D5.3

First report on workflows for the various flagship codes and on the capability of running them on pre-exascale machines

Marnik Bercx, Miki Bonacci, Emanuele Bosoni, Jens Bröder, Augustin Degomme, Andrea Ferretti, Alberto Garcia, Luigi Genovese, Sebastiaan P. Huber, Nicola Marzari, Giovanni Pizzi, Francisco F. Ramirez, Vasily Tseplyaev, Daniel Wortmann, and Aliaksandr Yakutovich

Due date of deliverable:	31/05/2020
Actual submission date:	31/05/2020
Final version:	31/05/2020

Lead beneficiary:	EPFL (participant number 6)
Dissemination level:	PU - Public

Document information

Project acronym:	MAX
Project full title:	Materials Design at the Exascale
Research Action Project type:	European Centre of Excellence in materials modelling, simulations and design
EC Grant agreement no.:	824143
Project starting / end date:	01/12/2018 (month 1) / 30/11/2021 (month 36)
Website:	www.max-centre.eu
Deliverable No.:	D5.3

Authors: M. Bercx, M. Bonacci, E. Bosoni, J. Bröder, A. Degomme, A. Ferretti, A. Garcia, L. Genovese, S. P. Huber, N. Marzari, G. Pizzi, F. F. Ramirez, V. Tseplyaev, D. Wortmann, and A. Yakutovich

To be cited as: M. Bercx, et al. (2020): First report on workflows for the various flagship codes and on the capability of running them on pre-exascale machines. Deliverable D5.3 of the H2020 project MAX (final version as of 31/05/2020). EC grant agreement no: 824143, EPFL, Lausanne, Switzerland.

Disclaimer:

D5.3 First report on workflows for the various flagship codes and on the capability of running them on pre-exascale machines

Content

1 Executive Summary

One critical goal of MAX is to enable computation of materials properties on current and future (pre)-exascale machines.
Beside the critical aspects of scaling the simulation codes (see WP1, WP2, WP3 and WP4 of MAX) to be able to run efficiently on large supercomputers, in WP5 we focus more specifically on how to reach exascale via high-throughput calculations, and more specifically by a convergence of high-performance computing (HPC), high-throughput computing (HTC) and high-performance data analytics (HPDA).
One of the key aspects of scaling to exascale computing *via* HTC is a robust and scalable workflow engine. In the case of MAX, this is AiiDA. In the past 18 months, AiiDA has seen a complete restructuring, with major rewrite of critical parts of the code (like the workflow engine) and major optimizations on storage, querability and efficiency. These have been released in AiiDA 1.0 in October 2019, and additional features have been added in versions 1.1 and 1.2 of AiiDA, released in February and April 2020 respectively. We present the major improvements of AiiDA in Sec. 2, outlining how they now make it possible to deal with tens of thousands of "processes" (calculations and workflows) per hour, making AiiDA ready to scale to exascale HTC research projects composed of multiple HPC runs.
A second key aspect of HTC is the robustness against code failures. Beside a number of key improvements in AiiDA to recover from errors and problems common to all plugins (node crashes, walltime errors, network failures, ...), we have moved into AiiDA core an essential software components, called the BaseRestartWorkChain, that facilitates implementing error recovery for any code, using a common interface, that of course needs then to be customised by each plugin to take care of the code-specific possible failures. These aspects are also discussed in Sec. 2.
From Secs. 3 to 8 we then detail the work performed on the automated workflows for each of the MAX codes. For each of them, we have decided to address workflows that are critical to the strength points and relevant aspects of each of them.
In particular, in Quantum ESPRESSO (Sec. 3) there has been a lot of focus on increasing the robustness and the success ratio of calculations for very common and essential tasks like computing the total energy and performing a variable-cell relaxation. Additionally, support of GPU-accelerated Quantum ESPRESSO using the SIRIUS library has been implemented, together with accurate verification and validation benchmarks of the GPU results against the standard Quantum ESPRESSO CPU-only results.
For Siesta, that is based on a localised basis set, focus was on major improvements of the workflows to compute STM microscopy images. Moreover, automatic computation of equation of states has been implemented, needed both to compute elastic properties of materials, and to benchmark the code against other ones. Additionally, work has been

performed to benchmark GPU performance and to define automated protocols to select numerical code parameters.
For Yambo, the majority of the human and CPU time is typically spent in the complex convergence analysis that needs to be run for each material. Therefore, sophisticated workflows are now available to perform, automatically and efficiently, convergence studies for GW calculations. Moreover, verification of the GPU versions of the code has been performed.
A strength point of FLEUR is its capability to compute magnetic properties. Therefore, workflows to compute the magnetic anisotropy energy, the Dzyaloshinskii-Moriya interaction and the Heisenberg exchange interaction energies have been developed. In addition, also in the case of FLEUR significant efforts have been made to increase the robustness against code failures.
For CP2K, robust workflows have been developed to address the use case of computing properties of nanoporous materials, critical in a number of applications like gas separation, sensing, ...
Finally, BigDFT has joined MAX in phase 2. Therefore, the first goal has been to develop a plugin to drive the code with AiiDA. Work is now devoted to the implementation of robust workflows to compute materials properties with BigDFT, leveraging existing python libraries developed by the BigDFT team to facilitate the selection of appropriate calculation parameters.
In Sec. 9, finally, we outline the current progress to provide common workflow interfaces across all codes to perform common materials-science oriented simulation tasks. A preliminary protocol has been already designed for calculations performing a relaxation of a crystal structure, and the various MAX flagship codes are now implementing it. This will also act as the template for further extensions to other relevant simulation types.

2 Improvements in AiiDA core

2.1 Engine and database performance

The engine and database of AiiDA have been significantly improved towards enabling managing high-throughput computational loads on pre-exascale clusters. The engine had to be adapted in order for it to be able to sustain the workloads that might involve tens of thousands of concurrent tasks per hour, which could furthermore be distributed on multiple computational resources. The database, on the other hand, needed to be modified to increase the performance of its queryability so that users would be able to apply data analysis to very large provenance graphs containing millions of nodes or more. These improvements have been released with AiiDA v1.0[1] and will be described below.

[1] S.P. Huber *et al*., arXiv:2003.12476 (2020)

The new implementation of the engine consists of independent operating-system processes that are executed in parallel, called runners, which can process tasks concurrently by distributing the workload involved in the execution of workflows and calculations among themselves. These are supervised by a daemon, which is in charge of monitoring them and relaunching them if they were to die, whereas task distribution is organized via a task queue implemented through RabbitMQ (which, in turn, uses the AMQP protocol). This guarantees a reliable scheduling of tasks and a reaction to events (such as the submission of calculations or the continuous steps of workflow) that is almost instantaneous.
Robustness is another important aspect to consider when managing big workloads: for this, an optimised algorithm to automatically reschedule failed tasks was implemented. This algorithm works through an exponential-backoff mechanism, which recognizes common transient errors (such as connection issues) and can recover from them. It will also pause the process if a task fails multiple times consecutively (for example because the remote machine went offline), which can then be resumed effortlessly by the user once the underlying problem is resolved. Moreover, the AiiDA engine will make sure that no data is lost when the calculation is resumed from the point where it had to be paused.
Finally, when doing high-throughput workflows, one also needs to take into account the technical limits on the rate of connections or scheduler requests to computing clusters. To avoid overloading these (which ends up rendering unresponsive not only to the responsible party but to other users as well), AiiDA implements a connection pooling algorithm. It essentially consists of grouping the requests made to the same computer, so that new ones make use of the available existing open connections instead of creating others. In case no available connection already exists, AiiDA will open a new one while guaranteeing that a minimal time interval between requests (which can be configured) is respected. For similar reasons, it will also cache the list of jobs in the scheduler queue and only refresh their state periodically, with a minimum configurable timeout.
In order to compare the performance of this new engine with respect to earlier AiiDA versions, a very minimal test with an example work chain that performs two simple and fast arithmetic operations was designed: it first submits a CalcJob to compute the sum of two inputs and then uses a calcfunction to perform a second addition. The “ArithmeticAddCalculation” implementation used for the CalcJob is a simple wrap of a Bash script that sums two integers. Each work chain then consists of the execution of three processes (top work chain, a calculation job and a calculation function) that make it representative of typical use cases. Figure 2.1 shows the rate of submission and process completion: for each benchmark, 400 work chains were submitted to the daemon on a machine with an Intel Xeon E5-2623 v3 CPU with 64GB of RAM. The benchmark for the new engine was run both by using an optimal set of parameters (such as number of workers, transport intervals, etc.) and under some artificial constraints that made the conditions more

similar to the ones under which the old engine was working (number of parallel workers were and value of the minimum interval between connections.

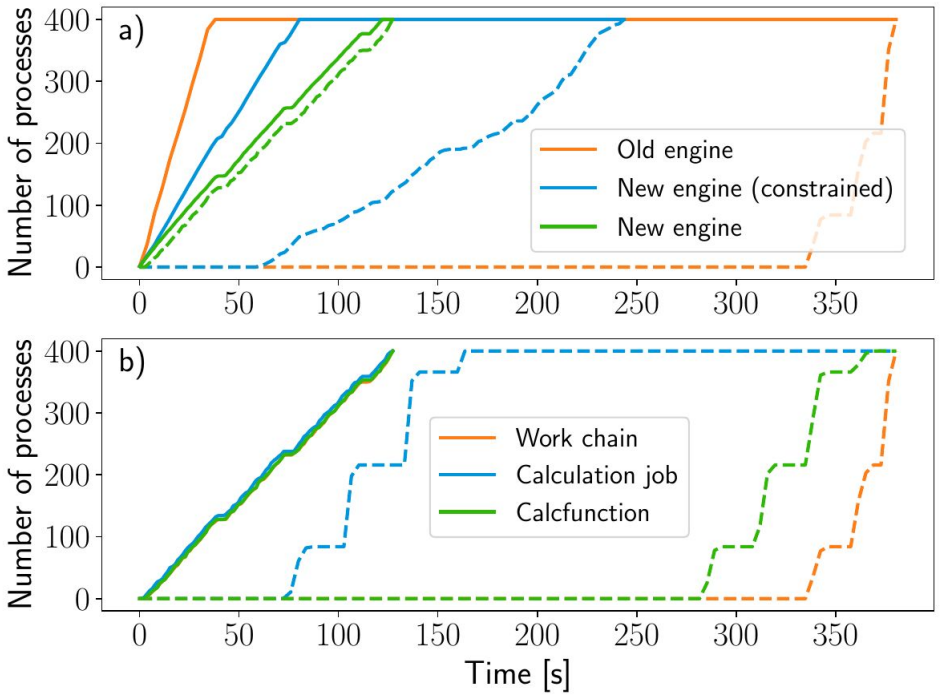

Figure 1: Process submission and completion rates for the old and new engine. (a) Number of submitted (solid lines) and completed (dashed lines) processes over time for the new engine (both with optimised parameters and with artificial constraints) and the old engine. (b) Number of completed processes for the old (dashed lines) and new (solid lines) engine, decomposed in the separate (sub)processes. Adapted from S. P. Huber et al., arXiv:arXiv:2003.12476 (2020).

