Reactive Molecular Dynamics Program Project Peport

Andrei Smirnov

May 30, 2009

Contents

1 Introduction

Molecular Dynamics simulations of reacting gaseous mixtures open new possibilities of investigating complex non-equilibrium phenomena occurring on micro-scales, such as chemical kinetics and diffusion in porous media, surface electrochemistry, and micro-scale mixing, to mention just a few. Parameters of operation of a typical fuel cell allow one to apply classical approximations of collision theory, thereby increasing the efficiency of simulations. However, there are several factors that need to be addressed to make these simulations practical. These include: account of detailed elementary reaction mechanisms, representing all internal degrees of freedom (internal energies) of reacting molecules, and the ability to simulate tens of millions of molecules in a reasonable time.

The Reactive Molecular Dynamics simulation program (ReMoDy) developed in the course of this study provided solutions to these problems, which were implemented in the novel time-stepping scheme based on collision detection algorithm and individual molecular time-steps. In this scheme each molecule advances directly to the time of the next collision, which leads to significant acceleration of the time-stepping scheme.

A common drawback of molecular dynamics (MD) simulations is the difficulty in describing reactive interactions including bond breaking and formation, which is essential in modeling of chemical reactions [3, 2]. The current program can perform simulation of complex chemical reactions between the molecules according to prescribed reaction mechanisms. Full account of internal degrees of freedom, activation energies and enthalpies of reactions is implemented.

In addition to these, the program provides the means to specify chemically active surfaces, thus enabling modeling of electrochemical reactions. These capabilities are of a special advantage in modeling of fuel cells, including reactions at micro-pores and at electrolyte interfaces.

2 Physical Model

The physical model of ReMoDy is uses the Collision theory[1] to describe elementary chemical reactions, and Kinetic theory[2] to describe molecular gas dynamics. Reactions occur probabilistically during molecular collisions, with the probabilities determined by the activation energies and reaction probabilities, if more than one outcome for the reaction exists.

The thermodynamic properties of the gas species include heat capacities[3] which are used to distribute thermal energy among molecular degrees of freedom during collisions.

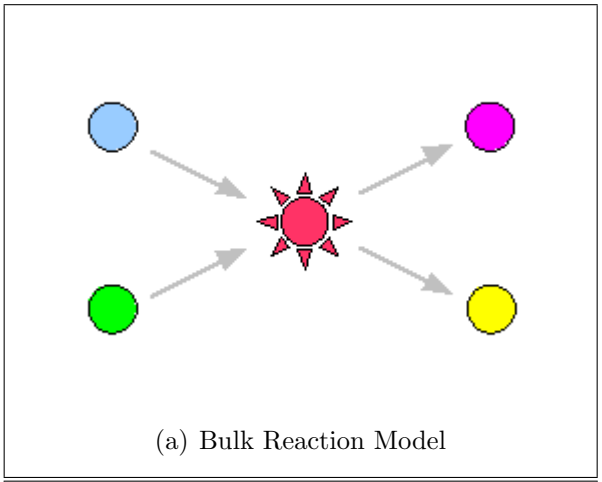
2.1 Chemical Reactions

In simulations of a gaseous phase and gas-solid interactions, each molecule is advanced following the classical laws of conservation of its linear and angular momenta. At the same time a reaction possibility is considered, which determines the transition event for the two species to enter into a reaction with the production of new species or forming a molecular bond. The reaction event occurs when the impact energy of the molecules plus their internal energies exceed the known activation energy for the reaction. Each molecule of a newly formed specie is treated as a rigid body subjected to Newton's laws of motion. Energy distribution among the molecules which result from the collision are calculated in accordance with the molecular degrees of freedom determined by molecule's specific heat constants.

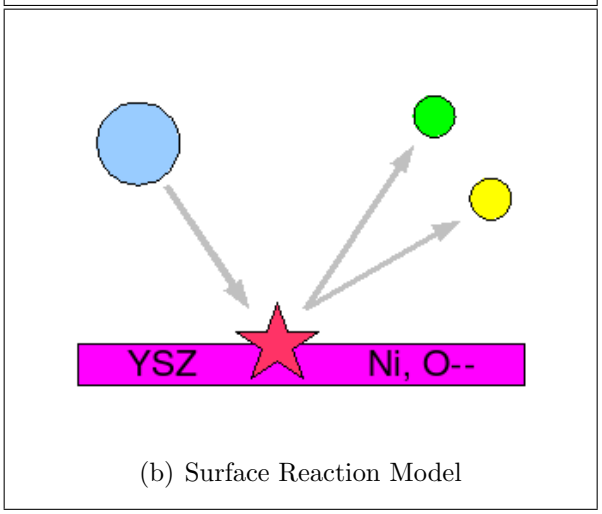
The interaction of the two molecules is modeled through the binary collision approximation whereby only two molecules can interact at a time. The interaction between the molecules can be of two kinds: (1) simple mechanical collision, (2) collision with subsequent chemical reaction. The fact of the collision is detected when the distance between the centers of two molecules becomes less than the sum of their radii, and their relative velocities are directed toward each-other.

Collision is modeled in the center-of-mass (CM) frame of reference. The velocities of the two molecules are first recalculated into the CM frame. The total energy of the two molecules is first calculated as their combined internal energy plus their combined kinetic energy in the CM coordinate frame.

