

Manual for Schmidt Group Monte Carlo Code(s)

Authors: Jesse McDaniel, Kuang Yu

Last updated January 2015 by Jesse McDaniel

Table of Contents:

General Monte Carlo Code (Page 2):

Section 1: Outline of source code

Section 2: Notes for compiling and running

Section 3: List of examples

Section 4: Solute and Crystal configuration input files (*.conf, *.fconf)

Section 5: Force field input file (*.pmt)

Section 6: Simulation parameters input file (simulation_parameters.pmt)

Lattice Model Code (Page 20):

Section 1: Constructing the lattice model

Section 2: Coarse grained solute-solute interactions

Section 3: Lattice Monte Carlo simulation

Section 4: Using the scripts

Section 5: Examples

General Monte Carlo Code:

This code is a general Monte Carlo program that can perform simulations in standard ensembles (nvt, npt, uvt, etc.). It can also be used for Molecular Dynamics simulations (currently, nve only). The primary motivation for writing this code is that we needed a Monte Carlo program that was compatible with our symmetry-adapted perturbation theory (SAPT) based force-fields.¹⁻⁵ As these force fields utilize Drude oscillators to explicitly model many-body polarization (see Ref 5 for a complete description of the force fields), the efficient implementation of Drude oscillators was a central goal of the code. Here we briefly list the important capabilities (as well as limitations) of this code.

Capabilities:

- Supports nvt, npt, uvt (grand-canonical), egc (expanded grand-canonical) ensembles for Monte Carlo simulations. Supports nve Molecular Dynamics as well.
- Supports standard force fields (Lennard-Jones) as well as our SAPT-based force fields (When utilizing our SAPT-based force fields, an explicit energy decomposition is available with the trajectory)
- Supports fast and accurate particle-mesh Ewald (PME) treatment of long-range electrostatic interactions (Also supports PME treatment of solute-framework *dispersion* interactions in GCMC simulations)
- Supports collective hybrid MD/MC moves, as well as standard single molecule moves
- Supports Drude oscillators
- Supports three-body, Axilrod-Teller dispersion interactions
- Supports Feynman-Hibbs effective potentials for treatment of nuclear quantum effects

Limitations (that could be remedied with further development)

- All molecules are treated as rigid bodies. Molecule flexibility could be easily implemented in hybrid MD/MC moves as forces are already coded in.
- Currently, there are no thermostats/barostats for Molecular Dynamics module.

Section 1: Outline of Source Code

This program is written in Fortran 90, and besides the source code files listed below, it requires MKL libraries (for fast-Fourier transforms) and Open-MP libraries (for threading) for successful compilation (see Section 2). The source code contains modules that range from being very general for molecular simulation programs, such as energy (total_energy_forces.f90, pme.f90, pair_int.f90, electrostatics.f90) and sampling (sampling.f90, rigid_body_kinematics.f90) routines, to modules that are very specific for our force field implementation (sapt_ff_routines.f90, penetration_ff_routines.f90). However, one can get a pretty good idea of how the code is structured by just focusing on the general modules such as (in structure-flow order) main_mc.f90, sampling.f90, and total_energy_forces.f90.

As a primary use of this code is to run Monte Carlo simulations employing our SAPT-based force fields, hybrid MD/MC type moves are most often used for these simulations as such simulations employ Drude oscillators. This is because single molecule MC moves are very inefficient, as one cannot use an order(N) energy calculation to find energy differences between two such configurations for a non-pairwise additive potential. The hybrid MD/MC moves serve to create new molecular configurations by moving all molecules simultaneously, guided by the forces on each molecule, to generate new trial configurations which are accepted or rejected using the standard Metropolis criteria. At each trial configuration, the Drude oscillators on all solute molecules are optimized self-consistently to their minimum energy configurations.

Source files:

main_mc.f90 :: This contains the main program call, which controls the loop over MC/MD moves.

read_simulation_parameters.f90 :: This reads the simulation_parameters.pmt file

mc_routines.f90 :: The module in this file controls everything to do with the initialization of the program.

sapt_ff_routines.f90 :: The module in this file is used specifically to initialize the use of our “decomposed”, SAPT-based force fields.

general_routines.f90 :: This file contains a scattered collection of subroutines that are commonly used by many other modules.

sampling.f90 :: The module in this file controls the generation and acceptance of all types of Monte Carlo moves (and controls MD integration).

rigid_body_kinematics.f90 :: This file contains routines designed to integrate Newton's equations for arbitrary rigid bodies.

eq_drude.f90 :: This file contains a subroutine to self-consistently optimize Drude-oscillator positions using a conjugate gradient algorithm.

pme.f90 :: This file contains subroutines to carry out Particle-mesh Ewald electrostatics, as well as a PME dispersion treatment of solute-framework interactions for GCMC adsorption simulations.

electrostatics.f90 :: The subroutines in this file include electrostatic cutoff routines, and old Ewald routines that were used for debugging. Note that most likely the electrostatics will be treated using Particle-mesh Ewald, and therefore most of the time the electrostatic subroutines in this file will not be used.

pair_int.f90 :: The module in this file controls the calculation of all non-electrostatic, pairwise-additive energy and force terms for either a Buckingham or Lennard-Jones potential.

explicit_three_body_interaction.f90 :: This file contains routines that calculate explicit three-body interactions, such as Axilrod-Teller like three-body dispersion energies.

total_energy_forces.f90 :: The subroutine here outputs the total energy and forces of the system.

insertion_bias_routines.f90 :: This file contains subroutines for orientation-bias and energy/cavity-bias molecule insertion/deletion moves.

expanded_grand_canonical_routines.f90 :: The subroutines in here are related to the use of the expanded grand canonical ensemble.

penetration_ff_routines.f90 :: These subroutines are used for piecewise force fields, in which pairwise interactions are switched between the standard "ZIF FF" functional form, and a single repulsive term at close distances.

glob_v.f90 :: This module contains global variables for the program. One can look at this file to see (change?) the values of parameters which have been hard-coded in for the convenience of condensing input files. The size of many data arrays are set through the parameters "MAX_N_MOLE" and "MAX_N_ATOM" which are found in this module. One can lower these values to free up memory, or increase them if more molecules or atoms are needed than allowed.

Section 2: Notes for compiling and running

A Makefile is included with the source code, and this can be used to compile the code after a couple paths are changed. In particular, the code uses the discrete Fourier transform routines from the Intel math kernel (MKL) library. The required links are listed in the “LDFLAGS” variable, but the user must set the “MKLLIB” to the appropriate path where these libraries can be found. In addition, the code uses the open MP algorithms (OMP) for shared-memory parallelization. Therefore the user must also set the “OMPINCLUDE” variable to the appropriate path where the open MP libraries can be found. After these paths are set, executing the command “make” in the source directory should compile the code and produce the executable “main_mc”.

Concerning memory requirements, most of the global data arrays are not allocatable, and the size of most of these arrays is governed by the parameters “MAX_N_MOLE” and “MAX_N_ATOM”, with values hard coded in the file glob_v.f90. Depending on the size of the systems that are studied, the user may want to change the values of these parameters before compiling the code. MAX_N_MOLE should be set to a value larger than the total number of molecules in the system (which may be somewhat undetermined in a GCMC simulation), and MAX_N_ATOM should be set to a value larger than the maximum number of atoms per molecule. Because this code is threaded, setting these values too large may require explicitly setting the stack size on the compute node before running the program.

Thus it may be necessary to explicitly set the variable OMP_STACKSIZE before running the code. A command such as “export OMP_STACKSIZE=20M” will do this. If this variable is not explicitly set, a “segmentation fault” might occur when trying to run the code.

The code can then be run (assuming the executable and all required input files are in the current directory) using the command

```
./main_mc ifile.conf [ifile.fconf] ifile.pmt simulation_parameters.pmt ofile1 ofile2
```

Where “ifile.conf” is the solute configuration input file, “ifile.fconf” is the [optional] crystal framework configuration input file used in a nvt/ugt gas adsorption simulation, “ifile.pmt” is the force field parameters input file, “simulation_parameters.pmt” is the simulation parameters input file, “ofile1” is the first output file containing configurations of the trajectory, and “ofile2” is the second output file containing run information as well as energetics of the trajectory.