The results displayed above in Figure 1 are organized into two different graphs: one that considers each of the 400 workchains as units (a) and another that shows the completion for each component independently (top work chain and each of the two subprocesses). To understand the second graph is also important to know that, although the workflow only runs two subprocesses, the internal logic consists of multiple steps (such as a setup and checks), which can be seen being executed by the old engine between 0 and 50s and between 175s and 275s. From a deeper analysis of these graphs one can derive the following observations:

1. (Panel a) The new engine seems to have a slightly slower submission rate compared to the old engine, due to the extra complexity added because of the communication with the daemon workers. Even so, the new engine completes all processes faster in both conditions, and achieves a total completion time three times shorter than that of the old one in the optimized example.

2. (Panel a) The old engine finishes all work chains roughly at the same time, towards the end of the time window, and it does so in discontinuous jumps because of its polling-based design. On the other hand, the new engine has a much smoother completion rate due to its continuous operating character, in which processes are managed concurrently and executed immediately, and there is no polling interval acting as a bottleneck.
3. (Panel b) These curves highlight the effect of the concurrency and one of the main drawbacks of the polling mechanism. Whereas the new engine can execute all steps of the workflow in quick succession as the previous finishes, the old implementation needs to wait for the whole polling interval and is only able to process one of the steps during each, thus creating this step-function shape.

Finally, it is important to mention that the increase in the efficiency achieved by the new engine would be even more pronounced in a real high-throughput research situation, since the new daemon would never be idle between polling intervals. Moreover, this new implementation is scalable and it is able to dynamically increase the number of daemon workers when it has to distribute heavier workloads. This is further supported by the performance tests, in which one can see that the new engine takes half the time to complete all processes when it runs on optimal conditions, compared to the constrained situation. Therefore, the results obtained here of roughly 35000 processes per hour on a modest workstation can easily be scaled when using a more powerful machine by simply running more daemon runners.

In regards to the database efficiency, when storing the attributes of a node in the database, the exact schema for these will depend on the node type and cannot be statically defined *a priori*, which is in direct conflict with the relational database model used by the SQL DB engines. The way in which this was overcome originally was by implementing an extended entity-attribute-value (EAV) table which would allow us to store these values as arbitrarily nested serialisable attributes. This solution came with an increased storage cost and a reduction of the querying efficiency, due to the considerable overhead involved in the (de-)serialisation of data. Luckily, as storing semi-structured data became a more common requirement for other applications, PostgreSQL eventually added support for a native JSON and JSONB data type. The addition of support for this efficient storage and indexing format allowed us to drop the custom EAV implementation in favour of PostgreSQL's native JSONB solution, which ended up yielding significant improvements in both storage cost and query efficiency.

The reduction of storage cost is a product of both (a) the data itself being stored more compactly in a single column instead of in an entire table, and (b) being able to remove database indexes while still providing a superior query efficiency. To show the effects of this, benchmarks were performed by measuring the space occupied when storing 10 000 crystal structures either as raw files on disk, or with the new ORM backends of AiiDA 1.0, or with

the old EAV format. For the raw files, the XSF format[2] was used since it contains only the information that is absolutely necessary, and PostgreSQL 9.6.8 was the underlying database engine used. The results, presented on Figure 2a, clearly show a decrease of almost two orders of magnitude in storage space of the new schema with respect to the old one (and even by a factor of 1.5 when comparing to the raw file size).

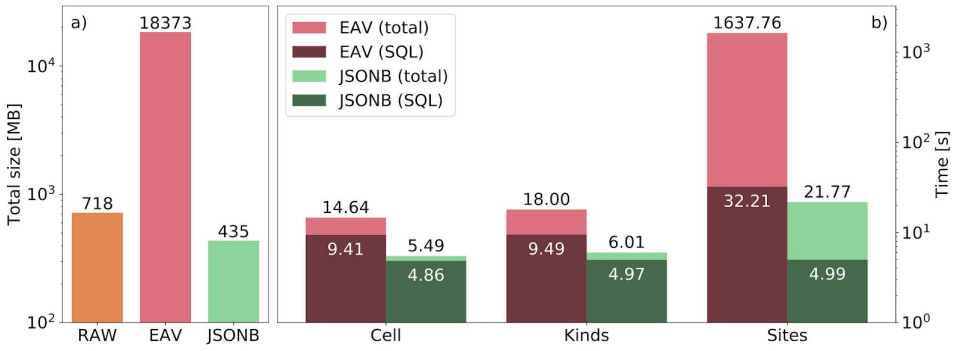

Figure 2: Comparison in a log scale of the space requirements and time to solution when querying data with the two AiiDA ORM backends. (a) Space needed to store 10000 structure data objects as raw text files, using the existing EAV-based schema and the new JSON-based schema. (b) Time for three different queries that return attributes of different size for the same set of nodes. The benchmarks are run on a cold database, meaning that the database caches are emptied before each query. We indicate separately the database query time (SQL time) and the total query time which includes also the construction of the Python objects in memory.
Adapted from S. P. Huber et al., arXiv:arXiv:2003.12476 (2020).

Changing to the JSONB-based format also significantly improved the speed of the queries. One of the main benefits to this comes from the reduced amount of data transferred from the database to AiiDA thanks to the more compact schema with respect to the EAV, which is the exact same underlying cause of the previously mentioned reduced disk usage. This also results in a reduced cost when deserializing the raw query result into Python objects. To measure this improvement, speed benchmarks were carried out on a machine with an Intel i7-5960X CPU and 64GB of RAM running PostgreSQL 10.10. Three different kinds of attributes were queried on a subset of 300000 crystal-structure data nodes of an AiiDA database of 7318371 total nodes, corresponding to the results of one of the group's publications[3]. The attributes were:

1. cell: is a 3 × 3 array of float;
2. kinds: contain information on each atomic species;
3. site: contains the coordinates for each atom.

[2] http://xcrysden.org/doc/XSF.html
[3] N. Mounet, et al. Nature Nanotech. 13, 246–252 (2018).

There is one cell array per structure, and typically one kind per chemical element contained in it, whereas there is a site for each of the individual atoms in the unit cell of the structure. Given the specific format of the EAV, this schema needed to retrieve more rows per crystal structure node than the new one. The effects of these different sizes becomes noticeable both in the SQL time (which only includes the time to perform the query and to get the result from the database) and in the total amount of time spent (which also includes the deserialisation into Python objects of the raw results). It can be seen in Figure 2b how the total querying time for the site attributes in the new format is 75 times smaller than the equivalent query in the old one. However, this same case has an SQL time for the JSONB version that is roughly 6.5 times smaller compared to the EAV one. This dramatic difference given by the final speedup increase can be explained thanks to the JSON-based schema being able to mostly avoid the overhead involved in serialising the attributes at the Python level.

2.2 The BaseRestartWorkChain implementation in AiiDA core

The codes that are currently typically run with the help of AiiDA are high-performance computing codes, such as the MAX flagship codes. These often implement complex computational methods that accept a wide variety of inputs that, unfortunately, lead to many possible failure modes. In addition to the intricacies of the codes themselves, they are typically run on advanced computational infrastructures that add another layer of complexity and yet more potential manners in which a calculation can fail. All of these potential failures need to be handled by an automated workflow for these codes to become robust and reliable tools.

In the first MAX project, the aiida-quantumespresso plugin included a BaseRestartWorkChain, that provides the basic infrastructure to make it easy to add error handlers to any calculation run through AiiDA. Even though originally applied for the Quantum ESPRESSO codes, it was designed from the beginning to be generically applicable and soon found adoption in the plugins of the other MAX codes. Due to the success of the concept of the BaseRestartWorkChain, it has been moved to the core code of AiiDA and is available in the AiiDA as of v1.1. This base class can now be used to very quickly and easily write a robust workflow for any HPC code with error handlers for a wide variety of generic and code specific problems, and is already being adopted by all other MAX flagship codes.

3 AiiDA-Quantum ESPRESSO workflows

3.1 Improvements in workflows

The workflows of the aiida-quantumespresso[4] package, centered around the pw.x code of the Quantum ESPRESSO suite, have seen a lot of development. The main goal has been to develop a workflow, the PwWorkChain, that can compute the relaxed ground-state of any crystal structure without any further user inputs. This workflow would be a powerful turn-key solution that can be used as the starting point for many other tools computing complex structural properties (see also discussion on common workflows at the end of this report). The PwWorkChain first performs a reconnaissance SCF calculation (using the PwBaseWorkChain to automatically recover from code failures) to determine the electronic and magnetic properties of the structures, which are then passed to an input protocol that determines the optimal parameters to be used for the variable-cell relaxation calculation in order to compute the relaxed ground state of the structure (through the PwRelaxWorkChain). The workchain has been applied to the structures of three databases of inorganic compounds, COD[5], ICSD[6] and MPDS[7], to create a single database of the relaxed ground state of all inorganic compounds as computed with DFT. This project is a work in progress and will be published soon.

A lot of development and care has been dedicated to improve the parser responsible for the parsing of the output of pw.x. Not only does it now support the new XML format (the new default since Quantum ESPRESSO v6.2), the parser also comes with a lot of improvements to detect various subtle problems that can occur during relaxation calculations. These new features are now automatically tested through an extensive unit test suite that is run on every commit through continuous integration.

Going into the technical details, the PwBaseWorkChain is the lowest level workflow that wraps the PwCalculation which is the AiiDA plugin to run a pw.x calculation. This base workchain, implemented using the BaseRestartWorkChain as described in Section 2, implements all the logic to recover from any potential error that might be encountered when running a Quantum ESPRESSO pw.x calculation. Thanks to this error handling, workflows that would have normally failed can now continue and finish successfully due to the workchain automatically responding to the problem. Figure 3 shows a schematic of the logic of the workchain and an example provenance graph that is automatically generated by AiiDA when the workchain is run. The histogram in the top right, showing how many iterations were needed for the calculation to complete successfully, illustrates the criticality

[4] https://github.com/aiidateam/aiida-quantumespresso
[5] https://www.crystallography.net/cod/
[6] https://icsd.fiz-karlsruhe.de/
[7] https://mpds.io/

of this automated error handling. Indeed, a big part of calculations requires at least one restart. As a note, we emphasize despite the error handling there are still some workchains that fail to finish successfully, but are able to achieve a current success rate of 91%, and further work is undergoing to increase further this rate.