[1] `http://en.wikipedia.org/wiki/Collision_theory`
[2] `http://en.wikipedia.org/wiki/Kinetic_theory`
[3] `http://en.wikipedia.org/wiki/Heat_capacity`



(a) Bulk Reaction Model



(b) Surface Reaction Model

Figure 1: Reaction Model

Chemical reactions are triggered when this energy is in excess of the activation energy for the reaction. In this case the enthalpy of the reaction is added to the total energy. This energy (E) is then redistributed between the degrees of freedom (dof) of the product molecules according to the following scheme[4]:

$$E = KE + IE + enthalpy$$

$$KE_X = \frac{3}{dofk}E$$

$$IE_X = \frac{dofi}{dof}E$$

$$dof = dof_A + dof_B$$

$$dof_A = DOF(Cp_A)$$

$$dof_B = DOF(Cp_B)$$

where X represents any of the two interacting molecules $X = (A, B)$, $dofk$ and $dofi$ are the kinetic and internal degrees of freedom respectively, dof_X are total degrees of freedom (kinetic + internal), and Cp_X is the specific heat of molecule X. The function $DOF(Cp)$ for coputing the degrees of freedom from the specific heat is defined as:

$$DOF = 3 + (2Cv/R - 3)/2 = 3 + Cv/R - 1.5 = Cv/R + 1.5 = Cp/R + 0.5$$

where $Cv = Cp - R$. Since the combined kinetic degrees of freedom are 6, then the combined internal degrees of freedom are calculated as:

$$dofi = dof - 6$$

Then for each molecule the ratio of its internal degrees of freedom to the total internal degrees of freedom is computed as:

$$rdofi_A = (dof_A - 3)/dofi$$

$$rdofi_B = 1 - rdofi_A$$

[4]The notation here is kept close to that used in the code.

The calculations of energy redistribution between colliding molecules is done in the center-of-mass system (CM). The number of kinetic (translational) degrees of freedom of two colliding molecules in CM system ($dofk_{CM}$) will be less than that in the laboratory system, since the CM system already has 3 translational degrees of freedom associated with its center of mass. Thus, the total number of kinetic degrees of freedom in CM system will be:

$$dofk_{CM} = 6 - 3 = 3$$

And the total number of dof in the CM system will be reduced accordingly: $dof_{CM} = dof - 3$. The number of internal degrees of freedom in CM system remain the same as in the laboratory system: $dofi_{CM} = dofi$.

The procedure for calculating new velocities and internal energies in CM system during the collision of two molecules ('a' and 'b') is as follows:

1. The old kinetic energy of the two molecules are computed as:

$$ke_{old} = \frac{ua_{old}^2}{2} + \frac{ub_{old}^2}{2}$$

2. It is combined with the old internal energy to form the new total energy in CM system as:

$$e_{old} = ke_{old} + iea + ieb$$

   where iea, ieb are the internal energies of molecules a and b respectively.

3. The new combined energy (e) of the two molecules is calculated by adding the enthalpy of reaction (h) to the old total energy:

$$e = e_{old} + h$$

4. Kinetic energy in the CM system is computed as the share of the new total energy distributed according to degrees of freedom:

$$ke = \frac{dofk_{CM}}{dof_{CM}}\, e$$

$$ie = energy - ke$$

5. The new velocities of the molecules in the CM frame are then updated from the old ones. The old velocities were computed according the elastic collision scheme between two hard-balls. So, if the energy was released or absorbed during the reaction, and some of it was absorbed into the internal degrees of freedom, the old velocities should be recalculated. The kinetic-energy is redistributed equally between the two molecules:

$$\frac{ma\,ua^2}{2} = \frac{mb\,ub^2}{2} = \frac{ke}{2}$$

which gives:

$$ua = \sqrt{ke/ma}$$
$$ub = \sqrt{ke/mb}$$

This is used to build the ratio of new to old velocities (see interact(...) function in domain.cc):

$$uratio_A = ua/ua_{old}$$
$$uratio_B = ub/ub_{old}$$

which are used to update the velocity vector for each molecule as:

$$u_X(i) = u_X(i) * uratio_X$$

where i=(x,y,z) is the Cartesian direction of velocity vector. The remaining internal energy, ie, is distribued among the internal degrees of freedom of each molecule X as:

$$InternalEnergy_X = rdofi_X\, ie$$

It should be noted, that in the above calculations, the value of kinetic energy was that computed in CM frame of referecne. This means that he actual kinetic energy of the molecule will have a somewhat higher value that what would be expected from an equal distribution of energy among the internal and exeternal degrees of freedom. Nevertheles, this scheme is considered accurate, since the resistribution of energy is indeed taking place in the center-of-mass reference frame, and this will inevitably lead to a higher contribution of energy to kinetic degrees of freedom.

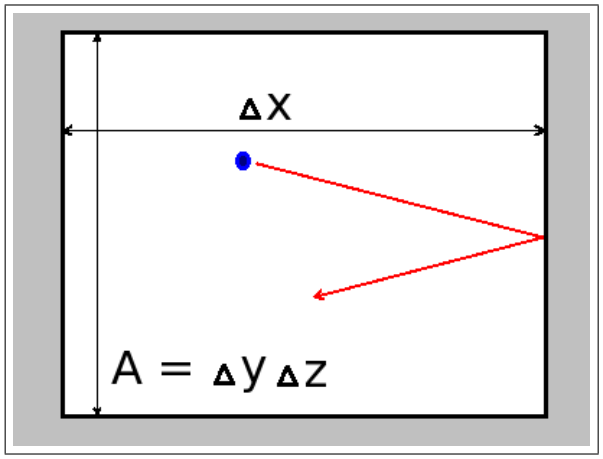


Figure 2: Injection frequency calculations

2.2 Cross-Boundary Species

The code provides the possibility of specifying cross-boundary gases, which can enter the computational domain from the other side of the open boundary (the boundary with the type="open" inside the `<boundary>` tag of the XML input file (see Sec.4.2).

