chimes_calculator Documentation

Release 0.0.1

Rebecca K. Lindsey, Nir Goldman, & Laurence E. Fried

Mar 16, 2023

CONTENTS

The **Ch**ebyshev **I**nteraction **M**odel for **E**fficient **S**imulation (ChIMES) is a machine-learned interatomic potential targeting chemistry in condensed phase systems. ChIMES models are able to approach quantum-accuracy through a systematically improvable explicitly many-bodied basis comprised of linear combinations of Chebyshev polynomials. Though originally developed to enable description of organic molecular materials, ChIMES has successfully been applied to systems spanning ambient water to molten carbon, and leveraged as correction for density functional based tight binding simulations.

The ChIMES calculator comprises a flexible tool set for evaluating ChIMES interactions (e.g. in simulations, single point calculations, etc). Users have the option of directly embedding the ChIMES calculator within their codes (e.g. see ''The ChIMES Calculator," for advanced users), or evaluating interactions through the beginner-friendly serial interface, each of which have Python, C++, C, and Fortran API's. Files necessary for linking to popular simulation codes are being continually added with ancillary support. For more information see the links below.

The ChIMES Calculator is developed at Lawrence Livermore National Laboratory with funding from the US Department of Energy (DOE), and is open source, distributed under the terms of the LGPL v3.0 License.

Note: This documentation is under still construction.

CHAPTER

ONE

CHIMES CALCULATOR QUICKSTART GUIDE

For more detailed instructions, see the *Getting Started* page.

1.1 Obtain a copy

1. Create a fork of the code
2. Clone a copy of the code to your computer or high-performance computer (HPC)

1.2 Installing

If your environment is correctly configured, you can install by simply executing `./install.sh`.

If you are on a HPC using module files, you may need to load them first. Module files are already configured for a handful of HPC - inspect the contents of modfiles to see if yours is listed. If it is (e.g., `LLNL-LC.mod`), execute `export hosttype=LLNL-LC; ./install.sh` to install. Otherwise, load the appropriate modules by hand before running the install script.

Note: Consider submitting module files and corresponding install.sh changes as a pull request, for your HPC!

1.3 Running

You can test your installation by running an example job, e.g., by executing the following in your base chimes_calculator directory:

```
serial_interface/examples/cpp/chimescalc \
serial_interface/tests/force_fields/published_params.liqC.2b.cubic.txt \
serial_interface/tests/configurations/liqC.2.5gcc_6000K.OUTCAR_#000.xyz | tee my_test.log
```

CHAPTER

TWO

GETTING STARTED

2.1 Obtaining the code

2.1.1 LLNL Employees:

Please see the special Bitbucket instructions

2.1.2 GitHub Users:

Please see the special Github instructions

2.2 Compiling and running the code

As described above, the `chimes_calculator` comprises library tools for evaluating ChIMES interactions. However, the repository contains several usage examples (see, e.g. *ChIMES Calculator* and *ChIMES Calculator Serial Interface* examples.). These examples can be compiled by navigating to a given example sub directory (e.g. `chimes FF/examples/cpp/`) and typing `make`.

Alternatively, the entire software suite can be compiled at once using CMake, via the install script - note that C++, C, *and* Fortran compilers are all required to use the this approach.

If your environment is correctly configured, you can simply execute `./install.sh`.

If you are on a HPC using module files, you may need to load them first. Module files are already configured for a handful of HPC - inspect the contents of modfiles to see if yours is listed. If it is (e.g., LLNL-LC.mod), execute `export hosttype=LLNL-LC; ./install.sh` to install. Otherwise, load the appropriate modules by hand before running the install script.

Note: Consider submitting module files and corresponding install.sh changes as a PR, for your HPC!

, or executing the appropriate CMake commands by simply running `./install.sh` from the base `chimes_calculator` directory. If the latter option is used, a list of generated executables/library files and their respective install locations can be found in the generated `build/install_manifest.txt` file. Note that C++, C, *and* Fortran compilers are all required to use the `./install.sh` approach.

Sample ChIMES parameter and input files are provided in the `serial_interface/tests/force_fields` and `serial_interface/tests/configurations` directories, allowing compiled executables to be tested via, e.g.:

```
serial_interface/examples/cpp/chimescalc \
serial_interface/tests/force_fields/published_params.liqC.2b.cubic.txt \
serial_interface/tests/configurations/liqC.2.5gcc_6000K.OUTCAR_#000.xyz | tee my_test.log
```

For additional details on using, integrating, and compiling, and contributing, see:

- *The ChIMES Calculator*
- *The ChIMES Calculator Serial Interface*
- *Contributing*

CHAPTER

THREE

CHIMES CALCULATOR RELEASES

- v1.0.2: (Jan. 6 2022) Test suite bug fixes, file renaming, documentation update
- v1.0.1: (Dec. 22, 2021) CMake/Make bugfixes
- v1.0.0: (Dec. 12, 2021) First stable release

CHAPTER

FOUR

THE CHIMES CALCULATOR

4.1 Overview

ChIMES is a reactive explicitly many-bodied machine learned interatomic potential (ML-IAP) for which interactions are computed on the basis of atom clusters. For example, the total ChIMES energy is given as:

$$E_{n_\mathrm{B}} = \sum_{i_1}^{n_a} {}^1E_{i_1} + \sum_{i_1>i_2}^{n_a} {}^2E_{i_1 i_2} + \sum_{i_1>i_2>i_3}^{n_a} {}^3E_{i_1 i_2 i_3} + \cdots + \sum_{i_1>i_2\cdots>i_{n\mathrm{B}-1}>i_{n\mathrm{B}}}^{n_a} {}^nE_{i_1 i_2 \ldots i_n} \tag{4.1}$$

where E_{n_B} is the total ChIMES system energy, n_B is the maximum bodiedness, ${}^nE_{i_1 i_2 \ldots i_n}$ is the n-body ChIMES energy for a given set of n atoms with indices $i = i_1, i_2, \ldots, i_n$, and n_a is the total number of atoms in the system. In the ChIMES framework, single-body energies are constant values and n-body energies are constructed from the product of polynomials of transformed atom pair distances. Thus, a 2-body interaction would involve a single pair, ij, while a three-body interaction would involve three pairs, ij, ik, and jk, a 4-body interaction would involve $\binom{4}{2}$ pairs, and so on. Currently, the ChIMES calculator supports up to 4-body interactions.

For further details of the ChIMES ML-IAP equations, the reader is referred to the following. For a complete set of ChIMES references, see *Citing ChIMES*.

1. R.K. Lindsey*, L.E. Fried, N. Goldman, *J. Chem. Theory Comput.*, **13** 6222 (2017). (link)
2. R.K. Lindsey*, L.E. Fried, N. Goldman, *J. Chem. Theory Comput.* **15** 436 (2019). (link)
3. R.K. Lindsey*, N. Goldman, L.E. Fried, S. Bastea, *J. Chem. Phys.* **153** 054103 (2020). (link)
4. R.K. Lindsey*, L.E. Fried, N. Goldman, S. Bastea, *J. Chem. Phys.* **153** 134117 (2020). (link)

Corresponding authors are indicated with an asterisk (*).