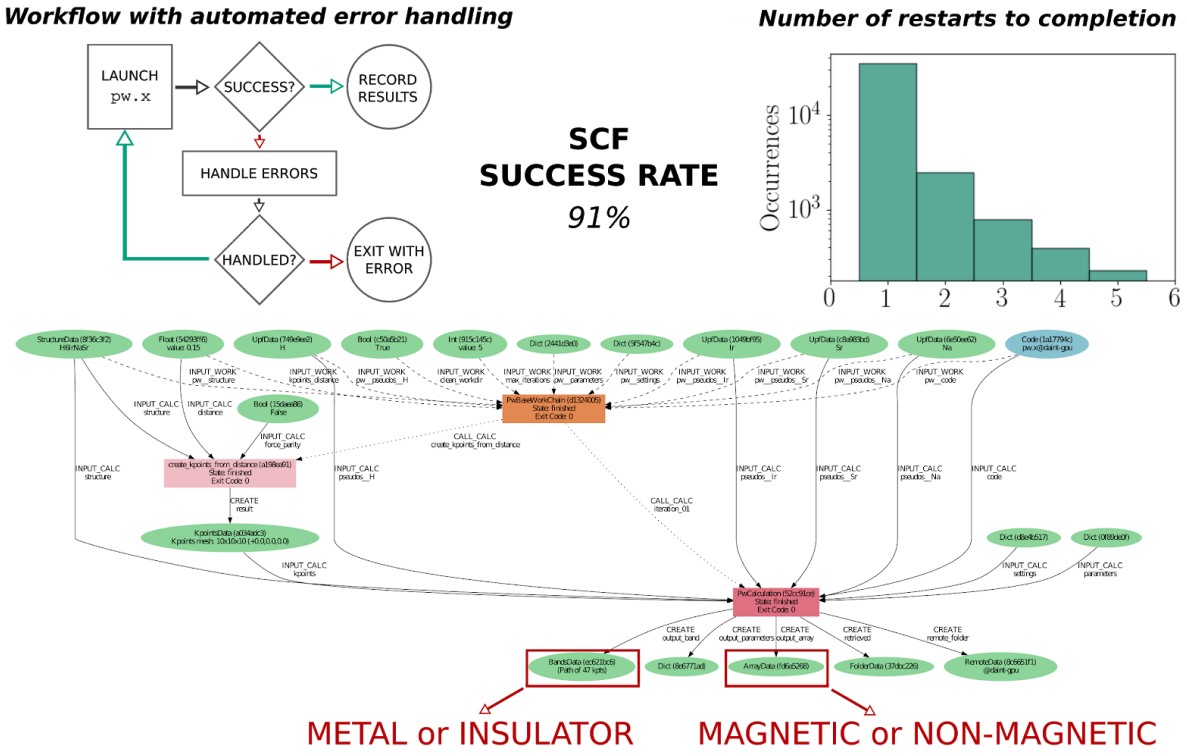

Figure 3: The PwBaseWorkChain is the base workflow used to run pw.x calculations and includes various error handlers to recover from common problems such as the job running out of walltime or the calculation having included insufficient electronic bands. In the turn-key solution workflow, the PwBaseWorkChain is used to run a first reconnaissance SCF calculation in order to determine the electronic and magnetic configuration of the material.

The PwRelaxWorkChain is a higher level workflow that is designed to compute the relaxed ground state of a given structure. It launches multiple pw.x calculations in variable-cell relax mode, in order to relax both the cell as well as the atomic positions, until the all forces and stresses are below a desired threshold. If the relaxed cell volume between two consecutive relaxations is below a certain threshold, the structure is considered relaxed. This additional layer of convergence prevents local minima from being incorrectly labeled as the relaxed ground state. For each relaxation calculation the PwRelaxWorkChain leverages the PwBaseWorkChain to run the actual calculation as to profit from the error handling that is implemented on the latter. Figure 4 shows an example provenance graph created by running a PwRelaxWorkChain. The two consecutive relaxations performed by the PwBaseWorkChain

are highlighted. Once relaxed, a final SCF calculation is performed, again using the PwBaseWorkChain to obtain the charge density of the relaxed ground state. The histogram at the bottom right shows the relative volume change of the input crystal structure compared to the computed ground state. The current work-chain has a success rate of 81%. The majority of failures come from a failure of pw.x to reach convergence in the self-consistent cycle.

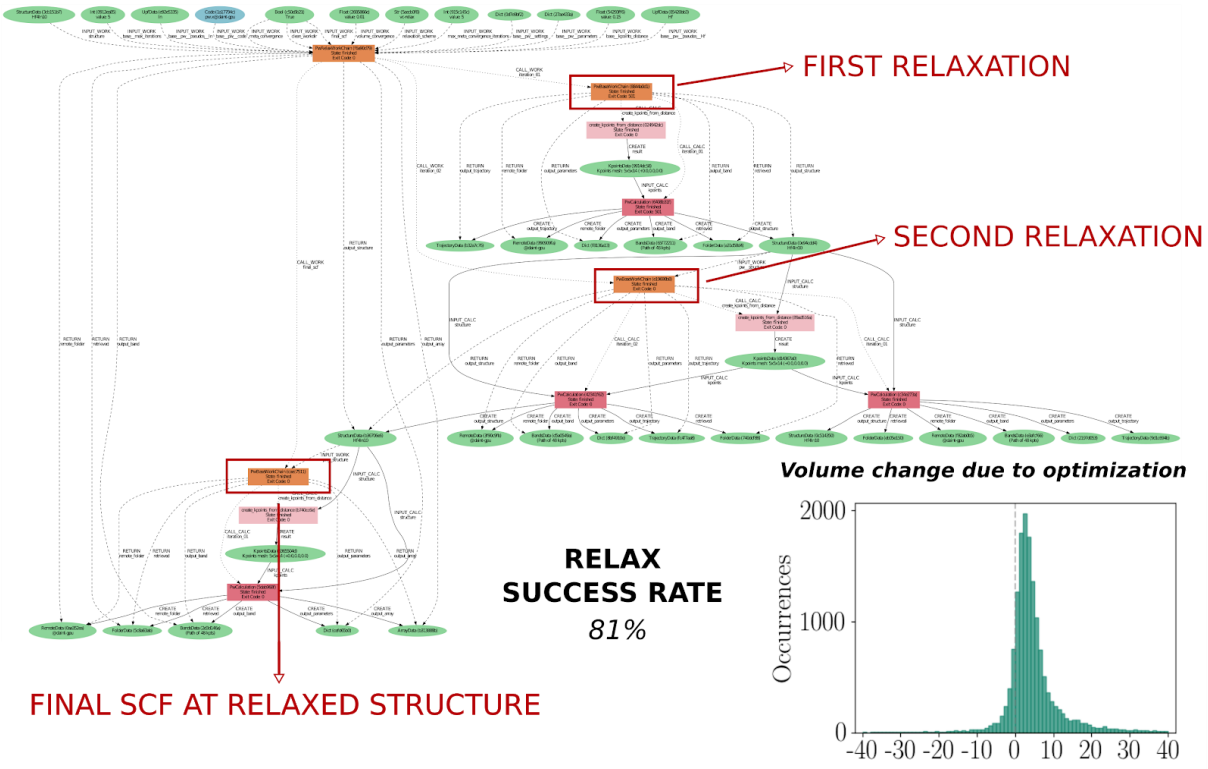

Figure 4: example provenance graph of an execution of the PwRelaxWorkChain. The first and second relaxation, performed through a PwBaseWorkChain are highlighted. The output structure of the second iteration is then used for a final SCF calculation to compute the ground state charge density. The current version of the PwBaseWorkChain has an effective success rate of 81%.

All the code described above is publicly available through the aiida-quantumespresso package.

3.2 Support for GPU with SIRIUS

One of the goals of the MaX project is to refactor the flagship codes in order to separate the low-level mathematical and system libraries that interface with the underlying hardware architecture. The SIRIUS library is designed for this purpose and aims to optimize the

performance of electronic structure calculations relying on plane wave codes such as Exciting, Elk and Quantum ESPRESSO (QE). By isolating the bottlenecks in these DFT codes, the software also becomes both easier to maintain and more portable towards different hardware architectures. SIRIUS has recently been extended to offer support of single-GPU architectures such as Piz Daint @ CSCS, but this feature has yet to be thoroughly tested. By leveraging the workflows in AiiDA to test both the reliability and performance of the SIRIUS-enabled QE code versus the native QE parallelization, we performed an extensive benchmark of SIRIUS-QE. Once confident that the new code produces results consistent with native QE, we are using it to improve the speed of the aiida-quantumespresso workflows on GPU architectures. Moreover, SIRIUS is currently being extended with variational techniques for solving the electron density, which have the potential of significantly improving the robustness of the electronic optimization for complex systems.

To test the performance of the SIRIUS-QE we have benchmarked it versus native QE based on a static calculation of 518 structures, using the SCF mode of the PwBaseWorkChain described earlier. The atomic positions of the structures were randomly perturbed by a small amount to obtain forces and pressures that deviate from zero.

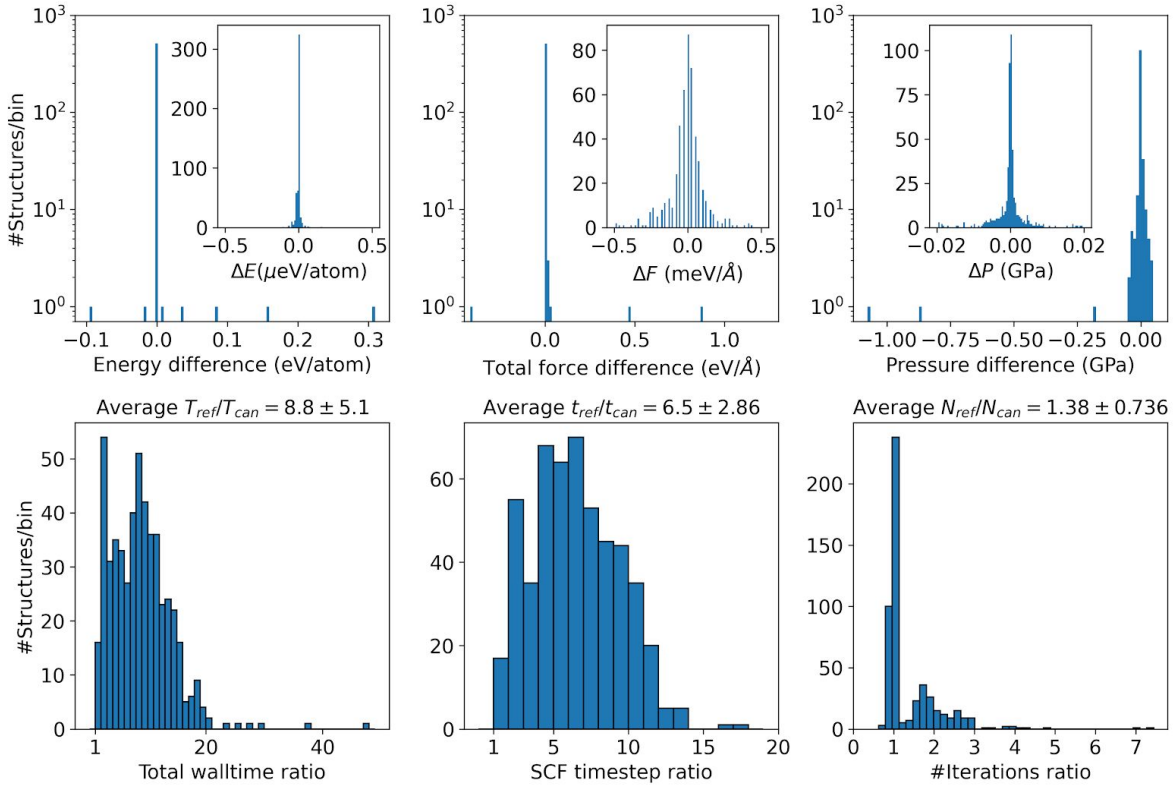

Figure 5: Benchmark results comparing the energies, forces and pressures of the 518 completed static runs for SIRIUS-QE and native QE. The top figures plot the distribution of the differences in energy per atom, total force and pressure, where the insets focus on a smaller range to give a clearer

picture of the distribution. Note that the bins of the force difference histogram are separated by a certain value, which is due to the precision which QE uses to print the total force. The bottom figures show the ratios of native QE (reference) to SIRIUS-QE (candidate) for the total time to completion T_{ref}/T_{can}, the average time per self-consistent field (SCF) step t_{ref}/t_{can} and the number of iterations required to reach convergence N_{ref}/N_{can}.

