D2.2

Code interoperability

Pietro Delugas, Alberto Garcia, Ada Böhm, Ivan Carnimeo, Carlo Cavazzoni, Andrea Ferretti, Luigi Genovese, Cristiano Malica, Nicola Marzari, Davide Sangalli, Nicola Spallanzani, Matteo Vandelli, Daniele Varsano, Daniel Wortmann

Due date of deliverable	30/06/2024 (**month 18**)
Actual submission date	28/06/2024
Final version	28/06/2024
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.	D2.2
Authors	Pietro Delugas, Alberto Garcia, Ada Böhm, Ivan Carnimeo, Carlo Cavazzoni, Andrea Ferretti, Luigi Genovese, Cristiano Malica, Nicola Marzari, Davide Sangalli, Nicola Spallanzani, Matteo Vandelli, Daniele Varsano, Daniel Wortmann
To be cited as	P. Delugas et al. (2024): Code Interoperability Deliverable D2.2 of the HORIZON-EUROHPC-JU-2021-COE-01 project MAX (final version as of 28/06/2024). EC grant agreement no: 101093374, SISSA, Trieste, Italy.

Disclaimer

Contents

Executive Summary

This report contains an update on plans for the design and implementation of mechanisms for interoperability of the MaX lighthouse codes in the context of exascale workflows, as pertains to the task T2.4 of WP2.

Deliverable D1.1 [1] already provided, in the context of the sister task T1.4, some initial ideas and implementation plans, with a focus on work to be carried out in the codes themselves. Here, with the extra perspective afforded by the workflow-design ideas presented in D2.1, we provide general objectives that are seen as desirable for the achievement of interoperability, and suggest further specific implementation plans that should be taken up by T1.4.

After a general introduction to frame the subject of interoperability, we discuss the main areas of development towards enhancing it, which are structured I/O and error logging, checkpointing and recovery data, performance models, and data-exchange techniques. In addition, we comment on the special role of the `HyperQueue` tool and `AiiDA` framework, and on their ongoing and planned enhancements in the context of exascale workflows.

The document then gathers the reports on current work and development plans of the individual flagships codes with regard to interoperability. While there are common themes and techniques, each code has its own interoperability baseline and implementation priorities, which are also linked to the specific kind of exascale workflows, addressing scientific showcases, in which a particular code participates.

1 Introduction

Interoperability is a fundamental principle driving the development of the MAX software platform. Through the use of well-defined APIs and standardised data structures and protocols, MAX has aimed from its inception to achieve a highly integrated and interoperable ecosystem of codes, libraries, and workflows for materials science at the exascale.

In its third phase, MAX has identified a set of scientific challenges that will benefit from the development of exascale workflows. An analysis of the main characteristics of those workflows was carried out in D2.1 [2], which offered an approximate classification in terms of workflow archetypes. The D2.1 report also provided a discussion of the technical issues involved in the design of the workflows, with particular attention to issues of orchestration (control, resource allocation and handling, and data exchange). In that analysis, codes featured as workflow elements, using and producing data within the workflow with the allotted resources, all within the scripted logic of the overall calculation.

To implement in practice this "block diagram" view of a workflow, one has to keep in mind that the quality and robustness of the integration of a code within a workflow depend on how well and how flexibly the code can interact with the outside world.

At the simplest level, a code has some input and output channels, typically implemented through the file system. Some workflows will just use those channels (maybe wrapped through lightweight plugins) to encapsulate the functionality of the code. More substantial plugins (such as those of the `AiiDA` framework) would provide more options, such as error logging and recovery, necessitating some extra features in the code(s) to support the additional options. These extra features fall under the umbrella of interoperability enhancements, and this report is devoted to document the design and implementation plans for them in the MAX flagship codes in the context of exascale workflows.

Beyond questions of data exchange (which could be implemented more abstractly, possibly coupled with control logic, as detailed below), our list of identified interoperability measures includes performance models, checkpointing, task spawning and hints for resource allocation, and structured output and error logging.

These measures are to be implemented in each code in partnership with Task 1.4 of WP1, but we should stress that their design benefits from common patterns and the use and further development of MAX tools such as `HyperQueue`, `AiiDA`, and `remotemanager`. This report is, therefore, structured around a general discussion of interoperability traits and their specific implementation plans for individual MAX lighthouse codes. The latter are modulated, both in time and depth, by the need to develop the specific exascale workflows in which the code is involved.

2 Issues regarding interoperability for exascale workflows

In the next sections, we present a discussion of selected issues related to code interoperability in the context of MAX exascale workflows. The first two sections are concerned with the `HyperQueue` and `AiiDA` tools, as they play a special role in our workflow-design ideas. The rest addresses the basic interoperability traits already identified as being relevant for the flagship codes, and serve as a general introduction to the code-specific implementation plans to be discussed below.

2.1 Resource allocation: Interfaces to HyperQueue functionality

Flow-control in a workflow can be managed by an external orchestrator, but sometimes individual codes might spawn new tasks when needed (e.g., for an analysis stage of just-produced data). When the resources for the workflow are handled by `HyperQueue` [3] (HQ), it is convenient to launch the new task under the control of this tool. For this, an interface to the "submit" functionality of HQ is needed.

This interface can be as simple as a Fortran-level "system" call using the HQ CLI commands. Ideally, codes might want to leverage a programmatic API, but the current offering by HQ is a Python API that is built directly on the underlying Rust framework. A C interface (which can then be wrapped in Fortran using the iso_c_binding module) might be implemented if feasible. Otherwise, a lighter-weight Fortran wrapper over the CLI commands might be used.

Note that recently, HQ has implemented a feature to enable server resiliency. It works by keeping a journal file that records the activity of the server, and that can be reloaded in future instantiations of the server. This can be seen as a global checkpointing feature that keeps track of resources and pending tasks, supplementing data-only checkpointing measures.

2.2 Extensions and refactoring of `AiiDA` functionality and the code plugins

Workflows developed within the `AiiDA` framework have their interoperability needs solved from the start, as code plugins map code entry-points and input-output to the framework control and data structures. Successful workflows (including exascale-oriented workflows such as, e.g., that described in Secs. 3.3.2) are being routinely developed within `AiiDA`, and we consider it to be an important asset of the MAX project. `AiiDA` is being actively developed, and has a very permissive licensing model.

On the other hand, the built-in interoperability in `AiiDA` is obviously only as good and sophisticated as the plugins and the framework support. In particular, as explained in D2.1, data exchange is currently implemented via filesystem objects, the workflow logic and control is encoded via the scripting idioms of `AiiDA`, and resource specification is static to a large degree (only leaf jobs can be dynamically grouped by `HyperQueue`, for example, when this tool is used as scheduler back-end).

The more relevant changes needed for an upgrade of the interoperability efficiency, as needed for large exascale workflows, are enhancements and further abstractions of the `AiiDA` data structures, and a mechanism for resource specification. Then, code plugins will need to be updated to match this new functionality. Plans for this work are being coordinated with an ongoing `AiiDA` review within the context of the Swiss MARVEL

project. It is worth noting here that, besides `AiiDA`, MAX is also considering leveraging other tools, such as `remotemanager` (see below in Sect. 3.1.1), which is a lighter weight orchestrator that has a different underlying philosophy.

2.3 Structured I/O and error logging

A very important desired development is the general availability of structured output and log files that can be processed easily by the codes participating in a given workflow. For quite some time, BigDFT has employed the YAML format for its output (and input) files, leveraging the dictionary implementation, I/O primitives, and other support in the `Futile` library. QUANTUM ESPRESSO has recently implemented schemas and code to mark up its output in XML, SIESTA is able to generate XML output in the Chemical Markup Language [4], YAMBO is able to parse (and write if needed) XML, and have an exploratory interface with `Futile` for YAML output.

