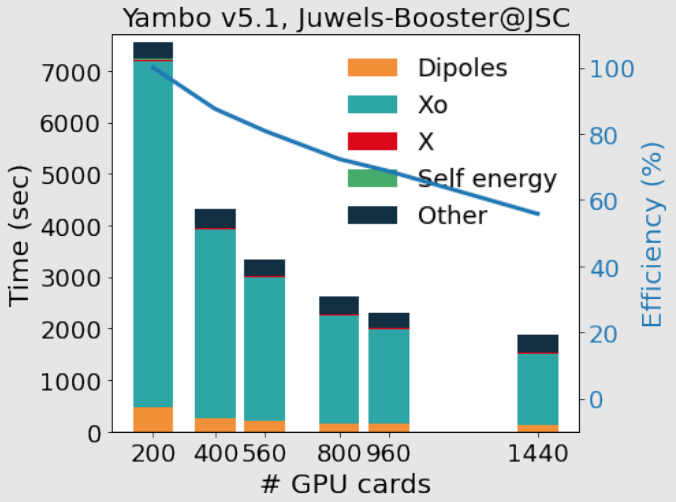
YAMBO

URL: https://www.yambo-code.org

Description and scientific targets. YAMBO is an open-source code released within the GPL licence that implements ground-state as well as excited-state properties in an ab initio context. The code implements MBPT, DFT and Non-Equilibrium Green's Function Theory (NEGF) in order to allow the user to calculate a wealth of physical properties: reliable band gaps, band alignments, defect quasi-particle energies, optical and non–equilibrium properties. YAMBO resorts to previously computed electronic structure, and for this reason it is interfaced with other DFT codes, among which QUANTUM ESPRESSO.
Among the variety of physical quantities that can be described with YAMBO, we mention

- Electronic properties: quasi-particle energies, linewidths, and renormalization factors;
- Linear optical properties, capturing the physics of excitons, plasmons, and magnons;
- Temperature dependent electronic and optical properties via electron-phonon coupling;
- Non–equilibrium and Non–linear optical properties via NEGF real–time simulations;
- Advanced post-processing tools to analyse the simulation flow of data.

All these properties are ubiquitous for the understanding of the optical and electronic properties of a wealth of advanced materials. More importantly, YAMBO provides a unique approach for the non–equilibrium regime, where ab initio numerical tools are scarse, able to model Pump&Probe experiments and to capture coherent electron dynamics. The code is under a constant development and fully documented.
Diffusion. YAMBO has attracted over the years a growing community of code users and developers. The code is routinely used to organise hands–on schools where the most fundamental concepts of the underling theory are described. A dedicated user forum (with more than 900 subscribers) is actively used to answer users' questions and doubts. The code has been used to produce the results published in hundreds of papers by many different groups all over the world. The YAMBO reference papers,[**?**, **?**] have been cited more than 800 times to date (Mar 2022). At the moment YAMBO counts about 20 active developers and the source project is publicly hosted on the github platform.



Performance in Parallel Computation environments. YAMBO has a user-friendly command-line interface, flexible I/O procedures, and it is parallelised by using an hybrid MPI plus OpenMP infrastructure, very well integrated with support of GPGPU-based heterogeneous architectures. This makes it possible to distribute the workload to a large number of parallel levels. In practice, depending on the kind of calculation, all the variables to be used (k/q grids, bands, quasi-particles, etc) are distributed along the different level of parallelisation. At present YAMBO has been shown to be efficient in large-scale simulations (several upto few tens of thousands MPI tasks combined with OpenMP parallelism) for most of its calculation environments.
The GPU porting was first made using CUDA-Fortran, and more recently enlarged to other programming models (like OpenACC and OpenMP5, both in development). To avoid code duplication, we make an intense use of pre-processor macros that activate the language chosen at compile time. This allows YAMBO to optimally integrate MPI-OpenMP with programming models for GPGPU. The outcome of this integration is well exemplified by the scaling tests reported in figure, for the calculation of quasi-particle corrections on a graphene/Co interface (GrCo) composed by a graphene sheet adsorbed on a Co slab 4 layers thick, and a vacuum layer as large as the Co slab. The test represents a prototype calculation, as it involves the evaluation of a response function, of the Hartree-Fock self-energy and, finally, of the correlation part of the self-energy. The scalability and relative efficiency are reported in the Figure as a function of the number of GPU cards and show a very good scalability up to 1440 GPUs (360 nodes on Juwels-Booster@JSC, 4 NVIDIA A100 per node).

References

[1] Ge, X., Binnie, S. J., Rocca, D., Gebauer, R. & Baroni, S. turbotddft 2.0—hybrid functionals and new algorithms within time-dependent density-functional perturbation theory. *Comput. Phys. Commun.* **185**, 2080–2089 (2014).

[2] Hutter, J. Excited state nuclear forces from the Tamm–Dancoff approximation to time-dependent density functional theory within the plane wave basis set framework. *J. Chem. Phys.* **118**, 3928–3934 (2003).

[3] Handy, N. C. & Schaefer, I., Henry F. On the evaluation of analytic energy derivatives for correlated wave functions. *J. Chem. Phys.* **81**, 5031–5033 (1984).

[4] Guzzo, M. *et al.* Valence electron photoemission spectrum of semiconductors: Ab initio description of multiple satellites. *Phys. Rev. Lett.* **107** (2011).

[5] Caruso, F., Lambert, H. & Giustino, F. Band structures of plasmonic polarons. *Phys. Rev. Lett.* **114** (2015).

[6] Romaniello, P., Bechstedt, F. & Reining, L. Beyond the GW approximation: Combining correlation channels. *Phys. Rev. B* **85** (2012).

[7] Ren, X., Marom, N., Caruso, F., Scheffler, M. & Rinke, P. Beyond the GW approximation: A second-order screened exchange correction. *Phys. Rev. B* **92** (2015).

[8] Vacondio, S., Varsano, D., Ruini, A. & Ferretti, A. Numerically precise benchmark of many-body self-energies on spherical atoms. *J. Chem. Theory Comput.* (2022).

[9] Maggio, E. & Kresse, G. GW vertex corrected calculations for molecular systems. *J. Chem. Theory Comput.* **13**, 4765–4778 (2017).

[10] Kutepov, A. L. Electronic structure of van der waals ferromagnet CrI_3 from self-consistent vertex corrected GW approaches. *Phys. Rev. Mater.* **5** (2021).

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