Section 3: List of examples.

****Note: In all of the examples below, the “name.out” output file containing the trajectory xyz-coordinates has been removed to limit the file size of the download****

a) ZIF-68 (“ZIF FF”)/CO₂ (“SYM”) simulation at 298K, .4bar.

This is an example calculation of a GCMC simulation for CO₂ in ZIF-68 at 298K, .4bar. The “SYM” force field is used to describe CO₂/CO₂ interactions, and the “ZIF FF” force field is used to describe CO₂/ZIF-68 interactions. The solute (CO₂) configuration input file is named “co2.conf”, and it contains 62 CO₂ molecules. Note that the “box vectors” in “co2.conf” are unimportant, as these will always be taken from the “*.fconf” (in this case, “zif68_relabel_2x2x2.fconf”) crystal configuration input file. The only time the box vectors will be taken from the solute configuration file is if a neat fluid simulation is carried out. The crystal configuration file, “zif68_relabel_2x2x2.fconf”, contains the configurations for a 2x2x2 super cell, and the atoms have been labeled to correspond to the atomtypes in the force field, “co2_zif68.pmt”, file. This super cell consists of 4800 atoms, and the code will treat the framework atoms individually when calculating interactions within the cutoff range. ZIF-68 has a “non-cubic” unit cell, and therefore at the bottom of the “zif68_relabel_2x2x2.fconf” file, three vectors are given describing the box edges in Cartesian coordinates.

The force field file, “co2_zif68.pmt”, begins with the “solute_species” section, which lists the unique atom types for all solute molecules (in this case just C and O), and their force field parameters for *solute-solute* interactions. These are the “SYM” force field parameters. As we are using different solute parameters for *solute-framework* interactions, these are given under the next section labeled “solute parameters for framework cross terms”. These are the “ZIF FF” parameters for carbon and oxygen of CO₂. The next section titled “solute dhf cross terms” is to give the “SYM” C-O Adhf parameter as this can’t be generated using a combination rule. The next section titled “solute-solute exponents” gives the “SYM” exponents for (in order) C-C, C-O, and O-O interactions, as the C-O exponent can’t be generated using a combination rule. The final section, titled “framework_species”, gives the “ZIF FF” parameters for all atom types in the ZIF-68 crystal. Note that the atomtype labels are consistent with those in the “zif68_relabel_2x2x2.fconf” crystal configuration file. Note also that charges are given, which have been derived by fitting point charges to a distributed multipole analysis (DMA), (see Ref 3 for details), and charges are available for nine different ZIFs in the supporting information of reference 4.

The “simulation_parameters.pmt” file contains general simulation parameters, and the options are described in detail in section 6. Note that the chemical potential given in this file, as described in section 6, must not contain any rotational/vibrational contribution. The script

"co2_chempot_eos.pl" (located in the /useful_scripts/ directory) can be used to generate the chemical potential for CO₂ as a function of temperature and pressure using the Peng Robinson equation of state.

This example job can be submitted and run using the script "submit.sh". The first output file for this job, "zif68_298K_4bar.out", contains the trajectory of solute molecules during the simulation, with snapshots printed in order for every "n_output" (found in simulation_parameters.pmt) number of MC moves. The second output file, "zif68_298K_4bar.out1", contains simulation parameter information, along with a brief (non-comprehensive) list of force-field parameter information. After these sections, energy information is printed for the corresponding snapshots (of the first output file) from the trajectory. Because we are using our "SAPT-based" force fields, we have an explicit energy decomposition for each of these snapshots, and this is accessible in this output file. Also printed is the kinetic energy, number of solute molecules, volume, and long range dispersion correction (which is always zero for a crystal simulation). A trajectory of the density of gas molecules can be generated using the script "density.pl" (`./density.pl zif68_298K_4bar.out1`). Note that the total potential energy reported in the "zif68_298K_4bar.out1" file is the total potential energy of all solute molecules in the unit (super) cell. The only crystal-crystal interactions that are calculated during the simulation (reciprocal space pme electrostatics) have been subtracted out, so that the potential reported is entirely the potential energy of the solute molecules.

b) ZIF-69 ("ZIF FF")/N₂ ("SYM") simulation at 298K, .6bar.

This is very similar to the example above, only this is a simulation of N₂ uptake in ZIF-69, again using the "SYM" force field for solute-solute interactions, and "ZIF FF" for solute-framework interactions. The important thing to notice here is that the exponents in the force field parameter file "N2_zif69.pmt" are the ZIF FF exponents scaled by a factor of 0.98, as detailed in reference 4. The chemical potential for N₂ as a function of temperature and pressure can be calculated with the script "N2_chempot_eos.pl" using the Peng Robinson equation of state.

c) ZIF-8 (UFF)/CO₂ ("EPM2") simulation at 303K, 1bar.

Again, this example is very similar to the previous two, but in this case a Lennard-Jones force field is used with the UFF parameters describing the ZIF-8 framework atoms, and the EPM2 force field (J. G. Harris and K. H. Yung, JPC 1995, 99, 12021-12024) is used for the CO₂ molecules (notice the change in bond length of CO₂ molecules in the co2.conf file!). The format of the force field parameter "*.pmt" file is similar, but simpler, as the only necessary

parameters are charge, epsilon, and sigma values (with some additional zero's entered to make the parameter reading subroutines happy). Also notice the changes in the "simulation_parameters.pmt" file, which are consistent with the LJ force field and UFF parameters. In the "*.out1" output file, the energy is now only explicitly divided into two components, a contribution from point charges ("Electro_static"), and everything else (listed under "Buckingham", sorry Lennard-Jones).

d) MOF-5 ("ZIF FF") / CO₂ ("SYM") simulation at 298K, 15bar.

This example illustrates the use of an "exact" PME treatment of long range van-der-Waals interactions (between the crystal and solute molecules), to limit the errors introduced when employing a cutoff. While long-range corrections are routinely used in isotropic systems to remedy errors introduced by cutoffs, there is no such satisfactory correction for a non-isotropic, crystalline system. By using an Ewald sum for these interactions, no such errors are introduced. To use the PME treatment for dispersion, the parameter "pme_disp" is set to "yes" in the simulation_parameters.pmt input file.

e) Neat Acetone npt simulation with 3-body dispersion.

Here, we illustrate how to run a neat fluid simulation in the npt ensemble, using our SAPT-based force fields with 3-body Axilrod-Teller dispersion interactions. The parameter input file is similar to before, with the addition of a three body dispersion parameter section, and the format for this section is commented on in the input file. There are a few changes to be aware of in the simulation_parameters.pmt file. First, the parameter "damp_solute_solute_dispersion" is set to "yes", as should always be done for interactions that employ a $C_6/R^6 + C_8/R^8 + C_{10}/R^{10} + \dots$ treatment of dispersion. This will employ a Tang-Toennies damping function for all such interactions. The reason that this parameter was set to "no" in the previous examples is that for CO₂-CO₂ and N₂-N₂ interactions, we were using the "SYM" forcefield, which employs an "effective" C_6/R^6 dispersion interaction, and is thus undamped. Second, a parameter called "three_body_dispersion" is included and is set to "yes". In addition, there are parameters specific to the npt ensemble that control volume steps (pressure, vol_step, cycles_per_vol_step, max_vol_step, target_acceptance_v). Since we are simulating a neat fluid, we can employ a long-range correction to the cutoff van-der Waals interaction, and this is done by setting dispersion_lrc = 1 (and hence we use a shorter cutoff of $l_j_cutoff = 12$). The last settings that we mention are for the 3-body dispersion evaluation; parameters "na_nslist", "nb_nslist", and "nc_nslist" are grid sizes for the cell-list algorithm used for looping over 3-body interactions, and three_body_dispersion_cutoff determines the cutoff for these interactions.

f) Neat Acetone OPLS npt simulation.