Offering *some kind* of structured markup is a step in the right direction, and it can be argued that it is enough for interoperability, as appropriate converters can be employed when needed. This is relevant, because in addition to the low-level coding details of, say, the YAML-production feature, one must decide on the logical structure and semantic details of the output. Experience in many workshops and committees makes us skeptical about achieving any kind of common standards along these lines, even in a restricted setting such as the MAX project.

Still, it is worth identifying those I/O sections in each code that contain information needed for interoperability in our exascale workflows, in order to achieve consistency in signal handling and data events. It should be possible to work on a relevant subset of I/O events that can be formatted appropriately, ideally with a common interface. Extension to further sections in an incremental manner should then be easier.

Error logging can be seen as another form of structured output, but with a higher priority and close monitoring to trigger recovery mechanisms. Those at the code level are linked to checkpointing (see next section) and internal logic. Workflow-level triggers do depend on standardised error logs, so this information should be included in the common essential set of I/O sections mentioned above.

2.4 Checkpointing, semaphores, and recovery-data

Checkpointing can be seen as a special kind of data transfer: the data is needed (maybe) in the future by the same process creating it. It may seem that interoperability issues would not be relevant here, but the extra resilience offered by proper checkpointing is essential for the proper functioning of a workflow. Ideally, checkpointing data should be written asynchronously, taking advantage of opportunities for the overlap of data transfer and computation. This should be easier in the checkpointing scenario than in the more general one of data exchange among workflow actors. Checkpointing design and implementation is specific to each code and will be detailed below.

2.5 Performance models

Performance models are essential for developing efficient workflows, particularly in high-throughput computing scenarios. Workflows often involve executing numerous calcula-

tions with varying input parameters and computational requirements. Performance models can guide the optimal allocation of computational resources to different tasks within a workflow by predicting their performance characteristics on specific hardware. For instance, in a workflow that involves molecular dynamics simulations coupled to property calculations, performance models can help determine the appropriate number of processors or the optimal distribution of tasks across CPUs and GPUs for each element.

Furthermore, performance models can contribute to the design of more robust and adaptable workflows. By considering the performance implications of different code paths and algorithmic choices, workflows can be designed to dynamically adjust their execution strategies based on the available resources or the characteristics of the input data. In this respect, the QUANTUM ESPRESSO development team was involved in the development of an initial performance model for QUANTUM ESPRESSO during the second phase of MAX . Its experience and techniques can be leveraged by other code teams.

2.6 Communication layer for exchanging potentially large data-sets

As discussed in D2.1, data exchange should preferably be abstracted, at the code level, in an API that is as close as possible to the native file-oriented semantics. We have a long combined experience with I/O libraries such as netCDF, in which basic operations are represented with the idioms:

```
(Declaration section)
nf90_def_var(ncid,'gridfunc',nf90_float
         (/n1_id,n2_id,n3_id,spin_id/),gridfunc_id)

(Data Transfer)

nf90_put_var(ncid,gridfunc_id,grid_buf(1:npt_node(BNode)),
                start=(/ lb(1), lb(2), lb(3), ispin/),
                count=(/nel(1), nel(2), nel(3), 1 /) ) )
```

With appropriate wrappers, full datasets can be transferred transparently:

```
data_send(ncid, charge_density)
```

Further, the destination might not be a filesystem object, but another process in the workflow that is listening to a socket. Here the low-level calls in the stack are to a communication library such as ZeroMQ [5], and the wrappers are of course different, but offering a similar top-level call.

Most MAX lighthouse codes already implement the "file-system" approach above, with either NetCDF or the lower-level HDF5 libraries to transfer large datasets. The work still to be done in each code is to create the top-level wrappers if not yet existing, and to do the same with the ZeroMQ wrappers. The underlying ZeroMQ interface is common to all codes, and the main task to be carried out is to refine its Fortran layer so that it offers support for more kinds of data types on which to build the needed wrappers.

3 Design and implementation work and plans

In this Section, we provide an update on the design work being carried out in the context of interoperability (which is formally part of WP2, but it is projected on implementations done within WP1 by the code owners), and further plans. The plans are informed by the current development of exascale workflows, which are at various levels of advancement.

3.1 BIGDFT

There is a strong emphasis within the BigDFT project on using Jupyter notebooks to drive complex simulation and analysis workflows. The PyBigDFT Python library, developed within the second phase of the MAX project, integrates well with Jupyter notebooks. Its modules can be used to set up the calculations (defining atomic positions, pseudopotentials, etc.) and then post-process the (structured) BigDFT output data for analysis within the notebook itself.

Recent developments, showcased in the next section, take further the approach by enabling seamless remote execution of the glue python code that is used to achieve interoperability among codes.

3.1.1 RemoteManager as an interoperability engine

In order to illustrate the present and future focus of the work within the BigDFT project on interoperability, we present some sample uses of the PyBigDFT and `remotemanager` combination in substantial simulation and data analysis workflows. The `remotemanager` tool was already presented in D2.1. We emphasize that these are not predefined workflows with well-defined input parameters. Rather, they are custom combinations of calculation and analysis built on high-level Python data structures. It is key to realize that PyBigDFT and `remotemanager` can be used for driving calculations and developing new functionality on top of existing, *unmodified* codes. In what follows, the underlying DFT calculations are performed with unmodified versions of the core BigDFT Fortran code [6], NWChem [7], and also PSI4 [8]; these are well established codes with different computational capabilities and underlying methodologies. We also present program snippets that demonstrate key operations.

Recently, we have introduced and implemented the transition-based constrained-DFT method for studying excited states [9]. The aim of T-CDFT is to provide a method for studying excitations in disordered supramolecular morphologies, which can easily be used in conjunction with BigDFT's molecular fragment approach [10, 11] to reach the required large system sizes. In practice, this requires a workflow that involves many steps, from setting up a system containing "active" (i.e., those undergoing an excitation) and "environment" molecules, to defining the transition to be imposed (e.g., based on the outcome of a time-dependent (TD) DFT calculation), to generating basis sets for template gas phase molecules, to calculating excitation energies for pure, e.g., HOMO to LUMO, and/or mixed transition constraints on the full systems. The workflow thus combines standard DFT/TDDFT calculations with a variety of post-processing steps. For the latter steps, it is first necessary to post-process several calculations to combine the converged density kernels associated with the pure constraints, a process which is run on the front end, as illustrated in the List. 1

```python
@RemoteFunction
def combine_density_kernels(kernel_info):
    # Combine the Density Kernels from Calculations with a Pure Constraint
    for p, kernel_dir in enumerate(kernel_info['kernel_dirs']):
        kernel_file = join(kernel_dir, 'density_kernel_sparse.mtx')
        rho_a_p = mmread(kernel_file)

        # Set the Kernel to Zero on the First Step
        if p == 0:
            rho_a = deepcopy(rho_a_p)
            rho_a.data[:] = 0.0

        rho_a += kernel_info['transition_weights'][p] * rho_a_p

    # Write the Kernel for Reading by BigDFT
    mmwrite(join(kernel_info['target_dir'], 'density_kernel_sparse.mtx'), rho_a)
```

Listing 1: A `@RemoteFunction` used as a post-processing step in the T-CDFT workflow. By decorating this function, it can now be called by any `Dataset`. The function takes density kernels generated from T-CDFT calculations with pure constraints, and combines them according to weights taken from the transition breakdown coming from a TDDFT calculation, to be used in a calculation with a mixed constraint. For the generic case of a mixed transition, it is then enough to split the T-CDFT approach into multiple constraints. We can define the energy of the mixed transition by the SCF energy obtained from the density operator $\hat{\rho}^a = \sum_p \mathcal{P}_p^a \hat{\rho}_p^a$, where $\hat{\rho}_p^a$ is the SCF density obtained from the pure T-CDFT calculation with the constraint $\mathrm{Tr}(\hat{\rho}\hat{T}_p^{(a)}) = 1$.