4.2 Sections

4.3 The ChIMES Calculator

The ChIMES Calculator source files are located in `chimesFF/src`. To use in a C++ code, simply `#include "chimescalc.h"` in the target code and instantiate a `chimesFF` object. As described in greater detail below, `chimesFF` objects take information on individual atom clusters and provide the corresponding ChIMES energy, stress tensor, and forces. Any such code must at least include the following operations, in order:

```
int my_rank = 0;
chimesFF my_chimesFF_object;      // Instantiate
my_chimesFF_object.init(my_rank); // Set MPI rank (replace with zero if used␣
↪in serial code)
my_chimesFF_object.read_parameters("my_parameter_file");
```

Note that the ChIMES calculator `chimesFF` class provides users with the following functions:

<table>
<tr><th>Return Type</th><th>Name</th><th>Arguments and Description</th></tr>
<tr><td>void</td><td>init</td><td><table><tr><th>Type</th><th>Description</th></tr><tr><td>int</td><td>MPI rank</td></tr></table>Set the MPI rank. With the exception of error messages, the ChIMES calculator will only print output for rank 0.</td></tr>
<tr><td>void</td><td>read_parameters</td><td><table><tr><th>Type</th><th>Description</th></tr><tr><td>string</td><td>Parameter file</td></tr></table>Read the chimes parameter file.</td></tr>
<tr><td>void</td><td>set_atomtypes</td><td><table><tr><th>Type</th><th>Description</th></tr><tr><td>vector<string></td><td>List of atom types defined by parameter file (updated by function)</td></tr></table>Update the input vector with atom types in the parameter file.</td></tr>
<tr><td>double</td><td>max_cutoff_2B</td><td><table><tr><th>Type</th><th>Description</th></tr><tr><td>bool</td><td>Flag: If true, prints largest 2-body cutoff</td></tr></table>Returns the maximum 2-body outer cutoff distance.</td></tr>
<tr><td>double</td><td>max_cutoff_3B</td><td><table><tr><th>Type</th><th>Description</th></tr><tr><td>bool</td><td>Flag: If true, prints largest 3-body cutoff</td></tr></table>Returns the maximum 3-body outer cutoff distance.</td></tr>
<tr><td>double</td><td>max_cutoff_4B</td><td><table><tr><th>Type</th><th>Description</th></tr><tr><td>bool</td><td>Flag: If true, prints largest 4-body cutoff</td></tr></table>Returns the maximum 4-body outer cutoff distance.</td></tr>
<tr><td>void</td><td>compute_1B</td><td><table><tr><th>Type</th><th>Description</th></tr><tr><td>int</td><td>Atom type index</td></tr><tr><td>double</td><td>Energy (updated)</td></tr></table>Update energy with the single atom contribution.</td></tr>
<tr><td>void</td><td>compute_2B</td><td><table><tr><th>Type</th><th>Description</th></tr><tr><td>double</td><td>Distance between two atoms, i and j</td></tr><tr><td>vector<double></td><td>Distance vector components for each atom</td></tr><tr><td>vec-</td><td>Type indices for atoms i</td></tr></table></td></tr>
</table>

4.3.1 The C API

The C API (`chimescalc_C*`) is located in `chimesFF/api`. This wrapper provides C style name mangling and creates a set of C-style wrapper functions. The latter are needed for compatibility with std::vector which is heavily used in `chimesFF` and not supported in most other languages. Any C code attempting to use the ChIMES Calculator should `#include "chimescalc_C.h"` and at least include the following operations, in order:

```
int my_rank = 0;
set_chimes();          // Instantiate
init_chimes(my_rank); // Set MPI rank (replace with zero if used in serial␣
↪code)
chimes_read_params("my_parameter_file");
```

For additional information on compiling, see *Implementation Examples*.

Note that the ChIMES calculator `chimescalc_C` API provides users with the following functions:

<table>
<tr><th>Return Type</th><th>Name</th><th>Arguments and Description</th></tr>
<tr><td>void</td><td>set_chimes</td><td>No arguments. Instantiates a pointer to a <code>chimesFF</code> object.</td></tr>
<tr><td>void</td><td>init_chimes</td><td><table><tr><th>Type</th><th>Description</th></tr><tr><td>int</td><td>MPI rank</td></tr></table>Set the MPI rank. With the exception of error messages, the ChIMES calculator will only print output for rank 0.</td></tr>
<tr><td>void</td><td>chimes_read_params</td><td><table><tr><th>Type</th><th>Description</th></tr><tr><td>char*</td><td>Parameter file</td></tr></table>Read the chimes parameter file.</td></tr>
<tr><td>int</td><td>get_chimes_2b_order</td><td>No arguments. Returns the two body order set by the parameter file.</td></tr>
<tr><td>int</td><td>get_chimes_3b_order</td><td>No arguments. Returns the three body order set by the parameter file.</td></tr>
<tr><td>int</td><td>get_chimes_4b_order</td><td>No arguments. Returns the four body order set by the parameter file.</td></tr>
<tr><td>double</td><td>get_chimes_max_2b_cutoff</td><td>No arguments. Returns the two body maximum outer cutoff set by the parameter file.</td></tr>
<tr><td>double</td><td>get_chimes_max_3b_cutoff</td><td>No arguments. Returns the three body maximum outer cutoff set by the parameter file.</td></tr>
<tr><td>double</td><td>get_chimes_max_4b_cutoff</td><td>No arguments. Returns the four body maximum outer cutoff set by the parameter file.</td></tr>
<tr><td>void</td><td>chimes_compute_2b_props</td><td><table><tr><th>Type</th><th>Description</th></tr><tr><td>double</td><td>Distance between two atoms, i and j</td></tr><tr><td>double array</td><td>Distance vector components for each atom</td></tr><tr><td>char* array</td><td>Atom types for atoms i and j</td></tr><tr><td>double array</td><td>Forces for atoms i and j ([atom index (out of 2)][component index (i.e. fx=0, fy=1, fz=3)]) (contents updated by function)</td></tr><tr><td>double array</td><td>Stress tensor ([s_xx, s_xy, s_xz, s_yx, s_yy, s_yz, s_zx, s_zy, s_zz]) (contents updated by function)</td></tr><tr><td>double*</td><td>Energy (updated by function)</td></tr></table></td></tr>
<tr><td>4.3. The ChIMES Calculator</td><td></td><td>Update the force, stress tensor, and energy with the two-atom contribution.</td></tr>
<tr><td>void</td><td>chimes_compute_3b_props</td><td></td></tr>
</table>

4.3.2 The Fortran API

The Fortran API (`chimescalc_F.f90`) is located in `chimesFF/api`. This wrapper enables access to `chimesFF` functions through the C API and handles other details like differences in array storage order.

Any Fortran code attempting to use the ChIMES Calculator should `use chimescalc` and at least include the following operations, in order:

```
integer(C_int) :: my_rank
call f_set_chimes()          ! Instantiate
call f_init_chimes(my_rank) ! Set MPI rank (replace with zero if used in␣
↪serial code)
call f_chimes_read_params(string2Cstring("my_parameter_file"))
```

For additional information on compiling, see *Implementation Examples*.