Here we illustrate how to run a simple (rigid-body) OPLS npt simulation of neat acetone. These input files should be simple and self-explanatory. The only thing one would have to change for any other standard Lennard-Jones force field is the combination rule setting in `simulation_parameters.pmt` (`lj_comb_rule = "opls"` uses the opls combination rule which is a geometric mean for both epsilon and sigma, `lj_comb_rule = "standard"` uses Lorentz-Berthelot combination rules). This simulation was run at STP, and one can compare the density and enthalpy of vaporization from this simulation (0.794 g/cm^3 and 7.32 kcal/mol respectively) to the values reported in the original OPLS paper- JACS, 118, (1996), 11225-11236 (0.795 g/cm^3 and 7.24 kcal/mol). The minor differences are due to the fact that we have used a rigid model for acetone, whereas the reference work uses a flexible model with OPLS bonded parameters.

Section 4: Solute and Crystal configuration input files (`*.conf`, `*.fconf`)

a) `"*.conf"` file

The `*.conf` file contains the initial configuration of the system. For a neat fluid simulation, this should therefore contain coordinates of all the molecules in the system. The reason why we refer to this as the "solute" configuration file, is that for the case of a GCMC gas adsorption simulation in a porous material, the system is separated into "solute" and "framework" components. This file would then contain the initial coordinates for the solute only. For a GCMC simulation, while the initial number of solute molecules can vary, at least one molecule must be present in this file, as the molecular geometry and atom types are determined by the labels and configurations. Also for a simulation in a crystal framework, care must be taken to provide an initial configuration where there is no "overlap" between solute molecules and the framework atoms.

The format of this file is as follows:

First line (unformatted): Number of total solute atoms in this initial configuration.

Next (# solute atoms) lines: Configurations of molecules, this is FORMATTED INPUT! The format for these lines must be (in Fortran 90 syntax) `'(I5,2A5,I5,3F14.6)'`, corresponding to the molecule number, molecule name, atomtype, # of atom entry, and x, y, z Cartesian coordinates (given in Angstroms) respectively. Note this is very similar to GROMACS `.gro` file formats, with the main difference being coordinates are given in Angstroms rather than nanometers and no velocities are given.

Last line(s)(unformatted): This line(s) gives the box vectors (in Angstroms). Note that these box vectors are only used in a neat fluid simulation; if a “*.fconf” file is used, the box vectors from that file are used in the simulation. The box vectors can be entered in two different ways. For a rectangular box, only the lengths of the sides are needed, and these can be entered on the same line, and read in as “length1 length2 length3”. If this is the only line present, then this input format is assumed, and the three box vectors are taken to be orthogonal. For a non-orthogonal box, (or, if you want, for an orthogonal box), the cartesian coordinates of the three box vectors are given explicitly one after the other, on separate lines. So the first line should contain Ax, Ay, Az; the second line Bx, By, Bz; and the third line Cx, Cy, Cz for box vectors A, B, C.

b) “*.fconf” file

This file is used for a simulation of a solute in a fixed crystal framework. For neat fluid simulations, this file is omitted. The format of this file is as follows:

First line: Number of crystal framework atoms

Next (# atoms) lines: These give the coordinates of all atoms in the crystal. These lines are read in as atomtype, x, y, z Cartesian coordinates (in Angstrom). These lines are not read in by an explicit fixed format.

*(As an option, charges can explicitly be read in for each atom from the *.fconf file. In this case, the previous lines will be read in as atomtype, x, y, z Cartesian coordinates, charge. Whether or not charges are being read will automatically be detected by the code based on the number of arguments given for these lines. If charges are read in here, they will overwrite any charges read in from the *.pmt file)*

Last line(s) :: The last line(s) give the unit (or super) cell vectors for the crystal. These vectors are read in the same way as in the “*.conf” file (see above).

Section 5: Force field input file (*.pmt)

The file has many possible formats, due to the fact that there originally was a lot of experimentation with force field functional forms and parameters. Unless a simple Lennard-Jones type force field is being used, the subroutines that read this file are all contained in “sapt_ff_routines.f90”. The best way to generate an appropriate “*.pmt” file is probably to use

one of the example files as a template. Here, we will briefly describe the main aspects of this file.

Units:

Charges are in a.u., all other units are in a combination of energy units of kJ/mol and distance units of Angstrom. So for a Lennard Jones force field, epsilon should be given in kJ/mol, and sigma should be given in Angstroms. For our SAPT force fields, the various exponential prefactors (A_{exch} , A_{elec} , A_{ind} , etc.) should be given in kJ/mol, the exponents, "B", should be in inverse Angstroms, and C6, C8, C10, C12 coefficients should be in kJ/mol*Å⁶, kJ/mol*Å⁸, kJ/mol*Å¹⁰, kJ/mol*Å¹² respectively. Polarizabilities of solute atoms should be in Angstrom³.

General Structure:

The .pmt file is divided into sections, and the number of sections and format of each section depend on the type of simulation and the force field employed.

a) "solute_species" section.

This section is required for all types of simulations, and it is the only section required for neat fluid simulations. This section is (and must) be labeled with the keyword "solute_species". After this heading line, a comment line must be included, followed by a line with the number of unique atom types for all the solute molecules. After this, every solute atom type is listed followed by the force field parameters (the format of which depends on the type of force field used, see below).

b) "framework_species" section.

The phrase "framework_species" is needed for recognition of this section. This section is used only for a simulation of adsorbates in a porous crystalline framework. After the heading line, a comment line must be included, followed by a line with the number of atom types for the crystal framework. After this, every framework atom type is listed, followed by the force field parameters.

Specific Force Fields:

Specific force fields require specific formats for entering the force field parameters as well as possibly extra sections in the "*.pmt" file. Here we discuss these specifics, according to the type of force field

SAPT Force fields-SYM CO₂/N₂ model: While the SYM force fields for CO₂ and N₂ require a somewhat “specialized” input section, these input sections are clearly illustrated in the provided examples.

a) “solute_species” section: The format for the force field input lines is as follows: atomtype label (must match that used in “*.conf” file), charge, A_{exch} , A_{elec} , A_{ind} , A_{dhf} , A_{disp} (zero), C6,C8,C10, (C8 and C10 zeroed for “SYM” CO₂/N₂ models), and finally polarizability (for drude oscillators). (NOTE that C12 parameters are not usually input here, as we don’t use C12 coefficients for CO₂, N₂ “SYM” force fields. However, C12 parameters can be read in, as long as the setting `solute_cross_parameter_set=“yes”` is used).

b) “solute parameters for framework cross terms” section: This section must include the key phrase “framework cross terms” for recognition. This section is used only for a simulation of adsorbates in a porous crystalline framework, in which different force field parameters are used to create the solute-framework cross terms. If this section is present, the variable **solute_cross_parameter_set** should be set to “yes” in the `simulation_parameters.pmt` file. There should be as many lines for this section, as there are solute atom types in the previous “solute species” section, and the original order should be preserved. “solute parameters for framework cross terms” section: The entries for each line are as follows: A_{exch} , A_{elec} , A_{ind} , A_{dhf} , A_{disp} (zero), B, C6,C8,C10, (and C12 iff parameter `C12_dispersion` is set to “yes”). Here, note the sign of the A_{dhf} parameter, as this can sometimes be negative (see “SAPT Force fields-general” section for further elaboration on the A_{dhf} sign convention).

c) “solute dhf cross terms” section: The phrase “solute dhf cross terms” is needed for recognition of this section. This section is only used for the CO₂ “SYM” model, in which the C-O A_{dhf} parameter isn’t generated by a simple combination rule. This parameter should be input under the section heading.

d) “solute-solute exponents” section: The phrase “solute-solute exponents” is needed for recognition of this section. This section lists the all the exponents, including cross terms, for the solute atoms of the “SYM” force field, since these exponents can’t be generated using simple combination rules. The format for “N” atom types, is a single line under the heading line, containing the $(N * (N+1)) / 2$ possible atom type pairs, in an order analogous to the loop structure `i=1,N ; j=i,N`.

e) “framework_species” section: The format for the force field input lines is as follows: atomtype label (must match that used in “*.fconf” file), charge, A_{exch} , A_{elec} , A_{ind} , A_{dhf} , A_{disp} (zero), B, C6,C8,C10, (C12 iff parameter `C12_dispersion` is set to “yes”), and polarizability (zero). Again, note the sign of the A_{dhf} parameter, as this can sometimes be negative (see “SAPT Force fields-general” section for further elaboration on the A_{dhf} sign convention).

