29-03-2021


# D2.1 – Report on a first alpha release of the I/O library, ready for WP4

Version V1.0

## GA no 952165

Dissemination Level

- ☒ PU: Public
- ☐ PP: Restricted to other programme participants (including the Commission)
- ☐ RE: Restricted to a group specified by the consortium (including the Commission)
- ☐ CO: Confidential, only for members of the consortium (including the Commission)

# Document Information

| | |
|---|---|
| Project Title | Targeting Real Chemical accuracy at the EXascale |
| Project Acronym | TREX |
| Grant Agreement No | 952165 |
| Instrument | Call: H2020-INFRAEDI-2019-1 |
| Topic | INFRAEDI-05-2020 Centres of Excellence in EXascale computing |
| Start Date of Project | 01-10-2020 |
| Duration of Project | 36 Months |
| Project Website | https://trex-coe.eu/ |

| | |
|---|---|
| Deliverable Number | D2.1 |
| Deliverable title | D2.1 – Report on a first alpha release of the I/O library, ready for WP4, as seen in GA |
| Due Date | M06 – 31-03-2021 (from GA) |
| Actual Submission Date | 29-03-2021 |

| | |
|---|---|
| Work Package | WP2 – Code modularization and interfacing |
| Lead Author (Org) | Sandro Sorella (Scuola Internazionale Superiore di Studi Avanzati di trieste (SISSA)) |
| Contributing Author(s) (Org) | Anthony Scemama (Centre National de la Recherche Scientifique (CNRS)) |
| Reviewers (Org) | François Coppens (Université de Versailles Saint-Quentin-en-yvelines (UVSQ)), Pablo Oliveira (UVSQ), Matúš Dubecký (Slovenská technická univerzita v Bratislave (STUBA)), Jan Beerens (Universiteit Twente (UT) |
| Version | V1.0 |
| Dissemination level | PU |
| Nature | Report |
| Draft / final | Final |
| No. of pages including cover | 29 |

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

## Disclaimer

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

The content of this document does not represent the opinion of the European Union, and the European Union is not responsible for any use that might be made of such content.

# Versioning

| Version | Date | Authors | Notes |
|---|---|---|---|
| 1.0 | 29-03-2021 | Sandro Sorella (SISSA) | First Official Release |

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

# Abbreviations

| | |
|---|---|
| **AO** | Atomic Orbital |
| **API** | Application Programming Interface |
| **BSE** | Basis Set Exchange |
| **CI** | Configuration Interaction |
| **CNRS** | Centre National de la Recherche Scientifique |
| **CoE** | Center of Excellence |
| **ECP** | Effective Core Potential |
| **ERI** | Electron Repulsion Integral |
| **EZFIO** | Easy Fortran Input/Output |
| **GPFS** | General Parallel File System |
| **HDF5** | Hierarchical Data Format |
| **HPC** | High Performance Computing |
| **I/O** | Input/Output |
| **MO** | Molecular Orbital |
| **QMC** | Quantum Monte Carlo |
| **SISSA** | Scuola Internazionale Superiore di Studi Avanzati di trieste |
| **STUBA** | Slovenská technická univerzita v Bratislave |
| **TREX** | Targeting REal chemical accuracy at the eXascale |
| **TREXIO** | TREX Input/Output |
| **UT** | Universiteit Twente |
| **UVSQ** | Université de Versailles Saint-Quentin-en-yvelines |

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

# Table of Contents



TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

# List of Figures



TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

# List of Tables



TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

# 1 Introduction

We build a library to help inter-operability between codes in the field of quantum chemistry, primarily focused on enabling the communication of data between the flagship codes of the Targeting REal chemical accuracy at the eXascale (TREX) Center of Excellence (CoE) (NECI, GammCor, Quantum Package, QMC=Chem, CHAMP, TurboRVB, QML). We expect this library to be also adopted by the community beyond the TREX CoE.

The data that needs to be stored is the electronic wave function, which is obtained from a post-Hartree-Fock calculation, or the one- and two-body density matrices, together with the one- and two-electron integrals that are necessary to compute the energy or other properties. As a wave function can be obtained by executing multiple codes in a complex workflow, the library should give the possibility to build the files incrementally using multiple codes.

