1 Introduction

ProtoMol is an "object-oriented, component based, framework for molecular dynamics (MD) simulations". Written in C++, ProtoMol can simulate classical mechanical systems with very high accuracy and efficiency. For a detailed review of ProtoMol, see Ref [**?**]. In the remainder of this manual, we first explain the ProtoMol installation process, then we detail the usage of ProtoMol and its add-ons. At last, we describe the processing of writing customized add-ons for future simulation studies.

2 Obtaining ProtoMol

The original ProtoMol is hosted at SourceForge. The download address for linux version is `http://sourceforge.net/projects/protomol/files/ProtoMol/Protomol%203.3`

ProtoMol is an open-source and cross-platform software hosted at SourceForge. Considerable modifications have been added for simulations in this dissertation, so it is highly recommended to have the modified version to start with, which can be found at github `https://github.com/kuangchen/ProtoMol`. Any git tool should be able to clone this online repository into local copies.

3 Compiling ProtoMol

To compile modified ProtoMol, a C++ compiler compatible with C++11 standard is needed. Modified ProtoMol also depends on two external library, Lua v5.2 and HDF5 v1.8.10, which can be found in the package repository (for Linux system) or downloaded online (for Windows system) at the following URL,

- Lua `http://code.google.com/p/luaforwindows/`
- HDF5 `http://www.hdfgroup.org/HDF5/release/obtain5.html`

The compilation and installation process is automated by CMake tool. On Linux system, installation is accomplished by the following commands,

```
# Remove previous CMakeCache and CMakeFiles folder
rm -rf CMakeCache.txt CMakeFiles

# Turn on build_lapack switch and set the build_lapack_type to lapack
cmake -DBUILD_LAPACK=ON -DBUILD_LAPACK_TYPE=lapack

# Tell cmake to generate Makefiles
cmake .

# Install ProtoMol into system folder
sudo make install
```

The basic usage of CMake is documented at `http://www.cmake.org/cmake/help/documentation.html`. To toggle other build switches (cluster computing, $\cdots$), see CMakeLists.txt for detail.

4 Basic Usage

To start a simulation a user needs to provide a configuration file (in plain text) that defines simulation initial conditions, integrator, outputs, and other properties of simulation. These entries are organized in the format of self-explanatory "keyword - value" pairs. The definition of integrators and forces have their own special syntax. A sample configuration file is presented below, with the meaning of each entry explained in the comment line. To see a full list of keywords, please see Ref [**?**, **?**].

```
# Number of steps in simulation
numsteps 100000000

# The number assigned to the first step
firststep 0

# Random Number seed
seed 11

# Initial temperature of particles
temperature 1e4

# Simulation cell size
cellsize 5000000

# Boundary Conditions
boundaryConditions vacuum

# Cell Manager
cellManager Cubic
exclude none

# Initial position and velocitiy definition
posfile ion_neutral_cooling_ini_pos_32.xyz
psffile ion_neutral_cooling_32.psf

# Par file definition
parfile ion_neutral_cooling.par

# Output Setting
outputfreq 10000

# Add IonSnapshot as the output
IonSnapshot ss.lua

# Integrator Setting
integrator {
  # 0th level integrator
  level 0 LeapfrogBufferGas {
    timestep 1e8
```

```
    filename buffer_gas.lua

    # Add Coulomb force between ions
    force Coulomb
    -algorithm NonbondedSimpleFull

    # Add ion trap force
    force LQT
    -lqt_filename trap.lua
  }
}
```

5 Using ProtoMol Add-Ons

A set of new forces, integrators and outputs have been added specifically to simulate ion trap dynamics, whose usage are detailed below.

5.1 New Forces

Rf-trapping force
Rf-trapping force is defined with the following syntax,

```
force LQT
    -lqt_filename your_file
```

where `LQT` is the name of the force, and the argument of `-lqt_filename`, i.e. `-your_file` is a LUA file that defines an ion trap.

The trap definition declares a variable named `trap`, a LUA table that contains the mass of the ion in the trap, the physical dimensions of the trap and the trap voltage. A sample trap definition file is given below, where the meanings of each filed are the same as in Eq. **??** and their units are given in the comment.

```
trap = {
  m = 173,          -- AMU
  r0 = 12e-3,       -- m
  z0 = 21.5e-3,     -- m
  v_rf = 175/2,     -- V (Note arithmetic operations can be done in Lua file)
  v_ec = 10,        -- V
  eta = 0.1275,     -- 1
  omega = 300e3     -- Hz
}
```

Harmonic static trapping force
The harmonic static trap is introduced to simulate ion dynamics under the secular approximation, in which ions are confined by a harmonic potential with the trapping frequency equal to ion's secular frequency. Harmonic trapping force is defined with the following syntax,

```
force HarmonicTrapForce
    -ht_def your_file
```

where `HarmonicTrapForce` is the name of the force, and the argment of `-ht_filename`, i.e. `-your_file` is a LUA file that defines an harmonic trapping force.