SAPT Force fields-general: Beyond the “SYM” force fields, we have developed transferable force fields for other systems. We have tried to keep the force field parameters as simple as possible, avoiding explicit cross terms and using a consistent treatment of dispersion (damped C6-C12 terms) for all interactions.

a) “solute_species” section: The format for the force field input lines is as follows: atomtype label, charge, A_{exch} , A_{elec} , A_{ind} , A_{dhf} , A_{disp} (zero), C6,C8,C10, C12, alpha. Besides the charge and A_{dhf} term, all parameters should be positive here. The A_{dhf} term can either be positive or negative, and the sign of the cross term will depend on setting of the parameter “dhf_combination_rule” in the global_variables module (the current setting of this parameter will be printed to the output file). The most common settings used are either “2”, meaning the cross term A_{ij} is positive if and only if A_i and A_j are both positive, and negative otherwise; or “3”, meaning A_{ij} is always negative (attractive interaction) regardless of the signs of A_i and A_j . More recently, we have adopted the convention to use a strictly attractive convention for these terms (see ref 5), but in order to be compatible with some of the older force fields (ref 4) that use both positive and negative A_{ij} terms, we most often set “dhf_combination_rule=2” and make all A_{dhf} terms negative if we want strictly attractive terms (see neat_acetone_3body example).

b) “framework_species” section: The format for the force field input lines is as follows: atomtype label, charge, A_{exch} , A_{elec} , A_{ind} , A_{dhf} , A_{disp} (zero), B, C6,C8,C10, C12 and polarizability (zero). See the above discussion in the “solute_species” section for comments on the sign of A_{dhf} . The polarizability should always be set to zero for framework atoms, as Drude oscillators are not implemented for fixed framework atoms.

Other Force fields: Generic Lennard-Jones and Buckingham type force fields can also be used. The input here is relatively simple, and is described briefly below. See the examples “zif8_UFF_CO2_303K_1bar” or “neat_acetone_opls”. The setting of the parameter “lj_comb_rule” in the “simulation_parameters.pmt” file determines the combination rule to be used for cross terms, and should be set either to “standard” for Lorentz-Berthelot rules, or “opls” for opls combination rules.

a) “solute_species” section: The format for the force field input lines is as follows: atomtype label (must match that used in “*.conf” file), charge, then 3 parameters, either epsilon, sigma, and junk (best to set junk=0.0 so that it is obvious that a LJ potential is being used) for a LJ force field, or A, B, C6 for a Buckingham force field; and finally the polarizability is listed (most likely zero for these force fields).

b) “framework_species” section: The input is identical to that of the “solute_species” section, except these atom types correspond to crystal framework atoms.

Section 6: Simulation parameters input file (simulation_parameters.pmt)

General comments about structure of input file:

The parameters in this input file are read from two sections, the first section headed with the “Simulation Methodology” title with parameters read in as small character strings; and the second section headed with the “Simulation Parameters” title with parameters read in as either integers or real numbers depending on the parameter. It is important to use these exact titles for the sections, as the code will search for these exact headers. Each line after these headers will be read in, the first entry being the variable name (string), and the second entry being the value of this parameter. Any additional entries will be treated as comments (I put comments in with a “!”, but this is just for clarity). All entries on a given line are delimited with spaces.

Note that many times it is not necessary to explicitly set all of the possible parameters read in from this file, as many parameters are specific to certain simulation types or force fields. The best way to construct this file is probably to use one of the example files as a template. If an essential parameter is missing from this file, the code should stop with an error message, and the code does perform some internal checks on consistent parameter settings.

Section: Simulation Methodology (string variables)

select_ensemble :: This variable determines the ensemble for the MC simulation. Currently implemented choices are “nvt”, “npt”, “uvt”, “egc”, corresponding to the canonical, isothermal-isobaric, grand-canonical, and expanded grand canonical ensembles respectively. Note that the isothermal-isobaric ensemble cannot be used if there is a fixed crystal framework. Also “egc” has not been tested thoroughly and should only be trusted after validating with “uvt” results.

Molecular Dynamics (MD) simulations can be run in the nve ensemble by setting “select_ensemble=md”.

sapt_type_ff :: This variable is set to “yes” if SAPT-based force fields^{1,3,4} (SYM¹, ZIF-FF⁴) are being used in which there is an explicit energy decomposition. Setting this variable to “yes” determines how the force field parameter file is read, and explicitly prints an energy decomposition to the output file. This variable should be set to “no” for any Lennard-Jones force field, or standard Buckingham force field.

solute_cross_parameter_set :: This variable is set to “yes” only if different parameters are used to generate solute-solute and solute-framework force field cross terms (This is used, as in Ref 4, to treat solute-solute interactions with SYM and solute framework interaction with ZIF-FF). This variable should be set to “no” for any non-Buckingham, non-Energy decomposed force field. The setting of this variable determines how the force field parameter file is read in.

C8_C10_dispersion_terms :: This variable is set to “yes” if C8 and C10 terms are used for dispersion. This can only be used with a Buckingham force field, of the energy decomposed type. This variable should be set to “no” for any Lennard-Jones force field.

C12_dispersion :: This variable is set to “yes” if C12 terms are used for dispersion. This can only be used with a Buckingham force field, of the energy decomposed type. If omitted in the input file, this variable will automatically be set to “no”.

damp_solute_solute_dispersion :: This variable should be set to “yes” if C6 through C12 dispersion terms are used for solute – solute interactions, as such interactions should be damped (for the SYM force field, this variable should be set to “no”, as only an effective C6 term is used). For any generic Lennard Jones force field, this should be set to “no”.

hybrid_md_mc :: This variable should be set to “yes” if hybrid md/mc type translation and rotation moves are desired. Although this code can run MC simulations with ordinary single molecule moves, it was written with the intent of carrying out hybrid md/mc moves and is not optimized for single molecule moves. It is therefore recommended that this variable be always set to “yes”.

electrostatic_type :: Determines the treatment of electrostatics, set to either “pme” (Particle-Mesh Ewald), “cutoff”, or “none”. Although regular Ewald routines are contained in the source code, these were not generally implemented as they are slow.

pme_disp :: If set to “yes”, solute-framework dispersion interactions in a GCMC simulation are computed “exactly” using the generalized PME method for $1/R^n$ potentials. In this implementation, the reciprocal space potential is gridded for every atom type, and on-the-fly interpolation (using 2nd order Lagrange functions) is used to generate the potential at all spatial points. The real space part of the Ewald sum for the solute-framework interactions uses the parameter “ewald_cutoff” to determine the cutoff, which is consistent with PME electrostatics.

Solute-solute interactions are still treated strictly in real space with a cutoff, using the value of “lj_cutoff”. If the “pme_disp” setting isn’t present in the parameter file, it will be automatically set to “no”.

three_body_dispersion :: If set to “yes”, an Axilrod-Teller (C9) treatment of three-body dispersion interactions is employed. See the “neat_acetone_3body” example.

lj_comb_rule :: Determines the type of combination rule to use for Lennard Jones or Buckingham parameters. For Lennard Jones, set to “opls” or “standard” (for Lorentz-Berthelot combination rules), for Buckingham, set to “standard” (all geometric) or “ZIFFF” (for ZIF-FF combination rules) .

verlet_list :: If set to “yes”, uses a verlet neighbor list to calculate pairwise Lennard Jones or Buckingham interactions. The default setting is not to use a neighbor list for these interactions.

Feynmann_Hibbs_correction :: (Aside: The two “n’s” is not a typo! Sorry, bad spelling when writing code, don’t want to correct this for compatibility issues). If set to “yes”, a Feynman-Hibbs effective potential is used (see references in source code). Such a potential is meant to incorporate nuclear quantum effects in a variationally optimized way. This can be used to simulate hydrogen at low temperatures. The Feynman-Hibbs contribution to the energy is either computed with a cutoff, or with a generalized PME implementation, so that it is consistent with the “pme_disp” setting. The Feynman-Hibbs potential should only be used for atoms, or linear molecules (see supporting info of ref 6 for discussion).

