# **Ab initio GCMC simulations using LAMMPS and VASP**

## **Description:**

In this program, LAMMPS and VASP are coupled using a client-server library (called CSLIB) to perform ab initio GCMC simulations. LAMMPS acts as the ‘client’, which generates the trial configuration (via insertion/deletion/translation). The configuration is communicated to VASP (the ‘server’), which relaxes the structure and calculates the energy. The energy is then communicated to LAMMPS, which uses the GCMC acceptance criteria to accept/reject the configuration. If the move is accepted, VASP communicates the relaxed coordinates and atom types to LAMMPS; else, the previous configuration is restored. This completes the timestep. LAMMPS then uses the current energy, coordinates, and atom types to generate a new configuration, and this process continues until the specified number of GCMC trial moves has been reached.

Please read the fix gcmc doc page to learn about how LAMMPS performs GCMC simulations. The Howto client/server page gives an overview of client/server coupling of LAMMPS with another code and describes how the two codes to work in tandem to perform a simulation.

This document describes the input files needed, how to run the calculations, how to build LAMMPS to perform the calculations, helpful links, and some miscellaneous notes. This program was primarily built using the source code for fix gcmc and fix client/md commands in LAMMPS.

## **Input files:**

Here is the description of the input files needed and the information they contain:

|  |  |
| --- | --- |
| **Input file** | **What it contains** |
| in.lammps | Input script to specify parameters for LAMMPS |
| data.in | Starting structure configuration for LAMMPS |
| INCAR | Input script to specify parameters for VASP |
| KPOINTS | k-mesh |
| POSCARtemplate | Starting structure configuration for VASP |
| POTCAR | VASP pseudopotential file |
| vasp\_wrap\_gcmc.py | Wrapper script to call VASP and communicate energy and coordinates to LAMMPS |
| run.script | Job submission script |
| restart.run.script | Job restart script |

The command that calls the GCMC program is called fix client/gcmc. This is the syntax:

|  |
| --- |
| **fix ID group-ID client/gcmc N X M type seed T mu displace volume keyword values ...** |

* ID = user-assigned name of fix
* group-ID = ID of group of atoms to apply this fix to
* client/gcmc = style name of this fix command
* N = invoke this fix every N steps (usually N = 1)
* X = percentage of exchange moves (should be 100-M)
* M = percentage of translation moves
* type = atom type for inserted/deleted atoms
* seed = random # seed (positive integer)
* T = temperature of the ideal gas reservoir (temperature units)
* mu = chemical potential of the ideal gas reservoir (energy units)
* displace = maximum Monte Carlo translation distance (length units)
* volume = volume in which GCMC is being performed (length3 units) (put 0 if you want the program to calculate the volume for you, else specify the volume)
* zero or more keyword/value pairs may be appended to args

keyword = region, full\_energy, overlap\_cutoff

* region value = region-ID

region-ID = ID of region where GCMC exchanges and MC moves are allowed

* full\_energy = compute the entire system energy when performing GCMC exchanges and MC moves (always include this, this is not optional)
* overlap\_cutoff value = maximum pair distance for overlap rejection (distance units)

Example:

For example, if you have a system with Al and O (defined as type 1 and type 2, respectively), and would like to perform GCMC insertions and deletions of only oxygen atoms (70% of the time), but translation moves of both Al and O atoms (30% of the time), you may call your fix as:

|  |
| --- |
| fix 1 Al\_O client/gcmc 1 70 30 2 31782 800 -9.6476 5.0 437.113 region surf full\_energy overlap\_cutoff 1.0 |

Here, Al\_O has to be defined (in the input script in.lammps) as a group consisting of both Al and O atoms.

## **How to carry out the simulations:**

* Set up the input files listed in the table above. It is **highly encouraged** to read the LAMMPS documentation on the structure and information that should be present in in.lammps and data.in. Note that when using LAMMPS as a GCMC client, your LAMMPS input script should not normally contain force field commands, like a pair\_style, bond\_style, or kspace\_style command.
* Edit the line ‘vaspcmd’ at the top of vasp\_wrap\_gcmc.py to use the desired number of cores and VASP binary.
* You may need to edit the poscar\_write() function in vasp\_wrap\_gcmc.py to reflect your simulation requirements (such as if you are using selective dynamics, etc.).
* Use run.script or restart.run.script to run your calculations! Specifically,

|  |
| --- |
| ibrun -n 1 -o 0 lmp\_stampede -v mode file < in.lammps &  python vasp\_wrap\_gcmc.py file POSCARtemplate &  wait |

runs LAMMPS and VASP in client-server mode, and

|  |
| --- |
| sed -i 's/read\_data data.in/read\_restart gcmc.restart.\*/g' in.lammps |

restarts the run from the most recent completed timestep.

Note that the GCMC part uses only one processor, but VASP runs on the number of processors specified in vasp\_wrapper\_gcmc.py. This fix works on only 1 processor.

## **How to build LAMMPS to use the fix client/gcmc command:**

To runs the simulations, you need to first build LAMMPS with the fix client/gcmc program and with the MESSAGE package installed (so that it can communicate with VASP).