The trap definition declares a variable `trap`, a Lua table that contains a single field `freq`, which in turn contains three fields `x, y, z`. A sample trap definition file is given below,

```
trap = {
  freq = { x = 39.63e3 * 2 * 3.14159,
           y = 39.63e3 * 2 * 3.14159,
           z = 39.63e3 * 2 * 3.14159
         }
}
```

The meanings and units of these fields are given in the following Tab. 1.

Field	Unit	Meaning
`trap.freq.{x,y,z}`	Hz	$\omega_{x,y,z}$ as defined in $V = \frac{1}{2}(\omega_x^2 x^2 + \omega_y^2 y^2 + \omega_z^2 z^2)$

Table 1: Harmonic trap definition syntax.

Damping force

The damping force, *i.e.* $\vec{F} = -b\vec{v}$, is introduced to emulate the laser-cooling force, which quickly removes ions' kinetic energy, and cause the ions to form Coulomb crystal. Damping force is defined with the following syntax,

```
force DampingForce
    -damping_def your_file
```

where `DampingForce` is the name of the force, and the argment of `-damping_def`, i.e. `-your_file` is a LUA file that defines a damping force.

The trap definition declares a variable `damping` that contains the damping magnitude, and when the damping is on. A sample damping definition file is given below,

```
damping = {
  coeff = 3e-22,
  t_start = -1,
  t_end = 0.01
}
```

The meanings and units of the fields in the `damping` table are given in Tab. 2.

Field	Unit	Meaning
`coeff`	?	damping coefficient b, defined by $\mathbf{F} = -b\mathbf{v}$
`t_start`	s	start time. Laser is on only when `t_start` $\leq$ t $\leq$ `t_end`.
`t_end`	s	end time

Table 2: Damping force definition file

5.2 New Integrators

Buffer-gas Leapfrog Integrator V2
A leapfrog integrator is introduced to simulate random collisions with ultracold neutral atoms. It is defined with the following syntax,

```
level X LeapFrogBufferGas2 {
  timestep 1e8
  filename your_file
}
```

where `LeapFrogBufferGas2` is the name of the integrator, and the argument of `filename`, i.e.`your_file` is a LUA file that defines the buffer gas integrator, by declaring a table variable `neutral`. A sample buffer-gas leapfrog integrator definition file is given below

```
neutral = {
  mass = 40,
  polarizability = 50,
  temperature = 5e-3,
  density = 1e16
}
```

The meanings and units of the fields in the `neutral` table are given in Tab. 3

Field	Unit	Meaning
`mass`	AMU	Mass of neutral atom
`polarizability`	10^{24}cm^3	Polarizability of neutral atom
`temperature`	K	Temperature of neutral atom
`density`	m^3	Density of neutral atom

Table 3: Buffer-gas leapfrog integrator definition file

5.3 New output

Ion Snapshot
At each simulation time step, ions' positions and velocities form a *frame*. A consecutive series of *frames* form a *snapshot*. For analysis at a later time, an end user might want to record an array of snapshots starting at different time t_i $(i = 1, 2, 3, \cdots)$ during the simulation, and for simplify, let's suppose every snapshot contains the same number of frames. It is defined with the following syntax,

```
IonSnapshot your_file
```

where `your_file` is a LUA file that defines the ion snapshots.

The `IonSnapshot` definition declares a table variable `ss` that contains the number of snapshots, and frames in each snapshot. A sample `IonSnapshot` definition is given below,

```
start_time = {}
```

```
for i=1, 50 do
    start_time[i] = (i-1) * 2e-2
end

ss = {
  num_frame = 250,
  start = start_time
  dir = ./
}
```

The meanings and units of the fields in the `ss` table are given in Tab. 4

Field	Type	Unit	Meaning
`num_frame`	integer	1	N_f, if set to -1, then N_f is set to number of steps in the longest secular periods in x, y, z three directions
`start`	table	sec	t_i, i.e. the start time of each snapshot
`dir`	string	N/A	The name of directory where snapshot files are stored

Table 4: Snapshot Definition

All the snapshot files are stored under the same directory, specified by the `dir` keywords in the definition file. They are named by the pattern snapshot_`i`.hd5, where i ranges from 0 to $N_s - 1$.

The snapshot files are stored in HDF5 format, for its compact size and support by major programming languages, including C/C++/Python/Matlab. For an overview of HDF5 file, visit its official website at `http://www.hdfgroup.org/HDF5/`. Shown in Fig. **??**, every HDF5 file has a hierachical data structure, similar to the file system in an operating system. In addition to N_f consecutive frames, a `config` header is also included which consists of auxillary information intended to make the snapshot file self-inclusive. The meanings of various fields in `config` is explained in Fig. 5.

Field	Type	Unit	Meaning
`start`	float	s	Start time of the snapshot
`fps`	integer	1	Number of frames per second
`numFrames`	integer	1	Number of frames in the snapshot
`numAtoms`	integer	1	Number of atoms in the simulation
`numFrameperMM`	integer	1	Number of frames in one micromotion period

Table 5: Snapshot config header

6 Writing ProtoMol Add-Ons

ProtoMol makes use of object-oriented programming language, and it has been carefully designed to ensure extensibility for new model encapsulation. How to add new modules to ProtoMol is beyond the scope of this thesis.

For interested readers, it is recommended to read Ref. [**?**], combined with the newest ProtoMol source code to