MAGICS Workshop
RXMD Hands-on Session

Ken-ichi Nomura, Sungwook Hong, Ankit Mishra, Pankaj Rajak, Subodh Tiwari

November 12, 2018

Materials Genome Innovation for Computational Software

U.S. DEPARTMENT OF ENERGY | Office of Science

Basic Energy Sciences

Outline

- Software Setup
- Create Initial Configuration
- RXMD Input Parameters
- Hands-on : MoO_3 Self Reduction
- Hands-on : SiC Nanoparticle Oxidation

USC HPC Environment Setup

- Your home directory is under /home/rcf-40 but home directory has 1GB quota.

```
$ pwd
/home/rcf-40/magics60
```

- Each magics account has access to "staging" directory. The staging directory has fast disk access and no disk/file quota. We are going to use the staging directory for the rest of hands-on sessions.

```
$ cd staging
```

- Use **pwd** command, make sure you are in /staging/magics18/${USER}.

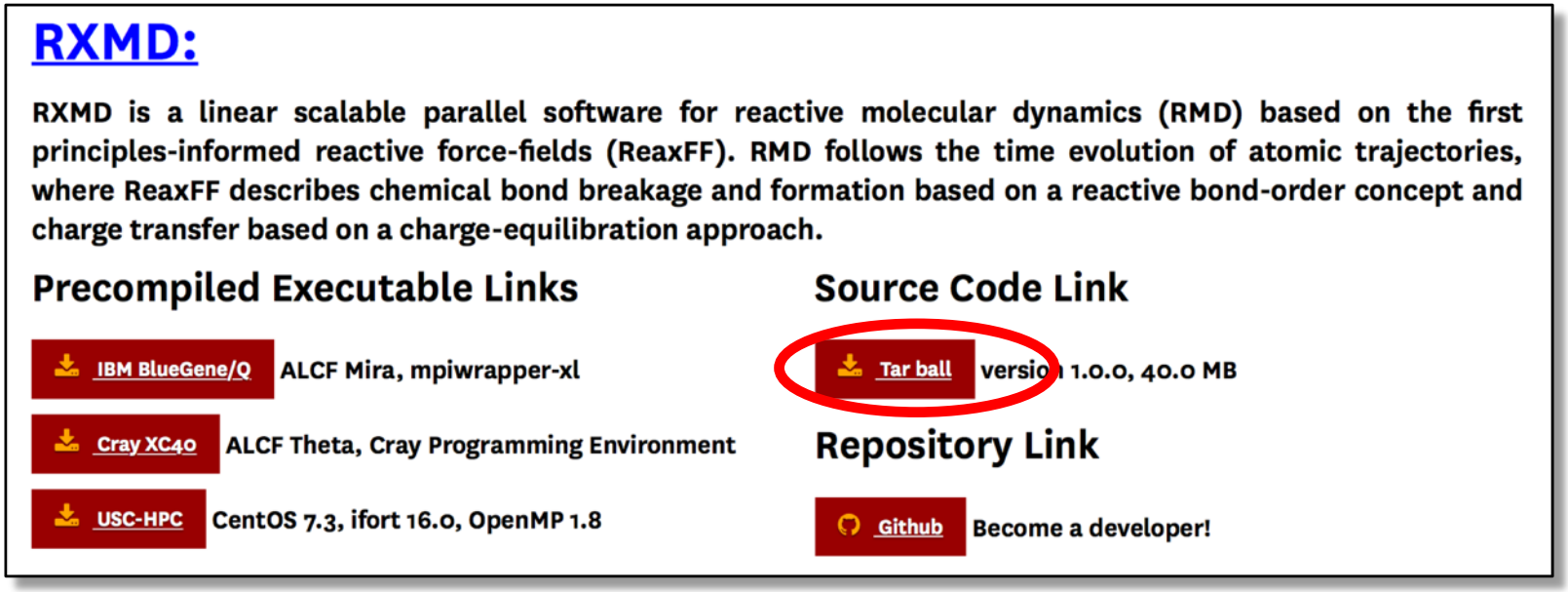
```
$ pwd
/staging/magics18/magics60
```

RXMD Hands-on: Software Setup

- Download RXMD source code. Go to the MAGICS website.

https://magics.usc.edu/home-old/software-downloads/

- Click "**Tar ball**" button under "Source Code Link".



- However, due to the network bandwidth issue, we will copy the zip file from local disk.

```
$ cp ~magics60/magics/rxmd-2017Nov03.zip .
```

RXMD Hands-on: Software Setup

- Unzipping the zip file will create a directory called **rxmd-master**

```
$ unzip rxmd-2017Nov03.zip
$ cd rxmd-master/
```

RXMD directory structure looks like this.

```
.
├── DAT
│   └── rxff.bin
├── ffield
├── init
│   ├── geninit.F90
│   ├── input.xyz
│   └── Makefile
├── Makefile.inc
├── rxmd.in
└── src
```

```
src/
├── bo.F90
├── cg.F90
├── comm.F90
├── fileio.F90
├── init.F90
├── main.F90
├── Makefile
├── module.F90
├── param.F90
├── pot.F90
├── qeq.F90
├── stress
└── stress.F90
```

RXMD Hands-on: Software Setup

Go to **init** directory and edit **Makefile** to choose a compiler you want to use.

```
$ cd init
$ nano Makefile
```

```
…
# Fortran Compiler
#FC = gfortran
#FC = xlf
FC = ifort          // <- enable Intel compiler
                    // and comment out other compilers
…
```

Then, type **make** to create an initial configuration.

```
$ make
```

RXMD Hands-on: Software Setup

```
ifort -c   geninit.F90
ifort -o geninit geninit.o
./geninit
-------------------------------------------------------------
        input file: input.xyz
       ffield file: ../ffield
     nprocs,vprocs:         1     1     1     1
          mctot,mc:         6     2     3     1
-------------------------------------------------------------
-------------------------------------------------------------
reading atom name in ../ffield
  1-O   2-S   3-Mo  4-Al
-------------------------------------------------------------
       64   MoO3 unit cell
1, L2, L3, Lalpha, Lbeta,
Lgamma:           7.920       7.390      13.860      90.000      90.000      90.0
00
rmin(1:3),rmax(1:3):     1.00000E-09    1.00000E-09    1.00000E-
09       9.97353E-01    9.31997E-01    9.64851E-01
 sum(lnatoms), lnatoms:          384         384
L1, L2, L3, Lalpha, Lbeta,
Lgamma:          15.840      22.170      13.860      90.000      90.000      90.0
00
cp -v rxff.bin ../DAT
rxff.bin -> ../DAT/rxff.bin
```

make command does a lot of things. We will look into it later.