Fig. 5 shows the benchmark results for the 518 structures. The average absolute value of the differences is 1.4 meV, 3.6 meV/Å and 7.5 MPa for the energies, forces and pressures, respectively. However, this relatively large value is mainly due to some outliers, visible in the top plots of Fig. 5. When taking the average over the range shown in the insets – which contain over 95% of all structures – the values drop down to 0.065 eV, 0.077 meV/Å and 2.8 MPa. We have reported the outliers to our collaborators at CSCS, and they are looking into the issue. Looking at the walltime ratios, SIRIUS-QE is consistently faster than native QE, with an average increase in performance of 780% for the total walltime. This is largely because of a reduction in the average time step, however the number of iterations to reach convergence is on average lower for SIRIUS-QE as well.

4 AiiDA-SIESTA workflows

The workflows and turn-key solutions built around the MAX flagship code SIESTA are collected in the python package aiida-siesta, already available in the MAX website and on GitHub[8]. The package is being continuously updated and improved. In particular, the documentation has been brought to a new level of clarity and an automatic-testing system has been put into place. This is of great benefit for the further development of the package.
In the next three subsections we will explain in detail the three main new features introduced in the workflows section of aiida-siesta. In the fourth subsection, an example of the use of the base aiida-siesta workflow to benchmark the performance of the Siesta code on pre-exascale machines is reported.

4.1 The Equation of state workflow and the Delta Test

The first enhancement concerns the introduction of a new AiiDA workflow called EqOfStateFixedCellShape. It is a tool for the calculation of the equation of state of a solid. Density Functional Theory (DFT) calculations with the SIESTA code are performed at 7 equidistant volumes around a starting volume in order to obtain the energy (E) versus volume (V) data. The workflow ensure robustness in the convergence of each SIESTA calculation thanks to the fact that each DFT run is submitted through the SiestaBaseWorkChain, that automatically manages some common failures (lack of

[8] https://github.com/albgar/aiida_siesta_plugin

electronic-structure or geometry relaxation convergence, termination due to walltime restrictions, etc). As the name of the class suggests, the EqOfStateFixedCellShape workflow is designed to obtain the E(V) curve under the restriction of fixed cell shape. This means that no algorithm for stress minimization is implemented in the workflow. However the option relaxation MD.ConstantVolume might be added into the input-parameters dictionary to let SIESTA to relax the structure at fixed volume. The workflow also tries to perform a fit on the calculated E(V) data with the Birch-Murnaghan function. This allows to directly extract some important properties of the solid under investigation, namely the equilibrium volume and the bulk modulus.

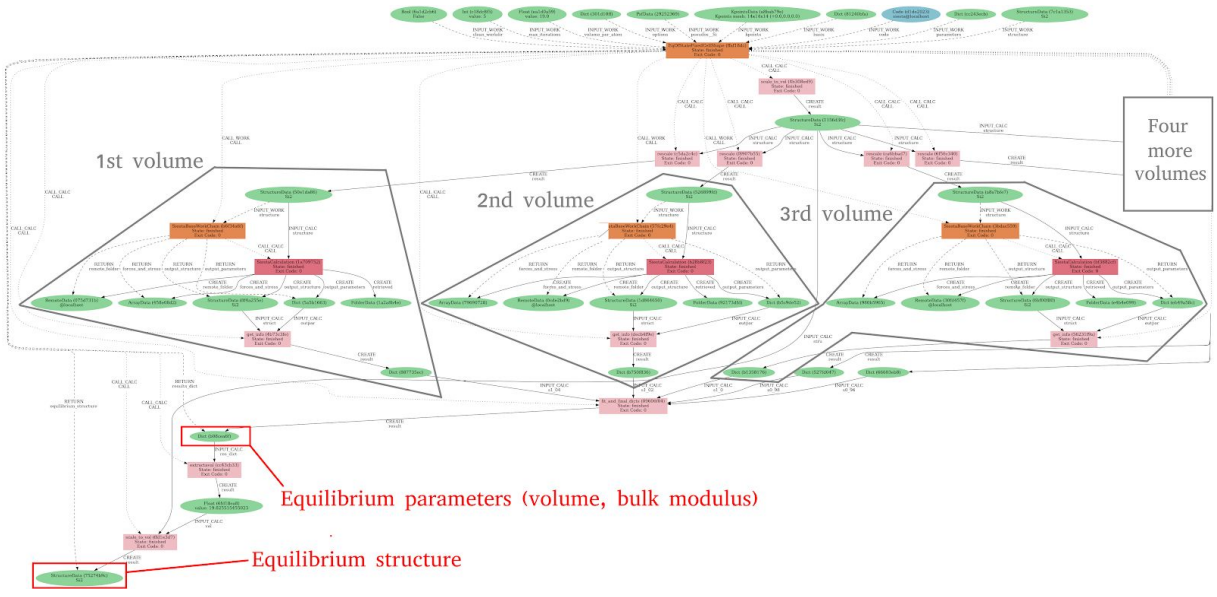

Figure 6: Example of the provenance graph of an EqOfStateFixedCellShape workflow. For readability, only the calculations for three out of seven volumes are reported on the provenance graph.

The EqOfStateFixedCellShape workflow is the essential building block for the automation of the so-called DeltaTest[9]. In the paper "Reproducibility in density functional theory calculations of solids"[10] this test was employed to assess the reproducibility (within a certain error) of the results obtained with 15 different DFT codes on 71 elemental crystals. SIESTA was not part of the study, mostly due to the difficulties at the time in selecting an appropriate basis set automatically. The EqOfStateFixedCellShape workflow will be harnessed to perform the test for the desired sets of SIESTA input parameters. In particular, it will be interesting to test the performance for a range of basis set choices. Even though

[9] https://molmod.ugent.be/deltacodesdft
[10] Kurt Lejaeghere *et al.*, Science **351**, aad3000 (2016)

the original DeltaTest involves only elemental crystals, the EqOfStateFixedCellShape workflow can be used on a broader set of materials as one of the tools to curate basis sets. This is a step towards the validation of the SIESTA code as an accurate tool for the calculation of materials properties on a large set of crystals, one of the objectives of WP5.

4.2 Re-design of the protocols system

The workflow SiestaBaseWorkChain allows to perform various tasks in a robust way, above all the relaxation of a structure and the calculation of the band dispersion. So far, for each system, the selection of the SIESTA input parameters to be used in the workflow was entirely up to the user. We implemented the class ProtocolRegistry with the purpose to collect some standard choices of inputs. These choices are made available to the users through specific APIs tailored for specific tasks. At the moment we have implemented two APIs, one for the task of the relaxation of a structure (class SiestaRelaxationInputsGenerator) and one for the generation of band structures (class SiestaBandsInputsGenerator). The method get_builder of the aforementioned classes returns a builder for the SiestaBaseWorkChain with pre-compiled inputs that can be directly submitted. The suggested inputs can be modified by the user before submission, leaving anyway full flexibility.
The protocols, at the moment, are embryonic and for real production calculations there is no guarantee that the suggested inputs are sufficiently accurate in any situation. Users are invited to tune the suggested inputs on the specific system under investigation. We are furthermore designing a mechanism for dynamic loading of (curated) protocols and therefore boost considerably the overall automation of any workflow making use of the SiestaBaseWorkChain. A battery of curated protocols is important for the wider use of the AiiDA-siesta interface in industrial applications.

4.3 Upgrade of the STM workflow

In the previous report, the workflow SiestaSTMWorkChain was introduced. It is the turn-key solution to the problem of producing simulated STM images for a given structure. The workflow performs a SIESTA run (through the SiestaBaseWorkChain) in order to produce a file with the local density of states (LDOS) in an energy window. The LDOS can be seen as a "partial charge density" to which only those wavefunctions with eigenvalues in a given energy interval contribute. In the Tersoff-Hamann approximation, the LDOS can be used as a proxy for the simulation of STM experiments. The workflow automatically processes the LDOS file (through the STMplugin distributed in aiida_siesta) to produce STM images.
The workflow has now been extended to include more features. In particular, the spin options have been added. It is now possible to discriminate the spin contributions to the LDOS and therefore plot STM images for each spin component. Please note that this includes non-collinear and spin-orbit calculations. Moreover, now both “constant-current” and

"constant-height" options are available. The first option plots LDOS iso-surfaces, representing the physical movement of the tip in direction perpendicular to the surface. The second option plots the value of the LDOS at a fixed height above the surface. Both experimental setups are commonly used for STM imaging. A prototypical result of a SiestaSTMWorkChain is presented in Fig. 7, where the spin-dependent simulated STM image for a monolayer of Cr is reported.

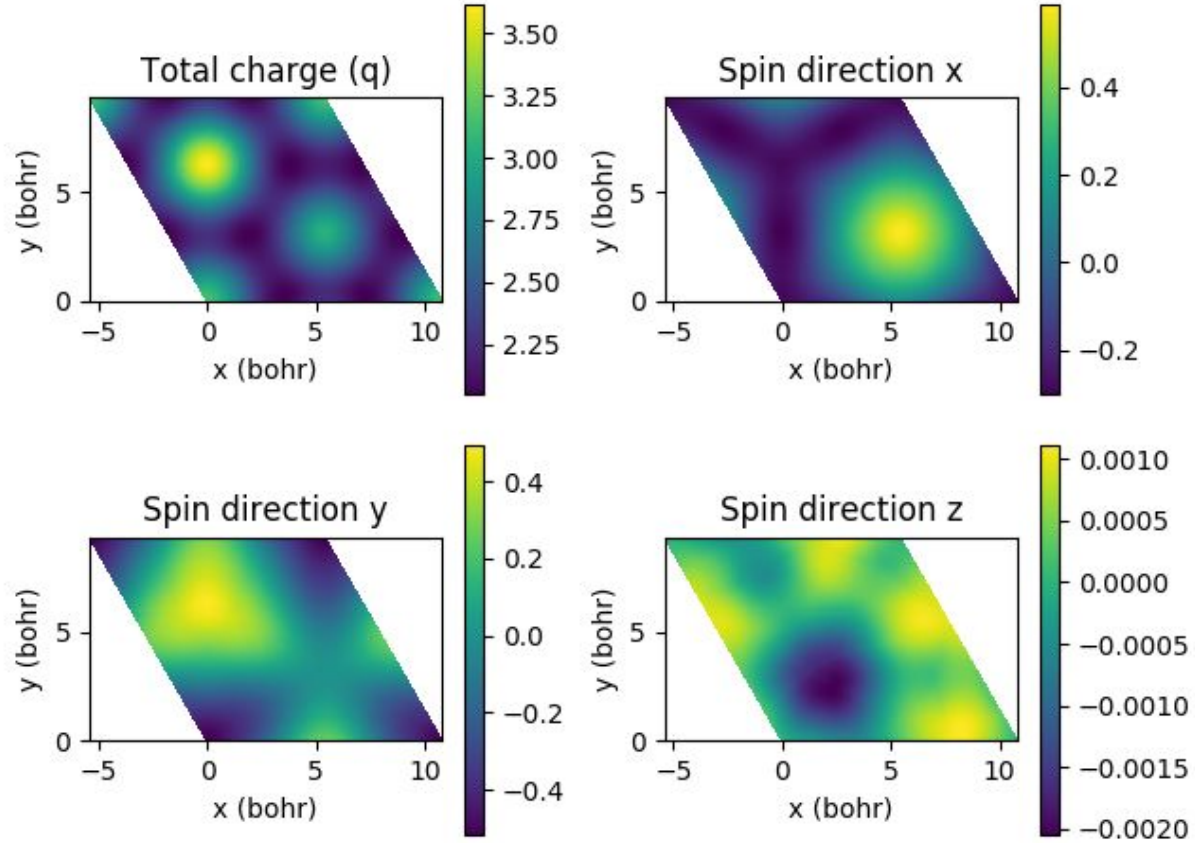

Figure 7: Simulated STM image of a Cr monolayer, resulting from a calculation with spin-orbit coupling. It reports a map of the partial charge density in units of 10^{-3} e/bohr^3 generated by wavefunctions in an energy window just above the Fermi energy and recorded at a height of 3 Ang above the surface. In the Tersoff-Hamann approximation, the partial charge density can be used as a proxy for the simulation of STM experiments. The STM signals associated to spin components are analyzed separately. The standard image (not spin-resolved) is reported in the top left panel.