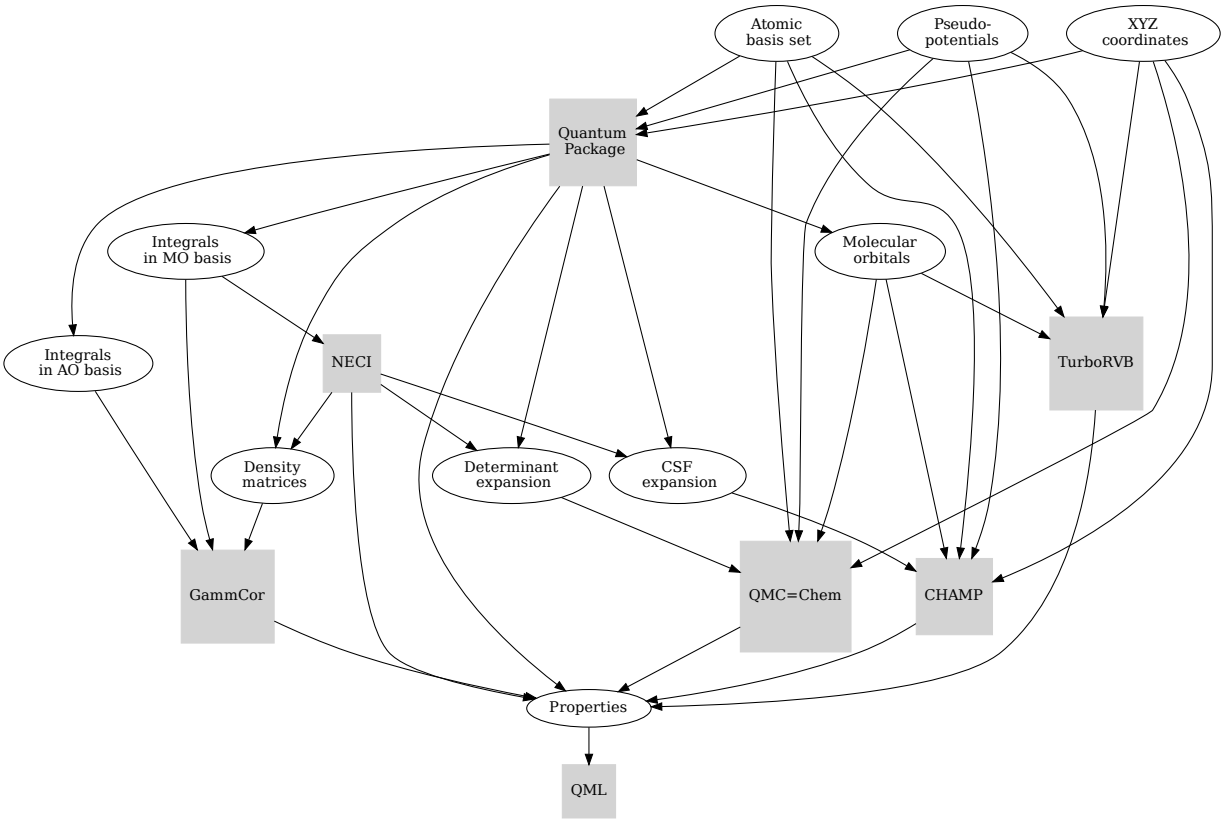


**Figure 1**: Dependencies between the codes and the data. Codes are represented in gray, and data are represented in white.

Fig. 1 shows which data are read and written by all of the codes of the TREX CoE. The objective of this library is to organize all the data in a file, and provide a common interface to make the data easily accessible to all the codes.

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

# 2 Content of the files

The files need to be self-contained: they should contain *all* the information needed to reconstruct the wave functions from an external program, without relying on any extra service to provide additional data. For instance, all the parameters of the atomic basis set should be explicitly stored instead of storing only the conventional name of the basis, which would require obtaining the parameters from another source. It is important for the files to be self-contained, as they are intended to be archived on an open-data repository: this reduces their dependencies on other services and therefore increases their re-usability.

## Metadata

As we expect our files to be archived in open-data repositories, we need to give the possibility to the users to store some metadata inside the files. We propose to store the list of names of the codes which have participated to the creation of the file, a list of authors of the file, and a textual description.

## Nuclei

We consider wave functions where the nuclei are considered as fixed point charges. The file should contain information describing the positions of the nuclei, their charges, their labels, and the point-group symmetry of the system.

## Electrons

We consider wave functions expressed in the spin-free formalism, where the number of $\uparrow$ and $\downarrow$ electrons is fixed.

## Atomic Basis set

We consider here basis functions centered on nuclei. Hence, we enable the possibility to define *dummy atoms* to place basis functions in random positions.

The atomic basis set is defined as a list of shells. Each shell $s$ is centered on a center $A$, possesses a given angular momentum $l$ and a radial function $R_s$. The radial function is a linear combination of $N_{\text{prim}}$ *primitive* functions that can be of type Slater ($p = 1$) or Gaussian ($p = 2$), parameterized by exponents $\gamma_{ks}$ and coefficients $a_{ks}$:

$$R_s(\mathbf{r}) = \mathcal{N}_s |\mathbf{r} - \mathbf{R}_A|^{n_s} \sum_{k=1}^{N_{\text{prim}}} a_{ks} \exp\left(-\gamma_{ks} |\mathbf{r} - \mathbf{R}_A|^p\right).$$

In the case of Gaussian functions, $n_s$ is always zero. The normalization factor $\mathcal{N}_s$ ensures that all the functions of the shell are normalized to unity. As this normalization requires the ability to compute overlap integrals, the normalization factors should be written in the file to ensure that the file is self-contained and does not require the client program to have the ability to compute such integrals.

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

## Atomic Orbitals

Going from the atomic basis set to Atomic Orbitals (AOs) implies a systematic construction of all the angular functions of each shell. We consider two cases for the angular functions: the real-valued spherical harmonics, and the polynomials in Cartesian coordinates. In the case of spherical harmonics, the AOs are ordered in increasing magnetic quantum number ($-l \leq m \leq l$), and in the case of polynomials we choose the canonical ordering of the `Libint`[1] library, i.e

$$\begin{aligned} p &: p_x, p_y, p_z \\ d &: d_{xx}, d_{xy}, d_{xz}, d_{yy}, d_{yz}, d_{zz} \\ f &: f_{xxx}, f_{xxy}, f_{xxz}, f_{xyy}, f_{xyz}, f_{xzz}, f_{yyy}, f_{yyz}, f_{yzz}, f_{zzz} \\ \text{etc.} & \end{aligned}$$

AOs are defined as

$$\chi_i(\mathbf{r}) = P_{\eta(i)}(\mathbf{r})\, R_{\theta(i)}(\mathbf{r})$$

where $i$ is the atomic orbital index, $P$ encodes for either the polynomials or the spherical harmonics, $\theta(i)$ returns the shell on which the AO is expanded, and $\eta(i)$ denotes which angular function is chosen.

## Effective core potentials

It is common to use Effective Core Potentials (ECPs) in Quantum Monte Carlo (QMC) calculations. An ECP $V_A^{\text{pp}}$ replacing the core electrons of atom $A$ is the sum of a local component $V_A^{\text{loc}}$ and a non-local component $V_A^{\text{non-loc}}$.[2] The local component is given by

$$V_A^{\text{loc}}(r) = -\frac{Z_A^{\text{eff}}}{r} + \frac{Z_A^{\text{eff}}}{r}\, \exp\left(-\alpha_A\, r^2\right) + Z_{\text{eff}}\, \alpha_A\, r\, \exp\left(-\beta_A\, r^2\right) + \gamma_A \exp\left(-\delta_A\, r^2\right),$$

and the component obtained after localizing the non-local operator is

$$V_A^{\text{non-loc}}(r) = \zeta_A\, \exp\left(-\eta_A\, r^2\right) |0\rangle\langle 0| + \mu_A\, \exp\left(-\nu_A\, r^2\right) |1\rangle\langle 1|$$

where $r = |\mathbf{r} - \mathbf{R}_A|$ is the distance to the nucleus on which the potential is centered, $Z_A^{\text{eff}}$ is the effective charge due to the removed electrons, $|0\rangle\langle 0|$ and $|1\rangle\langle 1|$ are projections over zero and one principal angular momenta, respectively (generalization to higher angular momenta is straightforward), and all the parameters labeled by Greek letters are parameters.