RXMD Hands-on: Software Setup

- Move back to work directory, and edit **Makefile.inc** to enable a compiler you want to use for RXMD executable build.

```
$ cd ..
$ nano Makefile.inc
```

```
…
# Intel Compiler
#FC = mpif90 -O3
FC = mpif90 -qopenmp -O3     // enable this line for Intel compiler
#FC = mpif90 -check all -traceback

# gfortran
#FC = mpif90 -O3 -ffast-math  // then, comment out this line
#FC = mpif90 -fopenmp -O3 -ffast-math
…
```

- Now move to **src** directory and build the RXMD executable.

```
$ cd src
$ make
```

RXMD Hands-on: Software Setup

- **make** compiles the RXMD source code and place the executable **rxmd** in work directory.

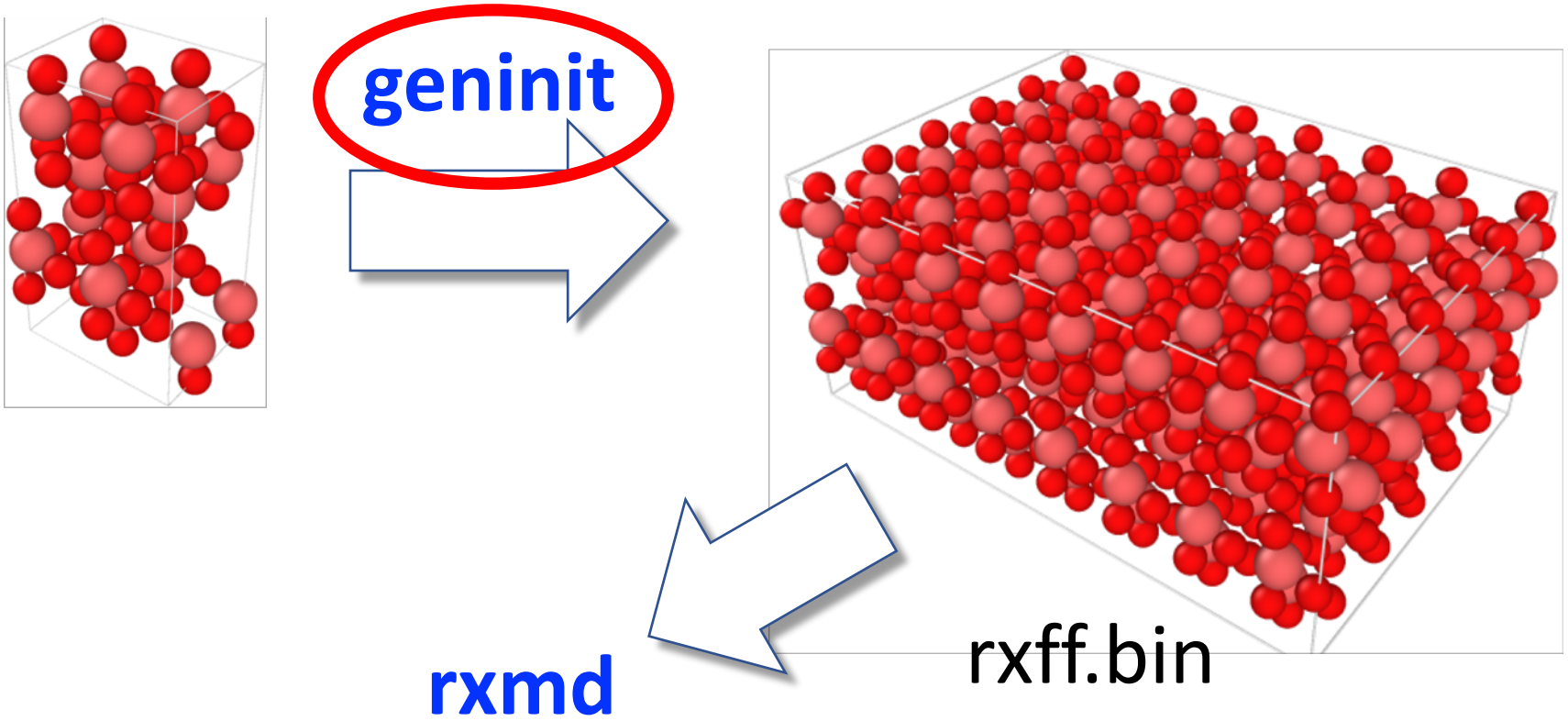
```
$ make
mpif90 -qopenmp -O3 -c   module.F90
mpif90 -qopenmp -O3   -c cg.F90
mpif90 -qopenmp -O3   -c pot.F90
mpif90 -qopenmp -O3   -c param.F90
mpif90 -qopenmp -O3   -c fileio.F90
mpif90 -qopenmp -O3   -c comm.F90
mpif90 -qopenmp -O3   -c init.F90
mpif90 -qopenmp -O3   -c bo.F90
mpif90 -qopenmp -O3   -c qeq.F90
mpif90 -qopenmp -O3   -c stress.F90
mpif90 -qopenmp -O3   -c main.F90
mpif90 -qopenmp -O3 -o rxmd  cg.o pot.o fileio.o comm.o init.o
bo.o qeq.o param.o stress.o main.o module.o
mv rxmd ../rxmd
```