4.4 GPU benchmarks and computations on pre-exascale machines

We have used the aiida-siesta package to drive benchmarks on HPC machines. In particular, we have focused on the new GPU features of Siesta, based on the interface to the ELPA[11] solver and the ELSI interface. The benchmark system is also of current relevance: a protein of the coronavirus COV-SARS-2, in an aqueous environment (with explicit water molecules). This is a milestone for Siesta and its AiiDA interface, as it is the first real-world, large-system test of the plugin and the robustness of the workflows. Benchmark results can be seen in the D4.3 deliverable report.

5 AiiDA-YAMBO workflows

5.1 Quick Overview on the plugin

The existing set of AiiDA-yambo workflows has seen significant improvements since the upgrade of the AiiDA-core package (version 1.0). The main functionality used in the plugin, the YamboCalculation, computes single-shot G0W0 quasiparticle corrections (through the yambo executable) on top of existing DFT calculations.

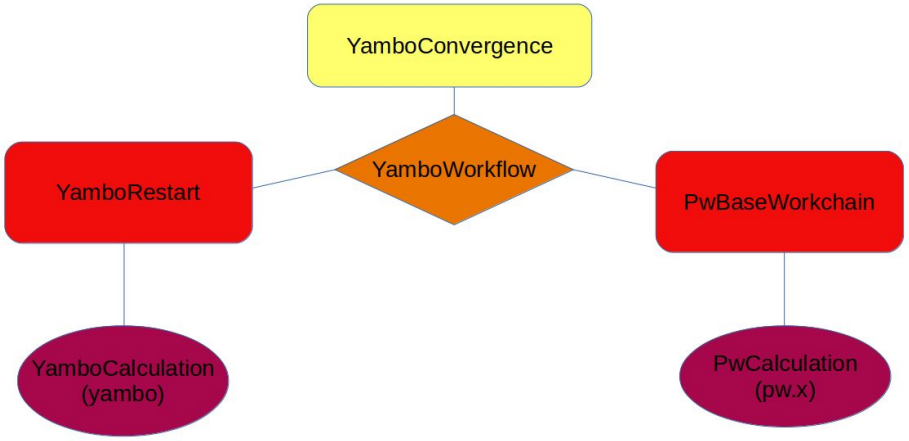

Figure 8: Schematic hierarchy of workchains used within the plugin. The highest level workchain is the YamboConvergence. The PwBaseWorkchain is called whenever the YamboWorkflow has no DFT calculations to be used as a starting point for the YamboRestart.

The main goal achieved up to now is the implementation of an automatic convergence workflow, called YamboConvergence, that can deal with the very complex multi-parameter dependence that characterises a G0W0 convergence calculation. This workflow is currently in use for the study of GW100 dataset[12] of molecules.

[11] P. Kus *et al.*, Parallel Comput. 85, 167 (2019).
[12] M. J. van Setten et al., JCTC 11, (2015).

The lower level workchain is the YamboRestart: it wraps the YamboCalculation and allows the user to run a calculation with automatic error handling and restarts. Such error handling includes walltime exceeding and parallelization/memory problems.
Now the workchain takes advantage of the BaseRestartWorkchain, provided in the AiiDA-core package; significant improvements were done in the error handling, including a better logic for parallelization and memory handling, two of the main sources of failure for a typical yambo calculation. Moreover, we started to include a pre-processing check for some of the inputs, in order not only to correct but also to prevent failures.
The YamboWorkflow lies on the top of YamboRestart and can also perform the needed DFT steps, by using the PwBaseWorkchain provided in the aiida-quantumespresso plugin.
In the future will include support also for Bethe-Salpeter optics and a turn-key solution that allows the user to obtain the fully converged band structure and optical spectra of a given system. Note that for now we can only converge a gap or an energy level, and the subsequent calculation of the bands along a given *k*-path has to be done by hand.

5.2 Insights on the YamboConvergence workflow

The new 'YamboConvergence' workflow revamps and extends the older 'YamboConvergenceWorkflow' and 'YamboFullConvergenceWorkflow'. This merging was done in order to have a single automatic convergence workflow that could perform both single and multi-parameter convergence. For this new workflow, a lot of effort was put in order to have a more flexible procedure to run multiple calculations, focusing on a very generic convergence scheme: setup, run a group of calculations, check the results, stop or do other calculations. Repeat. This allowed us to define different logic to perform convergence analysis by simply enriching the python functions and classes imported by the workflow. We can define this logic as inputs for the YamboConvergence. As an example, we have provided the possibility to choose between different quantities to be studied: gaps or single levels. As a second and more fundamental example, we can provide as input the basic logic of the convergence: automatic search for the convergence of study on a fixed-path given as input.

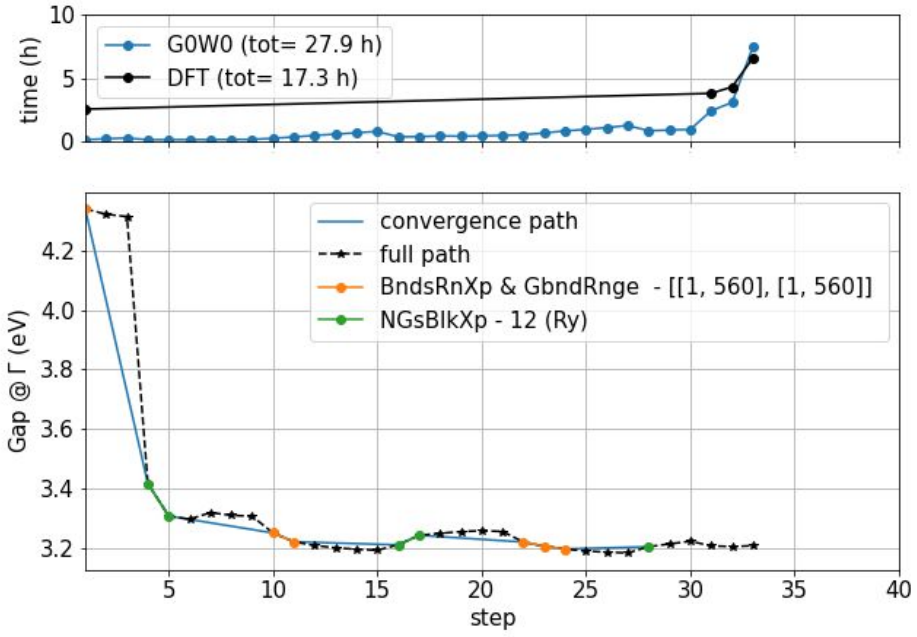

Figure 9: Convergence of the quasiparticle direct gap for bulk TiO2 rutile. The final result (of 3.20 eV) is consistent with previous literature (T. Rangel et al., "Reproducibility in G0W0 Calculations for Solids.", CPC, (2020)). The last three steps represent the convergence with respect to *k*-points mesh and, as over converged, are marked as black stars.

The automatic search requires the specification of the order in which the parameters have to be changed to obtain convergence: to deal with interdependencies, you may need to check multiple times the convergence of each single parameter. For example, in Fig. 9 we show a convergence curve for the direct transport gap of rutile TiO2. We can see that, after having reached convergence on bands (yellow dots) and then on G-vectors(green dots), we have to recompute the convergence on bands and so on. This is the meaning of interdependence. After some cycles, we obtain a flattened curve: the mutual dependence is defeated and the convergence is achieved.

The fixed-path investigation allows the user to compute the G0W0 corrections by varying an arbitrary number of parameters at the same time, providing the parameter space by hand. There is no convergence check here, and it can be used to perform post-processing studies like extrapolations. For example, Fig. 10 shows a typical outcome of the fixed-path philosophy. The HOMO level of the SiH4 molecule is computed against the variation of two parameters, and then the result is extrapolated, obtaining a value of -12.26 eV, consistently with literature results (M. J. van Setten et al., JCTC 11, (2015)).

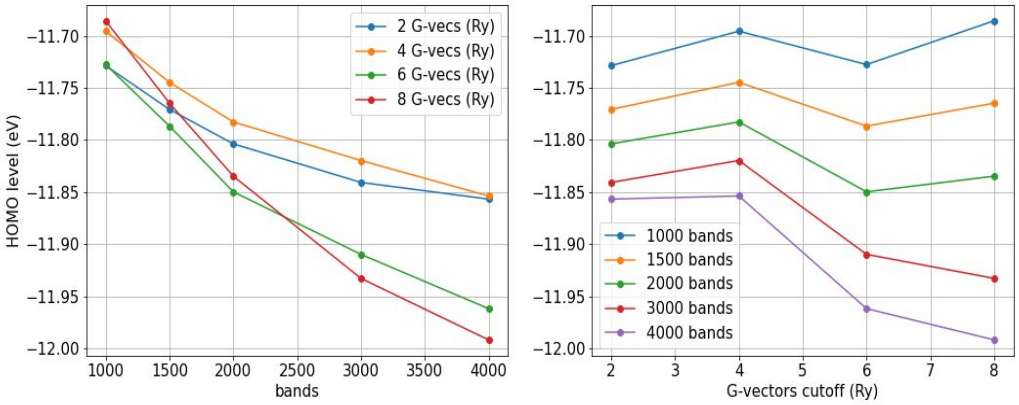

Figure 10: Study of the quasiparticle HOMO level for the silane (SiH4) molecule. Using these results, it is possible to perform an accurate 3D-extrapolation for the value of the Highest Occupied Molecular Orbital.

5.3 GPU-support and speedup for GW convergences

As the Yambo code has been ported on GPU-enabled HPC architectures, as a fundamental prerequisite to run on pre-exascale machines, the automatic convergence workflow can be viewed as a key tool allowing the user to obtain accurate results in a small amount of time. For example, the workflow of Fig. 9, performed 33 G0W0 + 4 DFT calculations in less than a day(the workflow can run multiple calculations in parallel), using for each Quantum ESPRESSO and Yambo simulation respectively 10 and 20 P100 NVIDIA GPUs, installed on the Piz Daint cluster. Note that the major time consuming part is coming from the convergence study on *k*-points (last three points of the curve), for both DFT (when varying k-point mesh, Kohn-Sham data need to be recalculated) and GW parts.
Moreover, we want to stress that the calculations above give a demonstration of using Yambo and QE with GPU support, combined with AiiDA, to run high-throughput style calculations on recent heterogeneous HPC machines (playing the roles of proxies for future pre-exascale architectures).