Note that the ChIMES calculator `chimescalc_F` API provides users with the following functions:

<table>
<tr><th>Return Type</th><th>Name</th><th>Arguments and Description</th></tr>
<tr><td>none</td><td>f_chimes_compute_2b_props_fromf90</td><td>
<table>
<tr><th>Type</th><th>Description</th></tr>
<tr><td>C_double</td><td>Distance between two atoms, i and j</td></tr>
<tr><td>C_double array</td><td>Distance vector components for each atom</td></tr>
<tr><td>C_char</td><td>Type for atom i</td></tr>
<tr><td>C_char</td><td>Type for atom j</td></tr>
<tr><td>C_double array</td><td>Forces for atoms i and j ([atom index (out of 2)][component index (i.e. fx=0, fy=1, fz=3)]) (contents updated by function)</td></tr>
<tr><td>C_double array</td><td>Stress tensor ([s_xx, s_xy, s_xz, s_yx, s_yy, s_yz, s_zx, s_zy, s_zz]) (contents updated by function)</td></tr>
<tr><td>C_double</td><td>Energy (updated by function)</td></tr>
</table>
Update the force, stress tensor, and energy with the two-atom contribution.
</td></tr>
<tr><td>none</td><td>f_chimes_compute_3b_props_fromf90</td><td>
<table>
<tr><th>Type</th><th>Description</th></tr>
<tr><td>C_double array</td><td>Distances between three atoms, ij, ik, and jk</td></tr>
<tr><td>C_double array</td><td>Distance vector components for each atom</td></tr>
<tr><td>C_char</td><td>Type for atom i</td></tr>
<tr><td>C_char</td><td>Type for atom j</td></tr>
<tr><td>C_char</td><td>Type for atom k</td></tr>
<tr><td>C_double array</td><td>Forces for atoms i, j, and k ([atom index (out of 3)][component index (i.e. fx=0, fy=1, fz=3)]) (contents updated by function)</td></tr>
<tr><td>C_double array</td><td>Stress tensor ([s_xx, s_xy, s_xz, s_yx, s_yy, s_yz, s_zx, s_zy, s_zz]) (contents updated by function)</td></tr>
<tr><td>C_double</td><td>Energy (updated by function)</td></tr>
</table>
Update the force, stress tensor, and energy with the three-atom contribution.
</td></tr>
<tr><td>none</td><td>f_chimes_compute_4b_props_fromf90</td><td>
<table>
<tr><th>Type</th><th>Description</th></tr>
<tr><td>C_double array</td><td>Distances between four atoms, ij, ik, il, jk, jl, and</td></tr>
<tr><td>C_double array</td><td>Distance vector components for each atom</td></tr>
</table>
</td></tr>
</table>

4.3.3 The Python API

The Python API (`chimescalc_py*`) is located in `chimesFF/api`. Like the Fortran API, this wrapper enables access to `chimesFF` functions through the C API, via ctypes.

Any python code attempting to use the ChIMES Calculator should `import chimescalc_py` and at least include the following operations, in order:

```
chimescalc_py.chimes_wrapper = chimescalc_py.init_chimes_wrapper("chimescalc_
↪dl.so") # Associate the wrapper with a compiled C API library file
chimescalc_py.set_chimes()  # Instantiate
chimescalc_py.init_chimes() # If run with MPI, an integer MPI rank can be␣
↪passed to this function. By default, assumes rank = 0
chimescalc_py.read_params("my_parameter_file")
```

For additional information on compiling (i.e. generation of `chimescalc_dl.so`), see *Implementation Examples*.

Note that the ChIMES calculator `chimescalc_py` API provides users with the following functions:

<table>
<tr><th>Return Type</th><th>Name</th><th>Arguments and Description</th></tr>
<tr><td>ctypes</td><td>init_chimes_wrapper</td><td><table><tr><th>Type</th><th>Description</th></tr><tr><td>str</td><td>C-wrapper library name (i.e. “lib-C_wrapper-serial_interface.so”)</td></tr></table></td></tr>
<tr><td>none</td><td>set_chimes</td><td>No arguments. Instantiates a pointer to a <code>chimesFF</code> object.</td></tr>
<tr><td>none</td><td>init_chimes</td><td><table><tr><th>Type</th><th>Description</th></tr><tr><td>int</td><td>MPI rank (optional parameter)</td></tr></table>Set the MPI rank. With the exception of error messages, the ChIMES calculator will only print output for rank 0.</td></tr>
<tr><td>none</td><td>read_params</td><td><table><tr><th>Type</th><th>Description</th></tr><tr><td>str</td><td>Parameter file</td></tr></table></td></tr>
<tr><td>float</td><td>get_chimes_max_2b_cutoff</td><td>No arguments. Returns the two body order set by the parameter file.</td></tr>
<tr><td>float</td><td>get_chimes_max_2b_cutoff</td><td>No arguments. Returns the three body order set by the parameter file.</td></tr>
<tr><td>float</td><td>get_chimes_max_2b_cutoff</td><td>No arguments. Returns the four body order set by the parameter file.</td></tr>
<tr><td>int</td><td>get_chimes_2b_order</td><td>No arguments. Returns the two body maximum outer cutoff.</td></tr>
<tr><td>int</td><td>get_chimes_3b_order</td><td>No arguments. Returns the three body maximum outer cutoff.</td></tr>
<tr><td>int</td><td>get_chimes_4b_order</td><td>No arguments. Returns the four body maximum outer cutoff.</td></tr>
<tr><td>none</td><td>chimes_compute_2b_props</td><td><table><tr><th>Type</th><th>Description</th></tr><tr><td>float</td><td>Distances between atoms i and j</td></tr><tr><td>float list</td><td>Distance vector components for each atom</td></tr><tr><td>str list</td><td>Types for atom i and j</td></tr><tr><td>float list</td><td>Forces for atoms i, and j ([atom index (out of 2)][component index (i.e. fx=0, fy=1, fz=3)]) (contents updated by function)</td></tr><tr><td>float list</td><td>Stress tensor ([s_xx, s_xy, s_xz, s_yx, s_yy, s_yz, s_zx, s_zy, s_zz]) (contents updated by function)</td></tr><tr><td>float</td><td>Energy (updated by function)</td></tr></table>Update the force, stress tensor, and energy with the two-atom contribution.</td></tr>
</table>

4.3.4 Implementation Examples

The following codes demonstrates how `chimesFF.{h,cpp}` can be used to obtain the overall stress tensor, energy, and per-atom forces for a given system configuration using C, C++ Fortran, and Python. See the `main.*` files in each corresponding subdirectory of `chimesFF/examples` for further implementation details. Note that sample system configurations (i.e. `*xyz` files) and parameter files can be found in `serial_interface/test/configurations` and `serial_interface/test/force_fields`, respectively. For user generated tests, note that `*.xyz` files must provide lattice vectors in the comment line, e.g. lx 0.0 0.0 0.0 ly 0.0 0.0 0.0 lz. Click *here* for an overview of ChIMES units.

Note: All implementation examples are intended to be run on Unix-based systems (e.g. Linux, OSX).

Warning: These codes are for demonstrative purposes only and come with no guarantees.

Note: All example executables can be compiled at once in `./build` with CMake, via `./install.sh` from the `chimes_calculator` base directory, and similarly uninstalled via `./uninstall.sh`. However, the examples below compile via the user-generated Makefiles located in each `examples` subdirectory, for demonstrative purposes.