Running this kernel on the remote machine avoids the need to copy matrices to one's own machine and allows us to build "Post-DFT" functionality on top of an unchanged DFT code (instead of incorporating this mixing into BigDFT's Fortran code). Figure 1 illustrates a generalised workflow for such a calculation, as well as optional additional post-processing steps, such as characterizing the imposed transitions via the means of a charge transfer parameter.

The interoperable workflow uses NWChem [7] for the initial step of identifying the transition constraint that will be imposed in later steps, although this step could equally be substituted with another code or approach, like YAMBO for instance. Nonetheless, this integration of NWChem into the workflow highlights the benefits of the ability to execute arbitrary Python code with `remotemanager`. While other packages provide access to a popular code like NWChem, they may not provide access to all calculation methods (for example, ASE does not have settings for TDDFT), or to all the computed properties. For a specific use case, though, it is easy to use Python as a glue code to build the appropriate input files and parse the results. The `remotemanager` class transparently allows the user to run that glue code on a remote machine over a large set of molecules.

Such approaches enable a seamless combination of different compute codes (or various pre- / post-processing treatments) from a Jupyter notebook on the end-user side. We are presently testing those approaches in combination with molecular dynamics calculations executed on different supercomputers to enable the usage of multiple HPC resources from the same orchestrator.

The showcase just described is an example of the dynamic coupling between scien-

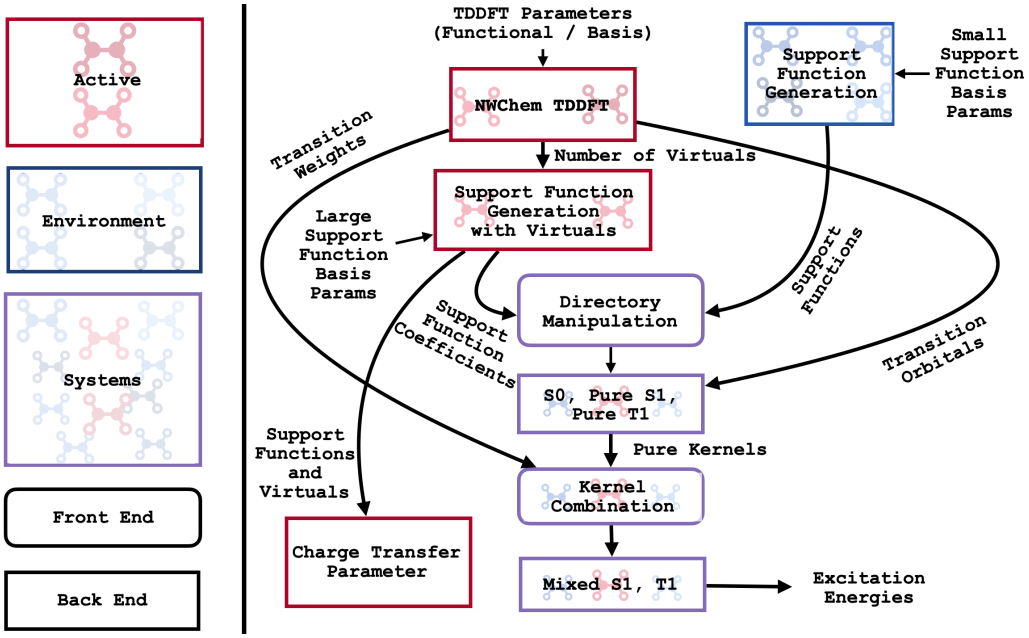

Figure 1: Flow chart of a workflow for calculating excited state singlet and triplet energies of large supramolecular morphologies using Transition-Based Constrained DFT, built on PyBigDFT and `remotemanager`. Calculations on active molecules, environment molecules, and the full systems are highlighted in red, blue, and purple, respectively. Calculations run on the back end of the remote machine are in squares, while front end calculations have rounded corners.

tific needs and technology advancement that is central to the development of fruitful new paradigms for simulation and workflows.

3.1.2 YAML-based I/O as a key element in the BigDFT toolchest

A feature of BigDFT that has served as a core principle for the development of the approach presented in the previous section is the use of YAML as a language to write input and output files. YAML allows for the serialisation of complex data structures by composing simple data types in a hierarchical fashion. YAML can thus serve as an intermediary between the ecosystems of a Python and Fortran code using dictionary data types (built in for Python, from our Futile library for Fortran). For example, the YAML input file for BigDFT can be generated via a python dictionary, filled either explicitly or by some pre-defined actions on the BigDFT. Inputfile class of the PyBigDFT module. Conversely, modules in PyBigDFT and in any YAML-enabled package can be used to post-process and in general act on the BigDFT output.

BigDFT is thus very well-placed to provide know-how to other MAX codes in regard to their efforts to implement flexible and structure output. There have been already contacts with the YAMBO and SIESTA development teams to explore the use of the `Futile` library in those Fortran codes.

3.2 FLEUR

FLEUR's strength lies in its ability to calculate complex properties, such as the spin-dependent electronic structure of materials, which is essential in understanding the behaviour of magnetic materials. This is particularly significant in the development of new materials for applications in data storage and spintronics devices. Moreover, FLEUR also focuses on the use of hybrid functionals, which provides a better description of materials properties compared to traditional density functional theory methods.

Interoperability is a crucial aspect of working with large-scale simulations, and FLEUR addresses this through the use of standard I/O Formats such as XML and HDF5. These formats allow for seamless data exchange between different codes, enabling researchers to combine data from different simulations and perform multi-physics analyses.

Additionally, FLEUR has traditionally relied on the `AiiDA` platform for automating and managing workflows. In this third phase of MAX , `AiiDA`, with some enhancements, is expected to continue to be the main workhorse for FLEUR's (sub)workflows.

3.2.1 Interoperability using improved output

The basic input and output of FLEUR was already switched to standard I/O Formats: In particular, it uses XML for input of parameters and structures and for various output properties. HDF5 on the other hand is used for large binary data. In the past, some parts of the I/O were removed to ensure better scalability. As a side effect, this unfortunately removed some possibility to construct workflows in which intermediate data is required. To mitigate this effect, FLEUR has added the capability to add structured IO on demand at several places in the code. A typical example for this is the refactoring and modifications of the I/O for the Wannier functionality.

3.2.2 Modifications of the AiiDA workflows

While the AiiDA-FLEUR plugin already features several high-level workflows suitable for the simulation of magnetic properties, we continue to work in two directions:

- The extension of the command-line interface of AiiDA-FLEUR: Many of our users do not appreciate the use of the workflows from within python scripts but prefer a call from the command line. Hence, we try to extend this interface. (This, by the way, is an example of the variability of use cases and user preferences in the different communities using the MAX codes.)
- The construction of additional workflows for the relaxation of magnetic states. Here, the inclusion of the magnetic torque calculated by FLEUR can lead to significant improvements. These workflows ultimately should couple the results of multiscale simulations back into the construction of larger magnetic supercells to be calculated on exascale machines.

3.2.3 Automatic determination of computational resources

The use of FLEUR on supercomputers is often a complex enterprise, in particular with respect to the choice of computational resources. Here, we already provide to our users a tool to simplify this process. This tool currently relies on a few simple assumptions, like the applicability of Amdahl's law, to predict the scaling of some code parts. In a more ambitious future version, we plan to include a more elaborate performance model, which was derived from actual performance measurements.