6 AiiDA-FLEUR workflows

6.1 Overview of workflows in v1.1.0

An overview on the workflows currently available within the AiiDA-FLEUR plugin package (v1.1.0) is shown in Fig. 11. The most basic workflows are the FleurBaseWorkChain and the SCF workflow to handle errors, restarts and the convergence of underlying calculations with the MaX flagship code FLEUR. All other workflows within AiiDA-FLEUR are higher-level

workflows to calculate specific properties and they deploy these two basic workflows as sub-workflows.

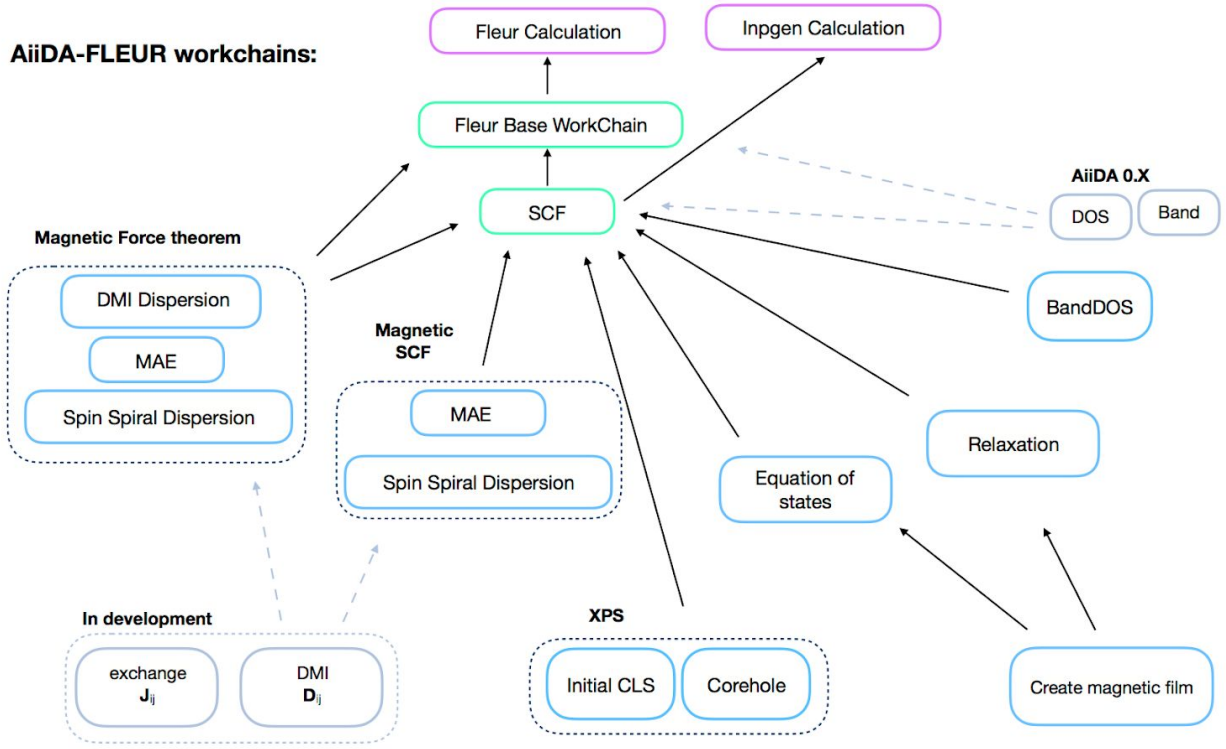

Figure 11: Overview diagram of available workflows in AiiDA-FLEUR (v1.1.0) and sub-workflows they deploy.

6.2 The magnetic workflows in detail

While the FLEUR quantum engine makes progress in exascale simulations of large magnetic structures such as skyrmions and Bloch points, it is essential to come up with tools to point at crystal structures that are worth investigating by an exascale ab-initio calculation. In other words, it would be more clever to make an analytical or a model Hamiltonian calculation for a given set of magnetic structures to get rid of ones giving the least promising target properties. This procedure would save computational resources, shrinking the set of magnetic candidate structures to study. In the majority of cases, analytical and model Hamiltonian models use a common set of magnetic properties, such as Magnetic Anisotropy energy (MAE), Dzyaloshinskii-Moriya Interaction (DMI) and Heisenberg exchange interaction energies. Fortunately, these essential parameters, defining the ground state of a material, can be accurately calculated by FLEUR using primitive unit cells, which usually does not require an extreme amount of computational resources. Indeed, similar calculations have

been performed in the past, however, the previous approach lacks robustness that might be a considerable obstacle to automatisation of subsequent exascale simulations.

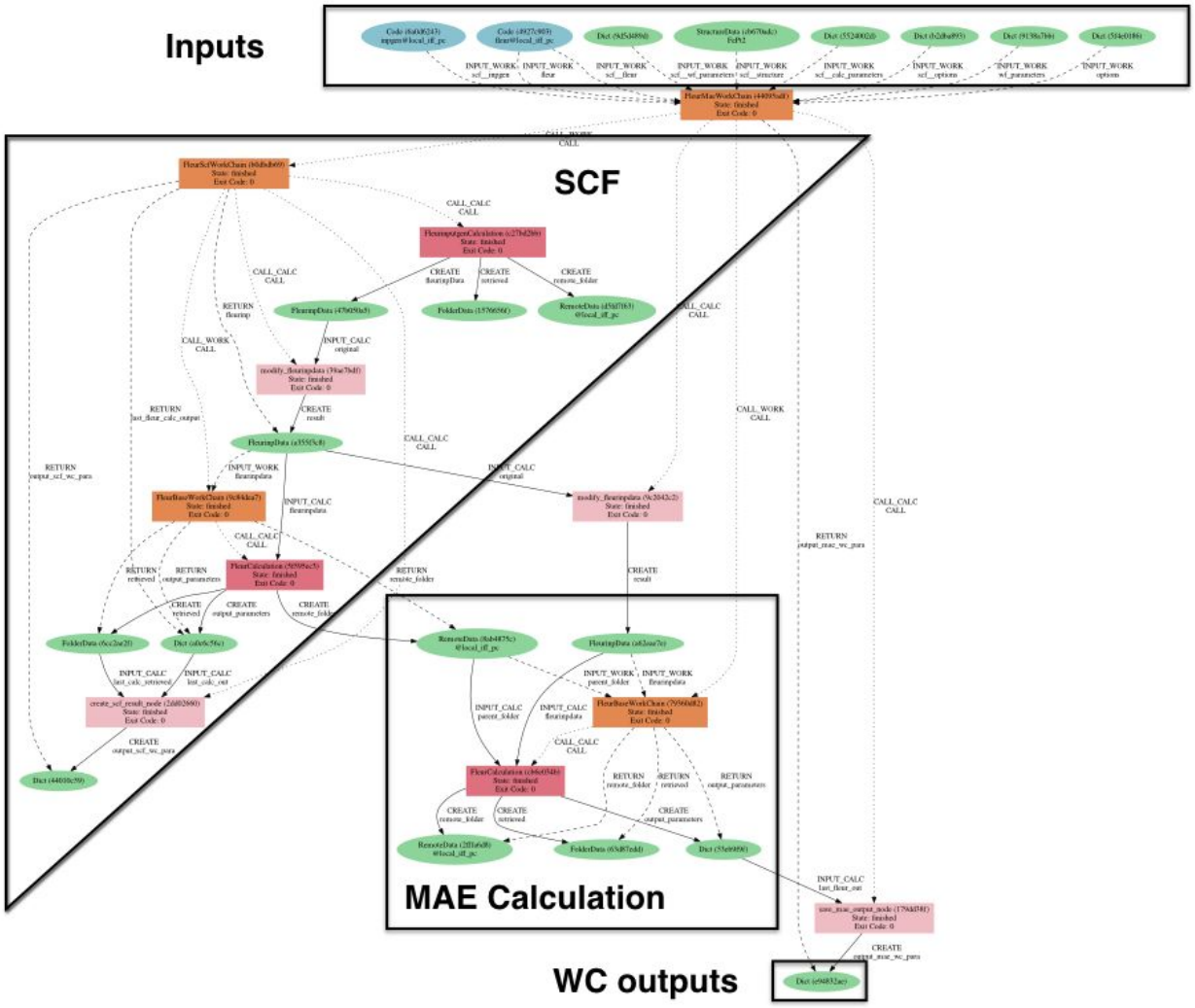

Figure 12: AiiDA provenance graph of a magnetic anisotropy energy calculation, applying the force theorem with the FLEUR flagship code through the FleurMaeWorkChain[13]. Since the recent AiiDA-FLEUR release FLEUR calculations are wrapped in a FleurBaseWorkChain[14] for automatic error handling.

As a part of this work, a set of workflows for computation of magnetic properties was developed. All of them are available in the current version of AiiDA-FLEUR 1.1.0. First of all, the CreateMagneticFilm workflow is responsible for construction and relaxation of a

[13] https://aiida-fleur.readthedocs.io/en/v1.1.0/user_guide/workflows/mae_wc.html
[14] https://aiida-fleur.readthedocs.io/en/v1.1.0/user_guide/workflows/base_wc.html

structure, which represents a substrate with deposited magnetic material. The resulting structure is ready-to-be-used in the other magnetic workflows: MAE, DMI dispersion and Spin Spiral dispersion workflows automate the calculation of corresponding magnetic properties. The AiiDA provenance graph of a MAE workflow is on display in Fig. 12, showing an SCF run and a single FLEUR run with the FleurBaseWorkChain. Finally, two other workflows Jij and Dij, that will be able to submit several DMI and Spin Spiral dispersion workflows, are responsible for calculation of final magnetic parameters, entering aforementioned analytical or model Hamiltonian models. At the moment the Jij and Dij workflows are in development.

6.3 Challenges towards pre-exascale for AiiDA-FLEUR

The pre-exascale capability of AiiDA-FLEUR workflows depends very strongly on the pre-exascale system size capability of the underlying quantum engine, FLEUR. All workflows of AiiDA-FLEUR have high-throughput capabilities that are throughput limited by the capabilities of AiiDA and the underlying computing and data infrastructure. So far the SCF workflow and the Initial CLS workflow have been deployed in a high-throughput way (>1000 different crystal structures per week) with a total failure rate in the latest project of 32% of which 26% was due to non convergence of FLEUR calculation of mostly magnetic systems and 4f systems. A peek view on the core-level shifts results from this study is displayed in Fig. 13.

Very important for successful high-throughput deployment of the workflows across all problem sizes, especially very large (pre-exascale to exascale) problem sizes is an accurate runtime prediction of the FLEUR quantum engine for given crystal structures, given FLAPW parameters and given computing resources. For small system sizes a slightly wrong resource choice is unproblematic, whereas for large system sizes (N) it will become more important because the overall $O(N^3)$ complexity makes it easier to launch calculations with very underestimated resources causing workflows to be unsuccessful. Also important is the robustness and fidelity of the workflows, since for very resource intensive pre-exascale calculation and high-throughput calculations errors are costly.

