Communication protocol of new QM/MM interface

November 28, 2017

1 Code suite description

The new QM/MM framework MiMiC (Multiscale Modeling in CPMD) is built following the MPMD model and uses a loose coupling between partner codes. MiMiC currently supports electrostatic embedding QM/MM which is what is described in this document. In this model CPMD computes the forces of the quantum system subject to the external electrostatic potential generated by the MM atoms, and GROMACS computes the forces acting on MM atoms due only to the MM atoms plus forces between bonded atoms at the QM-MM borders. MiMiC computes the forces on classical atoms due to the charge distribution of the QM domain, electrostatic forces on QM nuclei due to the MM atoms, and the electrostatic potential generated by the MM atoms on the electrons. CPMD takes care of evolving all the nuclear/atomic (i.e. QM and MM) positions. MiMiC takes care of managing data exchange to and from GROMACS and CPMD. In particular, MiMiC receives data from GROMACS for initialising the run (e.g. number of atoms, charges, masses, number of residues, etc.). At each time step MiMiC sends to GROMACS the present configuration of MM atoms and receives the corresponding MM forces (i.e. the forces acting on MM atoms due only to MM atoms). The workflow of a MiMiC-enabled QM/MM simulation is shown in Figure 1.

CPMD accesses MiMiC's functions through an API, while the data interaction with GROMACS is through an ad-hoc communication library. The communication library is based on an intercommunicator functionality introduced in MPI 2.0. The communication mechanism implemented is a client-server type where MiMiC is the server and GROMACS is the client. The data communicated through the communication library is row-major (i.e. in the C style). On the contrary, data accessible through MiMiC API column-major (i.e. Fortran style).

2 MiMiC QM/MM data protocol for GROMACS

This section describes the data flow between GROMACS and MiMiC on both initialization and MD loop stages. In the client-server scheme implemented in the communication library MiMiC will be the server and it will always have ID 0. MiMiC will assign an ID to GROMACS. Real and integer numbers are of C double and integer type, respectively. The communication library relies on the fact that the MPI has already been initialized, thus its functions can be called only in between MPI_Init and MPI_Finalize (this is not checked at the present stage).