3.3 QUANTUM ESPRESSO

The applications and libraries in the QUANTUM ESPRESSO suite are designed and implemented to be reciprocally interoperable and integrated into many complex workflows. For this reason, there is a strong stress on the support of interoperability in all the suite developments. Thanks to this effort, the interoperability of QUANTUM ESPRESSO applications has been rapidly extended to many other applications outside the suite [12]. The features enabling this interoperability —largely developed within the MAX activities— match the interoperability requirements presented in this plan on many points. The basic layer comprises formatted I/O files written in standardised schematisable data formats such as XML and HDF5. The applications that are purposely designed for workflows and molecular dynamics, such as `ph.x`, `neb.x`, and `cp.x` presents the stem for the more general checkpointing and semaphore system we aim to implement for our scientific workflow components.

Looking at these cases presented in the WP2 plan [2], QUANTUM ESPRESSO applications will be involved in various types of scientific workflows: NEB calculations, AiiDA-driven screenings, and non-adiabatic molecular dynamics (NAMD). It will also be involved in some of the many-body perturbation theory (MBPT) workflows performed by YAMBO with the `pw.x` and `ph.x` applications. For this reason, QUANTUM ESPRESSO will have to implement all 4 main interoperability traits and work with YAMBO to improve the interoperability between the two platforms. The rest of the Section is divided into four parts, each analysing the status and implementation plans for one of the four traits.

3.3.1 Formatted I/O

Interoperable I/O in QUANTUM ESPRESSO already adopts standard data formats such as `XML` for small-size data and `HDF5` for larger binary data sets. For XML interoperability, we rely on publicly released XSD schemas [13]. During the previous MAX phases, we have implemented a Python toolchain [14, 15] that allows for the automatic generation of Fortran module for reading, writing, checking the XML files; the same toolchain can be used for automatic parsing of the XML files and generating input files for executing the calculations [16].

For `HDF5`, during the previous MAX phase, we developed a small library for transparent interaction with HDF5 files. The library allows for writing and reading metadata, integer and arbitrary precision real and complex data sets, and reading and writing hyperslabs from data sets.

To implement a more robust interaction with the workflow managers, we have started experimenting with an alternative logging method that is more suitable for automatic parsing and detached from the main standard output and the XML I/O. For more streamlined logging that enables the possibility of appending events to pre-existing files, we have chosen a two-level `YAML` format. The format includes 4 types of events:

- *Status events*: These events register whenever the program crosses a checkpoint. It reports as attributes the CPU time plus a variable number of attributes that depend on the type of checkpoint that has been reached.
- *Warnings*: These events are logged whenever the program encounters a condition that has to be reported to the user. As attributes, it reports the severity of the warning, the name of the routine where it occurs, and a short message.
- *Errors*: Logged when the program encounters a condition for which the calculation can not progress further. It reports as attributes the name of the routine where this occurs and a short error message.
- *Debug*: Enabled at compilation time, log debug messages, such as entering targeted regions and kernels, report as attributes the label of the region that has been entered and the depth of the trace, if nested regions are defined.

3.3.2 Demonstrated workflow examples and their applications

As an example of a workflow using Quantum ESPRESSO, specifically managing the pw.x and hp.x routines, we present in Figure 2 the `AiiDA-quantumespresso-hp` workflow, which enables the self-consistent calculation of Hubbard parameters (namely U, V) with respect to the parameters themselves, the geometry and atomic arrangements of the input structure. The Bremen group is actively working on its optimisation and benchmark. We aim to publish the workflow details, possibly by the next review.

Subsequently, we plan to provide an extensive benchmark study of the workflow on lithium-containing materials from the MC3D database (high-throughput study). The data will be used to train state-of-the-art machine learning models to inexpensively predict these parameters for unknown structures. This represents the beginning of a larger effort to study batteries, focusing on finding new materials as candidates for lithium battery cathodes. This further justifies the research and use of the workflow self-consistent U

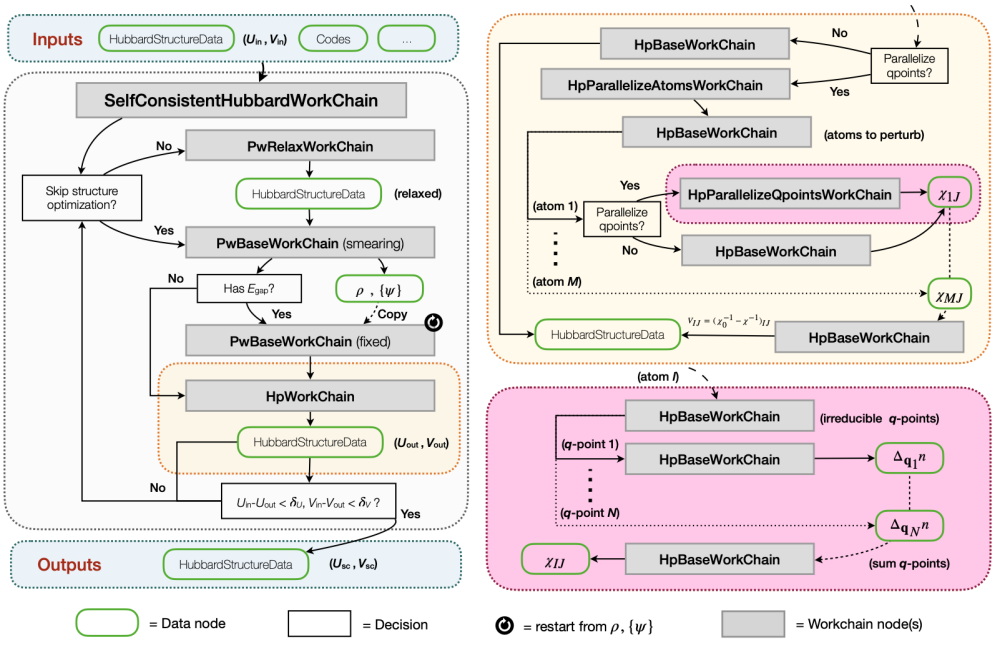

Figure 2: **Schematic representation of the `SelfConsistentHubbardWorkChain` that automates the self-consistent calculation of the Hubbard parameters.** The AiiDA workflow that automates the self-consistent calculation of U and V parameters, which iterates structural optimization via the `PwRelaxWorkChain`, ground-state calculation via the `PwBaseWorkChain`, and the ab-initio prediction via linear-response of the Hubbard parameters through the `HpWorkChain`. In particular, the latter can be used to fully exploit the parallel capabilities of the hp.x code, thus by (optionally) first parallelizing the calculations over independent atoms to perturb, using the `HpParallelAtomsWorkChain`, and then (optionally) parallelizing over irreducible monochromatic perturbations (q-points) via the `HpParallelQpointsWorkChain`. This nested calls can be visualized by the different coloured boxes.

and V parameters, which have recently proven essential for obtaining accurate cathodic properties in a highly relevant class of materials for lithium batteries [17].

3.3.3 Checkpointing and semaphores

The designed features for checkpointing are already implemented in one or more QUANTUM ESPRESSO applications. At this level, the next step to be taken will be the unification and extension of these features. This task also includes a possible API refactoring to streamline their usage. We present these features in separate paragraphs, each describing their current implementation, usage and necessary changes and updates.

Checkpointing and journaling of `ph.x` and `neb.x` workflows. The `ph.x` applications that compute vibrational spectra and `neb.x` that compute transition paths and activation energies are currently the two main applications in QUANTUM ESPRESSO that implement complex workflows composed of many autonomous stages. Depending

on the different dependencies, such stages have to be executed sequentially or concurrently, or as happens for the different perturbations in the phonon workflow, they can be processed asynchronously.