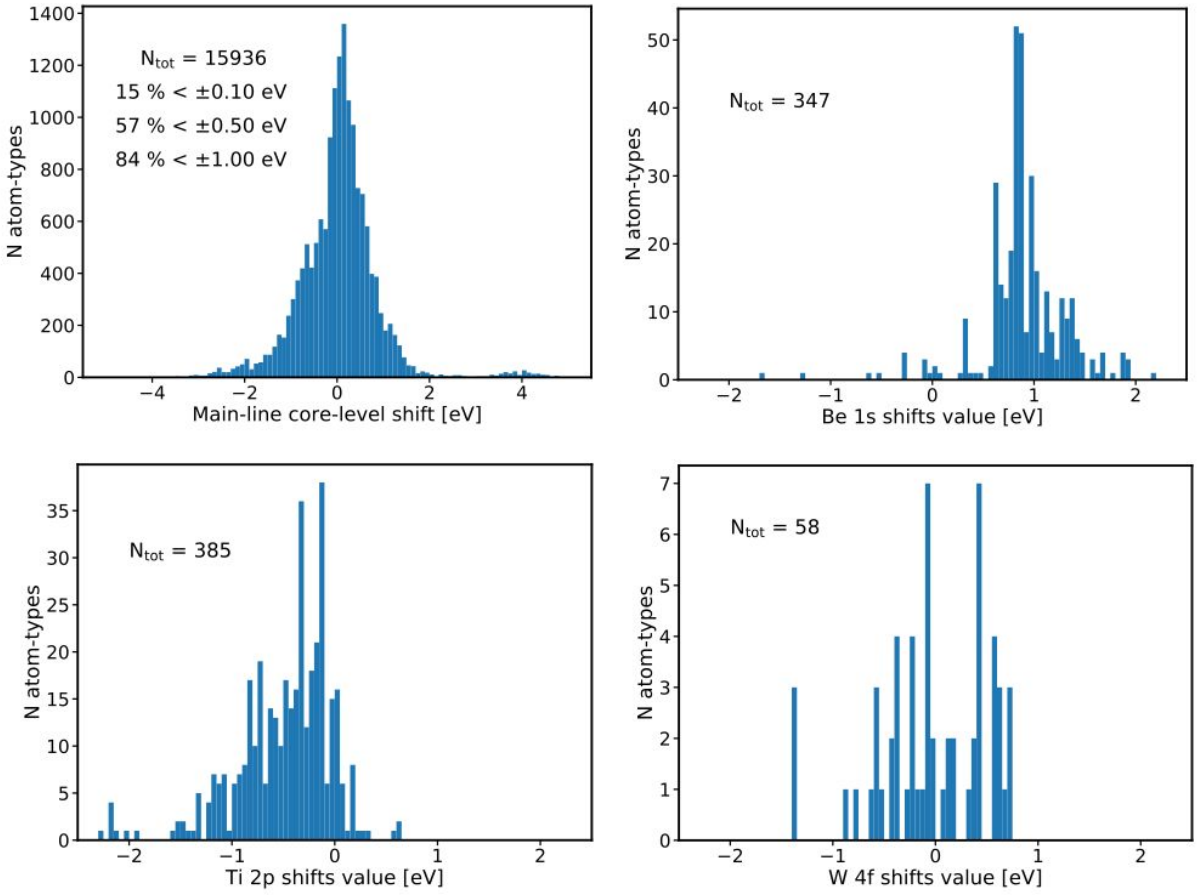

Figure 13: An overview of main line core-level shifts from the chemical environments of a binary metals HTC study with the Initial CLS workflow is shown in these histograms. The first histogram displays all XPS main-line core-level shifts of the converged systems of the structure set, while the other three histograms show distributions of the Be 1*s*, Ti 2*p* and W 4*f* core-level shifts from this structure set. Core-level shifts are essential for the chemical interpretation of X-ray photoemission spectra.

Therefore, it is crucial to absolutely minimize all possible sources of errors. The implementation of higher-level convergence strategies for example would improve the overall fidelity of the self-consistency workflow especially for magnetic and 4*f* systems. We are working to test and improve these features on all fronts within MAX.

Along this line AiiDA-FLEUR version 1.1.0 was released in January 2020 that supports the latest most scaling MAX release of FLEUR. This AiiDA-FLEUR release now only supports python3 and AiiDA-core versions beyond 1.0.0. The main highlight of this latest release is the first version of a workflow bundle to calculate magnetic properties. First continuous integration tests for workflows have been set up, which improved the CI test coverage to over 60% of the AiiDA-FLEUR code base.

7 AiiDA-CP2K workflows

7.1 Quick overview of the plugin

CP2K is a quantum chemistry and solid-state physics package for atomistic simulations that belongs to the MaX codes family. Thanks to the linear scaling DFT implementation CP2K can fairly easily deal with the structures containing thousands of atoms. To empower AiiDA users with full access to the whole spectrum of tools available within the CP2K package the AiiDA-CP2K[15] plugin has been developed by the AiiDA team in a close collaboration with the CP2K developers. Following the philosophy to "enable without getting in the way" the plugin provides access to all CP2K's capabilities through a small set of well tested features. To allow for a quick start we provide a set of ready-to-run examples both for simple calculations and workchains. To ensure the stability of the plugin those examples are executed and results are checked every time new changes are introduced. Currently, the plugin contains Cp2kBaseWorkChain that is meant to be a generic work chain used for a CP2K calculation that can be restarted. Another published work chain that is now a part of AiiDA-LSMO package[16] is Cp2kMultistageWorkChain. It provides a simple interface to run several step simulations with a goal to obtain an optimized structure and capability to escape metastable states.

7.2 Plugin and work chain design principles

While developing CP2K plugin and the workchains we adhere to the following basic principles for the separation of concerns:

- (level 1) The plugin must be as simple and as general as possible.
- (level 2) Cp2kBaseWorkChain with a list handlers must be the interface to run a specific type of calculations that can be restarted.
- (level 3) Combination of level 2 workchains (not necessarily of the same code) forms a work chain devoted to compute a specific property.

According to this approach we envision input plugin and output parser as a direct transformation of AiiDA objects to input files and output files to AiiDA objects. The job of Cp2kBaseWorkChain is to provide a set of handlers for restarting the calculations. Since the handlers can be switched on and off, before running a calculation user can configure only the ones that are needed. The level 3 work chains combine restart work chains of level 2 forming a set of calculations needed to compute a given property. The nested structure of different level work chains is shown in Fig. 14.

[15] https://github.com/aiidateam/aiida-cp2k
[16] https://github.com/lsmo-epfl/aiida-lsmo

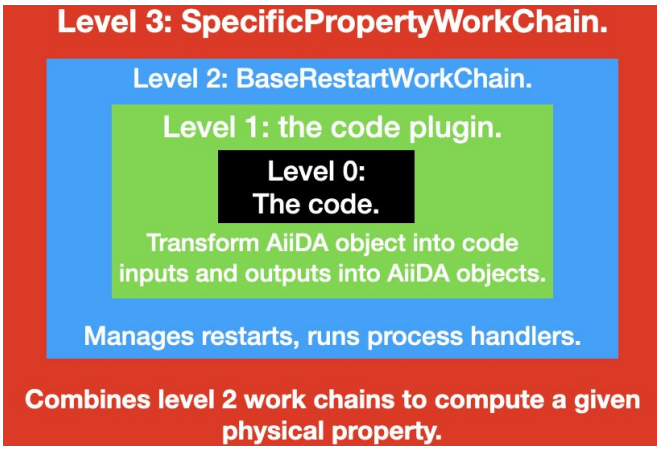

Figure 14: The nested structure of plugin and different level work chains. Code plugin interacts directly with the code and transforms AiiDA objects to the input files and output files to AiiDA objects. BaseRestartWorkChain interacts directly with the plugin and has the capabilities to restart calculations. SpecificPropertyWorkChain combines BaseRestartWorkChains to compute a given property.

In this frame Cp2kBaseWorkChain provided with the plugin belongs to the level 2, while Cp2kMultistageWorkChain belongs to the level 3 and aims to provide the optimized structure at the end of its execution.

7.3 Cp2kBaseWorkChain

The Cp2kBaseWorkChain is meant to be used for the calculations that can be restarted. The work chain is based on BaseRestartWorkChain of AiiDA and currently includes a handler to detect an unconverged geometry and to re-submit the calculation if possible.

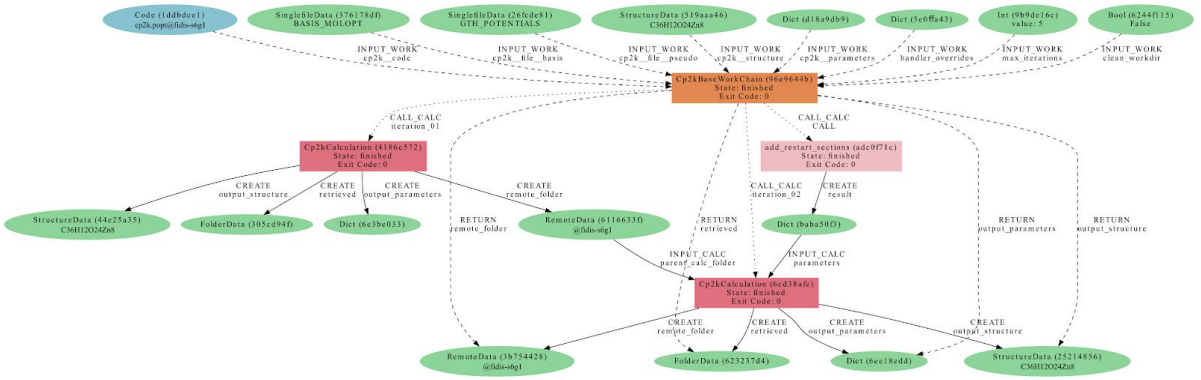

Figure 15: AiiDA graph of a two-step optimization for a metal organic framework. As the optimization could not be completed within one run, the Cp2kBaseWorkChain restarts it making sure to add the necessary restart information to the input dictionary of the second calculation.

If the handler identifies that the geometry is not converged, it tries to identify whether the calculation can be recovered at all. For instance, it can happen that the previous calculation did not do a single geometry step, so there won't be any possibility to recover from that calculation and the workchain must stop. If the calculation is recoverable the handler will modify the input dictionary adding a restart section and will instruct Cp2kBaseWorkChain to resubmit the calculation with a new input.
Since the handlers can be switched on and off, we will keep adding new ones to the workchain to expand its applicability beyond geometry optimization calculations.

7.4 Cp2kMultistageWorkChain

The Cp2kMultistageWorkChain belongs to the level 3 in the schema shown in Fig. 14 as it calls Cp2kBaseWorkChain and is devoted to obtaining an optimized structure of porous materials (such as covalent organic frameworks, metal organic frameworks, and zeolites). The work chain is a part of AiiDA-LSMO package. Given a YAML-formatted protocol with different settings, the work chain iterates them until the SCF calculation gets converged. The protocol can also contain a number of stages, i.e., different MOTION settings, that are executed one after another, restarting from the previous calculation. During the first stage, stage_0, different settings are tested until the SCF converges at the last step of stage_0. If this doesn't happen the work chain stops. Otherwise it continues running stage_1, and all the other stages that are included in the protocol. These stages can be used for running a robust cell optimization, i.e., combining first some MD steps to escape metastable geometries and later the final optimization, or ab-initio MD, first equilibrating the system with a shorter time constant for the thermostat, and then collecting statistics in the second stage.

