ARCHER2-eCSE03-10 Technical Report

Realising a modular data interface to couple quantum mechanical calculators with external data-driven workflows

PI: Dr Andrew Logsdail, Cardiff University

Co-Is: Dr Reinhard Maurer (University of Warwick), Prof. Volker Blum (Duke University), Dr Mariana Rossi (Max Planck Institute for the Structure and Dynamics of Matter), Dr Ben Hourahine (University of Strathclyde), Prof. Scott Woodley (University College London), Dr Thomas Keal (STFC Daresbury)

Technical Staff: Dr Pavel V. Stishenko (Cardiff University)

Abstract

The Atomic Simulation Interface (ASI) is a native C-style API that includes functions for the export and import of data structures that are used in electronic structure calculations and for classical molecular dynamics simulations. The ASI aims to be a uniform, generic and efficient interface for connecting various computational chemistry and materials science codes in multiscale simulation workflows, such as QM/MM, QM/ML, QM/QM. The ASI specifies functions, data types and calling conventions for export and import of density matrices, overlap and Hamiltonian matrices, electrostatic potential, atomic coordinates, charges, total energy and forces. We have implemented the ASI in FHI-aims and DFTB+ codes as demonstrations of capabilities. An ASI Python wrapper and ASE-complying (Atomic Simulation Environment) calculator have been written to simplify integrations with other Python codes. Proof-of-concept codes have been written for electrostatic potential embedding, density matrix prediction, and Hamiltonian matrix export.

Motivation

Exascale computing power enables the computational chemistry and materials science communities to conduct high-throughput computational screening of materials using models that explicitly consider electronic structure of simulated matter [1-3]. Complexity and diversity of electronic structure calculations, and subsequent analysis algorithms, have caused development of numerous computer codes tailored to different use cases [4]. As a consequence, the best algorithms for all problems are not available in all software packages, and computational scientists aiming to conduct research with the best algorithm implementations must put significant effort into organising data exchange between different programs [5]. In the computational chemistry and materials science domain, example cases include the growing adoption of hierarchical (e.g. QM/MM) approaches, with polarizable force-fields [6] that demand generic mechanisms to exchange data about electrostatic fields between software packages, as well as the recent promise offered by employing machine-learning (ML) methods at the electronic structure level [7-8]. Therefore, a generic and efficient way to extract electronic structure data, and derived properties, from electronic structure packages will allow for application in hierarchical modelling and training of ML models, as well as to employ ML predictors in electronic structure calculations, which will be of utility for large communities.

Implementation

General design considerations

Taking into account the potential size of data objects, as well as the large-scale application on HPC infrastructure, we decided to design the ASI in a way that allows minimal compute overhead with regards to interface implementation. Therefore, the ASI API is specified as a set of pure C functions with requirements on calling sequence; and no non-standard libraries are required to implement ASI API or to use codes with it. Although C-style API lacks expressiveness and means of compile-time error checking, it is undoubtedly the most portable interface that is crucial for interoperability and wide adoption, which is one of our overarching ambitions.

Support of distributed calculations

For distributed applications, we have designed the ASI API with ScaLAPACK as the suggested approach. In this regard, our functions that manipulate large matrices tend to pass data pointers to BLACS descriptors, along with pointers to matrices, rather than any complete data objects; these BLACS descriptors are optional, allowing codes with ASI API to also be built without ScaLAPACK if more appropriate (i.e, intermediate computing facility, and smaller).

Furthermore, electronic structure packages commonly parallelize calculations by discrete variables, such as k-points and spin channels; calculations for different values of k-points and spin channels are relatively independent, so such parallelization is efficient and easy to implement, providing excellent compute benefits. In our design and realisation, the ASI API provides special functions that return the list of local k-points and spin channel indices. The implementation of these functions allows client code receiving data to adapt to an already existing distributed data object.

Callbacks

To avoid unnecessary data copying, the ASI API functions that are designed for exporting of data provide pointers to internal data structures, letting the caller decide if copying is needed. It is a common practice in high performance computing to reuse data arrays for storing computation results; for example, LAPACK has many functions that place output in place of input arrays. Therefore, it is hard or impossible to guarantee data existence after full completion of computation, because intermediate data arrays that could be of interest for external codes can be overwritten during subsequent calculations. In this regard, it is necessary for our API to provide a means to pause main code execution, and thus allow for interesting data to be processed by external code (or transferred out). In the ASI approach, we decided to use the callback approach for this purpose. Callbacks are simple to introduce in existing codes, which is crucial choice for the ASI API to be widely adopted.