## One-electron integrals in the AO basis set

The one-electron integrals are of the form

$$O_{ij} = \int \chi_i(\mathbf{r})\, \hat{O}\, \chi_j(\mathbf{r}) \mathrm{d}\mathbf{r},$$

where $\hat{O}$ is a one-electron operator. The integrals needed to compute the energy are the overlap integrals $\mathbf{S}$ with $\hat{O} = \mathbf{1}$, the kinetic energy integrals $\mathbf{T}$ with $\hat{O} = \nabla^2$, the electron-nucleus potential integrals $\mathbf{V}$ with $\hat{O} = -\sum_A -Z_A^{\text{eff}}/|\mathbf{r} - \mathbf{R}_A|$, and the effective core potential integrals $\mathbf{V}^{\text{pp}}$ with $\hat{O} = V^{\text{pp}}$. It is also convenient to store the *core Hamiltonian* integrals, defined as the sum of all the previously mentioned one-electron integrals.

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

## Two-electron integrals in the AO basis set

Electron Repulsion Integrals (ERIs) are given by

$$W_{pqrs} = \iint \chi_p^*(\mathbf{r}_1)\, \chi_q^*(\mathbf{r}_2)\, \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|}\, \chi_r(\mathbf{r}_1)\, \chi_s(\mathbf{r}_2)\, \mathrm{d}\mathbf{r}_1 \mathrm{d}\mathbf{r}_2$$

ERIs have $(p, q, r, s)$ as indices, so their number grows formally as $N^4$ where $N$ is the number of AOs. However, due to the locality of the AOs these integrals are negligible when the overlap between $\langle\chi_p|\chi_r\rangle$ or $\langle\chi_q|\chi_s\rangle$ is close to zero. Hence, this data structure is sparse, and it is preferable to store only the non-zero values. One can also remark that the ERIs are symmetric with respect to the exchange of electron coordinates $\mathbf{r}_1$ and $\mathbf{r}_2$, and for real orbitals one can also exchange the indices $p$ and $r$ and/or $q$ and $s$. This enables an even more compact storage if only unique values are stored.

## One-electron integrals in the MO basis set

Post Hartree-Fock methods generally require integrals transformed from the AO basis set to the Molecular Orbital (MO) basis set, so it is convenient to be able to read them from a file. Some codes within TREX don't enforce the orthogonality between the MOs, so we provide the possibility to store the overlap matrix of the MOs in addition to all the other ones (kinetic, potential, ECP, core Hamiltonian).

## Two-electron integrals in the MO basis set

The transformation of ERIs from the AO to the MO basis can be expensive, as it scales as $N^5$. Therefore, storing these integrals is often necessary. The structure of these integrals is the same as the integrals in the AO basis set, with the same permutation symmetry in the indices:

$$W_{ijkl}^{\mathrm{MO}} = \iint \phi_i^*(\mathbf{r}_1)\, \phi_j^*(\mathbf{r}_2)\, \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|}\, \phi_k(\mathbf{r}_1)\, \phi_l(\mathbf{r}_2)\, \mathrm{d}\mathbf{r}_1 \mathrm{d}\mathbf{r}_2$$

However, as the MOs are generally delocalized, these data structures are less sparse than their AO counterparts.

## Summary

Table 1, summarizes the data that need to be stored in the files.

**Table 1**: Information that should be stored in the file.

| | |
|---|---|
| Metadata | File description |
| | Code used to write the file |
| | Authors of the file |

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

| | |
|---|---|
| Nuclei | Number of nuclei<br>Atomic charges<br>XYZ coordinates<br>Atom labels<br>Point-group Symmetry |
| Electrons | Number of $\uparrow$ and $\downarrow$ electrons |
| ECPs | Effective charge<br>Exponents of the local component<br>Coefficients of the local component<br>Powers of $r$ of the local component<br>Exponents of the non-local component<br>Coefficients of the non-local component<br>Powers of $r$ of the non-local component |
| Atomic basis set | Type : Gaussian or Slater<br>Cartesian or spherical coordinates<br>Normalization factors of the shells<br>Nuclei on which the functions are centered<br>Angular momenta<br>Exponents of the primitives<br>Coefficients of the primitives |
| One-electron integrals in the AO basis | Overlap integrals<br>Kinetic energy<br>Potential energy<br>Local component of the ECP<br>Non-Local component of the ECP<br>Core Hamiltonian |
| Two-electron integrals in the AO basis | Array of indices of ERI<br>Array of values of ERI |

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

| | |
|---|---|
| Molecular orbitals | Type : Hartree-Fock, Localized, Natural, ...<br>Coefficients<br>Class : Core, Inactive, Active, Virtual, Deleted, ...<br>Symmetry<br>Occupation number |
| One-electron integrals in the MO basis | Kinetic energy<br>Potential energy<br>Local component of the ECP<br>Non-Local component of the ECP<br>Core Hamiltonian |
| Two-electron integrals in the MO basis | Array of indices of ERI<br>Array of values of ERI |

# 3 Design of the library

The design of the library is split in two distinct sections: the front end, and the back end. The front end is the interface between the users and the library, and the back end is the interface between the library and the physical storage. The library is designed to decouple as much as possible the front end from the back end.

## License

The library is licensed under the open-source 3-clause BSD license to facilitate its adoption in all quantum chemistry software, commercial or not.