- Go back to work directory from **src**

```
$ cd ..
```

Outline

- Software Setup
- Create Initial Configuration
- RXMD Input Parameters
- Hands-on : MoO_3 Self Reduction
- Hands-on : SiC Nanoparticle Oxidation


init.xyz
geninit
rxff.bin
rxmd

Create Initial Configuration : geninit

- We use an executable called **geninit** (**gen**erate **init**ial config) to generate initial configuration for RXMD simulation.
- **geninit** reads unit cell information from **input.xyz** (by default) and ReaxFF force field file (**../ffield**) to find numerical IDs from element name (for example C (carbon) is 1, H (hydrogen) is 2), then creates a binary file **rxff.bin**, which will be input for RXMD.
- To build **geninit**, go to **init** directory and type **make**.

```
$ cd init
$ make
```

```
-----------------------------------------------------------
        input file: input.xyz
       ffield file: ../ffield
     nprocs,vprocs:          1      1      1      1
          mctot,mc:          6      2      3      1
-----------------------------------------------------------
…
```

Create Initial Configuration : geninit

- geninit command takes several options

```
$ ./geninit -help
./geninit -mc 1 1 1 -vprocs 1 1 1 -inputxyz
input.xyz -ffield ffield [-r or -n]
```

-mc or -m (3 integers) : Number of repetitions of unit cell.

-vprocs or -v (3 integers) : Number of processors in x,y, and z directions

-inputxyz or -i (string) : Filename contains unit cell configuration

-ffield or -f (string) : Filename contains ReaxFF force field parameters

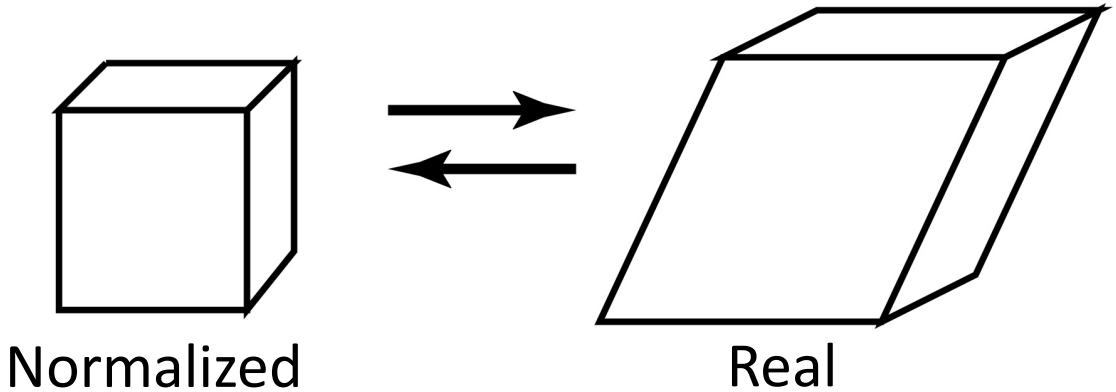
Create Initial Configuration : geninit

- **geninit** supports normalized and real coordinate conversion.

```
$ ./geninit -help
./geninit -mc 1 1 1 -vprocs 1 1 1 -inputxyz
input.xyz -ffield ffield [-r or -n]
```

-getreal or -r : Convert from normalized to real coordinates. Result will be stored in **real.xyz**.

-getnorm or -n : Convert from real to normalized coordinates. Result will be stored in **norm.xyz**.



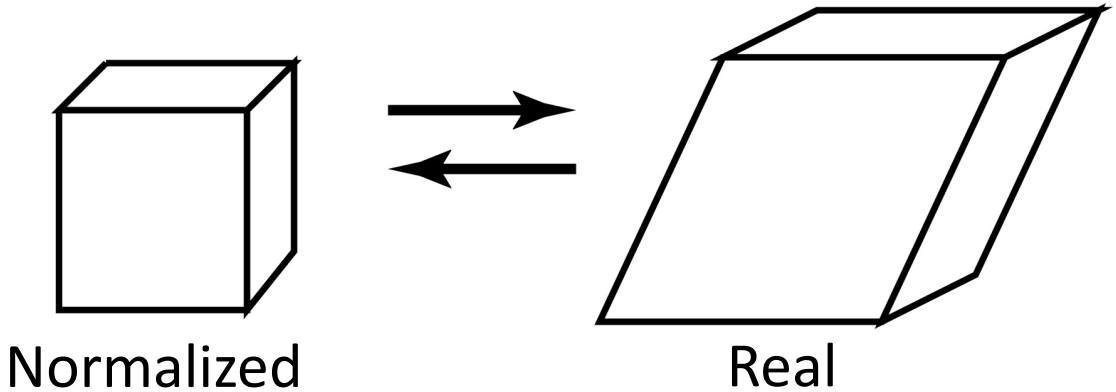
Create Initial Configuration : geninit

- **geninit** supports normalized and real coordinate conversion.

```
$ ./geninit -help
./geninit -mc 1 1 1 -vprocs 1 1 1 -inputxyz
input.xyz -ffield ffield [-r or -n]
```

- **-r** and **-n** flags can be used together with **–i** to specify input file name and **–mc** to repeat the unit structure but **-v** will be ignored.

Caveat! There is no check on the coordinates of input data. It is the user's responsibility to provide proper input coordinate data.



Create Initial Configuration : input.xyz

- Input file **input.xyz** resembles XYZ format but is slightly modified.
- **Line1 :** number of atoms in unit cell followed by a string to describe the unit cell.
- **Line2 :** six lattice parameters, *a, b, c* and *alpha*, *beta*, and *gamma*.

```
64 "MoO3 unit cell"
7.92   7.39  13.86  90.00  90.00  90.00
Mo 0.141162 0.137258 0.354299
…
O 0.0982146 0.62335 0.187911
```

Create Initial Configuration : input.xyz

- Line3-EOF : element name and x, y, and z positions.