The algorithm uses the density and temperature to calculate the frequency of injection of molecules of specie, s at the boundary. The injection frequency, f_s per unit area, A is computed as

$$f_s = \frac{N}{\Delta t}$$

where Δt_s is the time interval at which a molecule hits the boundary area A, and N is the number of molecules in a volume with the base A and length Δx as shown in the figure:

The time inteval, Δt, between the collisions can be related to the component of velocity of the molecule of species s, in direction x, v_{sx}, as follows:

$$\Delta t = \frac{2\Delta x}{v_{sx}}$$

The number of molecules, N can be related to density, ρ, as:

$$N = \frac{\rho}{\mu}\Delta x A$$

where μ is the mass of one molecule and $A = \Delta\, y\, \Delta\, z$. Thus, the frequency is:

$$f_s = \frac{\rho}{\mu} \Delta\, x\, A \frac{v_{sx}}{2\Delta\, x} = \frac{\rho\, v_{sx}}{2\mu} A$$

3 Implementation

3.1 Collision Detection Scheme

The discretization scheme is based on the combination of the Verlet list method [5] (Sec.3.1.3) with a space-time collision detection scheme (Sec.3.1.2), which enable to significantly accelerate the execution of the method. The standard space-discretization with a variable time-step is also available as an option.

3.1.1 Standard Time-Stepping Scheme

In a standard scheme all molecules are advanced using a single global time step. This time step is not constant and is automatically adjusted during the simulation. It is selected as the minimum time for a molecule to move at the distance of its radius:

$$dt = \frac{d}{v}$$

where 'd' is the on the order of molecule radius and 'v' is the velocity of a molecule. The minimum is sought for all molecules. After each time step the relative positions of all molecules are analyzed to determine if there are any overlaps (collisions) between pairs of molecules.

This choice of time-step, dt, above guarantees that no molecule will miss a collision event. However, since each molecule usually travels much longer distances than its length between the collisions, this scheme leads to unnecessary many steps calculations for an essentially straight trajectory path between the collisions as illustrated in the figure below.

Also, the time step is selected with respect to the smallest molecule, which usually also have the highest velocity. Thus, in the case of a gas mixture, this time-step will be excessively small for bigger and slower moving molecules.

Considering this, and the fact that the molecules travel on straight paths between the collisions, the standard scheme is very inefficient, since it does the majority of computations for molecules undergoing no collisions, but rather traveling on straight paths.

3.1.2 Local Time-Stepping Scheme

The new *local* time-stepping scheme is based on earlier ideas of event-driven MD simulations [4, 1], which uses collision detection. However, in the cur-

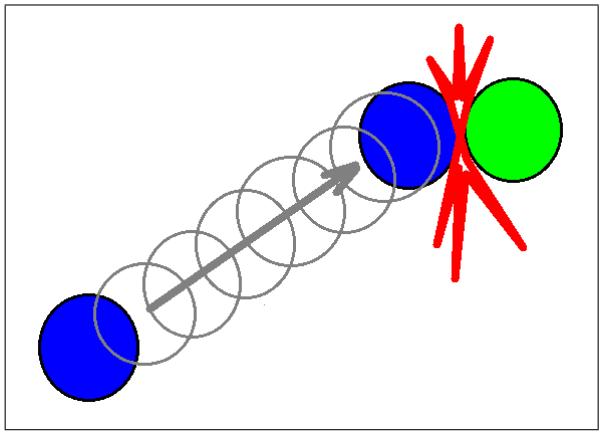


Figure 3: Standard time-stepping involves many unnecessary calculations

rent scheme, instead of moving all molecules with the same time step, each molecule is advanced directly to the site of its next collision, avoiding any calculations on the straight path, as shown in the figure.

Now, instead of using a single global time step for all molecules, each molecule has it's own time, and its own time-step. Thus, instead of a 3D position vector and velocity vector, the kinematic parameters for each molecule include a 4D space-time vector (x,y,z,t), velocity vector, (vx,vy,vz) and the time step, dt. The time-step for each molecule is calculated as the time to the next collision. Thus, one can estimate the time of the next collision event, t_{coll}, for each molecule by adding its current time to its current time step:

$$t_{coll} = t + dt$$

The time-steps of molecules are constantly updated within the loop, where all pairs of molecules are analyzed for possible collisions on the basis of their velocities and radii. If the molecules are found to be heading for a collision, then the time of that collision event is compared to the currently estimated next collision time for each molecule. The time-steps for the molecules are updated if the new collision event is found to occur earlier than both estimated collision events for each molecule. This results in selection of the earliest possible collision events. The collision events also include the collisions with the walls.

In the same loop where the molecular time-steps are updated using collision-detection scheme, the actual collision events are processed for those molecules

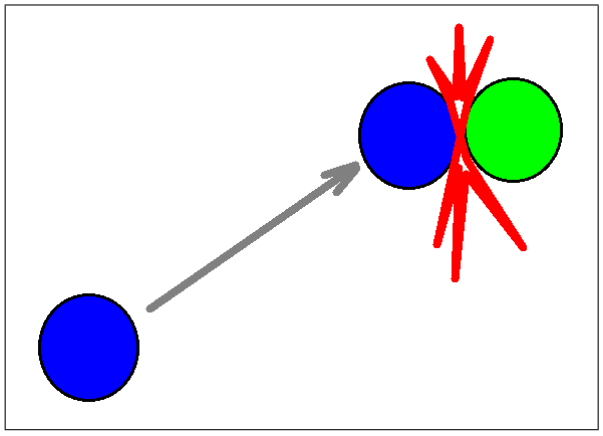


Figure 4: Collision detection moves molecule directly to the site of next collision

which are found to be within the interaction distance from each other. In the course of collision molecules change their velocities in compliance to momentum and energy conservation, as well as exchange their kinetic and internal energies according to the equipartition principle, that is, the total energy is equally distributed among the combined degrees of freedom of two molecules.

After the time-steps of all molecules are updated and collisions processed in the collision-detection loop, the molecules enter the time-advancement loop, where each molecule is advanced by its time-step to its nearest collision event. These two loops are iterated over and over until the termination time for the simulation.