To enable all these types of execution, the two codes have implemented a checkpointing mechanism based on an XML status file. Within this file, each scheduled step of the workflow is described by a record that allows the driving application to know its status, whether a temporary or final data structure for the step is available, and the location of the data. Temporary and final data are organised in a unique directory; files can be saved in different formats according to user preferences.

Each step of the workflow also keeps an internal checkpoint system for restarting in case the execution exits before the end of the task (see paragraph below). We plan to extend this workflow journaling system to all applications in QUANTUM ESPRESSO and implement a meta-structure that allows for unified and coherent programming of the workflow and its journaling and checkpointing systems. This will be first implemented with the refactoring of the `ph.x` workflow.

Trajectories and restart data in `cp.x`. The molecular dynamics workflow collects sequences of many thousands of time-ordered dependent steps. `cp.x` prints an appendable trajectory file with a synthetic description of each time step; restart data and checkpointing are saved for the 2 or 3 last time steps, and the detailed snapshot with density and wave functions may also be saved at fixed intervals. This mechanism allows for performing general molecular dynamics analysis and more refined on-the-fly tasks on the snapshots. During the previous MAX phase, WP5 has implemented a fully functional `AiiDA` plug-in for `cp.x` so that the trajectory file is, in principle, fully interoperable with workflow managers. To streamline this interaction, we plan to explore the conversion of the interoperable trajectory file to YAML format Other planned actions for trajectories and molecular dynamics concern:

- Refactoring of the molecular dynamics output for `pw.x` to make it compatible with the one of `cp.x`.
- Streamlining of the on-the-fly elaboration of molecular dynamic snapshot, adopting XML status descriptors similar to the ones adopted in `ph.x` and `neb.x`.
- Expanding and enhancing the molecular dynamics output, enabling managing and storing multiple trajectory cases such as Tully surface hopping or path-integral molecular dynamics.

Internal checkpointing for recovery and restart. A third type of checkpointing, more relevant for robustness and resilience, is implemented in most QUANTUM ESPRESSO applications. At several execution stages, intermediate recovery dates are saved, and the data can be used to resume interrupted calculations. While conceptually similar to the checkpointing mechanism for workflows we have seen before, the size of stored data is usually larger. For efficiency, each MPI rank saves its data in a distinct file. For this functionality, the planned work includes:

- Improve the integration with the journaling and snapshotting structures presented above for workflow and molecular dynamics.

- Implement whenever possible a parallelism transparent recovery system using collected data structures.
- Implement additional check-point for hybrid functional or currently unprotected lengthy steps.

As an example of the work needed for inserting new checkpoints and the collateral advantages that may come about in terms of interoperability, we report here what done for the **EXX-ACE checkpointing**. Specifically, we have implemented an improved checkpointing [18] of the ACE operator for Fock potential in QUANTUM ESPRESSO [19, 20, 21]. The ACE projector, whose computation is rather expensive, is now stored on disk whenever it is updated during an SCF calculation. It is saved in the same format used for wavefunction I/O and requires the same memory and disk space. The program stores it in the distributed form during the internal checkpoints and as a collected file when the SCF calculation is concluded. These two improvements significantly speed up the restart of hybrid calculations and enable the reuse of the computed Fock potential for band structure calculations on an arbitrary number of (generalised) Kohn-Sham orbitals. This is a substantial improvement in the interoperability with other codes, such as Wannier90 and YAMBO.

Another point to stress is the overall increased efficiency for band structure calculations because, thanks to this feature, the heavy SCF can be split into two steps: a cheaper SCF on occupied bands only and a subsequent non-SCF run, including virtual orbitals. In this way, we avoid evaluating the ACE operator on the virtual manifold at each SCF outer iteration. Since the virtual orbitals are not included in the first SCF step, in the non-SCF procedure, the ACE potential is first read from the file and used as is for a first diagonalization, then it is updated with the new virtual orbitals and a second diagonalization is performed, to get correct virtual band energies. Ongoing work on this point concerns the reuse of the stored ACE with different **k**-point meshes. If only the generalised Kohn-Sham eigenvalues are needed, we have also implemented a computationally inexpensive method [22, 23] to interpolate them from a uniform Monkhorst-Pack grid.

3.3.4 Performance models

Also for this feature, our work stems from work planned and prepared in previous MAX phases, particularly for enhancing and streamlining the execution of high-throughput computations (HPC) with `AiiDA`. Most of the needs in terms of performance models are indeed similar. The main information that our performance model needs to provide are:

- The class and size of HPC resources that must be allocated to efficiently solve a particular computation. Such information must necessarily be accessible to the workflow manager or resource allocator.
- Given the class and size of the allocated HPC resources for the given task, what are the optimal parameters for an efficient parallel execution? This information can be accessed by the resource allocator and then passed to the workflow component as options or computed directly by the task.

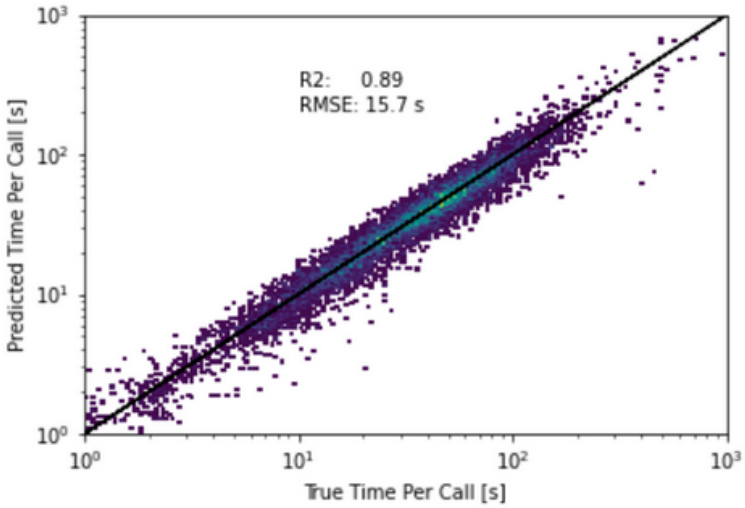

Figure 3: 2D histogram for the true (x-axis) vs the predicted time per call for a validation set of $\sim$ 8000 QUANTUM ESPRESSO calculations. The colour is proportional to the density of points, going from blue (low density) to yellow (high density). The black line is a visual aid that indicates a perfect fit.

While for the optimal parallelization parameters, few simple algorithmic solutions have been tried and hardcoded in `pw.x`, looking for more general algorithms adaptable to all applications and HPC systems is daunting, and we have instead resorted to ML/AI-trainable models that can more easily be adapted and updated for new systems.

Class and size of HPC allocations. For this estimate, CINECA has worked on an AI model designed and trained to estimate the time per call on a given HPC system [24]. A shallow artificial neural network estimates the time-per-call from the input parameters to DFT calculations. The algorithm was trained on a data set containing over 40,000 entries gathered from calculations performed using QUANTUM ESPRESSO as a case study on the Marconi and Piz-Daint supercomputers. The data set spans more than 20,000 chemically diverse systems and consists of calculations run using different parallelization setups and computational resources. The model accuracy (see Figure 3) is sufficient for optimising the computation time allocated by load managers, allowing for shorter queuing times on the users' side and more efficient resource management.