## The front end

From table 1, it appears clearly that the data can be organized in a tree structure, where the root of the tree is the file, elementary pieces of data are leaves of the tree, and the nodes between the root and the leaves constitute groups. Hence, the data is organized in *groups* (the left column of Table 1), and each group contains the associated data.

Most of the codes of the CoE are written in Fortran, with some scripts in Python. Therefore, the Application Programming Interface (API) should be such that the functions can be called easily in Fortran, and such that a Python interface to the library is easy to write. These constraints have lead us to the choice of implementing the library in C, with interfaces for Fortran and Python.

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

To maximize the portability of the library, the data exposed to the user in the API are reduced to the subset of elementary C types: 32-/64-bit integers and floats, scalars or arrays. The integer types defined in `<stdint.h>` (`int64_t` and `int32_t`) are used instead of the native C integer types. Boolean variables are stored as integers, `1` for `true` and `0` for `false`, and complex numbers are represented as an array of two floats, the real part at address `0` and the imaginary part at address `1`.

The library allows the user to create, open or close a file, and check if the file exists. The file may be opened in read-only mode to protect it from accidental data corruption.

Once the file is open, any kind of data can be read by calling the corresponding function. The function names for data access obey the following structure:

```
trexio_<read|write|has>_<group>_<data>
```

where `<data>` is the data to access, `<group>` is the group in which the data belongs, and `<read|write|has>` can read, write, or test if the data exists in the file. For example, if a user wants to read the array of nuclear coordinates, the function `trexio_read_nucleus_coord` should be called.

As the different data types are rather limited and the rules to build the function names are simple, the best strategy is to generate automatically the C code for all these functions from a simple configuration file, `trex.json` as:

```
{
    "metadata": {
        "code_num"           : [ "int" , [                                ] ]
      , "code"               : [ "char", [ "metadata.code_num"   , "128"  ] ]
      , "author_num"         : [ "int" , [                                ] ]
      , "author"             : [ "char", [ "metadata.author_num", "128"  ] ]
      , "description_length" : [ "int" , [                                ] ]
      , "description"        : [ "char", [ "metadata.description_length" ] ]
    },

    "electron": {
        "up_num"             : [ "int", [] ]
      , "dn_num"             : [ "int", [] ]
    },

    "nucleus": {
        "num"                : [ "int"  , [                    ] ]
      , "charge"             : [ "float", [ "nucleus.num"      ] ]
      , "coord"              : [ "float", [ "nucleus.num", "3" ] ]
      , "label"              : [ "char" , [ "nucleus.num", "32" ] ]
      , "point_group"        : [ "char" , [ "32"               ] ]
    },

    "ecp": {
        "lmax_plus_1"        : [ "int"  , [ "nucleus.num"                          ] ]
      , "z_core"             : [ "int"  , [ "nucleus.num"                          ] ]
      , "local_n"            : [ "int"  , [ "nucleus.num"                          ] ]
      , "local_num_n_max"    : [ "int"  , [                                        ] ]
      , "local_exponent"     : [ "float", [ "nucleus.num", "ecp.local_num_n_max"   ] ]
```

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

```
      , "local_coef"          : [ "float", [ "nucleus.num", "ecp.local_num_n_max"     ] ]
      , "local_power"         : [ "int"  , [ "nucleus.num", "ecp.local_num_n_max"     ] ]
      , "non_local_n"         : [ "int"  , [ "nucleus.num"                            ] ]
      , "non_local_num_n_max": [ "int"  , [                                          ] ]
      , "non_local_exponent" : [ "float", [ "nucleus.num", "ecp.non_local_num_n_max" ] ]
      , "non_local_coef"      : [ "float", [ "nucleus.num", "ecp.non_local_num_n_max" ] ]
      , "non_local_power"     : [ "int"  , [ "nucleus.num", "ecp.non_local_num_n_max" ] ]
    },

    "basis" : {
        "type"                : [ "char" , [ "32"              ] ]
      , "shell_num"           : [ "int"  , [                   ] ]
      , "shell_factor"        : [ "float", [ "basis.shell_num" ] ]
      , "shell_center"        : [ "int"  , [ "basis.shell_num" ] ]
      , "shell_ang_mom"       : [ "int"  , [ "basis.shell_num" ] ]
      , "shell_prim_num"      : [ "int"  , [ "basis.shell_num" ] ]
      , "prim_index"          : [ "int"  , [ "basis.shell_num" ] ]
      , "prim_num"            : [ "int"  , [                   ] ]
      , "exponent"            : [ "float", [ "basis.prim_num"  ] ]
      , "coefficient"         : [ "float", [ "basis.prim_num"  ] ]
    },

    "ao" : {
        "num"                 : [ "int"  , [          ] ]
      , "cartesian"           : [ "int"  , [          ] ]
      , "shell"               : [ "int"  , [ "ao.num" ] ]
      , "normalization"       : [ "float", [ "ao.num" ] ]
    },

    "basis_1e_int" : {
        "overlap"             : [ "float", [ "ao.num", "ao.num" ] ]
      , "kinetic"             : [ "float", [ "ao.num", "ao.num" ] ]
      , "potential"           : [ "float", [ "ao.num", "ao.num" ] ]
      , "ecp_local"           : [ "float", [ "ao.num", "ao.num" ] ]
      , "ecp_non_local"       : [ "float", [ "ao.num", "ao.num" ] ]
      , "core_hamiltonian"    : [ "float", [ "ao.num", "ao.num" ] ]
    },

    "basis_2e_int" : {
        "eri"                 : [ "float sparse", [ "ao.num", "ao.num", "ao.num", "ao.num" ] ]
    },

    "mo" : {
        "type"                : [ "char" , [ "32"                          ] ]
      , "num"                 : [ "int"  , [                               ] ]
      , "coef"                : [ "float", [ "basis.shell_num", "mo.num" ] ]
      , "class"               : [ "char" , [ "mo.num", "32"                ] ]
      , "symmetry"            : [ "char" , [ "mo.num", "32"                ] ]
      , "occupation"          : [ "float", [ "mo.num"                      ] ]
    },

    "mo_1e_int" : {
```

```
        "kinetic"            : [ "float", [ "mo.num", "mo.num" ] ]
      , "potential"          : [ "float", [ "mo.num", "mo.num" ] ]
      , "ecp_local"          : [ "float", [ "mo.num", "mo.num" ] ]
      , "ecp_non_local"      : [ "float", [ "mo.num", "mo.num" ] ]
      , "core_hamiltonian"   : [ "float", [ "mo.num", "mo.num" ] ]
    },

    "mo_2e_int" : {
        "eri"                : [ "float sparse", [ "mo.num", "mo.num", "mo.num", "mo.num" ] ]
    },

    "rdm" : {
        "one_e"              : [ "float", [ "mo.num", "mo.num" ] ]
      , "two_e"              : [ "float sparse", [ "mo.num", "mo.num", "mo.num", "mo.num" ] ]
    }
}
```