It should be noted, that the described scheme will not work well, if the times of all molecules start deviate from each other by too large a value, causing some molecules to go too far ahead in time compared with other molecules. Thus, it is necessary to periodically synchronize the molecules, by bringing them all to the same time level. This is done by introducing the global time step Δt, such that all the molecules exceeding the next global time level will not move to the next collisions until all the molecules reach that time level. Then the global time level is incremented by the global time-step and the procedure is repeated. This is illustrated in Fig.5, where dt designates an individual time step for a molecule till its nearest collision site, and Δt is the global time step. Each molecule is advanced directly to the place of the nearest collision with another molecule or to the intersection with

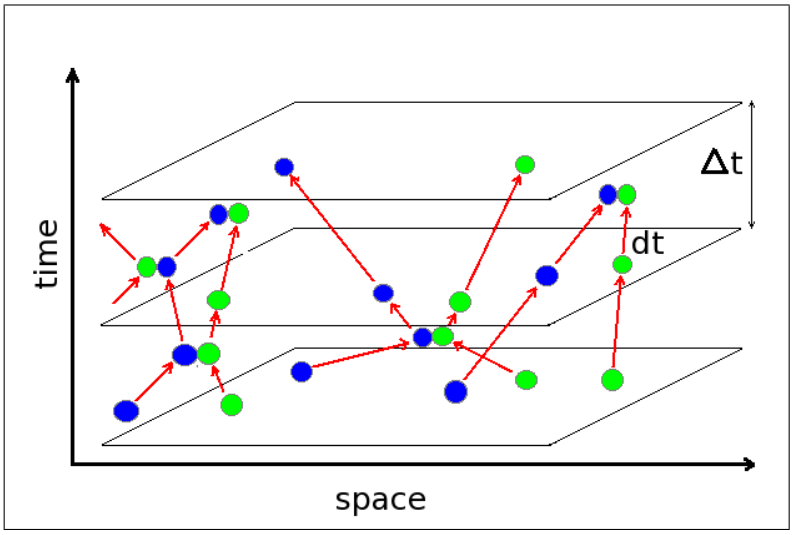


Figure 5: Collision detection with the local time-stepping scheme: all molecules collide at different times, which necessitates the introduction of individual times and time-steps for each molecule.

the next time-level plane. After crossing the time-level plane the molecule is not advanced ant further till the global time changes to the next level. These changes in global time occur when all molecules have advanced to the current global time-level.

The value of the global time step is selected on the order of molecular mean-free-path:

$$dt = \frac{D}{v} \tag{1}$$

where 'D' is on the order of several mean free-path lengths (or inter-molecular distances). This choice guarantees that the molecular collision times will not go out of sync by too much to cause miscalculations in collision events.

The flow-chart of the main calculation loop is shown in Fig.6. There are two nested loops: the outer loop iterates over global time, following global time-steps, while the inner loop goes over individual molecular time-steps. This nesting of the loops guarantees that the processing of the molecules that crossed the next global time-level will be postponed till all the molecules have

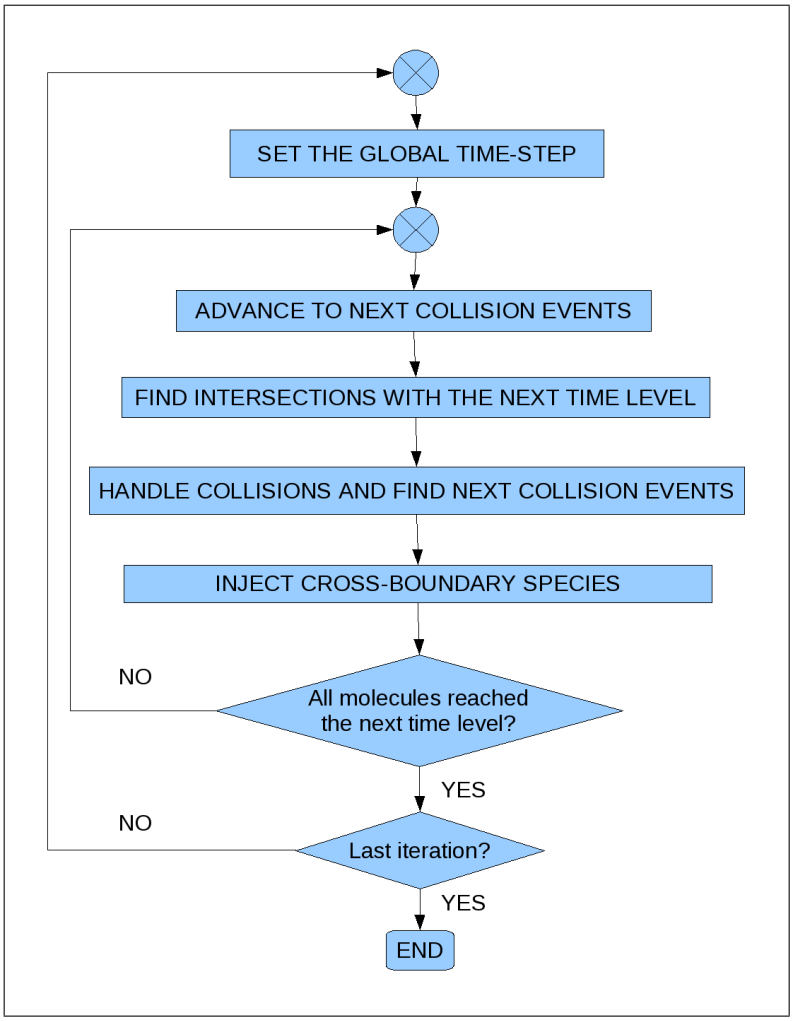


Figure 6: Flowchart of the main loop.