- **C Example:** The `main` function of this example includes the C API, `chimescalc_C.{h,cpp}`, which creates a global static pointer to a `chimesFF` object. The `chimesFF` pointer object is set up, i.e. by `set_chimes()`, and used for access to `chimesFF` member functions, etc.
  - Navigate to `chimesFF/examples/c`
  - Compile with: `make all`
  - Test with: `./chimescalc-test_direct-C <parameter file> <xyz file>`
  - Additional notes:
    - `*.xyz` files must not contain any information beyond atom type and x-, y-, and z- coordinate on coordinate lines.
    - This implementation does NOT use ghost atoms/layering thus the input system MUST have box lengths greater than two times the largest outer cutoff, or results will not be correct.
- **C++ Example:** The `main` function of this example creates an instance of `serial_chimes_interface` (i.e. a class inheriting `chimesFF`, which computes energy, per-atom forces, and stress tensor for an overall system). For additional details, see *The ChIMES Calculator Serial Interface*
  - Navigate to `chimesFF/examples/cpp`
  - Compile with: `make all`
  - Test with: `./chimescalc <parameter file> <xyz file>`
- **Fortran Example:** Similar to the C example, this `main` function establishes a pointer to a `chimesFF` object via `f_set_chimes()`. The `f_set_chimes()` function call is defined in `chimescalc_F.f90`, a wrapper for the C API `chimescalc_C.cpp` (i.e which facilitates C-style access to `chimesFF` member functions, etc). Actual linking is achieved at compilation. See the `Makefile` for details.
  - Navigate to `chimesFF/examples/fortran`
  - Compile with: `make all`

  - Test with: `./chimescalc-test_direct-F <parameter file> <xyz file>`
  - Additional notes:
    * `*.xyz` files must not contain any information beyond atom type and x-, y-, and z- coordinate on coordinate lines.
    * This implementation does NOT use ghost atoms/layering thus the input system MUST have box lengths greater than two times the largest outer cutoff, or results will not be correct.

- **Python Example:** This example accesses `chimesFF` functions through `chimescalc_py.py`, a ctypes-based python API for access to the C API functions (i.e. through `chimescalc_C.cpp`). Once `chimescalc_py.py` is imported, it is associated with a compiled C API library file, i.e. `chimescalc_dl.so` and can be used to access `chimesFF` member functions.
  - Navigate to `chimesFF/examples/python`
  - Compile `chimescalc_dl.so` with: `make all`
  - Test with: `python main.py <parameter file> <coordinate file>`
  - Additional notes:
    * Requires `chimescalc_dl.so` in the same directory, which is generated via `make all`
    * Expects to be run with Python version 3.X

Warning: This Python implementation example does NOT use ghost atoms/layering thus the input system MUST have box lengths greater than two times the largest outer cutoff, or results will not be correct.

CHAPTER

FIVE

THE CHIMES CALCULATOR SERIAL INTERFACE

5.1 Overview

The ChIMES calculator serial interface provides an easier means of evaluating ChIMES interactions for a given system. In constrast to the ChIMES calculator (i.e. `chimesFF`), which takes information on *individual* atom clusters and returns the cluster energy, stress tensor, via `compute_xB` functions, the serial interface (i.e. `serial_chimes_interface`) takes *overall* system information and returns *overall* system energy, stress tensor, and forces. Though far less flexible than direct use of `chimesFF`, `serial_chimes_interface` allows users to leverage ChIMES with much less coding. For further details on `chimesFF`, see *The ChIMES Calculator*. For a complete set of ChIMES references, see *Citing ChIMES*. Note that this functionality is primarily intended for instructive purposes, and is not recommended for large scale simulations.

5.2 The ChIMES Calculator Serial Interface

The ChIMES calculator serial interface source files are located in `serial_interface/src/`. To use in a C++ code, simply `#include "serial_chimes_interface.h"` in the target code and instantiate a `serial_chimes_interface` object. As described in greater detail below, `serial_chimes_interface` objects take information on the overall system and provide the corresponding ChIMES energy, stress tensor, and forces. Any such code must initialize the calculation the with following operations, in order:

```
int my_rank = 0;
// Instantiate
serial_chimes_interface chimes;
// Specify the parameter files and set  the MPI rank (replace with zero if␣
↪used in serial code)
chimes.init_chimesFF("my_parameter_file", my_rank);
```

Warning:

For small simulation cells (e.g., a single atom in a face-centered cubic unit cell), the ChIMES calculator must be instantiated via `serial_chimes_interface chimes(true)`. This allows for automatic replication in situations where the ChIMES outer cutoff is greater than one half of the smallest supercell length. Please note that use of extra-small simulation cells is ill-advised for aything except crystalline systems and should be used with caution.

Developer note: To recover behavior of the research code, instantiate with: `serial_chimes_interface chimes(false)`.

Please see the following example of interfacing a C++ code with the ChIMES calculator: `serial_interface/examples/cpp/main.cpp`. Note that the ChIMES calculator `serial_chimes_interface` class provides users with the following functions:

<table>
<tr><th>Return Type</th><th>Name</th><th>Arguments and Description</th></tr>
<tr><td>void</td><td>init_chimesFF</td><td>
<table>
<tr><th>Type</th><th>Description</th></tr>
<tr><td>string</td><td>Parameter file</td></tr>
<tr><td>int</td><td>MPI rank</td></tr>
</table>
Instantiates <code>serial_chimes_interface</code> object, sets rank, reads parameter file. With the exception of error messages, the ChIMES calculator will only print output for rank 0.
</td></tr>
<tr><td>void</td><td>calculate</td><td>
<table>
<tr><th>Type</th><th>Description</th></tr>
<tr><td>vector<double></td><td>Vector of x-coordinates for system atoms</td></tr>
<tr><td>vector<double></td><td>Vector of y-coordinates for system atoms</td></tr>
<tr><td>vector<double></td><td>Vector of z-coordinates for system atoms</td></tr>
<tr><td>vector<double></td><td>System cell a lattice vector</td></tr>
<tr><td>vector<double></td><td>System cell b lattice vector</td></tr>
<tr><td>vector<double></td><td>System cell c lattice vector</td></tr>
<tr><td>vector<string></td><td>Vector of atom types for system atoms</td></tr>
<tr><td>double</td><td>Overall system energy (updated by function)</td></tr>
<tr><td>vector<vector<double> ></td><td>Vector of forces for system atoms (updated by function); ([atom index][fx, fy, fz])</td></tr>
<tr><td>vector<double></td><td>System stress tensor (updated by function); ([s_xx, s_xy, s_xz, s_yx, s_yy, s_yz, s_zx, s_zy, s_zz])</td></tr>
</table>
Takes system coordinates and cell lattice vectors, computes corresponding ChIMES energy, stress tensor, and system forces.
</td></tr>
</table>

5.2.1 The C API

The C API (`chimescalc_serial_C*`) is located in `serial_interface/api`. This wrapper provides C style name mangling and creates a set of C-style wrapper functions. The latter are needed for compatibility with `std::vector` which is heavily used in `serial_chimes_interface` and not supported in most other languages. Any C code attempting to use the ChIMES calculator serial interface should `#include "chimescalc_serial_C.h"` and initialize calculations with the following operations, in order:

```
int my_rank = 0;
set_chimes_serial();         // Instantiate; as for the C++ API (see warning␣
↪message), can pass 0/1 for false/true for small cells
init_chimes_serial("my_parameter_file", my_rank); // Set MPI rank (replace␣
↪with zero if used in serial code)
```

Please see the following example of interfacing a C code with the ChIMES calculator: `serial_interface/examples/c/main.c`. For additional information on compiling, see *Implementation Examples*.

Note that the ChIMES calculator serial interface `chimescalc_serial_C` API provides users with the following functions:

<table>
<tr><th>Return Type</th><th>Name</th><th>Arguments and Description</th></tr>
<tr><td>void</td><td>set_chimes_serial</td><td>Creates a pointer to a <code>serial_chimes_interface</code> object.