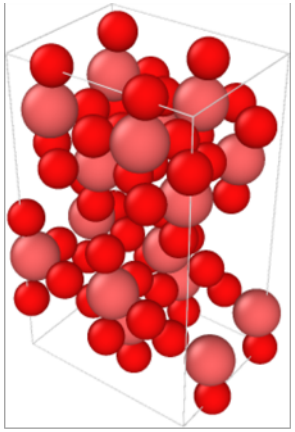
Caveat! element name must exist in ReaxFF force field file.

Caveat! atom coordinates are normalized by the lattice parameters.

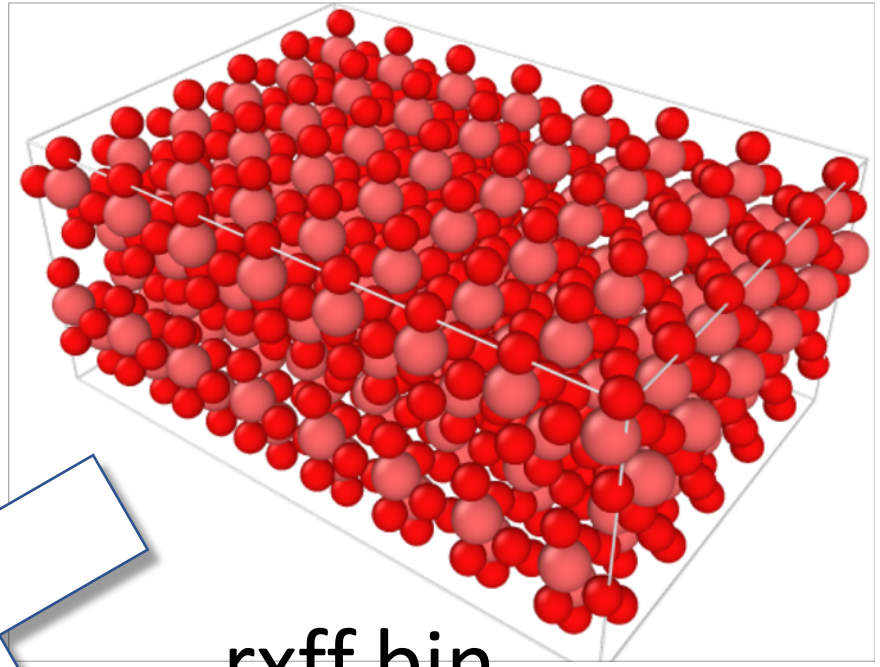
```
64 "MoO3 unit cell"
7.92   7.39  13.86  90.00  90.00  90.00
Mo 0.141162 0.137258 0.354299
…
O 0.0982146 0.62335 0.187911
```

Exercise : Create a MoO_3 slab

init.xyz



geninit



rxff.bin

rxmd

Exercise : Create a MoO_3 slab

1. Create a supercell and convert supercell from fractional/normalized coordinate into real coordinate

2. Add vacuum in the box

3. convert real to fractional/normalized coordinate

Exercise : Create a MoO_3 slab

- Change directory to **init**. Make sure you have the executable **geninit**. If not, type **make**.

```
$ ls
Makefile      geninit*      geninit.F90   input.xyz
```

- **input.xyz** in **init** directory contains a MoO_3 unit cell in normalized coordinates. Create a 2x2x1 unit cell and store it in the real coordinates. Use **–r** and **–m** flags. (**geninit** assumes **-i input.xyz -f ../ffield** by default)

```
$ ./geninit -r -m 2 2 1
...
          256    "MoO3 unit cell"
  15.84000  14.78000  13.86000    90.000    90.000    90.000
-----------------------------------------------------------
coordinates are saved in real.xyz
```

- Now we have the system configuration stored in **real.xyz**

Exercise : Create a MoO_3 slab

- Edit lattice parameter in **real.xyz** to insert vacuum in the z direction. Change the lattice parameter from **13.86** to **43.86**. This will insert 30(Å) vacuum on top of the slab in z-direction.

```
          256    "MoO3 unit cell"
  15.84000  14.78000  43.86000    90.000    90.000    90.000
Mo       1.118003040    1.014336620    4.910584140
Mo       1.097379360    4.559763020    4.922600760
Mo       5.079492000    1.014499200    4.910085180
```

- Normalize the atom coordinates. Use **-n** and **–i** flag.

```
$ ./geninit -n -i real.xyz
```

- Now the slab system is stored in **norm.xyz**.

Exercise : Create a MoO_3 slab

- Tell **geninit** how many MD domains you want to use. Here we use 2x2x1 domain decomposition. Use **–v** and **-i** flags.

```
$ ./geninit -v 2 2 1 -i norm.xyz
```

- This commands generate **rxff.bin**. Copy rxff.bin in ../DAT directory.

```
cp rxff.bin ../DAT
```

- That is all! Now you are ready to start RXMD simulation. Go back to the working directory, **rxmd-master**.

```
cd ..
```

Outline

- Software Setup
- Create Initial Configuration
- RXMD Input Parameters
- Hands-on : MoO_3 Self Reduction
- Hands-on : SiC Nanoparticle Oxidation

RXMD Input Parameters : rxmd.in

- When RXMD executable is invoked, it reads **rxmd.in** for various simulation-related parameters.

```
$ cat rxmd.in
```

```
1                    <mdmode>
0.25      5000       <dt> <ntime_step>
300  1.0   100       <treq> <vsfact> <sstep>
1000      100        <fstep>  <pstep>
1 1 1                <vprocs>
1  500  1.d-6  1     <isQEq> <NMAXQEq> <QEq_tol> <qstep>
1.0  180             <Lex_fqs> <Lex_k>
.true. .true. .true. <isBinary> <isBondFile> <isPDB>
1.d-8                <ftol>
```