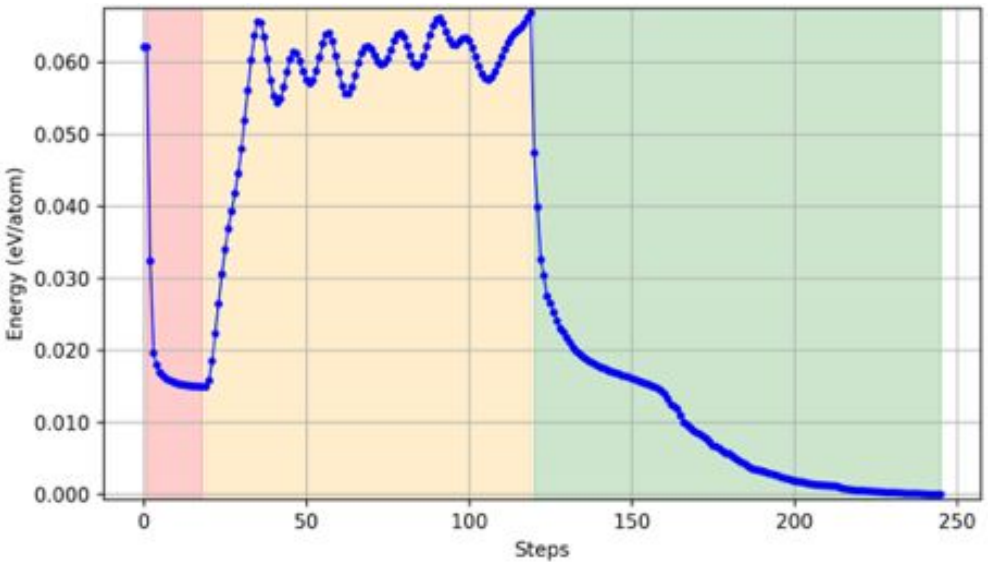

Figure 16: An energy profile obtained with a 3-step Cp2kMultiStageWorkChain for a covalent organic framework. The three steps are: fixed angle cell optimisation (in red), molecular dynamics (in yellow) and final cell optimisation (in green).

Some default protocols are provided in together with the aiida-lsmo package and can be imported with simple tags such as test, default, robust_conv. Otherwise, a user can take an inspiration from these protocols to write his own protocol and pass it to the work chain.
The Cp2kMultistageWorkChain work chain mentioned above was used in a middle-throughput study[17] where we simulated the performance of covalent organic frameworks for carbon capture. Thanks to the fully automated AiiDA infrastructure, even after the paper has been published we still keep adding new material, as it takes almost no human time to do so[18].

8 AiiDA-BigDFT workflows

8.1 Overview

BigDFT's AiiDA plugin is being developed in the context of the second MAX project, and released as a preliminary version 0.1.0a0 in December. This first preliminary version enabled simple AiiDA computation with BigDFT in AiiDA workflows, as well as insertion of AiiDA calculators in existing BigDFT workflows and notebooks. BigDFT indeed provides (with the PyBigDFT library[19]) many tools to prepare, run, and apply post-processing tasks to computations.

[17] Ongari et al, ACS Cent. Sci. 2019. (324 materials were published initially)
[18] https://archive.materialscloud.org/2019.0034 (505 materials available to date)
[19] https://bigdft-suite.readthedocs.io/projects/PyBigDFT

The second version of the plugin, 0.2.0 aims at allowing integration of existing PyBigDFT's operations in AiiDA workflows. This work takes place in the context of the development of common workflows for AiiDA codes, based on a single API, which is described later in this document. The plugin provides a basic workchain to wrap the actual computation of BigDFT and the gathering of necessary outputs. Each workflow will prepare the input parameters for these computations, call the workchain, and on completion, apply the post-processing needed through PyBigDFT calls, wrapped in AiiDA work functions.
The low level workchain is an instance of the BaseRestartWorkChain AiiDA concept, enabling simple error handling for common issues, such as an exceeded walltime, or errors in inputs, to be performed, and job restarting when possible.
An example of such a simple workflow would be the relaxation one, which for BigDFT only performs a single computation, while calling the PyBigDFT's optimize_geometry helper function from the InputActions module to setup the relaxation method and maximum number of steps for the computation. The logic of the relaxation itself is performed inside the BigDFT execution in this case, not in the workchain. After completion of the computation, the log files for the run, which in this case are several YAML dictionaries inside a single file, are gathered and parsed by the plugin, to present the user with meaningful results for the computation.

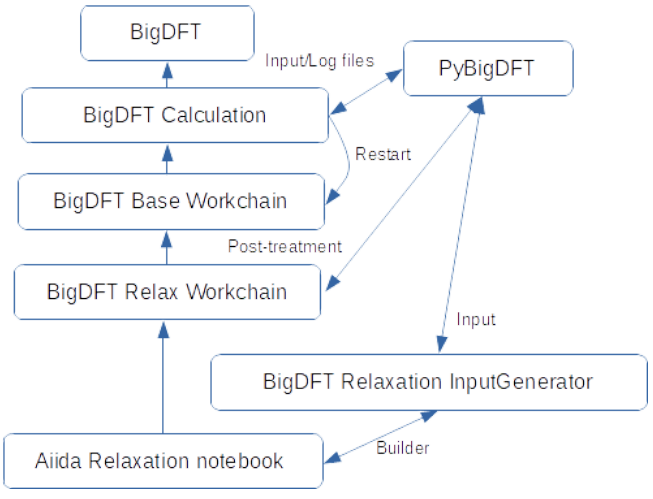

Figure 17: Example of a BigDFT relaxation computation structure, using the common interface.

The results can either be a PyBigDFT LogFile object encapsulating all individual sub-LogFile objects for further post-treatment tasks, a set of python dictionaries representing the output files for direct handling by the user, or in case of a call through the common API the resulting forces and final structure.

Each of the simple workflows provided by the BigDFT plugin will provide wrapper input generators following the common API, for instance a BigDFTRelaxationInputsGenerator tool with a get_builder method to provide the user the basic input sets to run a relaxation computation without knowing internals of BigDFT workflows, and allowing integration in more generic workflows.

8.2 Challenges and Future

In order to provide more parallelism, the logic of some computation performed inside the BigDFT executable will have to be performed in the workflows instead, allowing submission of several parallel jobs in some cases, while BigDFT was resorting on using only one large computation task. The good parallel efficiency of the code provided good results with this method, but in the future submitting many smaller tasks in parallel instead of a large one will often be necessary. This is already performed in a way with PyBigDFT's Datasets, which group several related computations and are able to submit them concurrently through AiiDA. Turning such workflows into proper AiiDA workflows would be possible, and is currently being evaluated.

9 AiiDA common workflows

Each MAX code presents some uniqueness in the capabilities and the tasks it can perform; however there are some basic materials properties that all the codes can calculate. For instance, the calculation of the electronic bands and the relaxation of a crystal structure can be obtained with all the MAX codes (except the relaxation with YAMBO). For these tasks, it makes sense to think about a common interface that facilitates the submission of the process that performs the task. In this section, we report about the first effort undertaken towards the creation of common workflow interfaces across different codes and plugins.
Two are the main objectives we want to achieve:

- Creating an interface to compute basic materials property that is common for all the MAX codes. This means, in other words, to standardize the access to automated "turn-key" workflows able to calculate specific materials properties.
- Provide a common background that will enable to run these turn-key solutions via simple GUIs, to the benefit of industry partners, experimental groups and for education purposes.

Both the above-listed objectives are key deliverables of the WP5.
The realization of the project requires, for a specific material property, the following three points:

- The standardization of the input options required to complete the task and of the output results. The actual implementation for the calculation of the property remains, instead, code specific.
- The creation of functions that return information about the allowed inputs and specifications for the task under consideration. These functions are used to automatically generate GUIs.
- The implementation of a function that translates the standard inputs to code-specific inputs and produces a "ready-to-submit" process-builder (typically for an AiiDA WorkChain) that can perform the task. The AiiDA workflow implementation is code-specific, but the outputs to be returned must follow agreed conventions, as stressed in the first point of this list.

The development of this project is hosted in the "aiida-common-workflows" GitHub repository[20] and, at the moment, it is at a very preliminary stage. It is the first step in preparation to the M36 deliverable D5.4 that includes a part on "code-agnostic workflows". So far, only a first test-case implementation has been produced for the specific task of the relaxation of a structure. This test-case is described in detail in the next subsection, with the scope to clarify the purpose and implementation of the common workflows idea.

9.1 The test case of the relaxation of a crystal structure

The relaxation of a structure in order to find its equilibrium configuration is one of the most common tasks in material science studies. It is often a required first step before calculating many other physical and chemical quantities and all the MaX codes (except Yambo) can perform it.

During the AiiDA Hackathon, hold in Cineca (Casalecchio di Reno, BO, Italy) in February 2020, we agreed on the relevant inputs for the relaxation task, they are:

- structure: the input structure to be relaxed
- relaxation_type: the specification of the type of relaxation to perform, for instance the request of relaxing only the atoms or also the simulation cell
- protocol: a string that summarizes the choice of the accuracy of the DFT calculation. The available options are code specific, but an idea could be to have a "accurate" and "test" protocol option
- threshold_forces: value of the atomic forces below which the relaxation is considered converged (in units of ev/ang)
- threshold_stress: value of the stress below which the relaxation is considered converged (in units of ev/ang^3)

[20] https://github.com/aiidateam/aiida-common-workflows

The available options for each input above can change code by code, only the schema must be the same. For instance, many codes allow several types of relaxation, while the FLEUR code only supports relaxation of atomic coordinates, but with fixed simulation cells. The important point is that all the codes implement a python function that returns the list of available relaxation types when interrogated. We call this function get_relaxation_types and it will be used to automatically generate a GUI. The same concept applies to the protocols; each code implements very code-specific protocols and the function get_protocol_names returns the available options.

A further important function is get_builder, that translates the selected options for the inputs listed above into inputs for a code-specific process. It returns an AiiDA builder for an AiiDA process (a WorkChain or CalcJob) that is ready to be submitted as it contains all the necessary inputs to complete the relaxation tasks. For instance, in the Siesta implementation, "get_builder" returns a builder for the AiiDA workchain SiestaBaseWorkChain, with the correct inputs to perform a robust relaxation of the structure.

All the functions mentioned above are methods of the classes <Code>RelaxationInputsGenerator, one for each MAX code. The idea is to have a collection of <Code><Task>InputsGenerator classes that constitute the base for the creation of GUIs where the selection of a code and few options allows a user to perform basic material science analysis in a straightforward way.

During the Hackathon we also reached an agreement on the standard outputs for the relaxation tasks. They are the final relaxed structure, the final forces on atoms and the final total energy. Depending on the relaxation options, also the final stress tensor is returned.