Section: Simulation Parameters (real/integer variables)

n_step :: Determines the number of total MC/MD steps to be carried out in a simulation. Things that count as one step are : An N molecule hybrid md/mc move , a particle insertion or deletion, a volume move, a single molecule move for non hybrid md/mc simulations, etc. This is usually at least Order(100,000) for a hybrid md/mc simulation.

n_output :: This is the frequency for printing trajectory information (energies, number of molecules, configurations, etc.) to the output files. A value of “100” means print information every 100 steps.

n_update_stepsize :: This is the frequency for updating the step size in order to try to reach the desired move acceptance rate (time step in hybrid mc/md, volume step in npt, etc.). A value of “50” means update the step size every 50 moves.

target_acceptance :: This is the target acceptance for translation/rotation moves. A setting of “0.4” means the target acceptance is 40%.

too_close :: --IMPORTANT FOR BUCKINGHAM POTENTIALS-- This variable essentially defines a hard wall potential distance (in Angstroms) for all atom-atom interactions. If any trial move places two atoms from different molecules at a distance less than this value, the move will be automatically rejected. This hard wall is important for Buckingham potentials, especially in a GCMC simulation when molecules are being inserted randomly, because unlike Lennard-Jones potentials, Buckingham potentials do not have a divergent repulsive term. Therefore Buckingham potentials always have a divergent attractive well due to both the Coulomb and dispersion interactions, and therefore a repulsive wall is needed for certain types of random moves. This also saves time by not computing energies and forces for strongly repulsive configurations, and also prevents Drude-oscillator catastrophes from placing atoms too close together. Normally this can be set up to a value of 2 Angstroms without “affecting” the simulation results, since this value is well within the van-der-Waals contact distance for any two elements (if there are potential close-contact interactions such as hydrogen bonding, a more conservative value should be used of maybe 1.5 Angstroms).

delta_t :: This is the time step (in pico-seconds (ps)) for numerical integration in the hybrid md/mc translation/rotation moves. Note these moves do not (and probably should not for effective sampling) conserve energy, and hence much longer time steps can (and should) be used compared to MD simulations. A typical setting is 0.02 ps.

max_delta_t :: As variable “delta_t” will be updated during the simulation in an attempt to reach the desired move acceptance rate, this puts a limit on the maximum allowed size of “delta_t”. A setting of 0.1 ps is reasonable.

temperature :: This is the simulation temperature in Kelvin.

lj_bkghm :: This tells the code whether the simulation will employ a Buckingham (set to “1”) or a Lennard Jones type force field (set to “2”)

lj_cutoff :: This is the cutoff distance for all dispersion interactions in Angstroms. Despite the name, this controls the cutoff for all C6, C8, C10, C12 terms in a SAPT-based potential, as well as the long range term in a Lennard-Jones potential.

ewald_cutoff :: This controls the real space cutoff for interactions in Ewald (PME) electrostatics (and if used, PME dispersion) and is given in Angstroms. A value of “10” should probably be long enough with the PME setting of “alpha_sqrt=.4”.

Electro_cutoff :: This controls the cutoff for a non-Ewald (“cutoff”) treatment of electrostatics, and is given in Angstroms.

verlet_cutoff :: This is the Verlet skin thickness and is used if the Verlet neighbor list is employed.

dispersion_lrc :: This variable determines whether a long-range correction should be applied for dispersion interactions. A setting of “1” turns on the long-range correction, a setting of “0” turns it off. For any kind of crystal framework simulation, this should be set to “0”, as the dispersion long range correction involves setting $g(r)=1$ at the cutoff for all atom type pairs, which is a good approximation only for isotropic systems, and doesn’t make sense for a crystal. Rather, the PME implementation of dispersion should be used in crystal simulations for a more exact treatment.

screen_type :: This controls the screening for electrostatic interactions. A setting of “1” uses a Tang-Toennies type screening function for all electrostatic interactions with the screening parameter equal to the Buckingham exponent for the atom pair. A setting of “0” uses no screening. For ZIF-FF, this should be set to “1”.

drude_simulation :: This setting determines whether or not Drude-oscillators are used in the force field and simulation. Set to “1” if Drude-oscillators are being used, “0” otherwise.

springconstant :: This is the springconstant for Drude oscillators in atomic units (au). Set to 0.1 au for SYM force field.

thole :: This is the thole parameter for intra-molecular screening of Drude oscillators. Set to 2.0 for SYM force field.

n_threads :: This determines the parallelization of the energy and force calculations. These routines have been parallelized using OpenMP (shared memory), and therefore this can be set as high as the number of processors on a shared memory node. For serial implementation, set this to “1”.

Ensemble specific parameters

“uvt” (and “egc”) ensemble:

gcmc_ins_rm :: This is the percentage of moves for which to perform an insertion/deletion of molecules. A setting of 0.3 means to do this for 30% of the moves.

chemical_potential :: This is the chemical potential of the solute in units of kJ/mol . Note this chemical potential SHOULD NOT include vibrational/rotational contributions, as this needs to be consistent with the acceptance criteria for the molecule insertion/deletion moves.

orientation_try :: This sets the number of attempted orientations for an orientation-biased insertion/deletion. A setting of “1” means that no orientation-biased moves are used.

“npt” ensemble:

pressure :: This is the simulation pressure in bar.

vol_step :: This is the volume step size to use in units of $\ln(V)$, as moves will be made in steps of $\ln(V)$. A reasonable setting is 0.05.

max_vol_step :: As the variable “vol_step” will be adjusted to meet the target acceptance, max_vol_step puts a limit on this step size. A reasonable setting is 0.2.

target_acceptance_v :: This is the target acceptance for volume moves, and the volume step size will be adjusted with frequency “n_update_stepsize” in order to try to meet this criteria. A value of 0.4 means that moves should be accepted 40% of the time.

cycles_per_vol_step :: This determines how often volume moves are done. A value of “1” means that on average, one volume move is carried out per cycle of translation/rotation moves. A “cycle” for translation rotation moves is either one hybrid md/mc move, or N single molecule moves.

Force field specific parameters

Three-body dispersion (these are only necessary for a setting of three_body_dispersion=“yes”):

three_body_dispersion_cutoff :: This is the cutoff used for 3-body interactions. In order for the ABC trimer interaction to be calculated, all distances r_{AB} , r_{AC} , r_{BC} must be less than this cutoff.

three_body_cutoff_type :: This determines whether the cutoff for three body dispersion interactions is applied on a molecule by molecule (setting = 0) or an atom by atom (setting = 1) basis.

na_nslst (nb_nslst, nc_nslst) :: These parameters control the number of cells along each box vector that are used for the cell link list algorithm that is used to loop over 3-body interactions.

Lattice Model Code:

This code implements the lattice simulation algorithm developed in the Schmidt group,⁷ that was designed as a substitute for GCMC simulations as it is 2-3 orders of magnitude faster, while introducing minimal error compared to the GCMC simulations. Briefly, the simulation box (which is usually a crystal structure, such as a metal-organic framework) is partitioned into a three-dimensional array of sub-cells, and the sub-cells are coarse grained rigorously (based on an atomistic force field) to build a lattice model, consisting of a free energy grid of solute-framework interactions at all the lattice sites. Creating this free energy grid is the most intensive and time consuming part of the simulation, and once the grid is created, fast lattice Monte Carlo simulations can be carried out to compute the adsorption loading isotherm. For all but low-loadings, solute-solute interactions are important, and in addition to the free energy grid, a coarse-grained solute-solute potential is needed in the lattice Monte Carlo simulation to give accurate results.