RXMD Input Parameters : rxmd.in

- **mdmode** decides overall behavior of RXMD simulation.
- **mdmode** = 1 is NVE run, 4-7 are various temperature control modes by velocity scaling, and 10 for structural optimization using conjugate gradient method.

```
1                    <mdmode>
0.25      5000       <dt> <ntime_step>
300  1.0   100       <treq> <vsfact> <sstep>
1000      100        <fstep>  <pstep>
1 1 1                <vprocs>
1  500  1.d-6  1     <isQEq> <NMAXQEq> <QEq_tol> <qstep>
1.0  180             <Lex_fqs> <Lex_k>
.true. .true. .true. <isBinary> <isBondFile> <isPDB>
1.d-8                 <ftol>
```

RXMD Input Parameters : rxmd.in

- **dt** is one MD timestep in femtosecond unit. e.g. 0.25 = 0.25(fs)
- **ntime_step** is the number of MD steps to run.

```
1                    <mdmode>
0.25      5000       <dt> <ntime_step>
300   1.0    100     <treq> <vsfact> <sstep>
1000      100        <fstep>  <pstep>
1 1 1                <vprocs>
1   500   1.d-6   1  <isQEq> <NMAXQEq> <QEq_tol> <qstep>
1.0   180            <Lex_fqs> <Lex_k>
.true. .true. .true. <isBinary> <isBondFile> <isPDB>
1.d-8                 <ftol>
```

RXMD Input Parameters : rxmd.in

- When mdmode == 4, atom velocity is multiplied by **vsfact** every **sstep** MD steps.
- **treq** is not used with mdmode == 4.
- **sstep** is the interval of each velocity scaling, e.g. **sstep** == 100 means velocity scaling every100 MD steps.

```
4                    <mdmode>
0.25      5000       <dt> <ntime_step>
300   1.0    100     <treq> <vsfact> <sstep>
1000      100        <fstep>  <pstep>
1 1 1                <vprocs>
1   500   1.d-6   1  <isQEq> <NMAXQEq> <QEq_tol> <qstep>
1.0   180            <Lex_fqs> <Lex_k>
.true. .true. .true. <isBinary> <isBondFile> <isPDB>
1.d-8                 <ftol>
```

RXMD Input Parameters : rxmd.in

- **treq** is used when mdmode == 5, 6 and 7 where atom velocity is scaled to **treq** (K) every **sstep** MD steps.
- **sstep** is the interval of each velocity scaling, e.g. **sstep** == 100 means velocity scaling every100 MD steps.

```
5                    <mdmode>
0.25      5000       <dt> <ntime_step>
300   1.0    100     <treq> <vsfact> <sstep>
1000      100        <fstep>  <pstep>
1 1 1                <vprocs>
1   500   1.d-6   1  <isQEq> <NMAXQEq> <QEq_tol> <qstep>
1.0   180            <Lex_fqs> <Lex_k>
.true. .true. .true. <isBinary> <isBondFile> <isPDB>
1.d-8                <ftol>
```

RXMD Input Parameters : rxmd.in

- **fstep** is the interval of check-pointing, i.e. save atom data and connectivity data on to disk. Type of data to be saved is determined by **isBinary**, **isBondFile**, and **isPDB** logical variables.
- **pstep** is the interval of displaying ReaxFF energy terms on standard output.

```
1                    <mdmode>
0.25      5000       <dt> <ntime_step>
300  1.0    100      <treq> <vsfact> <sstep>
1000      100        <fstep>  <pstep>
1 1 1                <vprocs>
1  500  1.d-6  1     <isQEq> <NMAXQEq> <QEq_tol> <qstep>
1.0  180             <Lex_fqs> <Lex_k>
.true. .true. .true. <isBinary> <isBondFile> <isPDB>
1.d-8                 <ftol>
```

RXMD Input Parameters : rxmd.in

- **vprocs** is the number of processors in x, y, and z directions, dividing the total simulation box into smaller subdomains.

  Caveat! vprocs must be either 1 or even number.

```
1                    <mdmode>
0.25      5000       <dt> <ntime_step>
300   1.0    100     <treq> <vsfact> <sstep>
1000      100        <fstep>  <pstep>
1 1 1                <vprocs>
1   500   1.d-6   1  <isQEq> <NMAXQEq> <QEq_tol> <qstep>
1.0   180            <Lex_fqs> <Lex_k>
.true. .true. .true. <isBinary> <isBondFile> <isPDB>
1.d-8                 <ftol>
```

RXMD Input Parameters : rxmd.in

- **isQEq** is a logical flag to enable the variable charge (isQEq == 1) or disable it (isQEq == 0).
- QEq minimize the electrostatic energy using conjugate gradient algorithm. **NMAXQEq**, **Qeq_tol**, and **qsteps** are the maximum number of iteration, the convergence tolerance and interval of QEq subroutine call, respectively.

```
1                    <mdmode>
0.25      5000       <dt> <ntime_step>
300   1.0    100     <treq> <vsfact> <sstep>
1000      100        <fstep>  <pstep>
1 1 1                <vprocs>
1   500   1.d-6   1  <isQEq> <NMAXQEq> <QEq_tol> <qstep>
1.0   180            <Lex_fqs> <Lex_k>
.true. .true. .true. <isBinary> <isBondFile> <isPDB>
1.d-8                 <ftol>
```

RXMD Input Parameters : rxmd.in

- **isBinary**, **isBondFile**, and **isPDB** are logical flags to save check-point data, bond connectivity and Protein Data bank (PDB) file or not.
- Data dumping happens every **fstep** MD steps and these files are saved into **DAT** directory (by default).

```
1                    <mdmode>
0.25      5000       <dt> <ntime_step>
300  1.0   100       <treq> <vsfact> <sstep>
1000      100        <fstep>  <pstep>
1 1 1                <vprocs>
1  500  1.d-6  1     <isQEq> <NMAXQEq> <QEq_tol> <qstep>
1.0  180             <Lex_fqs> <Lex_k>
.true. .true. .true. <isBinary> <isBondFile> <isPDB>
1.d-8                 <ftol>
```