<table><tr><th>Type</th><th>Description</th></tr><tr><td>int</td><td>Boolean: Allow for small cell replication? (0/1 for false/true); default = true</td></tr></table></td></tr>
<tr><td>void</td><td>init_chimes_serial</td><td><table><tr><th>Type</th><th>Description</th></tr><tr><td>string</td><td>Parameter file</td></tr><tr><td>int</td><td>MPI rank</td></tr></table>
Sets rank and reads the parameter file to the <code>serial_chimes_interface</code> object. With the exception of error messages, the ChIMES calculator will only print output for rank 0.</td></tr>
<tr><td>void</td><td>calculate_chimes</td><td><table><tr><th>Type</th><th>Description</th></tr><tr><td>int</td><td>number of atoms in system</td></tr><tr><td>double array</td><td>Vector of x-coordinates for system atoms</td></tr><tr><td>double array</td><td>Vector of y-coordinates for system atoms</td></tr><tr><td>double array</td><td>Vector of z-coordinates for system atoms</td></tr><tr><td>char array</td><td>System cell a lattice vector</td></tr><tr><td>double array</td><td>System cell b lattice vector</td></tr><tr><td>double array</td><td>System cell c lattice vector</td></tr><tr><td>double array</td><td>Vector of atom types for system atoms</td></tr><tr><td>double*</td><td>Overall system energy (updated by function)</td></tr><tr><td>double array</td><td>Vector of forces for system atoms (updated by function); ([atom index][fx, fy, fz])</td></tr></table></td></tr>
</table>

5.2.2 The Fortran90 API

The Fortran90 API (`chimescalc_serial_F.f90`) is located in `serial_interface/api`. This wrapper enables access to `serial_chimes_interface` functions through the C API and handles other details like differences in array storage order.

Any Fortran90 code attempting to use the ChIMES Calculator should `use chimescalc_serial` and at least include the following operations, in order:

```
integer(C_int) :: my_rank
! Instantiate; as for the C++ API (see warning message), can pass 0/1 for␣
↪false/true for small cells
call f_set_chimes()
! Specify the parameter files and set  the MPI rank (replace with zero if used␣
↪in serial code)
call f_init_chimes(string2Cstring("my_parameter_file"), my_rank)
```

Please see the following example of interfacing a Fortran90 code with the ChIMES calculator: `serial_interface/examples/fortran/main.F90`. For additional information on compiling, see *Implementation Examples*.

Note that the ChIMES calculator serial interface `chimescalc_serial_F` API provides users with the following functions:

<table>
<tr><th>Return Type</th><th>Name</th><th>Arguments and Description</th></tr>
<tr><td>none</td><td>f_set_chimes</td><td>Creates a pointer to a <code>serial_chimes_interface</code> object.

<table><tr><th>Type</th><th>Description</th></tr><tr><td>C_int</td><td>Boolean: Allow replication? (0/1 for false/true); default = true</td></tr></table></td></tr>
<tr><td>none</td><td>f_init_chimes</td><td><table><tr><th>Type</th><th>Description</th></tr><tr><td>C_char</td><td>Parameter file</td></tr><tr><td>C_int</td><td>MPI rank</td></tr></table>
Sets rank and reads the parameter file to the <code>serial_chimes_interface</code> object. With the exception of error messages, the ChIMES calculator will only print output for rank 0.</td></tr>
<tr><td>void</td><td>f_calculate_chimes</td><td><table><tr><th>Type</th><th>Description</th></tr><tr><td>C_int</td><td>number of atoms in system</td></tr><tr><td>C_double array</td><td>Vector of x-coordinates for system atoms</td></tr><tr><td>C_double array</td><td>Vector of y-coordinates for system atoms</td></tr><tr><td>C_double array</td><td>Vector of z-coordinates for system atoms</td></tr><tr><td>C_char array</td><td>System cell a lattice vector</td></tr><tr><td>C_double array</td><td>System cell b lattice vector</td></tr><tr><td>C_double array</td><td>System cell c lattice vector</td></tr><tr><td>C_double array</td><td>Vector of atom types for system atoms</td></tr><tr><td>C_double *</td><td>Overall system energy (updated by function)</td></tr><tr><td>C_double array</td><td>Vector of forces for system atoms (updated by function); ([atom index][fx, fy, fz])</td></tr><tr><td>C_double array</td><td>System stress tensor (updated by function); ([s_xx, s_xy, s_xz, s_yx, s_yy, s_yz, s_zx, s_zy, s_zz])</td></tr></table></td></tr>
</table>

5.2.3 The Fortran2008 API

The Fortran2008 API (`chimescalc_serial_F08.f90`) is located in `serial_interface/api`. This wrapper enables access to `serial_chimes_interface` functions through the C API and handles other details like differences in array storage order.

Any Fortran2008 code attempting to use the ChIMES Calculator should `use chimescalc_serial08, only : ChimesCalc, ChimesCalc_init` and at least include the following operations, in order:

```
! declare ChIMES object
type(ChimesCalc) :: chimes
! Initialize ChIMES calculator
! Note: ``param_file`` is the user-defined ChIMES parameter file, ``my_rank`` is␣
↪the MPI process rank (zero for a serial process), and ``small`` is set to 0/1␣
↪for false/true for small cells
call ChimesCalc_init(chimes, trim(param_file), my_rank, small)
! Set atom typesi for C++ interface, stored in the array atom_types in this␣
↪example.
call chimes%set_atom_types(atom_types)
! Get ChIMES contributions
call chimes%calculate(coords, latvecs, energy, forces, stress)
```

Please see the following example of interfacing a Fortran2008 code with the ChIMES calculator: `serial_interface/examples/fortran08/main.F90`.For additional information on compiling, see *Implementation Examples*.

Note that the ChIMES calculator serial interface `chimescalc_serial_F08` API provides users with the following functions:

<table>
<tr><th>Code Type</th><th>Name</th><th>Arguments and Description</th></tr>
<tr><td>subroutine</td><td>ChimesCalc_init</td><td>Creates a pointer to a <code>serial_chimes_interface</code> object through function calls to the Fortran90 API module.

<table>
<tr><th>Type</th><th>Description</th></tr>
<tr><td>ChimesCalc</td><td>Initialized chimes calculator instance on exit</td></tr>
<tr><td>character(*)</td><td>Name of the parameter file to use for the initialization</td></tr>
<tr><td>integer</td><td>MPI process rank</td></tr>
<tr><td>integer</td><td>Set to 0/1 for false/true for small cells</td></tr>
</table></td></tr>
<tr><td>subroutine</td><td><ChimesCalc>%set_atom_types</td><td>Converts Fortran char array to C/C++ string array.

<table>
<tr><th>Type</th><th>Description</th></tr>
<tr><td>character(*)</td><td>Fortran array of atom types. Subroutine converts to C/C++ string arrays.</td></tr>
</table></td></tr>
<tr><td>subroutine</td><td><ChimesCalc>%calculate</td><td>Performs ChIMES calculation based on simulation cell inputs

<table>
<tr><th>Type</th><th>Description</th></tr>
<tr><td>double precision</td><td>2D array of atomic coordinates with shape of (3,n_atom)</td></tr>
<tr><td>double precision</td><td>Lattice vectors. Shape: [3, 3], first index runs over x,y,z, second over lattice vectors.</td></tr>
<tr><td>double precision</td><td>Variable which should be increased by the ChIMES energy.</td></tr>
<tr><td>double precision</td><td>Forces, which ChIMES contribution should be added to. Shape: [3, nr_of_atoms].</td></tr>
<tr><td>double precision</td><td>Stress tensor, which the ChIMES contribution should be added to. Shape: [3, 3].</td></tr>
</table></td></tr>
</table>

5.2.4 The Python API

The Python API (`chimescalc_serial_py.py`) is located in `serial_interface/api`. Like the Fortran API, this wrapper enables access to `serial_chimes_interface` functions through the C API, via ctypes.