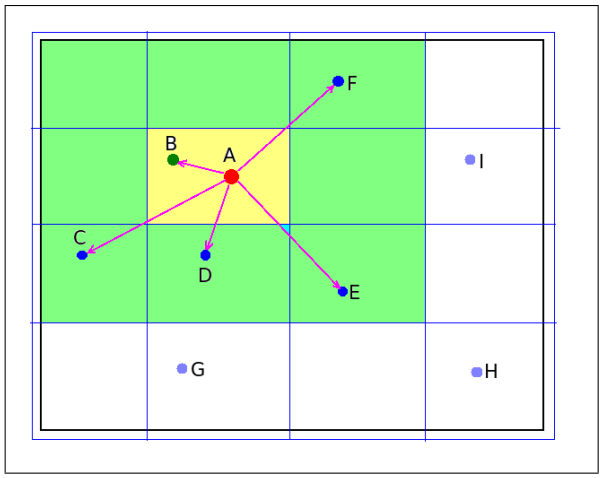
crossed that time level. This periodic synchronization of molecules in time is necessary to achieve effective capturing of all the possible collision events. All the boxes shown inside the loops indicate actions applied to the collection of all the molecules as a whole. In particular, after the global time-step is determined on the basis of (1) all molecules are advanced to their collision events using their individual time-steps. At the same boundary conditions are applied to the molecules crossing the domain boundaries. Depending on the type of the boundary, these can result in molecule being removed from the domain, bouncing back from the wall, reacting at the boundary with a resultant chemical transformation, etc.

After that (third block) the individual time-steps of molecules are reset to the maximum allowed by the current global time-step. Next all molecules currently in the state of collision are processed, resulting in possible changes of their velocities, internal energies, or even in chemical transformations to different molecular types in case when reactions took place. After that the new collision events are detected for each of the molecule. In the next step injection of molecules from across the domain boundaries is considered, where the 'open' type boundaries are specified. Such boundaries usually represent the open-end of the domain, which is physically a continuous region of gas. Example of this situation is when the computational domain is chosen to be smaller than the actual physical region.

Even though the current scheme is still relying on a global time-step, this time step is by far larger than that used in the standard time-stepping scheme discussed above. This is because in the standard scheme the global time-step is based on molecular size, and in the current scheme it is based on inter-molecular distance, which is far larger than the size of the molecule: $D >> d$. This leads to significant speed-up of calculations compared to the standard time-advance scheme.

This approach also presents a major improvement over the standard event-driven schemes [4, 1]. Indeed, the time-step in that case is selected each time on the basis of the nearest collision event for all the molecules in the domain. This time will be inversely proportional to the number of molecules, and at some point will be close to the time step used in a standard time-stepping scheme described above (see. 3.1.1). This is because with the large enough number of molecules, there will always be two molecules close enough to collision with each other at any given time. The new scheme does not suffer from this inverse dependence of the time step on the number of molecules. In this case each molecule moves directly to the next collision

Figure 7: Grid of Cells for Local Interaction Acceleration



point with its own collision partner, but not to the nearest collision point among all the molecules.

3.1.3 Verlet Lists

ReMoDy uses interaction acceleration scheme based on linked-cell technique, which is a variant of the Verlet list method [5]. In this method the whole computational domain is divided into box-shaped cells. Only interactions between the molecules from the same or adjacent cells are considered (see Figure).

The size of the grid cells is selected by optimizing the execution speed, and is usually on the order of several mean-free-paths. The method enables to achieve near linear dependence of execution time on the number of molecules ($\sim 5/4$). In contrast, looping over all molecules makes execution time proportional to the square of the number of molecules, which makes that impractical for large number of molecules.

3.1.4 Multi-Processor Implementation

The code can run in parallel on multi-core workstations. It uses the shared-memory OpenMP library to distribute processing of the molecules time-advancement and interaction loops among the available CPUs.

The two time-critical loops which run in parallel are time-advancement loop, implemented in subroutine Domain::step(), and interaction loop implemented in Domain::interaction() routine. Before the loops are entered the molecules are indexes sequentially. Each OpenMP thread selects the molecules which has index satisfying the criterion:

$$mod(i_{mol}, N_{threads}) = i_{thread}$$

where $mod(*,*)df$ is division by modulus. This way all molecules are distributed equally between active threads. The number of threads is selected as the maximum between the available processors, and the pre-defined constant `nthreads`.

In addition to this, in the interaction loop, the locking mechanism is used to prevent simultaneous processing of the same molecules by more than one processor. This can happen because the interaction procedure considers all pairs of molecules, which is done in a double-looping over all molecules. Thus, the selection of molecules from the primary loop, using the mechanism above will not prevent the possibility of simultaneous processing of the same molecule from the secondary loop by more than one processor. The locking mechanism operates by introducing the 'is-locked' flag for each molecule. Inside the nested loop the molecule is considered only when its state is not locked. If this is the case, then the molecule is temporarily locked while its interaction with the primary molecule from the main loop is being processed.

3.1.5 Dynamic Lists

The main data-structures holding molecules are dynamic lists: **Collection** and **Container**. *Collection* is a double-linked list of fixed number of items, with two pointers for each item, pointing to the next and the previous item. The list consist of two parts: active and dead. Each part forms a loop, such that following next or previous pointer from any element across the list will lead back to the same element. The procedure of moving elements between the active and dead parts is very simple, involving only several pointer reassignment operations. The figure below illustrates the operation of removing the active molecule from the computational space and assigning it to the pool of "dead" molecules.