Key ASI API functions

The most important functions implemented in the new ASI API are shown in Table 1, presenting a general picture of the API capabilities. For complete ASI API specification, please see the ASI API Specification in the project’s website (https://gitlab.com/pvst/asi).

Purpose: Control flow	
ASI_init()	Initialize calculations
ASI_run()	Do single point calculation. Can be called multiple times
ASI_finalize()	Finalize calculations, free resources
Purpose: Classical data object transfer (specific functions with limited dimensionality)	
ASI_set_atom_coords()	Set atomic coordinates
ASI_energy()	Get total energy
ASI_forces()	Get forces acting on atoms
ASI_atomic_charges()	Get atomic charges
Electrostatic potential management (higher dimensionality objects)	
ASI_calc_esp()	Get electrostatic potential and its gradient in arbitrary points

ASI_set_external_potential()
ASI_register_external_potential()
	Set electrostatic potential and its gradient in arbitrary points

Electronic structure matrix management (higher dimensionality objects)	
ASI_register_dm_init_callback()	Initialize SCF loop via density matrix
ASI_register_dm_callback()	Get density matrix on each SCF iteration
ASI_register_overlap_callback()	Get overlap matrix
ASI_register_hamiltonian_callback()	Get Hamiltonian matrix

Table 1. Key functions of the ASI API. For full list see ASI API Specification: https://gitlab.com/pvst/asi.

Peculiarities of implementations

In realising the application interfaces for the ASI API into existing packages, and thus demonstrating functionality, we discovered that uniform implementations of the ASI API where not fully possible in DFTB+ and FHI-aims. In particular, intrinsic differences of algorithms in these codes, and willingness to minimize existing code changes, led to a few implementation-specific peculiarities that are documented here:

The **DFTB+** software package was an ideal environment for test implementation, as it has a maintained C-style API with functions similar to ASI API functionality. Therefore, implementation has meant we have extended the already existing DFTB+ API to include missing functionality that supports export of large matrices. In this regard, the ASI API implementation is a separate wrapping library that essentially forwards ASI API calls to DFTB+ API.

The **FHI-aims** software package is quite contrasting, having only rudimentary support of functionality in the library form (but significant application potential). The ASI API was natively implemented in FHI-aims and, moreover, initialization code within the software package was refactored to make it more modular and extendable, i.e. delivering benefit of long-term sustainability to the FHI-aims community.

As a further nuance of the algorithmic differences between DFTB+ and FHI-aims, the ASI_register_dm_init_callback() is not implemented for DFTB+ as density matrix initialization of an SCF loop is meaningless for DFTB calculations, whereas it is highly beneficial in FHI-aims (as demonstrated below).

Interoperability with Python

To provide interoperability with the plethora of Python libraries available to computational chemistry and physics communities, two modules have been developed that allow interfacing with the ASI API. The primary pyasi module provides direct access to ASI API functions wrapping C arrays into NumPy counterparts. In addition, another module asecalc provides an ASI_ASE_calculator class that implements an instance of the ASE (Atomic Simulation Environment) Calculator interface, allowing subsequent use in a wide-range of computational chemistry packages. Both are distributed via the project repository.

Use cases

The ASI API is envisaged to have a wide range of applications, and here we demonstrate potential in three use cases that have been considered for testing purposes: (i) electrostatic potential embedding; (ii) density matrix initialization; (iii) and export of matrices related with DFT calculation.

Electrostatic potential embedding

Electrostatic embedding is an established QM/MM technique of approximation of interactions between subsystems. The ASI API supports the transfer for electrostatic information by providing functions that allow export and import of both the electrostatic potential and gradient (i.e. field) across arbitrary points. In the project repository there are test cases (tests/testcases/test_esp_sc.py.aims and tests/testcases/test_esp_sc.py.aims) for self-consistent calculation of electrostatic interactions between two water molecules. Note, that both FHI-aims and DFTB+ tests use the same Python code (tests/python/test_sc.py), which we believe helps to underline the generality of the ASI API.