RXMD Input Parameters : rxmd.in

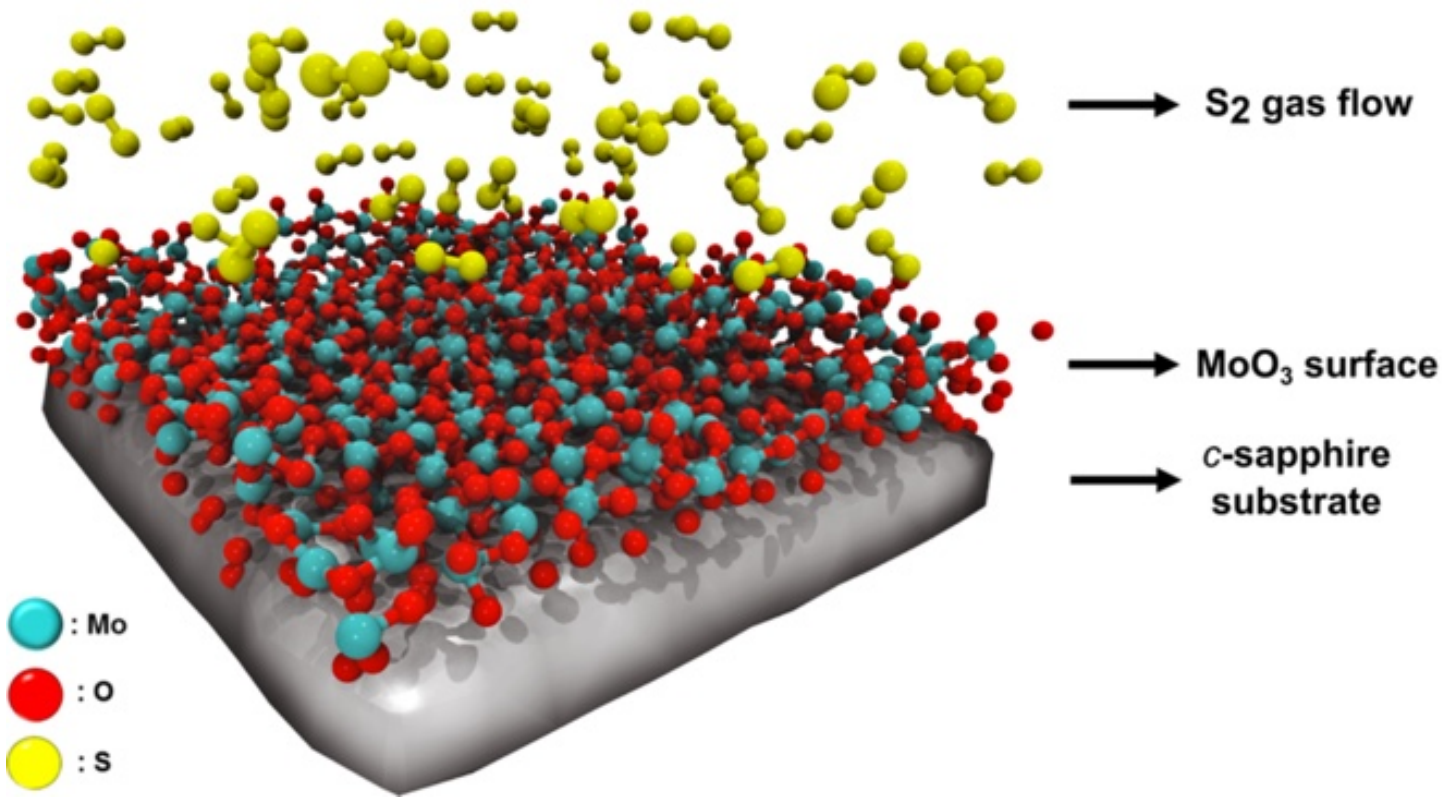
- **ftol** is the tolerance of conjugate gradient for structural optimization. Not for charge QEq.
- **ftol** is used when **mdmode** == 10.

```
1                    <mdmode>
0.25      5000       <dt> <ntime_step>
300  1.0    100     <treq> <vsfact> <sstep>
1000      100       <fstep>  <pstep>
1 1 1                <vprocs>
1  500  1.d-6  1    <isQEq> <NMAXQEq> <QEq_tol> <qstep>
1.0  180             <Lex_fqs> <Lex_k>
.true. .true. .true. <isBinary> <isBondFile> <isPDB>
1.d-8                <ftol>
```

Outline

- Software Setup
- Create Initial Configuration
- RXMD Input Parameters
- Hands-on : MoO_3 Self Reduction
- Hands-on : SiC Nanoparticle Oxidation

RXMD Hands-on : MoO_3 Self-Reduction Simulation



Computational synthesis of MoS2 layers by reactive molecular dynamics simulations, initial sulfidation of MoO3 surfaces S. Hong, et al. *Nano Letters* **17**, 4866-4872 (2017)

RXMD Hands-on : MoO_3 Self-Reduction Simulation

- First, copy a tar file **rxmd.moo3.tar.gz** for this hands-on session to your working directory.

```
$ cp ~magics60/magics/rxmd.moo3.tar.gz .
$ tar xvfz rxmd.moo3.tar.gz
```

- The tar command will create a directory called **moo3**. **moo3** is going to be your working directory in this hands-on. Change directory to **moo3**, then copy executable **rxmd** to **moo3**.

```
$ cd moo3
$ cp ../rxmd .
```

- You should have following files and directories.

```
01-relax.sh  02-heatup.sh  03-run.sh  cat_pdb.sh*
count_bond.py  rxmd.in
DAT/  ffield  init.moo3/  inputs/
```

RXMD Hands-on : MoO_3 Self-Reduction Simulation

- Change directory to **init.moo3** and type **make** to create initial config.

```
$ cd init.moo3/
$ make
```