The reverse procedure of "resurrecting" dead molecules and introducing them into the active pool is done in the same manner. This technique enables

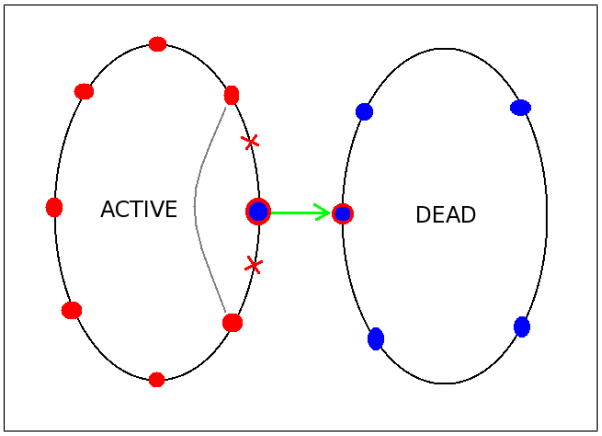


Figure 8: Collection Class

to avoid expensive memory allocation/deallocation operations, and save time on looping over the list of molecules, since all dead molecules are not in the active list, and are completely ignored by the looping procedure. Thus, no conditional if-statements are necessary, and shorter list sizes can be used.

The *Collection* class is convenient for the dynamic storage of a single collection of items, and is used to store all the molecules in the domain.

Container is a variable size list of pointers to items. Like a *Collection* it is also a double-lined list with the next and previous pointers for each item. But unlike a *Collection*, the *Container* all container lists share the same pool of dead items, or pointers, which can point to any item. Each container list can acquire items from the pool or return them back to the pool.

The *Container* list is convenient to store multiple lists of items where the number of items constantly change. Container lists are used in the implementation of the linked-cell method described below.

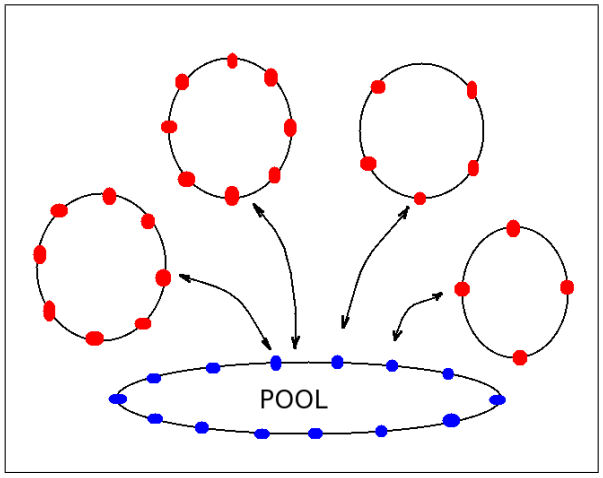


Figure 9: Container Class

4 Program Usage

4.1 Installation

4.1.1 Step 1: Extracting the archive

If you have the archive file, like remody.tbz (tar-bzipped) or remody.tgz (tar-gzipped) then files can be retrieved into a current directory as:

```
tar cvjf remody.tbz
```

or

```
tar cvzf remody.tgz
```

respectively.

4.1.2 Step 2: Compiling

To compile the executable on Linux run make from the remody root directory, where the makefile was saved. Note that the subdirectories src/, run/, and obj/ should be also present.

4.1.3 Step 3: Running

The executables are saved in the run/ directory. It should also contain the example remody.xml and demo.xml files, as well as an initial empty input file empty.dat.gz. To run the program with an OpenGL window active (slow, good for debugging), one can use the command:

```
./view -f demo.xml empty.dat.gz
```

This will start the program with the initial parameters read from the demo.xml file and the initial data read from empty.dat.gz file. On startup the program will open the window. One can point at the window with the mouse and press 'f' key to show the frame and 'r' key to run the simulation. Alternatively, one can use 's' key to run iterations step-by-step. Other key functions are described by pressing the '?' key.

To run the program in a batch mode without OpenGL output, one can use this command:

```
./job -f demo.xml empty.dat.gz &
```

This will start the run. To change the parameters of the job, one should modify the input xml file accordingly (demo.xml, remody.xml, etc.). A link with the name remody is also set to one of the executables, like

```
ln -s ./job remody
```

4.2 Configuration

4.2.1 Input File

Both OpenGL-based and batch-mode executables read the configuration xml file. By default the file will have the same name as the executable, for example: job.xml or view.xml. This can be overwritten with the '-f' option. Two example files `remody.xml` and `demo.xml` are provided with the distribution.

The default name of the config file is $< progname >$`.xml`, where `progname` is the name of the executable, such as `view`, `job`, or `remody`. One can override the default name by using -f command line option, like:

```
remody -f syngas.xml
```

4.2.2 Simulation Parameters

The sections of the input file should be enclosed in XML tag pairs, like $< tag > \ldots < /tag >$. where the tags are as follows:

- **iterations**: specifies the integer number of iterations the code should run
- **time**: several parameters of physical execution time in seconds, such as **start**=start time, **end**=end time, and **step**=time step. The code will run until either the number of iterations as specified in the **iterations** section or the end time is reached.
- **molecules**: integer number equal to the maximum number of molecules that the program will be able to handle. It will determine the memory allocated at the beginning of execution.
- **species**: this section lists all chemical species and reactions between them. Correspondingly, each specie is described in **specie** section as:

- **specie**: includes such parameters as **mass** in Atomic units [au], **size** in nanometers [nm], and specific heat, **cp** in kJ/(mol*K). The **specie** tag should also include an attribute **id**, identifying a specie chemical formula, such as CO2, H2O, etc.
- **reaction**: reaction tag has two attributes: **reactants** and **products**. The reactants tag should contain two species identifiers. Since only elementary (binary) reactions are considered, there should be exactly two identifiers for reactants. Each of the identifiers should correspond to one specie identifiers listed in the list of species. The products attribute should consist of one or two identifiers of reaction products. Also in the reaction section the following parameters are specified:

activation Determines the reaction activation temperature in K,

probability Determines the probability of reaction outcome as given by the

products This parameter should always be 1.0 if there is only one reaction with the given reactants. In case of several reactions with the same reactants, but with different products, this parameter should indicate the probability of this particular branch with given reaction products.

enthalpy The enthalpy of reaction in kJ/mol.