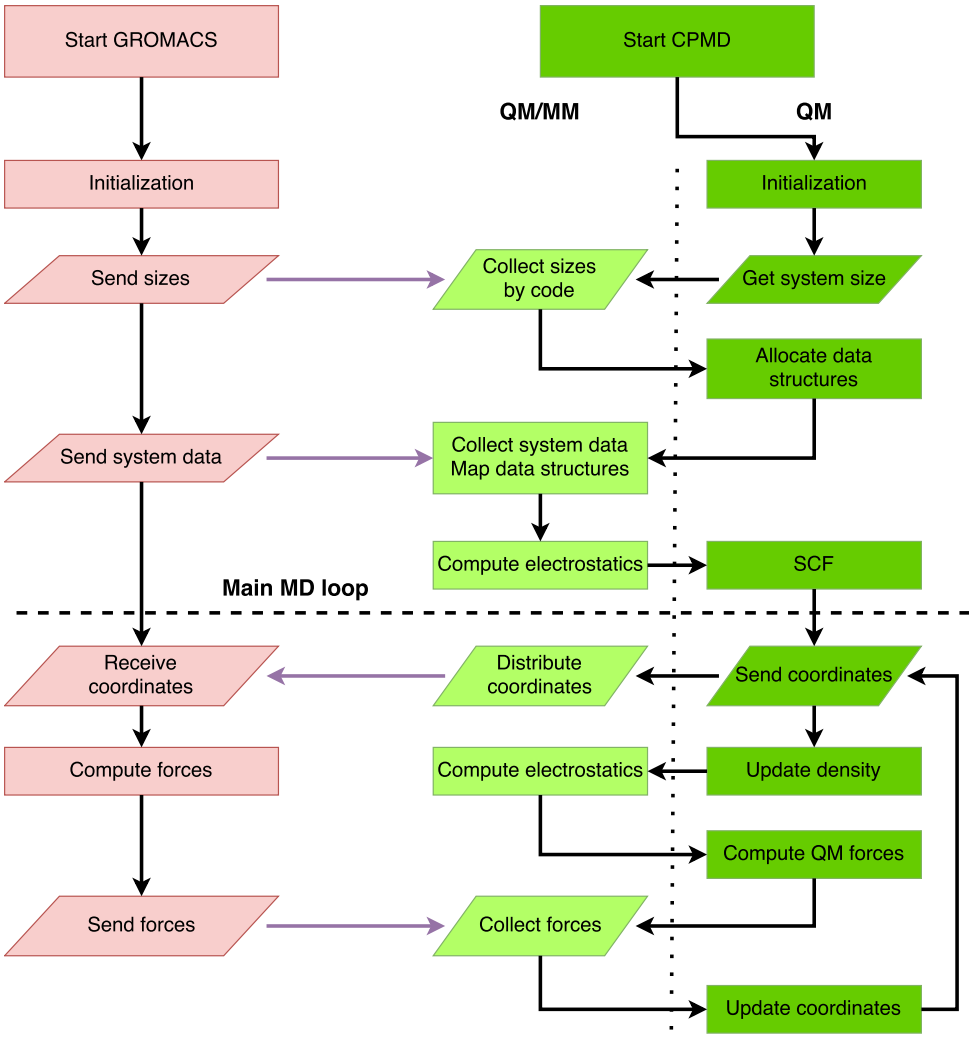

Figure 1 Workflow of a QM/MM run (purple arrows denote the communication through the communication library)

2.1 Initialization

1. Call the MCL_init_client function from the communication library API. A path to the working folder should be passed as an argument to this function.
2. Send total number of atoms (using the MCL_send function).
3. Send the number of atom types
4. Send array of atom types. Atomic species are identified by integer numbers IDs.
5. Send maximum order of multipoles used in MM computations (0 for point charges, only Cartesian multipoles are supported).
6. Send total number of fragments (e.g. charge groups) within the system.
7. Send array with the number of atoms per each fragment in the order of IDs of the fragments.
8. Send the total number of constrained bonds in the system

9. Send the total number of constrained angles
10. Send the flattened array of atoms in bonds (e.g. in case of two bonds involving atoms 1-2 and 2-3 the array should look like [1,2,2,3].
11. Send the equilibrium lengths of bonds (double precision).
12. Send the flattened index array of atoms in angles (e.g. for angles 1-2-3 and 4-5-6 the array will look like [1,2,3,4,5,6].
13. Send the array of equilibrium angle values (double precision).
14. Send flattened (1D) double precision array of multipole values for each atom in the order of atomic IDs (first n positions are for the multipoles of atom 1, second n positions for multipoles of atom 2, etc). Multipoles are ordered from the smallest to highest order, cycling always the rightmost index first, and only including non-redundant components, i.e. monopole (charge), x-, y-, and z-component of the dipole, xx-, xy-, xz-, yy-, yz-, zz-component of the quadrupole, etc.
15. Send double precision array of atomic masses ordered by atomic ID.
16. Send flattened (1D) integer array with the IDs of atoms belonging the each fragments. Atoms IDs have to ordered according to fragment ID, the first n_1 positions are for the atoms of fragment 1, the first n_2 positions are for the atoms of fragment 2, etc.
17. Send the integer array of element numbers
18. Receive the total number of MD steps.

2.2 MD Loop

1. Receive flattened (1D) double precision array of coordinates ordered by atom ID.
2. Send potential energy
3. Send flattened (1D) double precision array of forces ordered by atom ID.

3 Communication library documentation

Functions

- int MCL_init_server (char *paths, char delimeter)
- void MCL_init_client (char *path)
- void MCL_send (void *data, int count, int data_type, int destination)
- void MCL_receive (void *buffer, int count, int data_type, int source)
- void MCL_destroy ()

3.1 Detailed Description

External API of the library.

Provides the C API to call library functions NOTE! ALL API CALLS ARE BLOCKING!!!

3.2 Function Documentation

3.2.1 void MCL_destroy ( )

Destroy the endpoint

3.2.2 void MCL_init_client (char * *path*)

Initialize client endpoint

Parameters

path	string containing the path in the file system to this client

3.2.3 int MCL_init_server (char * *paths,* char *delimeter*)

Customizable init function. Uses arbitrary delimeter char

Parameters

paths	local paths of all clients (needed for addresses sharing)
delimeter	character which is used to extract individual paths

3.2.4 void MCL_receive (void * *buffer,* int *count,* int *data_type,* int *source*)

Receive data from a specified client

Parameters

buffer	buffer to store data
count	number of entities to receive
data_type	type of data to send
source	id of the client which is sending data

3.2.5 void MCL_send (void * *data,* int *count,* int *data_type,* int *destination*)

Send data to specified client

Parameters

data	pointer to the buffer with data
count	number of entities to send
data_type	type of data to send
destination	id of the client to receive data