Therefore, the steps in a Lattice simulation are:

- 1) Construct the lattice model – Map an atom-atom force field which describes the *interaction energy* between a solute molecule and the crystal framework at *all points in space* to a lattice model that gives the *free energy* of a solute molecule adsorbed at a *discrete lattice site* in the crystal. This is the rate-limiting step of the calculation. [Note that this free-energy grid may be useful by itself in analyzing adsorption sites in a crystal (see ref 6), without actually calculating an adsorption isotherm].
- 2) Discretize a continuous coarse-grained solute-solute interaction potential to a discrete coarse-grained solute-solute potential compatible with the lattice model.
- 3) Using the free energy grid lattice model and the discretized solute-solute coarse grained potential, run the lattice Monte Carlo simulation to get an adsorption isotherm (this is trivial compared to a standard atomistic MC simulation).

In constructing the free energy grid, this code utilizes PME for both electrostatic and long range dispersion interactions, and implements both link neighbor list and symmetry operations to accelerate the calculation speed.

All the force fields implemented in the Monte Carlo code are also supported here, and all the force field inputs are identical to the previous part of the manual. Note, Vegas and Miser

sampling methods are implemented, but not activated in the code. It can be activated by uncommenting certain part of the code, even though it is in general not helpful.

Section 1: Constructing the lattice model

The main computational effort involved in computing gas-adsorption isotherms is in constructing the free energy lattice grid. Therefore a large portion of this manual (Section 1) will be concerned with the logistics of this code. Other sections of the manual (Section 2, 3) will then focus on constructing the coarse grained solute-solute lattice model, and finally running the lattice Monte Carlo simulation.

Section 1a: Outline of Source Code

The source code for the lattice grid construction can be found in the “source_lattice/lattice_free_energy_grid/” directory. Besides the source code, this directory also contains a directory that has the GSL routines needed for compiling (see below)

Major source code files:

boltzmann_factor_mod.f90: This module defines the function that calculates the averaged Boltzmann factor of a particular sub-cell. ***Note, it is only used for Vegas sampling.***

sapt_ff_routines.f90: Inherited from the Monte Carlo code.

eq_drude.f90: Inherited from the Monte Carlo code, modified to use the new gridded energy and force subroutine.

initialization_routines.f90: Inherited from the Monte Carlo code (file mc_routines.f90), new codes are added to initialize g-function tables, utilized in the dispersion PME calculation.

glob_v.f90: Inherited from the Monte Carlo code. All new global variables are added in this file.

lattice_sampling.f90: This file contains the main subroutine used to sample each lattice grid site and average the Boltzman factor for each site.

link_list.f90: This file includes a link list construction function, which is called at the beginning of the simulation to construct the cell neighbor list.

main_lattice_grid.f90: The program declaration.

penetration_ff_routines.f90: Inherited from the MC code, barely relevant.

pme.f90: This contains major functions and subroutines used for electrostatic and van der Waals PME calculations. While these subroutines are partially inherited from the MC code, they contain fundamental differences. Namely, The Ewald sums are used to compute potentials at lattice sites, rather than total energy, and therefore there is no self-interaction as in standard Ewald sums.

read_simulation_parameters.f90: Inherited from the MC code.

general_routines.f90: Inherited from the MC code. Some extra subroutines related to the lattice calculation are added.

symmetry.f90: This module includes all the functions and subroutines related to the symmetry operations.

Qeq_charge_equilibration.f90: This module implements the Qeq charge equilibration method for periodic systems, to derive charges for a crystal framework.

Section 1b: Notes for Compiling and Running

A Makefile template is provided in the source directory and users can compile the code by running:

make

This will generate the major executable, namely, *main_lattice_grid*.

Several packages are prerequisite, including: MKL, GSL, FGSL. Depending on the implementation and installation details of these packages, users have to modify the path variables in the Makefile accordingly, before start compiling. The major path variables include:

OMPINCLUDE: open mp library path ***

MKLLIB: mkl library path

FGSLINCLUDE: fgsl head files path

FGSLLIB: fgsl library path

GSLINCLUDE: gsl head files path

GSLLIB: gsl library path

***NOTE this is not really used, as the code now has all parallel open mp sections commented. If one wants to get rid of this path, just comment the “use omp_lib” lines in the source code. We have left these parallel sections in for the option of future use, but it seems to us like it is not worth parallelizing, since most likely many different MOFs will be run at once, and one can “parallelize” instead over job submissions.

The command to run the lattice code is:

./main_lattice_grid ifile.conf ifile.fconf ifile.pmt simulation_parameters.pmt ofile symmetry.sym

This command runs the code to compute the averaged Boltzmann factor for each sub-cell. The adsorbate geometry file (ifile.conf), the MOF geometry file (ifile.fconf), and the force field definition file (ifile.pmt) are identical to the Monte Carlo code (refer to appropriate previous sections for details). The output file (ofile) contains all the free energy information for the lattice model. The other two input files (simulation_parameters.pmt and symmetry.sym) will be introduced in detail in the next section.

Section 1c: Simulation Parameters File and Symmetry File

The simulation parameters file (simulation_parameters.pmt) is inherited from the Monte Carlo code, and therefore has the same format. A few new variables are added in this file to specify the lattice construction parameters. These variables are described below:

energy_decomposition: analogous to parameter “sapt_type_ff” (see section 6 of MC code manual)

Qeq_charges: set to “yes” if Qeq charge equilibration method is to be used to derive atomic charges for the crystal framework.

pme_grid: size of the PME grid. The PME grid should be fine enough to ensure: 1, the convergence of the reciprocal part of the PME; 2, the accuracy of the reciprocal potential interpolation. Typically, the value of 120 should be fine enough for a 60Å*60Å*60Å simulation box.

alpha_sqrt: the \sqrt{a} value for the electrostatic PME.

lj_asqrt: the \sqrt{a} value for the dispersion PME. It can be either same or different to the \sqrt{a} value for the electrostatic PME (**alpha_sqrt**)

lj_cutoff: the real space cutoff for the dispersion energy. If **lj_asqrt** is set to be 0.6, typically it needs a 7.5Å real space cutoff to ensure convergence.

ewald_cutoff: the real space cutoff for the electrostatic PME. Typically, if **alpha_sqrt** is set to 0.6, then **ewald_cutoff** has to be set larger than 5.0Å.

cav_grid_a, cav_grid_b, cav_grid_c: the lattice size in three dimensions, respectively.

na_nlist, nb_nlist, nc_nlist: the grid size for the cell neighbor list. For a 60Å*60Å*60Å simulation box, it is recommended to use a 40*40*40 cell list.

start_grid_a, start_grid_b, start_grid_c, finish_grid_a, finish_grid_b, finish_grid_c (optional): These optional settings can be used to script the lattice code for trivially efficient parallelization. If these settings are present, then the free energy grid will only be calculated for lattice cells satisfying $\text{start_grid_a} \leq \text{cell_a} \leq \text{finish_grid_a}$, $\text{start_grid_b} \leq \text{cell_b} \leq \text{finish_grid_b}$, $\text{start_grid_c} \leq \text{cell_c} \leq \text{finish_grid_c}$. Thus **start_grid_a** must be greater than or equal to 1, and **finish_grid_a** must be less than or equal to **cav_grid_a** (and similarly for b and c lattice vectors)

orientation_try: the maximum sampling size for each sub-cell. If the sub-cell dimension is $\sim 0.7\text{-}1.0 \text{ \AA}$, then a value of 2000 is recommended.

REL_THRSH, ABS_THRSH, BZ_CUTOFF: the convergence threshold. If the averaged Boltzmann factor of a particular sub-cell is below BZ_CUTOFF, then the code adopts an absolute threshold for the Boltzmann factor sampling, and the threshold is set to be ABS_THRSH. Otherwise, a relative convergence threshold (controlled by variable REL_THRSH) is applied. The recommended medium precision setting for these three variables is (0.05, 3.0, 100.0). The users are referred to reference 7 for further details.