Optimal parallelization. For the estimate of the optimal parallelization parameters, we developed an algorithm based on random forest regression that informs the optimal choice for the number of threads and pools for DFT computations as a function of the computational resources available and of the computation input parameters (number of electrons, number of **k**-points, etc.). The algorithm was trained on a set of 1500 calculations run for the case study of QUANTUM ESPRESSO on the Marconi cluster for 5 chemical systems with varying system sizes, computational resources, and parallelization options.

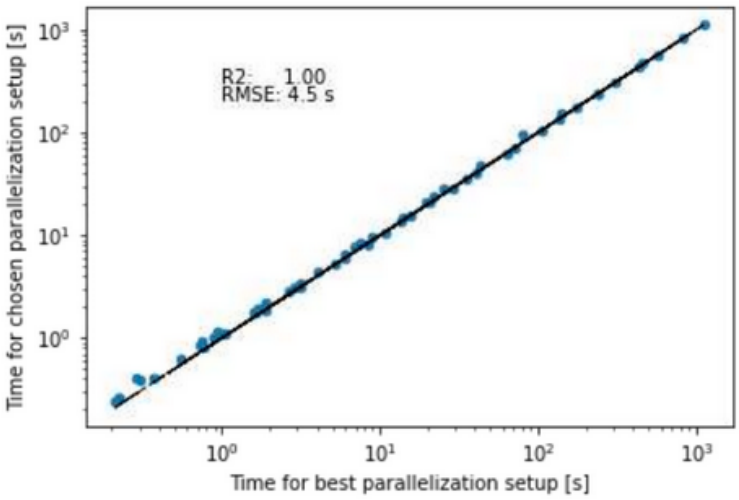

Figure 4: Scatter plot reporting the time required to launch a Quantum ESPRESSO calculation using the best possible combination of number of threads and number of pools (x-axis) and using the number of threads and pools proposed by the optimization algorithm (y-axis). The R2 score and root-mean-square error are reported in the graph.

The algorithm can predict the optimal number of threads and pools in 88% of verification cases. In the remaining 12% of cases where the algorithm does not choose the optimal parameters, the computation times are, on average, larger than 5% of the effective best computation time (see Figure 4). The software and data are available on the GitHub page of the code [25]. Work is ongoing to update the set of supported HPC platforms and extend the models to all relevant applications.

3.3.5 Binary data exchange

In QUANTUM ESPRESSO, the binary data exchange between different processes is currently implemented only via the successive access to binary files, possibly saved in `HDF5` format. Thanks to our internal APIs, such access can occur for slabs or hyperslabs of these files and has, in such a way, a sufficient degree of flexibility. Where needed, the access to such files is synchronised and coordinated by the workflow journaling seen above.

Because of the MAX scientific workflows, we need to work towards two main objectives:

- Improvement of the completeness and organisation of the data sets so that they are fully interoperable inside and outside QUANTUM ESPRESSO. For example, as a priority, we will work on engineering the format and content of the binary data files produced by `ph.x` and read by YAMBO or vice versa (see also subsection 3.5).
- Implement on-the-fly binary data exchange features. To minimize the changes and work in contiguity with the activity on the workflow journaling, we will try to implement these features keeping the same interface as for the binary and status files

access. The work for experimenting with the different back-ends `HyperQueue`, `ZeroMQ` will be done in synergy with the SIESTA group that is currently exploring these solutions more actively (see subsection 3.4).

3.4 SIESTA

3.4.1 Structured I/O

SIESTA currently produces an "output file" which is really a log of the calculation with further data as it is produced, and an XML file in the chemical markup language (CML) dialect, which contains a selection of simulation data and calculation metadata. Most humans use the former, and automated processes such as the SIESTA `AiiDA` plugin use the latter, which is too verbose to be useful to humans.

We are in the advanced stages of planning to consolidate both output files into a single structured file, which will be based on YAML, less verbose than the current CML, and amenable to treatment within various programming languages. Fortran support is available with BigDFT's `Futile` library, although this module integrates other bits of functionality that are not needed by SIESTA or compatible with its current internal operation. In particular, we need to decide whether the underlying dictionary implementation in futile can be adopted by SIESTA and extended to other uses within the code, such as the tables of exposed variables for the built-in Lua interpreter. In any case, other Fortran implementations of YAML parsers and writers are available.

3.4.2 Checkpointing

There is already support for checkpointing in SIESTA at two levels:

- The outer loop of molecular-dynamics or geometry optimisation (saving the current state and any extra variables needed for the restart of the coordinate-update algorithms).
- The self-consistent loop in the determination of the electronic structure. Here, SIESTA can write snapshots of the density-matrix at specific points.

The density matrix is a sparse data structure, but it is distributed over the MPI processes. Currently, the reading and writing is done in native binary form by the root process, which gathers the information from all processes. An improvement is already in the development pipeline: using the netCDF format to streamline the I/O, taking advantage of parallel filesystems when available, and using several MPI processes as writers. This has already been implemented elsewhere in the code for the output of data to be processed by external tools. After writing the appropriate code for the input phase, a further development is the use of extra temporary buffers to enable the asynchronous I/O of this data structure. This will enable overlap of computation (the next step in the SCF cycle, which is quite heavy in terms of compute cycles) and I/O (which is relatively lighter as the data is sparse).

3.4.3 Interface to HyperQueue

SIESTA, due to its intrinsic efficiency in the treatment of large systems, is well-placed to feature in a number of workflows, and in particular in the general kind denoted as

"producer-consumer" in the taxonomy presented in D2.1. Be it in the context of molecular-dynamics simulations, or in calculations of the electronic structure, SIESTA will need to signal the completion of a particular task and the availability of data, it might send the data to a worker process, or it might even spawn a worker process and exchange data with it. As described in the sections above on `HyperQueue` and ZeroMQ, these tools can be profitably used in the workflow logic and data exchange. In the current section and the next one, we outline our plans for their deployment.

With regard to `HyperQueue`, we will take the route of an ad-hoc lightweight wrapper over the shell functionality, avoiding for now the complications of a direct API built in layers over the Rust-based code. Only very few of the HQ primitives need to be wrapped (basically the "submit" one, and perhaps worker spawning, if dynamical load balancing features are needed). Server launching and reservations are to be left to the top level of the workflow. This basic functionality, perhaps coupled to the ability to tag jobs appropriately to give hints to the HQ scheduler, would be enough for most uses involving job spawning. A data layer is being developed for HQ, but we will not offer wrappers for it at this point, preferring to monitor its development and its synergies with ZeroMQ, which is discussed next.

3.4.4 Interface to ZeroMQ

Data exchange is the main functionality of this library, but its robustness and atomicity guarantees make it useful for resource control and load-balancing in the context of "producer-consumer" workflows, such as the computation of properties for a series of snapshots in a molecular-dynamics run, as exemplified in D2.1. An interface to ZeroMQ for MAX flagship codes is thus a key asset for the project. Its main design elements are stacked in layers:

- The basic C-language library code (libzmq [26]).
- A Fortran layer built on top of libzmq (fzmq [27]), developed under the auspices of the U.S. government, and distributed with a Mozilla license, which is compatible with work in MAX .
- A further higher-level layer in modern Fortran supporting basic data structures (e.g. arrays) as well as a mechanism for composability to support the transfer of arbitrary derived types with physical meaning (e.g. the “send the charge density” example mentioned above).

Work on the third layer was initiated by Riccardo Bertossa of SISSA near the end of the second phase of MAX . The SIESTA project, participating already in scientific showcases whose workflows are of the producer-consumer archetype, is working in the further development of the interface.