* Download the latest tarball of LAMMPS and extract it. This will create a LAMMPS directory with the version date in its name, e.g. lammps-3Mar20.
* Copy fix\_client\_gcmc.cpp, fix\_client\_gcmc.h, message.cpp into ./LAMMPS\_directory/ src/ MESSAGE.
* Build LAMMPS following the instructions on <https://docs.lammps.org/Build.html>. Note that LAMMPS has to be built with the MESSAGE package and CSLIB libraries installed. You could use the build\_lammps.sh script below as an example to build LAMMPS on Stampede2.

|  |
| --- |
| #!/bin/bash  # ================================================  #  # build\_lammps.sh  #  # Version: 3Mar20  # System: Stampede2, TACC  # Created by: Albert Lu ([alu@tacc.utexas.edu](mailto:alu@tacc.utexas.edu))  # Modified by: Vrindaa Somjit  # Last modified: 02/18/2022  #  # ================================================  VERSION=stable\_3Mar2020  module reset  module load intel/18.0.2  module load impi/18.0.2  module load python3/3.7.0  module load cmake/3.10.2  ROOT\_DIR=`pwd`  SRC\_DIR=${ROOT\_DIR}/${VERSION}  # LAMMPS PACKAGES  cd ${SRC\_DIR}/src  # Build lammps with its MESSAGE package installed  make yes-all  make no-lib  make no-ext  make no-gpu  make yes-message  make package-status  make lib-message args="-m"  # Build CSLIB to be used by vasp\_wrapper\_gcmc.py  cd ${SRC\_DIR}/lib/message/cslib/src  make shlib CC=mpicc zmq=no  # Build lammps  cd ${SRC\_DIR}/src  cat MAKE/OPTIONS/Makefile.intel\_cpu\_intelmpi | sed 's/-xHost/-xCOMMON-AVX512 -axMIC-AVX512/g' | sed 's/-restrict/-restrict -diag-disable=cpu-dispatch/g'> MAKE/MACHINES/Makefile.stampede  make -j 10 stampede  mkdir -p ../bin  mv lmp\_stampede ../bin |

## **Documentation to read:**

<https://docs.lammps.org/fix_gcmc.html>

[https://docs.lammps.org/Howto\_client\_server.html#](https://docs.lammps.org/Howto_client_server.html)

<https://docs.lammps.org/Build.html>

<https://docs.lammps.org/fix_client_md.html>

<https://lammps.github.io/cslib/>

See the examples/message directory for example scripts where LAMMPS is both the “client” and/or “server” code for this kind of client/server GCMC simulation. The examples/message/README file explains how to launch LAMMPS and another code in tandem to perform a coupled simulation. The directory ./LAMMPS\_directory/examples/COUPLE/lammps\_vasp may be helpful as well.

## **Miscellaneous notes:**

1. Important note: Similar to the fix gcmc command, in the fix client/gcmc command, all inserted particles are always added to two groups: the default group “all” and the fix group specified in the fix command (**group ID**). Different to the fix gcmc command, fix client/gcmc cannot be used to perform GCMC insertions or deletions of atoms other than the exchanged type (specified in **type**), but MC translations can be performed on any atom in the fix group.
2. This program was written by modifying fix gcmc and fix client/md. The fix gcmc command has a number of other keyword/values pairs that can be used; I have not edited them in fix\_client\_gcmc.cpp and fix\_client\_gcmc.h, so use/edit them at your own risk!
   1. keyword = mol, region, maxangle, pressure, fugacity\_coeff, full\_energy, charge, group, grouptype, intra\_energy, tfac\_insert, or overlap\_cutoff (have not edited fix\_gcmc source code for those in red)
3. The fix gcmc command can also perform GCMC moves on molecules. I have not included that in fix client/gcmc; however, GCMC of molecules can possibly be done with minor edits to the code
4. Important note: To perform the insertion/deletion moves on > 1 atom type, you will have to edit attempt\_atomic\_deletion\_full() and attempt\_atomic\_insertion\_full() in fix\_client\_gcmc.cpp. Specifically, in attempt\_atomic\_deletion\_full(), you will need to randomly select the atom to delete, specify the correct mass and chemical potential of the deleted atom and recalculate beta, lambda, sigma, and the zz factor. In attempt\_atomic\_insertion\_full(), you will need to randomly choose atom type to insert, specify the correct mass and chemical potential of the inserted atom, and recalculate beta, lambda, sigma, and the zz factor. Note that if you perform GCMC moves on > 1 atom type, you will need to specify the chemical potentials of the atom types you are inserting/deleting. These new variables need to be included in constructor of fix\_client\_gcmc.cpp and in fix\_client\_gcmc.h.
5. In principle, the server code can be any quantum code. The wrapper script will have to be prepared accordingly.
6. fix\_client\_gcmc.cpp usually outputs additional information, for example, insertion coordinates, updated coordinates of accepted moves, etc. You can Ctrl+F for ‘cout’ and comment out the print lines you do not require.