Such a simple file will be extremely valuable for the evolution of the library when some additional features will need to be implemented, such as the Configuration Interaction (CI) wave function, the Jastrow factor, data for periodic systems (cell tensor), range-separated ERIs, *etc.*

## Conventions

To facilitate the understanding of the users, we have defined strong conventions. Indeed, using as few exceptions as possible makes it easier for users to guess the answers to their questions.

- All the data are stored in atomic units.
- Preprocessor constants are expressed in upper case.
- Pointers are always set to `NULL` when not attached to a memory block.
- The singular is always used for the names of the variables.
- The `num` suffix denotes counting. For example `apple_num` is the number of apples.
- All the functions will be provided with two versions. One with a 32-bit representation of integers, and one with a 64-bit. This will reduce the risk of integer overflows.
- The first argument of the functions is always the file handle.
- The functions always return an exit code.
- Reading functions pass the argument in which to read the data by address, similarly to `scanf`.
- Writing functions pass the argument to be written by value, similarly to `printf`.

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

## Error handling

In case of error, the user of the library should be informed that the called function did not succeed in performing the requested work. The library should never make the calling program crash, nor decide to halt the execution. It should not even take the decision to print something on the terminal. The choice of how to handle the errors should be left completely to the code calling the function.

An effort is made in checking the validity of all the arguments of the functions to ensure that the preconditions are fulfilled. If some unexpected behavior happens, the function returns with an error code, which can be translated to a string using a function call.

## Safety

Multiple precautions need to be taken to prevent users from accidentally corrupting files. For the alpha release, we have chosen the simple model in which when the file is opened by a process, the file is locked until it is closed. This ensures that the data previously read from the file has not been changed by another process, so it is easy to ensure that the data read from the file and cached in memory is consistent with the data stored in the file.

A second level of safety needs to be added for multi-threaded environments, to avoid situations where multiple threads write the same data at the same time, which could lead to inconsistent data being written. All the provided functions are thread-safe.

## Prototype library

Before implementing the actual TREX Input/Output (TREXIO) library, a prototype was created using the design exposed in the previous section. This library is available in a repository under the TREX GitHub organization[1]. This prototype uses the Easy Fortran Input/Output (EZFIO) library generator[3] as a back end to generate the stored files. This enabled us to concentrate on the front end, which is the direct interface to the users, while being able to use it to see if it can be conveniently used in practice.

## The back end

We would like the data to be organized in the file, reproducing the hierarchy of the data. Using a binary format is desirable for the performance of large data sets, such as integrals or density matrices, but binary files are not necessarily compatible between different architectures because of the endianness of the binary representation. Hence, if we store data in binary format, the back end should make the files machine-independent by handling properly the endianness.

Finally, as the produced files are likely to be archived on open data repositories, it is desirable to have the possibility to compress them efficiently.

The EZFIO library, already used in QMC=Chem and Quantum Package, fulfills some of the requirements (automatically generated library and hierarchical storage). But this library is not well adapted to High Performance Computing (HPC) systems. EZFIO generates a large number of small

[1] https://github.com/trex-coe/trex-io-prototype

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

files, and large supercomputers use distributed file systems such as Lustre[4], General Parallel File System (GPFS)[5] or BeeGFS[6] which all suffer from a huge performance impact when multiple files are used. In addition, EZFIO is intended to be easy to use, but Input/Output (I/O) performance is not the main objective of this library.

The Hierarchical Data Format (HDF5) file format and library[7] address properly all the required aspects. The first version of HDF5 was released in 1998, so this file format has been present in the HPC landscape for a long time. The data can be stored in a hierarchy similar to a file system, exactly in the way we described in table 1. The library also provides compression possibilities, and is reputed for its high performance read/write operations.

Although HDF5 fulfills all our needs, we need to be careful about some important side effects of using such a library. First, as data are written in a binary format, it is possible to corrupt a file if the program crashes during a write operation. Secondly, HDF5 is a complex piece of software, which might not be installed (or even difficult to install) on some systems. If our library only provides an HDF5 back end, the users unable to install HDF5 will not be able to use our library, and therefore will not be able to use any of the TREX codes. Hence, we need to protect our users from these situations.

We also provide a text-file back end. This back end is by far less efficient, but is has the advantage that it requires no dependencies. We also provide a tool to convert files from the HDF5 format to the text format, to ensure that if some large data have been prepared in the HDF5 format, it can be converted to the text file format for following calculations.

The separation of the front end and the back end makes it easy to implement new back ends in the future.

