strucscan: A lightweight Python-based framework for high-throughput material simulation

Isabel Pietka[1], **Ralf Drautz**[1], **and Thomas Hammerschmidt**[1]

1 Interdisciplinary Centre for Advanced Materials Simulation (ICAMS), Ruhr University Bochum, Bochum, Germany

DOI: 10.21105/joss.04719

Software

- Review
- Repository
- Archive

Editor: Antonia Mey

Reviewers:

- @mturiansky
- @wcwitt

Submitted: 27 June 2022
Published: 23 November 2022

Summary

The development of new materials by computational materials science relies to a large degree on the prediction of material properties by simulation at different time and length scales. A common challenge at the atomic scale is the need for large numbers of calculations in order to sample, e.g., different chemical compositions, different crystal structures or different simulation settings. Typical examples of the required high-throughput calculations are (i) the sampling of the combinatorial space of structure and composition for determining the most stable structure of a mixture of chemical elements, (ii) the generation of a data set for constructing an interatomic interaction model or (iii) the generation of a data set for inferring properties by machine learning. Depending on the problem at hand, the number of required calculations may range from hundreds to millions.

The Python-based framework strucscan provides a robust solution to handle such high-throughput calculations in an efficient way on compute clusters with a queueing system or on the local host. The simple and transparent workflow of strucscan loops over a specified list of crystal structures and chemical compositions and computes a specified list of properties for each combination. The property calculations are represented as a pipeline of successive, interdependent steps which can easily be adapted and extended. The data is stored in a human-readable data tree with flat hierarchy. Strucscan performs a series of scalable and easily extendable pre-processing and post-processing steps and compiles the results in Python dictionaries for further evaluation. Data provenance for research-data management and analytics is realized in terms of the data-tree structure that includes all input files.

The present version of strucscan is tailored to the calculation of frequently needed material properties with widely used atomistic simulation codes on common scheduling systems. The implemented interfaces particularly support the VASP software package (Kresse & Furthmüller, 1996a), (Kresse & Furthmüller, 1996b), (Kresse & Joubert, 1999) for density-functional theory (DFT) calculations on SunGridEngine (Gentzsch, 2001) and slurm (Jette et al., 2002) scheduler systems. With the well-defined and documented interfaces, strucscan can be extended with basic programming skills to further scheduling systems, to further simulation codes and material properties at the atomic scale or to other simulation scales.

Statement of need

The need for high-throughput calculations in computational materials science lead to the development of several workflow managers and high-throughput frameworks ((Ong et al., 2013), (Hammerschmidt et al., 2013), (Mathew et al., 2017), (Janssen et al., 2019), (Huber et al., 2020), (Gjerding et al., 2021)). These software packages offer numerous features but often require a rather complex infrastructure with, e.g., external workflow managers (Jain et al., 2015) for interaction with compute clusters or SQL databases for storing results. Moreover, it is often not straightforward to extend these large software packages and to tailor them for particular needs. In many practical cases, the repetitive execution of the tasks does not benefit from a large toolbox of features or from a predefined database concept but rather needs a concise and transparent driver that can be customized to the particular high-throughput task and the specific data management solution. Strucscan is a lightweight driver with focus on atomistic simulations and offers the following features:

- Transparency: lean and lightweight Python code with transparent and robust handling of tasks and infrastructure
- Dependencies: no external workflow managers or database systems required, only NumPy and ASE (Larsen et al., 2017)
- Customization: straight-forward extension to further tasks and interfaces (simulation codes, schedulers) with only low-level programming experience
- Pipelining: simple and transparent realization of task sequences and task dependencies
- Restarts: seamless restart capabilities due to coherent interlinking of workflow organisation and data tree
- Post-processing: customizable post-processing within workflow with results stored in Python dictionaries for further post-processing
- Data provenance: human-readable data tree with flat hierarchy and storage of all input files for metadata generation

Strucscan

The strucscan framework is based on Python 3.6+ and requires the Atomic Simulation Environment (Larsen et al., 2017) and NumPy. It is available from a git repository and can be installed from there or with pip. Detailed documentation and usage examples of strucscan are available on readthedocs. High-throughput calculations with strucscan can be started from the command line, from a Python shell or from a Juypter notebook as shown in the examples in the strucscan git repository. The present version is focused on high-throughput DFT calculations on computer clusters with SunGridEngine (Gentzsch, 2001) and slurm (Jette et al., 2002) scheduler systems. Future extension to further simulation codes are planned.