Density matrix initialising

The efficiency that an electronic structure calculation can achieve is strictly limited by the matrix diagonalisation operations; however, if the quantity of these operations can be reduced, then significant compute gains can be made. Thus, the quality of the initial guess for the electronic structure orbital coefficients (stored in the density matrix) can reduce the quantity of self-consistent field (SCF) iterations and computation time, as well as reducing the number of non-converging simulations [9]. The ASI API provides a simple and efficient way to feed predicted density matrix in to the SCF loop, and this paves the way to exploration of various algorithms for density matrix prediction. As an exemplar case, and considering that development of such density matrix predictions was out of scope of this project, we are encouraged by preliminary experiments with the SchNOrb library [10] that demonstrate a reduction in the number of SCF steps by more than 70% for uracil and malondialdehyde molecules.
The exemplar python script in the repository uses precomputed ground-state density matrix as a mock-up of a ML-based density-matrix predictor (tests/python/test_dm_init.py).

Export of Hamiltonian, overlap, and density matrices

Our final example use case is the export of larger matrices related with electronic structure calculations, which is common for further integration with external software packages and post-processing. Example usage of these matrices, which can be highly memory consuming if written to file, are machine learning applications or electron transport calculations. The exemplar python script is in the repository (tests/python/test_expdmhs.py) and proves facile realisation.

Conclusions and future work

Unification of access to internal data structures of DFT codes is a challenge due to the monolithic nature of existing software packages, developed through significant coding endeavour; however, this modularisation and accessibility is necessary for employing machine learning advancements and for future development of multiscale simulation methods on new HPC infrastructure. Here, we have realised such a modular approach and demonstrated this with interfacing to two popular electronic structure packages that are commonly used on ARCHER2 and other HPC facilities. A benefit of implementing our universal API is the possibility to separate some auxiliary functionality from computational codebase in these packages, with long-term benefit for these communities; for example, charge partitioning, data formatting and parsing, electron transport calculations can be implemented as separate libraries relying on ASI API. We believe that the work realised here provides the foundation for an exciting new opportunity in accelerated atomistic modelling.

This work was funded under the embedded CSE programme of the ARCHER2 UK National Supercomputing Service (http://www.archer2.ac.uk).

References

(1) Torelli D, Moustafa H, Jacobsen KW, Olsen T. High-throughput computational screening for two-dimensional magnetic materials based on experimental databases of three-dimensional compounds. npj Comput Mater. 2020 Dec;6(1):158.

(2) Sarikurt S, Kocabaş T, Sevik C. High-throughput computational screening of 2D materials for thermoelectrics. J Mater Chem A. 2020;8(37):19674-83.

(3) Luo S, Li T, Wang X, Faizan M, Zhang L. High-throughput computational materials screening and discovery of optoelectronic semiconductors. WIREs Comput Mol Sci. 2021 Jan;11(1):e1489.

(4) Sherrill CD, Manolopoulos DE, Martínez TJ, Michaelides A. Electronic structure software. J Chem Phys. 2020 Aug 21;153(7):070401.

(5) Borini S, Monari A, Rossi E, Tajti A, Angeli C, Bendazzoli GL, et al. FORTRAN Interface for Code Interoperability in Quantum Chemistry: The Q5Cost Library. J Chem Inf Model. 2007 May 1;47(3):1271-7.

(6) Bondanza M, Nottoli M, Cupellini L, Lipparini F, Mennucci B. Polarizable embedding QM/MM: the future gold standard for complex (bio)systems?. Phys Chem Chem Phys. 2020;22(26):14433-48.

(7) Westermayr J, Gastegger M, Schütt KT, Maurer RJ. Perspective on integrating machine learning into computational chemistry and materials science. J Chem Phys. 2021 Jun 21;154(23):230903.

(8) Chandrasekaran A, Kamal D, Batra R, Kim C, Chen L, Ramprasad R. Solving the electronic structure problem with machine learning. npj Comput Mater. 2019 Dec;5(1):22.

(9) Lehtola S. Assessment of Initial Guesses for Self-Consistent Field Calculations. Superposition of Atomic Potentials: Simple yet Efficient. J Chem Theory Comput. 2019 Mar 12;15(3):1593-604.

(10) Schütt KT, Gastegger M, Tkatchenko A, Müller K, Maurer RJ. Unifying machine learning and quantum chemistry with a deep neural network for molecular wavefunctions. Nat Commun. 2019 Dec;10(1):5024.