# 4 Illustrative example of usage of the library

We propose here to show as an example a Fortran program that reads $x, y, z$ coordinates of a molecule and a basis set from a file obtained from the Basis Set Exchange (BSE) web site,[8] and stores it into a file using the TREXIO library.

```
subroutine fail_if_error(file,info)
  use trexio
  implicit none
  integer*8, intent(in)        :: file
  integer, intent(in)          :: info
  character*(*), intent(out)   :: message

  if (info /= TREXIO_SUCCESS) then
     call trexio_strerror(file, info, message)
     print *, info, message
     stop -1
  end if
end subroutine check_success

!-------------------------------------------------------------------------------

```

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

```f90
subroutine read_xyz(trex_file, xyz_filename)
  use trexio
  implicit none
  integer*8, intent(in)          :: trex_file
  character*(128), intent(in)    :: xyz_filename
  integer*8                      :: nucl_num         ! Number of nuclei
  character*(256)                :: title            ! Title of the file
  character*(32), allocatable    :: nucl_label(:)    ! Atom labels
  real*8, allocatable            :: nucl_charge(:)   ! Nuclear charges
  real*8, allocatable            :: nucl_coord(:,:)  ! Nuclear coordinates
  integer*8                      :: i
  integer                        :: j
  integer                        :: info
  double precision, parameter    :: a0 = 0.52917721067d0

  open(unit=10,file=xyz_filename)

  read(10,*) nucl_num

  allocate(nucl_label(nucl_num),  &
           nucl_charge(nucl_num), &
           nucl_coord(3,nucl_num) )

  read(10,'(A)') title

  do i=1,nucl_num
     read(10,*) nucl_label(i), nucl_coord(1:3,i)

     info = trexio_element_number_of_symbol(trim(nucl_label(i)), j)
     call check_success(info, 'Unable to convert symbol to number')

     nucl_charge(i) = dble(j)
  end do

  close(10)

  ! Convert into atomic units
  nucl_coord = nucl_coord / a0

  info = trexio_write_nucleus_num(trex_file,nucl_num)
  call check_success(info, 'Unable to write number of nuclei')

  info = trexio_write_nucleus_coord(trex_file,nucl_coord)
  call check_success(info, 'Unable to write nuclear coordinates')

  info = trexio_write_nucleus_charge(trex_file,nucl_charge)
  call check_success(info, 'Unable to write nuclear charges')

  info = trexio_write_nucleus_label(trex_file,nucl_label)
  call check_success(info, 'Unable to write nuclear labels')

  beta_num  = int(sum(nucl_charge(:)))/2
```

```
69    alpha_num = int(sum(nucl_charge(:))) - beta_num
70
71    info = trexio_write_electron_up_num(trex_file,alpha_num)
72    call check_success(info, 'Unable to write up electrons')
73
74    info = trexio_write_electron_dn_num(trex_file,beta_num)
75    call check_success(info, 'Unable to write dn electrons')
76
77  end subroutine read_xyz
78
79  !-------------------------------------------------------------------------------
80
81  subroutine read_basis(trex_file, basis_filename)
82    use trexio
83    implicit none
84    integer*8, intent(in)          :: trex_file
85    character*(128), intent(in)    :: basis_filename
86
87    integer*8                      :: nucl_num         ! Number of nuclei
88    character*(32), allocatable    :: nucl_label(:)    ! Atom labels
89    integer*8                      :: shell_num, prim_num
90    integer*8, allocatable         :: shell_center(:)
91    integer  , allocatable         :: shell_ang_mom(:)
92    integer*8, allocatable         :: shell_prim_num(:)
93    integer*8, allocatable         :: prim_index(:)
94    double precision, allocatable  :: shell_factor(:)
95    double precision, allocatable  :: exponent(:)
96    double precision, allocatable  :: coefficient(:)
97    character*(32)                 :: label
98    character*(80)                 :: buffer
99    integer                        :: i,j,k,n_shell,n_prim
100   integer                        :: info
101
102   open(unit=10, file=basis_filename)
103
104   info = trexio_read_nucleus_num(trex_file, nucl_num)
105   call check_success(info, 'Unable to read number of nuclei')
106
107   allocate(nucl_label(nucl_num))
108   info = trexio_read_nucleus_label(trex_file, nucl_label)
109   call check_success(info, 'Unable to read nuclear label')
110
111   shell_num = 0
112   prim_num  = 0
113   ! Find dimensioning variables
114   do i=1,nucl_num
115      info = trexio_element_name_of_symbol(nucl_label(i), label)
116      call check_success(info, 'Unable to get name of label')
117
118      ! Find element
119      rewind(10)
120      do
```

```
      read(10,*, iostat=j) buffer
      if (j < 0) exit
      if (trim(buffer) == trim(label)) then
         j=0
         exit
      end if
   end do
   if (j < 0) exit

   ! Read shell
   do
      read(10,*,iostat=k) buffer, j
      if (k /= 0) exit
      shell_num = shell_num + 1
      prim_num  = prim_num  + j
      do k=1,j
         read(10,*)
      end do
   end do

end do

buffer = 'Gaussian'
info = trexio_write_basis_type(trex_file, buffer)
call check_success(info, 'Unable to write basis type')

info = trexio_write_basis_shell_num(trex_file, shell_num)
call check_success(info, 'Unable to write basis shell_num')

info = trexio_write_basis_prim_num(trex_file, prim_num)
call check_success(info, 'Unable to write basis prim_num')

allocate(shell_center(shell_num),   &
         shell_ang_mom(shell_num),  &
         shell_prim_num(shell_num), &
         prim_index(shell_num),     &
         shell_factor(shell_num),   &
         exponent(prim_num),        &
         coefficient(prim_num))

shell_num = 1
prim_num  = 1
! Read data
do i=1,nucl_num
   info = trexio_element_name_of_symbol(nucl_label(i), label)
   call check_success(info, 'Unable to get name of label')

   ! Find element
   rewind(10)
   do
      read(10,*, iostat=j) buffer
      if (j < 0) exit
```

```
173         if (trim(buffer) == trim(label)) then
174            j=0
175            exit
176         end if
177      end do
178
179      ! Read shell
180      do
181         read(10,*,iostat=j) label, k
182         if (j /= 0) exit
183         select case (label(1:1))
184         case ('S')
185            shell_ang_mom(shell_num) = 0
186         case ('P')
187            shell_ang_mom(shell_num) = 1
188         case ('D')
189            shell_ang_mom(shell_num) = 2
190         case ('F')
191            shell_ang_mom(shell_num) = 3
192         case ('G')
193            shell_ang_mom(shell_num) = 4
194         case ('H')
195            shell_ang_mom(shell_num) = 5
196         case ('I')
197            shell_ang_mom(shell_num) = 6
198         case default
199            stop 'Too high angular momentum'
200         end select
201         shell_prim_num(shell_num) = k
202         shell_center(shell_num) = i
203         prim_index(shell_num) = prim_num
204         do j=1,shell_prim_num(shell_num)
205            read(10,*) buffer, exponent(prim_num), coefficient(prim_num)
206            prim_num = prim_num + 1
207         end do
208         shell_num = shell_num + 1
209      end do
210
211   end do
212
213   close(10)
214
215   info = trexio_write_basis_shell_center(trex_file, shell_center)
216   call check_success(info, 'Unable to write basis shell_center')
217
218   info = trexio_write_basis_shell_ang_mom(trex_file, shell_ang_mom)
219   call check_success(info, 'Unable to write basis shell_ang_mom')
220
221   info = trexio_write_basis_shell_prim_num(trex_file, shell_prim_num)
222   call check_success(info, 'Unable to write basis shell_prim_num')
223
224   info = trexio_write_basis_prim_index(trex_file, prim_index)
```