- **domain**: The **domain** section consists of the set of parameters describing the geometry and physical properties of the modeled media inside the computational domain, including:
  - **type**: specifies the geometry. Currently only **box** type is supported.
  - **grid**: Specifies parameters of the rectangular grid used in segmentation algorithm for accelerating the interaction scheme. In particular, the **cellsize** parameter determines the size of the grid cell. This size should be selected as small as possible but no less than twice the size of the largest specie. Decreasing the cell size will speed-up code execution in better than linear proportion of the cell-size (but no better than quadratic). At the same time it will increase memory utilization in proportion to it's 3-rd power. It is not recommended to increase the memory utilization above 90

- **energy**: Only used for Lennart-Jones type potentials, currently under development.
- **bounds**: specifies spatial bounds of the computational domain as: xmin xmax ymin ymax zmin zmax.
- **bulk**: this section specifies thermodynamic properties and gas composition inside the bulk of the domain. In particular, **temperature** is given in Kelvin [K], and for each specie, its **density** is specified in [kg/m$\hat{3}$].
- **boundary**: each boundary of the domain contains the description of thermodynamic properties and gas composition on the other side of the boundary in the same format as for the bulk of the domain. In addition to these, the boundary tag should have the boundary identifier **id**-attribute, such as "top", "bottom", "right", "left", "front", and "rear". Also additional boundary tag is **type**, which can be one of: "open", "elastic", and "periodic". In the case of open boundary the molecules can freely cross the boundary, in which case they will be removed from the domain. In case of elastic boundary the molecules will bounce from the boundary like from an elastic wall. For periodic boundaries, the molecules crossing the boundary will reappear from the opposite boundary. Boundary description can also contain the list reactions, between the boundary species given in the same format as in the **species** section.

- **gui**: the gui section describes parameters related to graphical output used when running an OpenGL based version with a visual window output.
  - **translation**: initial translation of the scene.
  - **vector**: parameters for displaying vectors.
  - **frame**: parameters for displaying a domain frame.
  - **mesh**: parameters for displaying mesh. In particular the **node** tag specifies the parameters for displaying particles, or molecules, such as using points or spheres (**type**), etc. Note that using 'spehere' for type may significantly slow-down simulation in GUI mode.

- **xterm**: if set to 1 this parameter will force terminal dumping of certain parameters, like temperature, energy, number of molecules, and species concentrations after every time-step. This can be useful if the time-dependence of concentrations and other parameters needs to be retrieved. In the last case the output can be redirected to a file, which can be processed with the **readlog** utility

4.3 Output

There are three data output methods that can be used:

1. Compressed snapshots of coordinates, velocities, and internal energies stored in gzipped ASCII format files. The files are dumped at time intervals specified in the input XML file under tag: $< time >< output > ... < /output >< /time >$ with the file names in format: $< task > - < step >$.dat.gz, where ¡task¿ is the name of the task, which is the same as the name of the executable file (job, view, or any other), and the $< step >$ is the sequence number of the dump, which is automatically incremented for each subsequent dump. The output is performed by `save()` function of Domain class implemented in `domain.cc` file. The date in the output file has the format:

   ```
   NAME X Y Z U V W E T DT
   ```

   where NAME is the name of the molecule, such as O2, CO2, H2O, etc., (X,Y,Z) and (U,V,W) are the position and velocity vectors, E is the internal energy of the molecules, and T and DT are the time and time-step for the molecule. A histogram of molecular distributions can be obtained using the `hist` utility (see Sec.4.5).

2. Terminal output of time sequence of average quantities, such as the number of molecules (N), average temperature (T), kinetic (K), internal (I), and total energy (E) per molecule and per degree of freedom. In addition to that is also outputs the physical time (Time) in nanoseconds and memory utilization (Memory). The terminal output can be toggled with $< xterm > 0, 1 < /xterm >$ flag in XML file. The output can be captured into a file using redirect command, like:

   ```
   ./job > log.log &
   ```

and post-processed using the Python `readlog.py` utility. The terminal output is performed inside the `Domain::run()` function at every designated time interval.

3. The last method is to dump the graphics window in X-Window dump format (XWD) and then convert into one of the common graphics formats, such as png or jpg using standard convert utilities, such as ImageMagic's convert. See also Sec. 4.4.

4.4 Graphics User Interface

ReMoDy can be compiled in two versions: with and without a graphical user interface (GUI). The former is used to visualize the simulation progress on a small set of data, while the latter is to perform larger-scale simulations where the concurrent visualization is not possible.

After the GUI-enabled version with the default name *view* is started, a display window pops up, showing the computational domain (Fig.10). The initial parameters of the scene can be set in the config file (see Sec. 4.2. The configfile can be reloaded by making the display window active and hitting the 'l' key for 'load'. The keyboard control of the simulation has two modes: *keystroke*, and *commandline*. This is similar to vi text editor in UNIX. Initially the keystroke mode is active.

In the keystroke model the single-key-strokes can be used while the display window is active. These include:

- **ESC**: exit. The same as 'q';
- **?**: show help.
- **:**: switch to a terminal command mode (similar to vi). The same as '.';
- **a**: show/hide the axes.
- **b**: show/hide boundary faces of the mesh (not used when mesh is absent).
- **c**: switch the color-scheme.
- **f**: show the box frame.
- **G**: show all the mesh edges.