Any python code attempting to use the ChIMES Calculator should `import chimescalc_serial_py` and at least include the following operations, in order:

```
# Associate the wrapper with a compiled C API library file
chimescalc_serial_py.chimes_wrapper = chimescalc_serial_py.init_chimes_wrapper(
↪"libchimescalc_dl.so")
# Instantiate; as for the C++ API (see warning message), can pass 0/1 for␣
↪false/true
chimescalc_serial_py.set_chimes()
# Read the parameter file, set MPI rank to 0 (i.e. no MPI used)
chimescalc_serial_py.init_chimes("my_parameter_file", 0)
```

For additional information on compiling (i.e. generation of `lib-C_wrapper-serial_interface.so`), see *Implementation Examples*.

Note that the ChIMES calculator serial interface `chimescalc_serial_py` API provides users with the following functions:

<table>
<tr><th>Return Type</th><th>Name</th><th>Arguments and Description</th></tr>
<tr><td>See description</td><td>init_chimes_wrapper</td><td>
<table>
<tr><th>Type</th><th>Description</th></tr>
<tr><td>string</td><td>Library name</td></tr>
</table>
Associate ctypes.CDLL (i.e. the return type) with a the compiled ChIMES calculator serial interface C-library.
</td></tr>
<tr><td>void</td><td>set_chimes</td><td>Creates a pointer to a <code>serial_chimes_interface</code> object.
<table>
<tr><th>Type</th><th>Description</th></tr>
<tr><td>bool</td><td>Allow replication? ; default = true</td></tr>
</table>
</td></tr>
<tr><td>void</td><td>init_chimes</td><td>
<table>
<tr><th>Type</th><th>Description</th></tr>
<tr><td>string</td><td>Parameter file</td></tr>
<tr><td>int</td><td>MPI rank</td></tr>
</table>
Sets rank and reads the parameter file to the <code>serial_chimes_interface</code> object. With the exception of error messages, the ChIMES calculator will only print output for rank 0.
</td></tr>
<tr><td>See description</td><td>calculate_chimes</td><td>
<table>
<tr><th>Type (input)</th><th>Description</th></tr>
<tr><td>int</td><td>number of atoms in system</td></tr>
<tr><td>float list</td><td>Vector of x-coordinates for system atoms</td></tr>
<tr><td>float list</td><td>Vector of y-coordinates for system atoms</td></tr>
<tr><td>float list</td><td>Vector of z-coordinates for system atoms</td></tr>
<tr><td>str list</td><td>System cell a lattice vector</td></tr>
<tr><td>float list</td><td>System cell b lattice vector</td></tr>
<tr><td>float list</td><td>System cell c lattice vector</td></tr>
<tr><td>float list</td><td>Vector of atom types for system atoms</td></tr>
<tr><td>float</td><td>Overall system energy</td></tr>
<tr><td>float list</td><td>Vector of forces for system atoms ([atom index][fx, fy, fz])</td></tr>
<tr><td>float list</td><td>System stress tensor ([s_xx, s_xy, s_xz, s_yx, s_yy, s_yz, s_zx, s_zy, s_zz])</td></tr>
</table>
Takes system coordinates and cell
</td></tr>
</table>

5.2.5 Implementation Examples

The following codes demonstrates how `serial_chimes_interface.{h,cpp}` can be used to obtain the overall stress tensor, energy, and per-atom forces for a given system configuration using C, C++ Fortran, and Python. See the `main.*` files in each corresponding subdirectory of `serial_interface/examples` for further implementation details. Note that sample system configurations (i.e. `*xyz` files) and parameter files can be found in `serial_interface/test/configurations` and `serial_interface/test/force_fields`, respectively. For user generated tests, note that `*.xyz` files must provide lattice vectors in the comment line, e.g. lx 0.0 0.0 0.0 ly 0.0 0.0 0.0 lz. Click *here* for an overview of ChIMES units.

Note: All implementation examples are intended to be run on Unix-based systems (e.g. Linux, OSX).

Warning: These codes are for demonstrative purposes only and come with no guarantees.

Note: All example executables can be compiled at once in `./build` with CMake, via `./install.sh` from the `chimes_calculator` base directory, and similarly uninstalled via `./uninstall.sh`. However, the examples below compile via the user-generated Makefiles located in each `examples` subdirectory, for demonstrative purposes.

- **C Example:** The `main` function of this example includes the C API, `chimescalc_serial_C.{h,cpp}`, which creates a global static pointer to a `serial_chimes_interface` object. The `serial_chimes_interface` pointer object is set up, i.e. by `set_chimes_serial()`, and used for access to `serial_chimes_interface` member functions, etc.
  - Navigate to `serial_interface/examples/c`
  - Compile with: `make all`
  - Test with: `./chimescalc-test_serial-C <parameter file> <xyz file>`
- **C++ Example:** The `main` function of this example creates an instance of `serial_chimes_interface` (i.e. a class inheriting `chimesFF`, which computes energy, per-atom forces, and stress tensor for an overall system). For additional details, see *The ChIMES Calculator*
  - Navigate to `serial_interface/examples/cpp`
  - Compile with: `make all`
  - Test with: `./chimescalc <parameter file> <xyz file>`
- **Fortran90 Example:** Similar to the C example, this `main` function establishes a pointer to a `serial_chimes_interface` object via `f_set_chimes()`. The `f_set_chimes()` function call is defined in `chimescalc_serial_F.F90`, a wrapper for the C API `chimescalc_serial_C.cpp` (i.e which facilitates C-style access to `serial_chimes_interface` member functions, etc). Actual linking is achieved at compilation. See the `Makefile` for details.
  - Navigate to `serial_interface/examples/fortran`
  - Compile with: `make all`
  - Test with: `./chimescalc-test_serial-F <parameter file> <xyz file>`
  - Additional notes:

- **Fortran2008 Example:** Similarly, this `main` function establishes a pointer to a `serial_chimes_interface` object via calls to `ChimesCalc_init()` and subroutine calls within the `ChimesCalc` class, defined in `chimescalc_serial_F08.f90`. Subroutines called from the Fortran2008 API act as an interface for the wrapper functions establied in the Fortran90 API. Actual linking is achieved at compilation. See the `Makefile` for details.
  - Navigate to `serial_interface/examples/fortran08`
  - Compile with: `make all`
  - Test with: `./chimescalc-test_serial-F08 <parameter file> <xyz file>`
  - Additional notes:
- **Python Example:** This example accesses `serial_chimes_interface` functions through `chimescalc_serial_py.py`, a ctypes-based python API for access to the C API functions (i.e. through `chimescalc_serial_C.cpp`). Once `chimescalc_serial_py.py` is imported, it is associated with a compiled C API library file, i.e. `lib-C_wrapper-serial_interface.so` and can be used to access `serial_chimes_interface` member functions.
  - Navigate to `serial_interface/examples/python`
  - Compile `libchimescalc-serial_dl.so` with: `make all`
  - Rename: `cp libchimescalc-serial_dl.so libchimescalc_dl.so`
  - Test with: `python main.py <parameter file> <coordinate file>`

CHAPTER

SIX

SUPPORT FOR LINKING WITH EXTERNAL CODES

6.1 Using the ChIMES Calculator with LAMMPS

We are currently working toward ChIMES calculator implementation in LAMMPS as a USER package. In the interim, the following provides a guide to implementing the ChIMES calculator as a LAMMPS pairstyle.