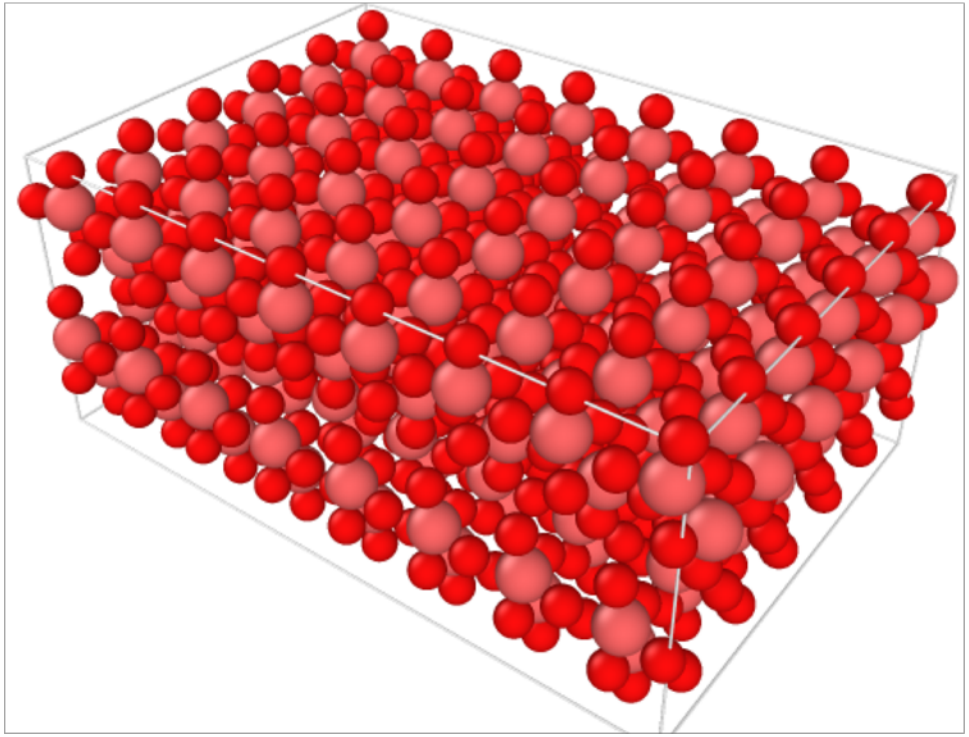
```
gfortran -c   geninit.F90
gfortran -o geninit geninit.o
./geninit input.xyz
 input file:   input.xyz
 ffield file:   ../ffield
 nprocs,vprocs        1     1     1     1
 mctot,mc       12     4     3     1
  1-O   2-S   3-Mo  4-Al
       64   MoO3 unit cell
...
...
cp -v rxff.bin ../DAT
'rxff.bin' -> '../DAT/rxff.bin'
```

- Move back to working directory.

```
$ cd ..
```

RXMD Hands-on : MoO_3 Self-Reduction Simulation

- The system looks like this,



- Number of Atoms : 768
192 Mo + 576 O
- Lattice Parameters:
31.68(Å)x22.17(Å)x41.58(Å)
90.0 90.0 90.0
- 30 (Å) vacuum in z-axis
- Relax free surface and heatup the system upto 1800(K)

Simulation Schedule

- First we relax the free surfaces by quenching, then increase the system temperature up to 1800K by velocity scaling.
- Simulation schedule and input parameters are following.

1. Surface Relaxation :

rxmd.in-00 : for 1000 MD steps
rxmd.in-01 : for 1000 MD steps
rxmd.in-02 : for 1000 MD steps

2. Heatup :

rxmd.in-03 : to 600K for 5000 MD steps
rxmd.in-04 : to 1200K for 5000 MD steps
rxmd.in-05 : to 1800K for 5000 MD steps

13. Measurement :

Keep temperature at 1800K and run.

Simulation Schedule : Surface Relaxation

rxmd.in-00

```
4                          <mdmod>
0.01    1000               <dt> <ntime_step>
100  0.5   100             <treq> <vsfact> <sstep>
100    100                 <fstep>  <pstep>
1 1 1                      <vprocs>
1  500  1.d-6  10   <isQEq> <NMAXQEq> <QEq_tol> <qstep>
1.0  180                   <Lex_fqs> <Lex_k>
.true. .true. .true. <isBinary> <isBondFile> <isPDB>
1.d-8                      <ftol>
```

rxmd.in-01

```
4
0.5 1000
100 0.5 100
100 100
1 1 1
1 500 1.d-6 10
1.0 180
.true. .true. .true.
1.d-8
```

rxmd.in-02

```
4
0.5 1000
100 0.9 100
100 100
1 1 1
1 500 1.d-6 10
1.0 180
.true. .true. .true.
1.d-8
```

- **YOUR TASK** : create the three RXMD input parameter files, **rxmd.in-00, rxmd.in-01 rxmd.in-02**, shown on left.
- We provide a template file (**rxmd.in**) that you can copy to **rxmd.in-00**, then edit the RXMD parameters that need to be updated.
- Copy **rxmd.in-00** to **rxmd.in-01**, then edit it.
- Repeat this for **rxmd.in-02**.