The basic workflow of strucscan is visualized in Figure 1 and includes the following steps:

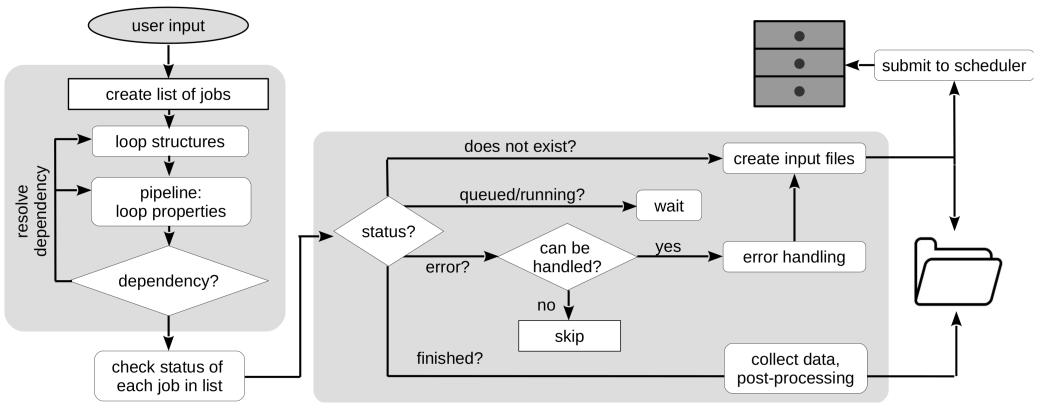

Figure 1: Workflow of the strucscan framework: the process starts with creating a list of jobs by looping over structures and pipelining the properties. Any required dependencies will be resolved and inserted into the list of jobs. Depending on the status of the job, the job is monitored if running (or queued), files will be created if necessary, and errors will be handled automatically. If the job is finished successfully, data is collected and a simple post-processing is conducted.

1. *Input from user*

The user specifies the materials simulation software and the usage of the scheduler system or localhost. The input information includes the chemical elements with the required parameter files, the structure files and the tasks to be performed. The currently implemented tasks are related to materials properties and include, e.g., the relaxation of the structure or the computation of the equilibrium lattice constant. The input can be organized in a YAML file format for starting strucscan from the command line or in a Python dictionary for starting strucscan from a python shell or a Jupyter Notebook.

2. *Initialization of workflow*

Based on the input from the user, strucscan generates of list of all necessary calculations by looping over the list of given structures and the list of tasks. In the context of materials properties, a common task is the full relaxation of a crystal structure from an initial guess of the atomic positions and simulation cell to a configuration that takes a minimum total energy in a DFT calculation. This task is often followed by a second task where the total energy is computed with DFT for a series of volumes of the simulation cell around the equilibrium volume and a subsequent fit of this energy-volume data to obtain the minimum with high accuracy. For such interdependent tasks, strucscan uses a pipelining concept that converts the final structure of one task to the initial structure of a subsequent task. I.e. each structure is pipelined through the list of properties. The pipelines are also treated as tasks and can easily be modified by the user. In this way the user can operate with higher-level pipeline names and strucscan will automatically insert all required tasks along a pipeline. Continuing with the above example, strucscan provides a pipeline 'EOS' that collects the values of the equilibrium volume after fitting the energy-volume data after computing the energy-volume data after performing a full relaxation. The result of the workflow initialization is a list of all necessary calculations, '*jobs*', that is directly reflected in the structure of the data tree. A restart of strucscan with the same user input will find the existing data tree and continue seamlessly after the last finished calculation.

3. *Execution of tasks*

After the initialization, strucscan identifies the status of each *job* by checking if the expected folders exist in the data tree, if it is waiting or running in the scheduler, if it is finished or if an error occurred. Depending on the status, strucscan will create the necessary input files, start the calculation or handle an error. This stage is repeated until the list of jobs is complete. In order to avoid uncontrolled restarts, a job is declared as finished if error handling has been attempted unsuccessfully for three times.

4. *Post-processing of results*

At the end of each workflow cycle, strucscan starts post-processing of the calculation results. It will collect the central results from the data tree and compile them in Python dictionaries in JSON format for further post-processing or for database upload. In the context of materials properties the post-processing by strucscan includes, e.g., the fitting of energy-volume data to an equation of state and the compilation of the resulting equilibrium volume, binding energy and bulk modulus in one JSON file for all given structures of a given chemical element or compound.