Another important input file is the symmetry file (symmetry.sym), which defines all the symmetry operations used in the simulation to accelerate the speed. Basically, once the code finishes sampling one particular sub-cell, it goes through all the symmetry elements and identifies all the symmetrically equivalent sub-cells. All these sub-cells are filled with the same free energy and are no longer sampled in the following calculation. The first line of the symmetry.sym file defines the number of the symmetry operations. And each following line represents one particular operation. Each symmetry operation can be written as:

$$\vec{x}' = M\vec{x} + \vec{s}$$

In this equation, \vec{x} is the three-dimensional coordinates before the operation, and \vec{x}' is the resulting coordinates after the operation. M is a 3*3 matrix and \vec{s} is the shift. In the symmetry.sym file, each line has the form (in FORTRAN format):

“(12F10.6)” $M(1,1), M(1,2), M(1,3), s(1), M(2,1), M(2,2), M(2,3), s(2), M(3,1), M(3,2), M(3,3), s(3)$

Generally, the space group symmetry information is given in the cif file with the format:

x, y, z

-x, -y, z

-x, -y, z+1/2

...

A python utility, cif2sym.py (which can be found in the directory “useful_scripts/”), is provided to convert the cif format to the sym format, which is used for our code (to run the utility, type: `./cif2sym.py MOF.cif > symmetry.sym`).

If no symmetry exists in the system, please write down 0 in the first line of symmetry.sym file.

CAREFUL!! : If you are simulating a *supercell*, the symmetry operations in the .cif file for the unit cell may not be appropriate! See examples for further discussion.

Section 2: Coarse grained solute-solute interactions

To incorporate solute-solute interactions into the gas adsorption isotherm, it is important to use a coarse grained potential in combination with the free-energy lattice grid. The coarse grained potential, which is continuous (although in practice is input as discretized), is then discretized to fit on the lattice grid. Two example coarse grained potentials (for CO₂ and N₂) are provided in the directory “lattice_coarse_grained_potentials/”, along with a script that will generate such files from a given epsilon and sigma parameter.

These two example coarse grained potentials have been taken from the literature, and have been tested against explicit atomistic GCMC simulations. To obtain a coarse grained potential for a different solute, one has a few choices. One can search the literature for previous benchmarked coarse-grained potentials and use these if available. If no such potential is available in the literature, one may use a force-matching approach to fit his/her own coarse grained potential to an accurate atomistic model. A third option, if one is interested in low-loadings only, is to use a hard sphere potential, which may (or may not!) be sufficiently accurate for low uptake. These three different methods have been discussed and compared in reference 7. Any newly used coarse-grained potential should be tested against explicit atomistic GCMC simulations.

Now, we'll assume that we have a coarse grained potential, and use the Liu_CO2_potential.xvg potential as an example. The (short) code used to discretize the potential to the lattice grid is located in the directory “source_lattice/coarse_grain_solute/”. As discussed in reference 7, there are a couple choices as to how to discretize the potential; namely one could average the Boltzman factor over all possible solute positions within the given two lattice cells, or one could simply use the distance between the centers of the two given cells. Because we have found that the later, simpler choice works as well as the former choice, as stated in said reference, this is the method that is currently implemented in the code (one could easily change the code to perform the first procedure, by changing the value of `n_sample` and choosing an appropriate temperature in `glob_v.f90`).

To run the code, use the command:

```
./discretize_solute_potential crystal.fconf potential.xvg ngrida ngridb ngridc > outputfile
```

Where `crystal.fconf` is the MOF crystal fconf file (used to get the unit cell vectors), `potential.xvg` is the coarse grained potential file described above, and `ngrida`, `ngridb`, `ngridc` are the number

of lattice cells along each box vector. It is IMPORTANT to make sure that `ngrida`, `ngridb`, and `ngridc` are the exact values used in the lattice free energy grid (`cav_grid_a`, `cav_grid_b`, `cav_grid_c` respectively). The output file will contain a list of lattice points (really these are displacement lattice vectors, between two coarse grained particles), with the corresponding potential printed in kJ/mol. This file can then be used for the lattice MC simulation.

Section 3: Lattice Monte Carlo simulation

Once the free energy lattice grid has been generated (Section 1) and the coarse-grained solute-solute potential has been discretized on the lattice (Section 2), we can run a lattice Monte Carlo simulation to compute the adsorption isotherm. The source code for this lattice MC simulation is located in the directory “`source_lattice/lattice_monte_carlo/`”. If one wants to modify the default values for simulation settings, the values of the parameters `n_equil` (10000000 default, number of equilibration steps), `n_steps` (10000000 default, number of production run steps), `n_output` (1000000 default, number of steps per block average), and `gcmc_ins_rm` (.4 default, fraction of moves which are insertions/deletions rather than lattice displacements) can be modified in `global_variables.f90`, and the code can be recompiled.

To run the lattice simulation, use the executable `lattice_MC` and run the command

```
./lattice_MC free_energy_grid_file solute_cg_file chem_potential mass_solute > ofile
```

Where the `free_energy_grid` input file is the output file from Section 1, the `solute_cg` file is the output file from Section 2, `chem_potential` is the chemical potential of the solute (the chemical potential should be in units of kJ/mol . Note this chemical potential SHOULD NOT include vibrational/rotational contributions, as this needs to be consistent with the acceptance criteria for the molecule insertion/deletion moves. Either see the CO₂/N₂ scripts in “`/useful_scripts/`” or see the examples) and `mass_solute` is the mass of the solute. The mass of the solute is needed for the insertion/deletion acceptance criteria, since it is in the ideal gas contribution to the chemical potential, and we want to keep the chemical potential definition the same as in the atomistic Monte Carlo code.

The output file will report the average density of adsorbed solute molecules (at the given chemical potential, and temperature of the free energy grid) for the different production run blocks. Because these lattice Monte Carlo simulations are very fast, one can generate a full isotherm (at say 10 different pressures) within minutes.

Section 4: Using the scripts

To make it easier to carry out a lattice simulation, we have tried to script as many of these steps as possible. So here's a comprehensive list of all the files you need to have in your directory, and the script(s) you need to run to compute a lattice model isotherm.

Files Needed (put all these files in the run directory)

Executable shell scripts : run_lattice.sh, construct_isotherm.sh (for isotherm)

Fortran binaries : main_lattice_grid (constructs free energy lattice grid), discretize_solute_potential (discretizes solute coarse-grained potential onto lattice), lattice_MC (runs the lattice Monte Carlo simulation; for isotherm)

Input files : name1.conf (Section 1), name1*.pmt (Section 1), simulation_parameters.pmt (Section 1), name2*.fconf (Section 1), name2.sym (Section 1), name3.xvg (Section 2), isotherm.pressures (for isotherm).

Running the command

```
./run_lattice.sh name1 name2
```

(e.g. in the zif8 example, one would use the command ./run_lattice.sh co2 zif8)

Will carry out the process of discretizing the solute-solute coarse grained potential (output file name3.txt) , and will submit a job named "name1_name2_lattice_grid" that creates the free energy lattice grid and produces an output file with the same name as the job.

While this free energy grid may be useful in its own right, i.e. to spatially visualize important adsorption sites in a MOF, we still have to run the lattice MC simulation to get an isotherm.

To run the lattice MC simulation create a file named isotherm.pressures

Which might look like this (for CO₂)

mass_solute 44

pressure(bar)	chemical potential
---------------	--------------------

1	-40.31693
---	-----------

5	-36.38414197
---	--------------

...

See Section 3 for definition of chemical potential, and the reason for including the solute's mass. Note the pressure here isn't actually used by the lattice MC code, but it is just used by the script to label the output files.

Then run the command
`./construct_isotherm.sh isotherm.pressures name1_name2_lattice_grid name3.txt`

Where name1_name2_lattice_grid is the lattice free energy output file, and name3.txt is the solute-solute discretized potential.

NOTE: The user will probably want to edit the `construct_isotherm.sh` script to change the way the executable is run. Currently, all isotherm points are run sequentially, on the local machine. This is probably not the ideal way to do this, as each different point on the isotherm can be run independently by a separate machine. Therefore the user should probably change the job submission of this particular script to better suit his/her computer resources.

Section 5: Examples

MOF-5 (CO₂)

The MOF-5 examples can be found in the directory “examples_lattice/mof5/”. In this directory, there are examples for a Lennard-Jones force field (UFF + ep_m2), as well as our SAPT-based force fields, and both examples use the coarse-grained Liu potential for solute-solute interactions. We will first discuss the Lennard-Jones force field simulations.