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

```
225   call check_success(info, 'Unable to write basis prim_index')
226
227   info = trexio_write_basis_exponent(trex_file, exponent)
228   call check_success(info, 'Unable to write basis exponent')
229
230   info = trexio_write_basis_coefficient(trex_file, coefficient)
231   call check_success(info, 'Unable to write basis coefficient')
232
233   return
234
235 10 continue
236   stop 'Unable to find element in basis write file'
237 end subroutine read_basis
238
239 !-------------------------------------------------------------------------------
240
241 program write_example
242   use trexio
243   implicit none
244
245   character*(128)               :: xyz_filename     ! Name of the xyz file
246   character*(128)               :: basis_filename   ! Name of the basis file
247   integer*8                     :: trex_file        ! Handle for the TREX file
248   integer                       :: i
249   integer                       :: info
250   character*(*), parameter      :: trex_filename = 'trex_file'
251
252   ! Get the xyz file name from the command line and user name
253   ! ========================================================
254
255   i = command_argument_count()
256   if (i /= 2) then
257      print *, 'Expected:'
258      print *, ' - xyz file as 1st argument'
259      print *, ' - basis file as 2nd argument'
260      print *, './fortran_write cyanoformaldehyde.xyz cc-pvtz'
261      stop -1
262   end if
263
264   call get_command_argument(1,xyz_filename)
265   call get_command_argument(2,basis_filename)
266
267   ! Open the TREX file
268   ! ------------------
269
270   trex_file = trexio_open(trex_filename,'w', TREXIO_HDF5)
271
272   ! Write the data
273   ! --------------
274
275   call read_xyz(trex_file, xyz_filename)
276   call read_basis(trex_file, basis_filename)
```

```
277
278   ! Close the file
279   ! ---------------
280
281   info = trexio_close(trex_file)
282   call check_success(info, 'Unable to close file')
283
284   print *, 'Wrote file '//trim(trex_filename)
285
286 end program write_example
```

To read back the data and print it, we can use the following program:

```
1  subroutine check_success(info,message)
2    use trexio
3    implicit none
4    integer, intent(in) :: info
5    character*(*)       :: message
6
7    if (info /= TREXIO_SUCCESS) then
8       print *, info, message
9       stop -1
10   end if
11 end subroutine check_success
12
13 !-------------------------------------------------------------------------
14
15 subroutine read_electrons(trex_file, alpha_num, beta_num)
16   use trexio
17   implicit none
18   integer*8, intent(in)          :: trex_file
19   integer*8, intent(out)         :: alpha_num, beta_num
20   integer :: info
21
22   info = trexio_read_electron_up_num(trex_file,alpha_num)
23   call check_success(info, 'Unable to read up electrons')
24
25   info = trexio_read_electron_dn_num(trex_file,beta_num)
26   call check_success(info, 'Unable to read dn electrons')
27 end subroutine read_electrons
28
29 !-------------------------------------------------------------------------
30
31 subroutine read_nuclei(trex_file, nucl_num, nucl_coord, nucl_charge, nucl_label)
32   use trexio
33   implicit none
34   integer*8, intent(in)          :: trex_file
35   integer*8, intent(in)          :: nucl_num
36   double precision, intent(out)  :: nucl_coord(3,nucl_num)
37   double precision, intent(out)  :: nucl_charge(nucl_num)
38   character*(32), intent(out)    :: nucl_label(nucl_num)
```

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

```
39
40    integer :: info
41
42    info = trexio_read_nucleus_coord(trex_file,nucl_coord)
43    call check_success(info, 'Unable to read nuclear coordinates')
44
45    info = trexio_read_nucleus_charge(trex_file,nucl_charge)
46    call check_success(info, 'Unable to read nuclear charges')
47
48    info = trexio_read_nucleus_label(trex_file,nucl_label)
49    call check_success(info, 'Unable to read nuclear labels')
50
51  end subroutine read_nuclei
52
53  !-------------------------------------------------------------------------
54
55  subroutine read_basis(trex_file, shell_num, prim_num, center, ang_mom, &
56       shell_prim_num, prim_index, expo, coef)
57    use trexio
58    implicit none
59    integer*8, intent(in)                :: trex_file
60    integer*8, intent(in)                :: shell_num, prim_num
61    integer*8, intent(out)               :: center(shell_num), shell_prim_num(shell_num)
62    integer  , intent(out)               :: ang_mom(shell_num)
63    integer*8, intent(out)               :: prim_index(shell_num)
64    double precision, intent(out)        :: expo(prim_num)
65    double precision, intent(out)        :: coef(prim_num)
66
67    integer                         :: info
68
69    info = trexio_read_basis_shell_center(trex_file, center)
70    call check_success(info, 'Unable to read basis shell_center')
71
72    info = trexio_read_basis_shell_ang_mom(trex_file, ang_mom)
73    call check_success(info, 'Unable to read basis shell_ang_mom')
74
75    info = trexio_read_basis_shell_prim_num(trex_file, shell_prim_num)
76    call check_success(info, 'Unable to read basis shell_prim_num')
77
78    info = trexio_read_basis_prim_index(trex_file, prim_index)
79    call check_success(info, 'Unable to read basis prim_index')
80
81    info = trexio_read_basis_exponent(trex_file, expo)
82    call check_success(info, 'Unable to read basis exponent')
83
84    info = trexio_read_basis_coefficient(trex_file, coef)
85    call check_success(info, 'Unable to read basis coefficient')
86
87  end subroutine read_basis
88
89  !-------------------------------------------------------------------------
90
```