Acknowledgements

The authors acknowledge funding by the Deutsche Forschungsgemeinschaft (DFG) through project C1 of the collaborative research center SFB/TR 103 (DFG project number 190389738).

References

Gentzsch, W. (2001). *Sun grid engine: Towards creating a compute power grid*. 35. https://doi.org/10.1109/CCGRID.2001.923173

Gjerding, M., Skovhus, T., Rasmussen, A., Bertoldo, F., Larsen, A. H., Mortensen, J. J., & Thygesen, K. S. (2021). Atomic simulation recipes: A python framework and library for automated workflows. *Computational Materials Science*, *199*, 110731. https://doi.org/10.1016/j.commatsci.2021.110731

Hammerschmidt, T., Bialon, A. F., Pettifor, D. G., & Drautz, R. (2013). Topologically close-packed phases in binary transition-metal compounds: Matching high-throughput ab initio calculations to an empirical structure map. *New Journal of Physics*, *15*, 115016. https://doi.org/10.1088/1367-2630/15/11/115016

Huber, S. P., Zoupanos, S., & al, M. U. et. (2020). AiiDA 1.0, a scalable computational infrastructure for automated reproducible workflows and data provenance. *Scientific Data*, *7*(300). https://doi.org/10.1038/s41597-020-00638-4

Jain, A., Ong, S. P., Chen, W., Medasani, B., Qu, X., Kocher, M., Brafman, M., Petretto, G., Rignanese, G.-M., Hautier, G., Gunter, D., & Persson, K. A. (2015). FireWorks: A dynamic workflow system designed for high-throughput applications. *Concurrency and Computation: Practice and Experience*, *27*, 5037–5059. https://doi.org/10.1002/cpe.3505

Janssen, J., Surendralal, S., Lysogorskiy, Y., Todorova, M., Hickel, T., Drautz, R., & Neugebauer, J. (2019). Pyiron: An integrated development environment for computational materials science. *Computational Materials Science*, *163*, 24–36. https://doi.org/10.1016/j.commatsci.2018.07.043

Jette, M. A., Yoo, A. B., & Grondona, M. (2002). *SLURM: Simple linux utility for resource management*. 44–60. https://doi.org/10.1007/10968987_3

Kresse, G., & Furthmüller, J. (1996a). Efficiency of ab-initio total energy calculations for metals and semiconductors using a plane-wave basis set. *Computational Materials Science*, *6*, 15–50. https://doi.org/10.1016/0927-0256(96)00008-0

Kresse, G., & Furthmüller, J. (1996b). Efficient iterative schemes for ab initio total-energy calculations using a plane-wave basis set. *Physical Review B*, *54*, 11169–11186. https://doi.org/10.1103/physrevb.54.11169

Kresse, G., & Joubert, D. (1999). From ultrasoft pseudopotentials to the projector augmented-wave method. *Physical Review B*, *59*, 1758–1775. https://doi.org/10.1103/PhysRevB.59.1758

Larsen, A. H., Mortensen, J. J., Blomqvist, J., Castelli, I. E., Christensen, R., Dułak, M., Friis, J., Groves, M. N., Hammer, B., Hargus, C., Hermes, E. D., Jennings, P. C., Jensen, P. B., Kermode, J., Kitchin, J. R., Kolsbjerg, E. L., Kubal, J., Kaasbjerg, K., Lysgaard, S., … Jacobsen, K. W. (2017). The atomic simulation environment—a python library for working with atoms. *Journal of Physics: Condensed Matter*, *29*, 273002. https://doi.org/10.1088/1361-648x/aa680e

Mathew, K., Montoya, J. H., Faghaninia, A., Dwarakanath, S., Aykol, M., Tang, H., Chu, I., Smidt, T., Bocklund, B., Horton, M., Dagdelen, J., Wood, B., Liu, Z.-K., Neaton, J., Ong, S. P., Persson, K., & Jain, A. (2017). Atomate: A high-level interface to generate, execute, and analyze computational materials science workflows. *Computational Materials Science*, *139*, 140–152. https://doi.org/10.1016/j.commatsci.2017.07.030

Ong, S. P., Richards, W. D., Jain, A., Hautier, G., Kocher, M., Cholia, S., Gunter, D., Chevrier, V. L., Persson, K. A., & Ceder, G. (2013). Python materials genomics (pymatgen): A robust, open-source python library for materials analysis. *Computational Materials Science*, *68*, 314–319. https://doi.org/10.1016/j.commatsci.2012.10.028