MOF-5: a) Lennard-Jones

The simulations using the UFF/ep_m2 force field are found in the directory “UFF”. In this directory (besides the directory “atomistic”), are five other directories, containing simulations with or without the use of symmetry, both for medium precision with a 30 lattice grid (no suffix) and with a 60 lattice grid (“60” suffix), as well as a simulation at high precision (suffix _HP, bigger grid, more sampling). For further discussion of the precision settings, see ref 7. Focusing on the “sym” directory (medium precision), we make a couple points.

-The time for constructing the lattice free energy grid can be found in the file “co2_mof5_lattice_grid.o###”, and one can see that this only took about 1 minute. The 7 points of the isotherm are in output files “uptake_at_#bar”, and one can see that it took about 6 minutes to generate all of these files. Therefore the isotherm (for this lowest sampling calculation) was generated in a total of about 7 minutes on a single processor, and in this case the rate limiting step was the actual lattice Monte Carlo simulations. We have not taken further steps to optimize the speed of the lattice Monte Carlo code, because, as we will see, constructing the free energy grid is almost always the rate limiting step. In this case,

constructing the free energy grid was very fast due the high symmetry of MOF-5, as well as the fact that a non-polarizable Lennard-Jones force field was used.

-The number of equilibration steps and number of steps of the production run can be found in any of the “uptake_at_#bar” output files. Examining the “30bar” output file (as this should take the longest to equilibrate) we see that the system has definitely been equilibrated over the 10,000,000 equilibration steps, and while the 1,000,000 step production run blocks are somewhat statistically different, the differences between these blocks are probably less than the errors introduced by the sampling in the generation of the free-energy grid (as discussed below). The user is encouraged to test the settings of “n_equl” and “n_steps” in the lattice Monte Carlo code, to balance the trade-off between statistical accuracy and simulation time.

One can then compare the differences in output between the five directories “sym”, “nosym”, “nosym60”, “sym60”, and “sym_HP”. The simulation times can be compared between the simulations of different grid sizes, precision, and use of symmetry. While the simulations with and without symmetry should give identical results in the limit of perfect sampling, they may not necessarily give numerically identical results if the sampling isn’t converged (since using symmetry effectively reduces sampling). To see this, one can view the file “compare_lattice_sym_mp” with xmgrace using the command `xmgrace -nxy compare_lattice_sym_mp``. This file contains the free energies of all identical lattice points from the free energy grids in the directories “nosym” and “sym” (medium precision), and compares them. One can see that the free-energy grids, are very similar, but not identical, due to somewhat incomplete sampling, as is expected at medium precision. Viewing the isotherm output files in these two directories, it is evident that the quantitative differences in the free energy grid translate to differences in the isotherm. One can see that these isotherms mostly compare to within ~10% error, which is on the order of the error expected between a lattice simulation using a good coarse grained potential and a fully atomistic simulation. However, comparing the isotherms from the nosym60 and sym60 directory, one sees that there are negligible differences when a 60 grid is used, so in the limit of perfect sampling, simulations with and without the use of symmetry should indeed converge to the same answer. Also, the isotherms from these 60grid simulations are very comparable to the isotherm in the directory “sym_HP”, indicating that these results from a 60 lattice grid are very nearly converged with respect to sampling. One can then compare these “converged” results to an isotherm from an analogous atomistic MC simulation (results in directory “atomistic” for certain pressures), and find that all points on the isotherm compare to within 10%, with the inherent discrepancy due to the employed coarse-grained solute-solute interactions in the lattice model isotherm.

MOF-5: b) SAPT force fields

The other directories in the “mof5” example directory contain lattice simulations using the SAPT force fields. Again, there are simulations with and without symmetry, and with different lattice grid sizes. One can see that these simulations do take longer, due to the fact that we’re now using a polarizable force field, but they still are very doable considering that a full isotherm for a system comparable in size to MOF-5 can be generated (at medium precision) in a few hours on a single processor (without symmetry).

We use these examples to illustrate an additional point related to simulations employing symmetry. Inspecting the crystal structures (*.fconf files) used in the directories “sym” and “nosym”, one can see that they are different, namely the structure in the “nosym” directory has been spatially translated from the structure in the “sym” file. In an atomistic Monte Carlo simulation, it is appropriate to use any translated unit cell, since the periodic boundary conditions cannot distinguish these systems. While this is also the case for a lattice simulation *without symmetry*, for a lattice simulation with symmetry, the symmetry operations must be consistent with the unit cell, **i.e. a unit cell cannot be translated in a simulation employing symmetry operations**. Therefore, if one were to use the translated crystal structure with the symmetry operations of MOF-5, the code would stop with an error message because the symmetry operations would be inconsistent with the unit cell.

ZIF-8 (CO₂)

In the directory, “examples_lattice/zif8/”, one can find examples of using the lattice model to compute CO₂ isotherms in ZIF-8. Here, we use our SAPT-based force fields, and therefore these simulations are on the slower side (for the system size), again due to the use of a polarizable force field. Again, one can compare the simulation times and statistical accuracy with and without the use of symmetry. The main point we wish to emphasize with this example is the system size. In a normal atomistic simulation, one would definitely need to use a supercell of ZIF-8, as the unit cell lengths are ~17 Angstroms, and therefore for simulating a unit cell, one would have to employ a cutoff less than ~8.5 Angstroms (half the box length), which is way too short for accurate results. However, in a lattice simulation, this is not an issue for two reasons; 1) Ewald sums are used for both electrostatics and vdw interactions, and therefore real-space cutoffs of < 8.5Å can easily be used; 2) there is no cutoff in the lattice Monte Carlo simulation, all solute-solute interactions are considered. Therefore we find that *another advantage of these lattice simulations is that a unit cell as small as ZIF-8 can accurately be used (without using a supercell)*.

ZIF-71 (CO₂)

While this last example doesn't really present anything new, we use it to make one final comment about the free energy grids. Compare all three free energy grids for MOF-5, ZIF-8, and ZIF-71 (i.e. look at the files "mof5/UFF/compare_lattice_sym_mp", "zif8/compare_free_energy_grids", and "zif71/compare_free_energy_grids"). One will see that these free energy grids are very qualitatively different! Therefore, it is clear by contrasting these three different examples that much information concerning the adsorption sites and topology of a particular MOF can be gleaned from the free energy grid (while such information may be very difficult to understand purely from an adsorption isotherm!).

- (1) Yu, K.; McDaniel, J. G.; Schmidt, J. R. Physically Motivated, Robust, Ab Initio Force Fields for Co(2) and N(2), *J. Phys. Chem. B* **2011**, *115*, 10054-10063.
- (2) Yu, K.; Schmidt, J. R. Many-Body Effects Are Essential in a Physically Motivated Co₂ Force Field, *J. Chem. Phys.* **2012**, *136*, 34503-34509.
- (3) McDaniel, J. G.; Yu, K.; Schmidt, J. R. Ab Initio, Physically Motivated Force Fields for Co₂ Adsorption in Zeolitic Imidazolate Frameworks, *J. Phys. Chem. C* **2011**, *116*, 1892-1903.
- (4) McDaniel, J. G.; Schmidt, J. R. Robust, Transferable, and Physically-Motivated Force Fields for Gas Adsorption in Functionalized Zeolitic Imidazolate Frameworks, *J. Phys. Chem. C* **2012**, *116*, 14031-14039.
- (5) McDaniel, J. G.; Schmidt, J. R. Physically-Motivated Force Fields from Symmetry-Adapted Perturbation Theory, *J. Phys. Chem. A* **2013**, *117*, 2053-2066.
- (6) McDaniel, J. G.; Yu, K.; Schmidt, J. R. Microscopic Origins of Enhanced Gas Adsorption and Selectivity in Mixed-Linker Metal–Organic Frameworks, *J. Phys. Chem. C* **2013**, *117*, 17131-17142.
- (7) Yu, K.; McDaniel, J. G.; Schmidt, J. R. An Efficient Multi-Scale Lattice Model Approach to Screening Nano-Porous Adsorbents, *The Journal of Chemical Physics* **2012**, *137*, 244102.