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

```
program read_example
  use trexio
  implicit none
  integer*8                      :: nucl_num         ! Number of nuclei
  character*(32), allocatable    :: nucl_label(:)    ! Atom labels
  real*8, allocatable            :: nucl_charge(:)   ! Nuclear charges
  real*8, allocatable            :: nucl_coord(:,:)  ! Nuclear coordinates
  integer*8                      :: alpha_num        ! Number of alpha electrons
  integer*8                      :: beta_num         ! Number of beta  electrons

  integer*8                      :: trex_file        ! Handle for the TREX file
  integer                        :: i,j,k
  integer                        :: info
  character*(*), parameter       :: trex_filename = 'trex_file'
  double precision, parameter    :: a0 = 0.52917721067d0

  integer*8                      :: shell_num, prim_num
  integer*8, allocatable         :: shell_center(:)
  integer  , allocatable         :: shell_ang_mom(:)
  integer*8, allocatable         :: shell_prim_num(:)
  integer*8, allocatable         :: prim_index(:)
  double precision, allocatable  :: shell_factor(:)
  double precision, allocatable  :: exponent(:)
  double precision, allocatable  :: coefficient(:)
  character*(32)                 :: bastype
  character*(32)                 :: label
  character, parameter           :: ang_mom(0:6) = (/ 'S', 'P', 'D', 'F', 'G', 'H', 'I' /)

  ! Read the data from the TREX file
  ! ================================

  ! Open the file
  ! -------------

  trex_file = trexio_open(trex_filename,'r',TREXIO_HDF5)

  ! Read the data
  ! -------------

  call read_electrons(trex_file,alpha_num,beta_num)
  print *, 'Electrons: ', alpha_num, ' up, ', beta_num, ' down'


  info = trexio_read_nucleus_num(trex_file,nucl_num)
  call check_success(info, 'Unable to read number of nuclei')

  allocate(nucl_coord(3,nucl_num), &
           nucl_charge(nucl_num), &
           nucl_label(nucl_num) )

  call read_nuclei(trex_file, nucl_num, nucl_coord, nucl_charge, nucl_label)

```

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

```
  do i=1,nucl_num
     print '(A4, 2X, F4.1,3(3X,F12.8))', nucl_label(i), nucl_charge(i), nucl_coord(1:3,i)
  end do


  bastype=''
  info = trexio_read_basis_type(trex_file, bastype)
  call check_success(info, 'Unable to read basis type')
  print *, 'Basis type: ', trim(bastype)

  info = trexio_read_basis_shell_num(trex_file, shell_num)
  call check_success(info, 'Unable to read basis shell_num')

  info = trexio_read_basis_prim_num(trex_file, prim_num)
  call check_success(info, 'Unable to read basis prim_num')

  allocate(shell_center(shell_num),   &
           shell_ang_mom(shell_num),  &
           shell_prim_num(shell_num), &
           prim_index(shell_num),     &
           shell_factor(shell_num),   &
           exponent(prim_num),        &
           coefficient(prim_num))

  call read_basis(trex_file, shell_num, prim_num, shell_center, &
       shell_ang_mom, shell_prim_num, prim_index, exponent, coefficient)

  k=0
  do i=1,shell_num
     if (shell_center(i) /= k) then
        k = shell_center(i)
        info = trexio_element_name_of_symbol(trim(nucl_label(k)),label)
        call check_success(info, 'Unable to read name of element :')
        print *, ''
        print *, trim(label)
     end if
     print *, ang_mom(shell_ang_mom(i)), shell_prim_num(i)
     do j=1,shell_prim_num(i)
        print '(I3,X,E16.8,3X,E16.8)', j, &
             exponent(prim_index(i)+j-1) , coefficient(prim_index(i)+j-1)
     end do
  end do


  ! Close the file
  ! --------------

  info = trexio_close(trex_file)
  call check_success(info, 'Unable to close file')

end program read_example
```

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

## References

[1] E. F. Valeev, "Libint: A library for the evaluation of molecular integrals of many-body operators over gaussian functions," http://libint.valeyev.net/, 2020, version 2.7.0-beta.6.

[2] M. Burkatzki, C. Filippi, and M. Dolg, "Energy-consistent small-core pseudopotentials for 3d-transition metals adapted to quantum Monte Carlo calculations," *J. Chem. Phys.*, vol. 129, no. 16, p. 164115, Oct 2008.

[3] "The Easy Fortran Input/Output (EZFIO) library generator," Mar 2021, [Online; accessed 5. Mar. 2021]. [Online]. Available: https://gitlab.com/scemama/EZFIO

[4] "Lustre file system," Mar 2021, [Online; accessed 5. Mar. 2021]. [Online]. Available: https://www.lustre.org

[5] "GPFS (General Parallel File System) - IBM," Mar 2021, [Online; accessed 5. Mar. 2021]. [Online]. Available: https://researcher.watson.ibm.com/researcher/view_group.php?id=4840

[6] "BeeGFS Documentation v7.2.1 — BeeGFS Documentation v7.2.1," Feb 2021, [Online; accessed 5. Mar. 2021]. [Online]. Available: https://doc.beegfs.io/latest/index.html

[7] The HDF Group. (2000-2010) Hierarchical data format version 5. [Online]. Available: http://www.hdfgroup.org/HDF5

[8] "Basis Set Exchange (BSE)," Mar 2021, [Online; accessed 5. Mar. 2021]. [Online]. Available: https://www.basissetexchange.org

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.