6.1.1 Quick start

Provided a system with a C++11-compatible compiler and an MPI compatible compiler are available, LAMMPS can be downloaded, installed, linked to ChIMES, and compiled all at once by navigating to `etc/lmp`, adding Intel compilers to your path and executing `./install.sh`. Once complete, the installation can be tested by navigating to `etc/lmp/tests` and running the example via `../exe/lmp_mpi_chimes -i in.lammps`.

As with installation of the ChIMES Calculator itself, if you are on a HPC using module files, you may need to load them first. Module files are already configured for a handful of HPC - inspect the contents of modfiles to see if yours is listed. If it is (e.g., LLNL-LC.mod), execute `export hosttype=LLNL-LC; ./install.sh` to install. Otherwise, load the appropriate modules by hand before running the install script.

Note that Intel oneapi compilers (which are now free) can be used to properly configure your enviroment for all Intel capabilities (e.g., icc, mpiicpc, mkl, etc.) - simply locate and execute the setvars.sh script within your Intel installation.

6.1.2 Detailed Compilation Overview

Note: This example assumes users have downloaded the 29 Oct 2020 release of LAMMPS (stable version as of 10/29/20), which can be downloaded here.

To integrate the ChIMES calculator in LAMMPS, locate the following files, and place them in the following destination among the LAMMPS source code:

File	Location	Destination	Description
chimesFF.{h,cpp}	chimesFF/src	src/MANYBODY	ChIMES calculator files
pair_chimes.{h,cpp}	etc/lmp/src	src/MANYBODY	ChIMES pair_style definition files
pair.{h,cpp}	etc/lmp/etc	src	Updated LAMMPS pair files (new ev_tally definition added)
Makefile.mpi_chimes	etc/lmp/etc	src/MAKE	Makefile for compiling with ChIMES support

Following, compile from the base LAMMPS directory with:

```
make yes-manybody
make mpi_chimes
```

Note that a successful compilation should produce an executable named `lmp_mpi_chimes`.

Tip: If you are using an intel compiler, either delete the `pair_list.*` files that appear in the src folder following the `make yes-manybody` command, or add `-restrict` to `CCFLAGS` in `MAKE/Makefile.mpi_chimes`. Note that the presently provided `Makefile.mpi_chimes` utilizes the latter approach.

6.1.3 Running

To run a simulation using ChIMES parameters, a block like the following is needed in the main LAMMPS input file (i.e. `in.lammps`):

```
pair_style  chimesFF
pair_coeff  * *   some_standard_chimes_parameter_file.txt
```

Note that the following must also be set in the main LAMMPS input file, to use ChIMES:

```
units       real
newton      on
atom_style  atomic
atom_modify sort 0 0.0
```

Warning:

1. Implementation assumes outer cutoffs for (n+1)-body interactions are always $\leq$ those for n-body interactions
2. This capability is still under testing - please let us know if you observe strange behavior
3. Assumes user wants single-atom energies to be added to the system energy. If you don't want to, zero the energy offsets in the parameter file

CHAPTER

SEVEN

CHIMES PARAMETER FILES

ChIMES parameter files are stored in `serial_interface/tests/force_fields`. A complete list of available force fields and corresponding references can be found on the *Citing ChIMES page*. These parameter files come with no guarantees and should only be used for system compositions and thermodynamic ranges indicated at the top of each file. If you are interested in developing a ChIMES model for a new material or range of conditions, please contact us via our Google group.

CHAPTER

EIGHT

CHIMES UNITS

ChIMES uses the following base units:

Property	Unit
Distance	Angstroms (“Ang”)
Energy	kcal mol $^{-1}$
Stress	kcal mol $^{-1}$ Ang $^{-3}$
Force	kcal mol $^{-1}$ Ang $^{-1}$

CHAPTER

NINE

CHIMES CALCULATOR UTILITIES

9.1 The PES Generator

9.1.1 Input

A utility for generating ChIMES potential energy surface scans for n-body clusters is available in `utils/pes_generator`. To use this utility, create a file name `config.py` in the desired working directory, structured as follows:

```
CHMS_REPO  = "/path/to/your/chimes_calculator/repository/"

PARAM_FILE = "/path/to/your/chimes_calculator/repository/serial_interface/tests/force_
↪fields/test_params.CHON.txt"

PAIRTYPES  = [0,    3,    5   ] # Pair type index for scans, i.e. number after "PAIRTYPE␣
↪PARAMS:" in parameter file
PAIRSTART  = [1.0,  1.0,  1.0 ] # Smallest distance for scan
PAIRSTOP   = [4.0,  4.0,  4.0 ] # Largest distance for scan
PAIRSTEP   = [0.01, 0.01, 0.01] # Step size for scan

TRIPTYPES  = [1,    4   ] # Triplet type index for scans, i.e. number after "TRIPLETTYPE␣
↪PARAMS:" in parameter file
TRIPSTART  = [1.0,  1.0 ] # Smallest distance for scan
TRIPSTOP   = [4.0,  4.0 ] # Largest distance for scan
TRIPSTEP   = [0.10, 0.10] # Step size for scan

# The example parameter file doesn't contain four body interactions, so the following is␣
↪not needed.
# If four body scans are desired, keep in mind a small step size will take a long time␣
↪to run
# Start with something very large to get a handle on run time, and modify from there
#
#QUADTYPES  = [7   ] # Triplet type index for scans, i.e. number after "TRIPLETTYPE␣
↪PARAMS:" in parameter file
#QUADSTART  = [1.0 ] # Smallest distance for scan
#QUADSTOP   = [4.0 ] # Largest distance for scan
#QUADSTEP   = [1.00] # Step size for scan
```

Variables `CHMS_REPO` and `PARAM_FILE` specify the `chimes_calculator` repository location, and path to the ChIMES parameter of file. Note that paths should be provided in their absolute form. Following these variables, three sets of options are provided. Focusing on options beginning with `PAIR`, one must provide the following:

- A list of pair type indices for which scans should be generated
  - Indices should correspond to values following `PAIRTYPE PARAMS:` in the target parameter file
- A list of the minimum pair distance for each pair type to consider during the scan
- A list of the maximum pair distance for each pair type to consider during the scan
- A scan step size

All input and output distances are in Angstroms, and all energies are provided in kcal/mol. Additionally, note that the penalty function will be included in scan results unless `PAIRSTART` is greater than the sum of the pair interaction inner cutoff and the penalty kick-in distance, or if the user has set `PAIR CHEBYSHEV PENALTY SCALING:` to zero in the parameter file. Similar variables must be set to specify desired 3- and 4-body scans. Note that empty lists can be provided if no scan is desired.

9.1.2 Output

All *n*-body scans will produce output scan files named like `chimes_scan_<n>b.type_<index>.dat`, where <n> is the bodiedness, and <index> is the `PAIRTYPES`, `TRIPTYPES`, or `QUADTYPES` index. Many-body scans will produce additional files named like `chimes_scan_2+3b.type_<index>.dat` or `chimes_scan_2+3+4b.type_<index>.dat`, which include contributions from lower-bodied interactions as well.