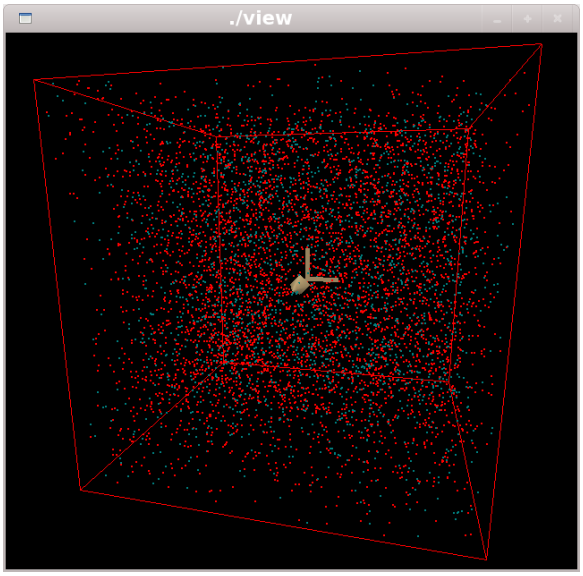


Figure 10: Snapshot of a GUI

- **g**: show only the boundary mesh edges.
- **l**: read configuration file.
- **m**: display the menue on the console.
- **N**: show mesh nodes.
- **n**: show boundary vertexes.
- **r**: start/stop the continuous run.
- **s**: perform one step of the simulation. Analogous to '+'.
- **v**: show boundary vectors.
- **w**: dump current window into a xwd file.
- **Z**: zoom-in view.
- **z**: zoom-out view.

The command mode is switched from the keystroke mode by hitting the ':' or '.' keys. To switch back from the command mode one has to enter an empty command, that is, press ENTER, while the command line is empty. In the command mode, the command line appears in the terminal windows from which the program has started. The commands one can enter in the command line include:

- **bg**: go to background run mode. During this mode the all controls are disabled.
- **cs**: switch the color scheme.
- **fg**: go to foreground mode.
- **?, h, or help**: show help.
- **r**: run a number of iterations.
- **sp**: toggle between spheres and points in particle (molecule) display.
- **wd**: start/stop window dump every fixed number of iterations.
- **dw**: dump current window picture into a xwd file.
- **e**: exit the program. Also saves current data.
- **q**: quit without saving.
- **.**: quit the command mode. Also hitting ENTER with empty command line.

4.5 Utilities

4.5.1 Pre-Processing

The `init.cc` utility can be used to generate the initial uniform distribution of molecules. For example, the command:

```
/init init
```

will generate file init.dat.gz which can be used as input to remody, like:

```
/job init.dat.gz
```

To se the parameters one should edit the init.cc file and compile the program as:

```
g++ -lz init.cc -o init
```

4.5.2 Post-Processing

1. Script `readlog.py` is used to parse the output file generated by the remody job, when run as:

   ```
   ./job -f remody.xml init.dat.gz > job.log \&
   ```

   The above command will start the `job` process, which will dump screen output into the job.log file. Note that the 'xterm' tag in the config file remody.xml should be set to 1 as $< xterm > 1 < /xterm >$ to generate the log file in the appropriate format. To extract the sequence of molecular concentrations of, say, hydrogen (H2) for the whole domain at different times, one should run the following script:

   ```
   python readlog.py H2 < job.log > H2.dat
   ```

   File `H2.dat` will contain time distribution of concentration of H2.

2. Program `hist.cc` is used to parse the dump file generated by remody when run as

   ```
   ./job -f remody.xml job-02.dat.gz \&
   ```

   where `job-02.dat.gz` is the dump file from the previous run used to restart the run. Usually the run will generate the output dump files at regular time intervals as specified in the remody.xml file inside the <time><output>...</output><time> tag. The output files will have names as: `job-XX.dat.gz` where XX is the next consecutive number of the output dump. To retrieve a histogram of molecular distributions inside the domain, one can run the hist utility as:

   ```
   gunzip -c job-XX.dat.gz | grep ^H2 | grep -v ^H2O
                           | cut -f 3 | hist > H2.dat
   ```

This will produce file H2.dat with the column of numbers corresponding to the number of molecules for different slices of the domain. Note, that the repeated `grep` functions were used to capture only the lines beginning with H2 and exclude those beginning with H2O. Other more efficient line retrieval can be arranged using awk, perl, or python.

To change the domain limits, slice width, the number of slices, and the direction of slicing one should edit the hist.cc file accordingly and recompile it as:

```
g++ -lm hist.cc -o hist
```

5 Conclusions

The moleculear dynamics simulation code ReMoDy was designed to enable efficient simulation of reacting gaseous mixtures. It is especially suitable for simulating non-equilibrium and transient processes in micro-pores of a fuel cell. The important features of the code include:

1. Account of both reactions in the bulk and with active surfaces.
2. Account of molecular internal energies and reaction activation energies in chemical reactions.
3. New space-time discretization scheme with collision detection.
4. Verlet list method for interaction accelaration.

These features enable to simulate tens of millions of molecules on a single workstation in a reasonable time.

References

[1] A. Donev, S. Torquato, and F.H. Stillinger. Neighbor-list collision-driven molecular dynamics simulation for non-spherical hard particles. *Journal of Comp. Phys.*, 202:737–764, 2005.

[2] J.M. Haile. *Molecular Dynamics Simulations: Elementary methods*. John Wiley & Sons, New York, NY, 1997.

[3] D.C. Rapaport. *The Art of Molecular Dynamics Simulation*. Cambridge University Press, New York, NY, 1997.

[4] D.C. Rapport. The event-driven approach to n-body simulation. In Y. Hiwatari and M. Isobe, editors, *Symposium on the 50th Anniversary of the Alder Transition*, 178, pages 5–14, 2009.

[5] L. Verlet. Computer "experiments" on classical fluids. *Phys. Rev.*, 159:98–103, 1967.