3.4.5 An enhanced SIESTA API

SIESTA has offered for a while a "SIESTA as subroutine/server" capability, allowing it to run as a subordinate process (in the shell or within an MPI context) to compute the energies, forces, and stress tensors of arbitrary atomic configurations prepared by a client (top level) process. The "subroutine" implementation is indeed based on pure Fortran

calls, using split MPI communicators to separate different SIESTA contexts. (This is needed as the program contains still some data structures that are global variables.) The "server" approach is implemented by launching different shell processes, with whom the client communicates via sockets or filesystem pipes. This is a very useful functionality for the creation of workflows, as it facilitates the interoperability of different pieces. This has been already demonstrated in the realm of advanced simulations by the interface to the i-Pi [28] driver program, using the sockets approach.

The (slight) downside of the latter approach is that a data exchange/handshaking protocol must be established for each application, as the socket interface (and also the one based on pipes) is relatively lightweight, handling only character data, and thus necessitating a very detailed interchange of messages to send and receive data. By leveraging the power of the ZeroMQ library discussed above, we plan to offer a higher-level scheme that can work with derived types appropriate to specific tasks, thus minimising code changes and gaining in robustness. Further, new capabilities beyond the computation of energies, forces, and stresses can be more easily developed.

The MPI-based API offers a more rigidly specified interoperability, as the MPI resources must be known in advance. The upside is the robustness of the function-call paradigm and the MPI communication primitives. Our plans here include a better encapsulation of basic data pieces (e.g., structures, charge densities, even hamiltonians) so that they can be treated as first-class objects and passed around as needed between the subprocesses. This will extend the applicability of the scheme to new problems. As a complementary measure, we have already started to implement modifications in the build system to streamline the compilation of new user-level applications that use the programmatic API.

3.4.6 Parametrization of a performance model for resource prediction

SIESTA does not yet have a performance model similar to the one developed for QUANTUM ESPRESSO during the second phase of MAX , but there are ongoing efforts along that line. In addition to the main factors affecting execution — type of machine and parallelisation details, size of the system, and basis set size — we are taking into account the particular features of the code:

- SIESTA's performance is heavily reliant on the use of external libraries, as most of the CPU time is spent on the solver part. Therefore, a performance model for SIESTA would need to consider the specific libraries used and their performance characteristics on different architectures. For instance, the choice of solver, such as PEXSI or ELPA, can significantly impact performance, particularly for large systems.

- SIESTA's reliance on sparse data structures, while memory-efficient (and essential for the operation of the PEXSI solver), can pose challenges for vectorisation and acceleration on GPUs. Therefore, a performance model would also need to account for the trade-offs between memory usage (algorithm choice) and computational speed on different hardware.

A complete performance model is a long-term effort. The essential features needed to offer approximate information to workflow orchestrators can be implemented in the form

of heuristics based on key parameters such as the number of orbitals, degree of sparsity of the Hamiltonian, and topology of the system (i.e. dimensionality and degree of overlap with the underlying real-space grid). The first steps aim at developing measures for the recording of these parameters, together with execution time information, and machine characteristics, as default for all SIESTA calculations. When enough data is available, heuristics can be mapped, and later artificial intelligence techniques used to refine the model.

3.5 YAMBO

YAMBO and QUANTUM ESPRESSO (QE) have a special relationship, due to their development history. QE is the main engine providing the initial electronic-structure data with which YAMBO performs calculations at a higher level of theory. Consequently, there is already a good basis for interoperability between the two codes. As shown in the sections below, refinements and further features are being developed and planned, and the experience gained will translate also to improvements in the interoperability of both codes (by themselves and in tandem) in larger workflows.

3.5.1 Yambo-QE interoperability: The Exciton-phonon coupling.

The implementation of exciton-phonon (exc-ph) couplings involves a complex workflow combining electron-phonon (el-ph) matrix elements and excitonic calculations, using `ph.x` (utility of the QE suite) and YAMBO, respectively. This implementation is currently ongoing, and has reached an advanced stage. A key aspect of this effort is the refinement of the interface between YAMBO and QE, for data exchange via HDF5 files. An initial version of the interface loaded the el-ph matrix elements (ME) stored by `ph.x` in binary format. However, these ME are not directly usable when computing exc-ph coupling, and require post-processing.

Over the past year, we have refined the interface, by leveraging an external code (LetzElPhC) [29] able to directly load the variation of the electronic potential in real-space, from `ph.x`. LetzElPhC combines this information with the wave-functions, and the convention for the symmetry operations, which are loaded from YAMBO, to generate ME in a format suitable for exc-ph simulations. This approach, which we are currently finalising, simplifies the exc-ph workflow by bypassing the need to compute the ME within `ph.x`. At present, the workflow is handled manually. However, we plan to use the yambopy code (see below) as a workflow manager for production runs.

As a future step, we will consider one or both of the following upgrades: (i) integrating the generation of ME, currently done via the LetzElPhC utility, within the YAMBO code; and (ii) bypassing data exchange via HDF5 files in favour of memory-based approaches. Step (i) is more likely, as it can leverage the interface with the QE-pseudo library, which is also needed for the Ehrenfest dynamics workflow, as discussed in the next Section.

3.5.2 Yambo-QE interoperability: Ehrenfest dynamics and exciton-phonon coupling in real-space.

Ehrenfest dynamics is an upgrade of the real-time module of the YAMBO code, aimed at integrating the electronic (or excitonic) dynamics with the classical displacement of atoms. The development started by the end of the second phase of MAX , and is currently ongoing (the reconstruction of the pseudo-potential has been achieved, and the implementation of the force terms is currently under development). In turn, when the whole approach will be demonstrated, we consider to directly exploit the UPFlib of QE to perform some of the basic operations involving pseudopotentials. We plan to continue this development to reach a first demonstration of the scheme in the second half of the MAX project. The long-term goal is to build a workflow in which data is exchanged between YAMBO and QE on the fly during real-time simulations. This workflow will be handled by yambopy and will use HDF5 for data exchange in its first version.

3.5.3 Yambopy as interoperability layer

As mentioned in the two previous subsections, we plan to take advantage of yambopy [30] to handle workflows which involve the combined usage of different codes and data exchange. Yambopy's current features include reading and editing YAMBO and QE input files, performing pre- and post-processing of the simulation data for these two codes by handling the YAMBO netCDF databases and QUANTUM ESPRESSO XML outputs directly, and providing visualization and plotting options. Besides, yambopy is being used to set up automatization workflows (e.g., convergence tests). These features are already appropriate for the development of the workflows integrating YAMBO and QE, as the ones described above. Further work on abstraction of the yambopy modules will allow the encapsulation of the functionality in larger workflows and enhance interoperability.

3.5.4 Advanced interfaces with linear algebra solvers as workflow elements

Advanced interfaces with linear algebra (LA) solvers are essential for tackling large-scale problems, such as those anticipated for MAX showcases. For non-accelerated HPC systems, YAMBO can automatically download and compile the necessary dependencies for LA operations. During configuration, users can specify which libraries to use among those adhering to the LAPACK and ScaLAPACK API standards. YAMBO includes separate interfaces that are activated depending on the presence of specific libraries.

Top pre-exascale and exascale supercomputers (with the sole exception of Fugaku) feature accelerated architectures, requiring specialised libraries. Focusing on architectures accelerated by Nvidia devices, YAMBO already features an interface that leverages the cuSOLVER LA solver, included in both the CUDA toolkit and Nvidia's SDK (NVHPC). However, this solver can only operate on a single GPU at a time, limiting its ability to address the large-scale problems inherent to the MAX project. Nvidia also introduced a GPU-accelerated distributed memory solver within its SDK (starting from version 21.11), compatible with the 2D block-cyclic data layout characteristic of ScaLAPACK, called cuSOLVERMp. The plan involves using a proof-of-concept mini-app, previously employed for studying cuSOLVER, to replicate the various interfaces in YAMBO and add the necessary interface for interacting with cuSOLVERMp. The primary challenge is creating an appropriate Fortran interfaces for the C-APIs provided by the library.