The first line in each output file provides a comment (prepended by a #) starting and stopping distances followed by the scan step size. Following, each line provides the *ij* (and if appropriate, *ik*, *il*, *jk*, *jl*, and *kl* distances, respectively) and the corresponding cluster energy. For example, consider the `test_params.CHON.txt` parameter file provided in `serial_interface/tests/force_fields/`, which contains the following 3-body interaction:

```
TRIPLETTYPE PARAMS:
   INDEX: 5 ATOMS: C H O
   PAIRS: CH CO HO UNIQUE: 54 TOTAL: 54
     index  |  powers  |  equiv index  |  param index  |       parameter
   ------------------------------------------------------------------------------
       0       0  1  1         0               0        6.500656496400314
       1       0  1  2         1               1        3.7493801790331345
       2       0  1  3         2               2        0.0
       3       0  2  1         3               3        -4.7147262741975711
       4       0  2  2         4               4        -2.0557295465375991
       5       0  2  3         5               5        -1.1723283559758126
```

In the above example, `TRIPTYPES` is 5, corresponding to i, j, and k atoms of type C, H, and O, respectively. Thus, lines in the corresponding resulting 3-body scan file would give the *ij* (C-H), *ik* (C-O), and *jk* (H-O) distances, followed by the corresponding cluster energy.

9.1.3 Visualizing

Two-body scans can be immediately be plotted by most software (e.g. matplotlib, xmgrace, etc.), however additional considerations are needed to plot the > 3 dimensional 3- and/or 4-body scans. Three body scans can visualized in slices. An additional utility is provided in utils/pes_generator (i.e. `gnuplotify`), which can be used to extract these slices in a gnuplot splot-friendly format. To use this script, the user must specify a 3-body scan file and a *ij* distance at which to make the slice. Note that the *ij* distance must be one listed in the 3-body scan file. For the `test_params.CHON.txt` and `config.py` example above, this can be achieved with:

```
python3.X gnuplotlify.py chimes_scan_2+3b.type_0.dat 2.5
```

This command will produce a file named like `chimes_scan_2+3b.type_0.dat.gnuplot.2.5` that can be plotted in gnuplot via:

```
splot 'chimes_scan_2+3b.type_0.dat.gnuplot.2.5' u 1:2:3 w pm3d
```

CHAPTER

TEN

CITING CHIMES

10.1 Reference Key for Methods/Applications

Key definitions are given *below*.

Method	Reference Key
2+3-body ChIMES	1. Carbon-1
ChIMES+DFTB	1. PuH-DFTB 2. DNTF-DFTB 3. TiH-DFTB 4. QMD-DFTB
Iterative Refinement	1. CO-1
Carbon Condensation	1. CO-1
2+3+4-body ChIMES	1. CO-2
Distributed LASSO	1. CO-2
Active Learning	1. CO-2
ChIMES+MSST	1. HN-1 2. DNTF-DFTB

10.2 Reference Key for Parameter Sets

Parameter set and key name are interchangeable. Key definitions are given *below*.

KEY	Material	Bodiedness	T (K)/ ρ (gcc) Range	Comments
Carbon-1	Molten Carbon	2	5000/2.43	N/A
Carbon-1	Molten Carbon	2	5000/2.43	N/A
Carbon-1	Molten Carbon	2+3	5000/2.43	N/A
Carbon-1	Molten Carbon	2+3	6000/2.25-3.00	N/A
Water-1	Water	2+3	298/1.00	N/A
PuH-DFTB	Pu/H	2+3	0-300/N/A	DFTB E_{rep}
CO-1	Carbon Monoxide (1:1)	2+3	6500-9350/2.5	N/A
CO-2	Carbon Monoxide (1:1)	2+3+4	2400/1.79	N/A
HN-1	Hydrazoic Acid H/N	2+3+4	300-4500/1-2	N/A
DNTF-DFTB	3,4-bis(4-nitrofurazan-3-yl)furoxan	2+3	300-9000/1.86-3.4	DFTB correction, Not applicable to other atom type ratios
TiH-DFTB	Ti/H	2+3	N/A/5.5	DFTB E_{rep}
QMD-DFTB	C/N/O/H (based on QM database)	2+3	0/ambient	DFTB correction

10.3 Reference Key Definitions

Corresponding authors are indicated with an asterisk (*).

Key	Link	Definition
Carbon-1	(link)	R.K. Lindsey*, L.E. Fried, N. Goldman, *J. Chem. Theory Comput.*, **13** 6222 (2017).
PuH-DFTB	(link)	N. Goldman*, B. Aradi, R.K. Lindsey, L.E. Fried, *J. Chem. Theory Comput.* **14** 2652 (2018).
Water-1	(link)	R.K. Lindsey*, L.E. Fried, N. Goldman, *J. Chem. Theory Comput.* **15** 436 (2019).
CO-1	(link)	R.K. Lindsey*, N. Goldman, L.E. Fried, S. Bastea, *J. Chem. Phys.* **153** 054103 (2020).
CO-2	(link)	R.K. Lindsey*, L.E. Fried, N. Goldman, S. Bastea, *J. Chem. Phys.* **153** 134117 (2020).
COND-1	(link)	M.R. Armstrong*, R.K. Lindsey*, N. Goldman, M.H. Nielsen, E. Stavrou, L.E. Fried, J.M. Zaug, S. Bastea*, *Nat, Commun.* **11** 353 (2020).
HN-1	(link)	H. Pham*, R.K. Lindsey, L.E. Fried, N. Goldman, *J. Chem. Phys.* **153** 224102 (2020).
DNTF-DFTB	(link)	R.K. Lindsey*, S. Bastea*, N. Goldman, L. Fried, *J. Chem. Phys.* **154** 164115 (2021).
TiH-DFTB	(link)	N. Goldman*, K. Kweon, R.K. Lindsey, L.E. Fried, T.W. Heo, B. Sadigh, P. Soderlind, A. Landa, A. Perron, J. Jeffries, B. Wood, *J. Chem. Theory Comput.* **17** 4435 (2021).
QMD-DFTB	(link)	C.H. Pham*, R.K. Lindsey, L.E. Fried, N. Goldman, *J. Phys. Chem. Lett.* **13** 2934 (2022).

CHAPTER

ELEVEN

CONTRIBUTING TO CHIMES

The ChIMES calculator is an open source project, and we welcome contributions, e.g. bug fixes, updates to the documentation, extensions, etc.

Contributions are made through the fork/pull request (PR) mechanism and generally, PRs should start from and target the develop branch. Additionally, PRs should include an attached test suite log file (see below).

11.1 Running the test suite

To run the ChIMES calculator tests, simply navigate to `serial_interface/tests/` an run `./run_tests.sh | tee run_tests.log`.

Note: The `run_tests.sh` shell script assumes that a binary named `python3.7` exists in the users $PATH. If it does not exist, users can set the `PYTH3` variable near the top of `run_tests.sh`

Tip: The above command (i.e. `./run_tests.sh | tee run_tests.log`) should be used generating a test suite log file for a PR, but if one desires quickers tests for debugging purposes, the test suite can be run as `./run_tests.sh SHORT | tee run_tests.log`, which reduces the number of test calculations by a factor of roughly ten.

For additional questions and concerns, we can be contacted through our Google group.

CHAPTER

TWELVE

LEGAL

GNU LESSER GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

   As used herein, "this License" refers to version 3 of the GNU Lesser General Public License, and the "GNU GPL" refers to version 3 of the GNU General Public License.

   "The Library" refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

   An "Application" is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

   A "Combined Work" is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the "Linked Version".

   The "Minimal Corresponding Source" for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

   The "Corresponding Application Code" for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

   You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

   If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

   a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or

b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

   The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

   a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.

   b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

   You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

   a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.

   b) Accompany the Combined Work with a copy of the GNU GPL and this license document.

   c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.

   d) Do one of the following:

      0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.

      1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.

   e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

   You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

   a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.

b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

CHAPTER

THIRTEEN

CONTACT

We can be contacted via our Google group.

This work was produced under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.