Once this interface is created and tested, it can be integrated into the YAMBO codebase. This enhancement will enable YAMBO to fully exploit the capabilities of modern accelerated architectures.

Given the importance of this need for most MAX codes and showcases, there are two complementary actions that could provide significant benefits to the project with minimal additional effort. The first action is to further encapsulate the mini-app code and turn it into a stand-alone library that could benefit all codes. This modularization has been a standard practice in the MAX project since the beginning. The second action highlights the blurred distinction between codes and workflows discussed in D2.1: the mini-app used to study and develop the accelerated linear algebra (LA) interfaces can be transformed into a code/module capable of participating independently in a workflow. This module could interoperate with various codes, including MAX flagships and others adhering to the proper protocols. This approach might be less intrusive for existing codes than introducing a new internal library interface. Effectively, this piece would act as a "solver server", which could be further developed to implement new methods as needed, such as those discussed in the paragraphs on Bethe-Salpeter Equation (BSE) below, or to exploit new architectures.

In most LA solvers, it is possible to specify whether the matrix involved in the computation is Hermitian or not. This allows for the use of more or less efficient algorithms accordingly. BSE (Bethe-Salpeter Equation) calculations mathematically require solving a large eigenvalue problem by diagonalizing the entire Bethe-Salpeter matrix. In the most general case, this matrix is pseudo-Hermitian, and currently, no library is available that explicitly considers this specific property.

Recognizing this gap, the developers of the YAMBO code have initiated a collaboration with a team of developers from the SLEPc library. The aim of this collaboration is to design an efficient algorithm for the diagonalization of pseudo-Hermitian matrices. This effort involves a thorough understanding of the unique characteristics of pseudo-Hermitian matrices and developing specialised techniques to handle them effectively, potentially leading to significant improvements in the performance of BSE calculations.

3.5.5 Yambo check-pointing, I/O and report formats

YAMBO performs intensive I/O operations, generating large files in NetCDF/HDF5 format via parallel I/O. This approach, built on top of the initial NetCDF3 implementation, has been crucial for improving MPI scaling and enabling the restarting of simulations in case of run failures. For instance, the `ndb.BS_PAR_Qn` files are generated when building the Bethe-Salpeter matrix. These files (one per each excitonic centre-of-mass momentum) can reach tens or even hundreds of GB each. They are generated at the beginning of the run, and progressively filled. If the run is interrupted, the subsequent simulations check how far the file was filled and continue from that point. The main drawback of the current implementation is the need of three layers of I/O libraries: NetCDF, NetCDF-FORTRAN and HDF5. The same functionality could be achieved by moving directly from the NetCDF API to the HDF5 API, which we plan to implement in the near future. Additionally, we also plan to apply this technique to other run levels, in particular in the calculation of quasi-particle corrections.

Another planned I/O improvement involves the format of report and input files. In the past, we have explored the use of the YAML format for the output report. As discussed

earlier in this document, YAML allows for easy parsing in Python and could also be used for the input file format. Our explorations are based on the `Futile` library from the BigDFT project.

4 Conclusions

The integration quality of a code within a workflow depends on the mechanisms it provides to interact with the external environment. These mechanisms are generally referred to as interoperability.

This report has documented the design choices and implementation plans for the interoperability enhancements in the MAX flagship codes in the context of exascale workflows. Identified interoperability mechanisms include performance models, checkpointing, task spawning, hints for resource allocation, structured output and error logging, and measures for efficient and flexible data exchange.

These measures are being implemented in each code in partnership with Task 1.4 of WP1, using the common patterns shown here, which involve in some cases MAX tools such as `HyperQueue`, `AiiDA`, and `remotemanager`.

References

[1] First report on max software architecture and implementation planning. URL https://www.max-centre.eu/sites/default/files/D1.1_First%20report%20on%20MAX%20software%20architecture%20and%20implementation%20planning_compressed.pdf.

[2] Exascale workflow design. URL https://www.max-centre.eu/sites/default/files/D2.1_Exascale%20workflow%20design%20%281%29_compressed.pdf.

[3] Hyperqueue. URL https://it4innovations.github.io/hyperqueue/stable.

[4] The chemical markup language. URL https://en.wikipedia.org/wiki/Chemical_Markup_Language.

[5] Hintjens, P. Ømq - the guide. https://zguide.zeromq.org/ (2019).

[6] Ratcliff, L. E. *et al.* Flexibilities of wavelets as a computational basis set for large-scale electronic structure calculations. *J. Chem. Phys.* **152**, 194110 (2020).

[7] Aprà, E. *et al.* Nwchem: Past, present, and future. *The Journal of Chemical Physics* **152**, 184102 (2020).

[8] Turney, J. M. *et al.* Psi4: an open-source ab initio electronic structure program. *WIREs Computational Molecular Science* **2**, 556–565 (2012).

[9] Stella, M., Thapa, K., Genovese, L. & Ratcliff, L. E. Transition-based constrained dft for the robust and reliable treatment of excitations in supramolecular systems. *Journal of Chemical Theory and Computation* **18**, 3027–3038 (2022).

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

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

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

[13] Delugas, P. schemas for qe applications. https://github.com/QEF/qeschemas (2018).

[14] Brunato, D. xmlschema. https://github.com/sissaschool/xmlschema (2018).

[15] Brunato, D. Xsdtools. https://github.com/QEF/xsdtools (2018).

[16] Delugas, P. Xsd schema for qe applications. https://github.com/QEF/qeschema (2018).

[17] Timrov, I., Aquilante, F., Cococcioni, M. & Marzari, N. Accurate electronic properties and intercalation voltages of olivine-type li-ion cathode materials from extended hubbard functionals. *PRX Energy* **1**, 033003 (2022).

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

[19] Carnimeo, I., Baroni, S. & Giannozzi, P. Fast hybrid density-functional computations using plane-wave basis sets. *Electronic Structure* **1**, 015009 (2019).

[20] Lin, L. Adaptively compressed exchange operator. *J. Chem. Theory Comput.* **12**, 2242–2249 (2016).

[21] Lin, L. & Lindsey, M. Convergence of adaptive compression methods for hartree-fock-like equations. *Commun Pure Appl Math* **72**, 451–499 (2019).

[22] Koelling, D. & Wood, J. On the interpolation of eigenvalues and a resultant integration scheme. *J. Comput. Phys.* **67**, 253–262 (1986).

[23] Pickett, W. E., Krakauer, H. & Allen, P. B. Smooth fourier interpolation of periodic functions. *Phys. Rev. B* **38**, 2721–2726 (1988).

[24] Pittino, F. *et al.* Prediction of time-to-solution in material science simulations using deep learning. In *Proceedings of the Platform for Advanced Scientific Computing Conference*, PASC '19 (Association for Computing Machinery, New York, NY, USA, 2019). URL https://doi.org/10.1145/3324989.3325720.

[25] Zeni, C. Qe optimal parameters model. https://github.com/QEF/optpara4QE (2018).

[26] Zeromq code repository. URL https://github.com/zeromq.

[27] Fortran binding to zeromq. URL https://github.com/richsnyder/fzmq.

[28] i-pi, a universal force engine. URL https://ipi-code.org/.

[29] Letzelphc: A complementary c code for yambo to compute electron-phonon coupling matrix elements with full crystal symmetries. URL https://github.com/muralidhar-nalabothula/LetzElPhC.

[30] Python utility for managing and postprocessig quantum espresso and yambo calculation. URL https://github.com/yambo-code/yambopy.