Submit job to USC HPC cluster

- We use SLURM (Simple Linux Utility for Resource Management) script to submit your job to HPC cluster.
- There are three SLURM scripts (**01-relax.sh, 02-heatup.sh, 03-run.sh**), each of which runs the relaxation, heatup, and measurement step.
- We will start from the surface relaxation step using **01-relax.sh**.

```
$ cat 01-relax.sh
#!/bin/bash
# SLURM and environment settings

for f in rxmd.in-0[0-2]; do
   cp -v $f rxmd.in
   srun –n 1 ./rxmd | tee log-${f}
done
```

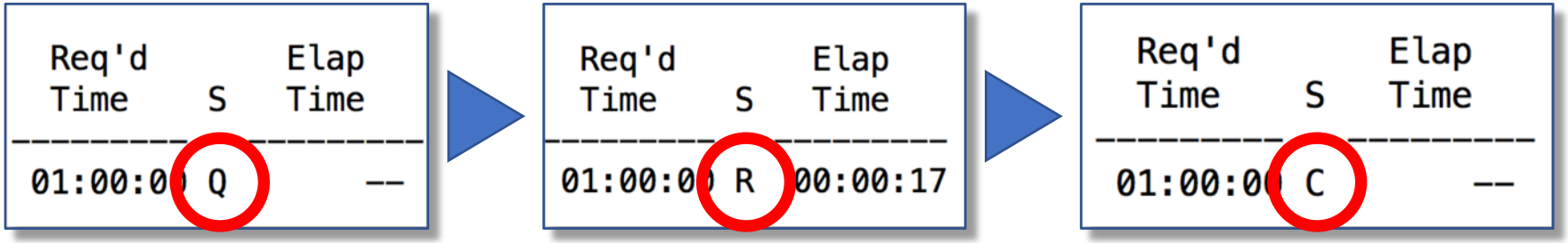
Submit job to USC HPC cluster

- Use **sbatch** to submit your job.

```
$ sbatch 01-relax.sh
```

- Monitor your job status using **squeue**. Your job status goes from **Q** (waiting in queue), **R** (running), and **C** (complete).

```
$ squeue -u ${USER}
```



- While the surface relaxation job is running, we will work on the input parameters for other steps.

Simulation Schedule : Heatup and Measurement Steps

rxmd.in-03

```
7                          <mdmod>
0.5    5000               <dt> <ntime_step>
600  0.9   100            <treq> <vsfact> <sstep>
100    100                <fstep>  <pstep>
1 1 1                     <vprocs>
1  500  1.d-6  10         <isQEq> <NMAXQEq> <QEq_tol> <qstep>
1.0  180                  <Lex_fqs> <Lex_k>
.true. .true. .true. <isBinary> <isBondFile> <isPDB>
1.d-8                     <ftol>
```

rxmd.in-04

```
7
0.5 5000
1200 0.9 100
100 100
1 1 1
1 500 1.d-6 10
1.0 180
.true. .true. .true.
1.d-8
```

rxmd.in-05

```
5
0.5 5000
1800 0.9 100
100 100
1 1 1
1 500 1.d-6 10
1.0 180
.true. .true. .true.
1.d-8
```

- **YOUR TASK** : create the three input files, **rxmd.in-03**, **rxmd.in-04**, **rxmd.in-05**, shown on left.
- You can copy **rxmd.in-02** to **rxmd.in-03**, then edit it wherever needs to be changed.
- Repeat this step, **rxmd.in-03** → **rxmd.in-04**, so on
- We will use **rxmd.in-05** for the Measurement step.

Submit job to USC HPC cluster

- After you have created all input parameter files, use **sbatch** to submit the heatup step.

```
$ sbatch 02-heatup.sh
```

- Monitor your job status using **squeue**.

```
$ squeue –u ${USER}
```

- When the heatup step finishes, i.e. job status becomes **C** (complete) status, submit the measurement step job using **03-run.sh**.

```
$ sbatch 03-run.sh
```

Analyze Simulation Result : Visualize Atom Trajectory

- While your job is running, checkpoint data (.bin), atom trajectory (.pdb), and connectivity information (.bnd) will be saved into **DAT** directory.

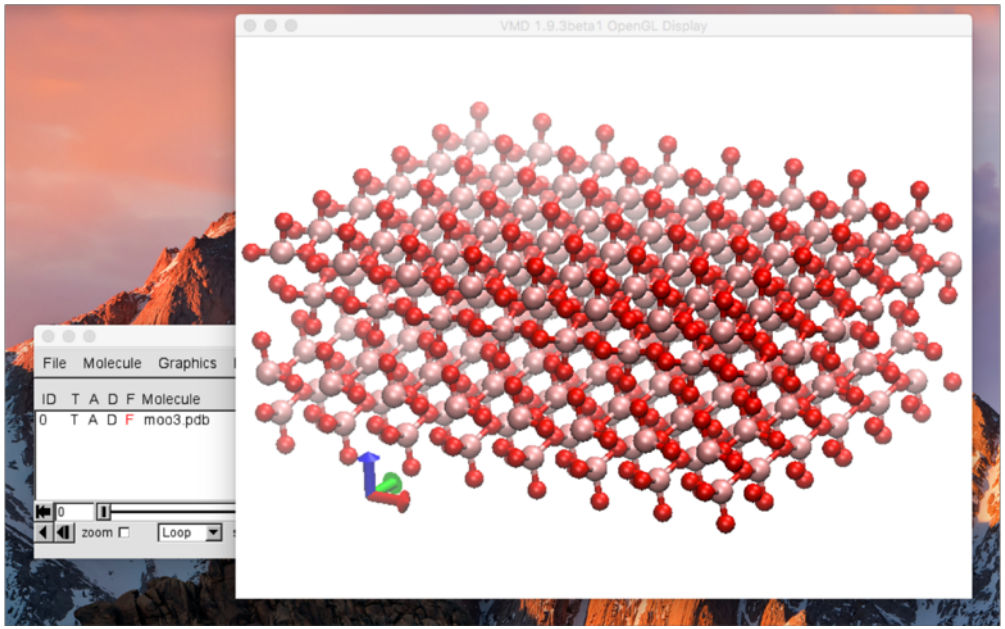
```
$ ls DAT/
000000000.bin
000000000.bnd
000000000.pdb
000000100.bin
000000100.bnd
000000100.pdb
...
```

- To visualize atom trajectory with VMD, we need to concatenate PDB files from different MD steps into one PDB file with a proper separator keyword [**END**].
- Also, every line must have the same atom through all MD frames.

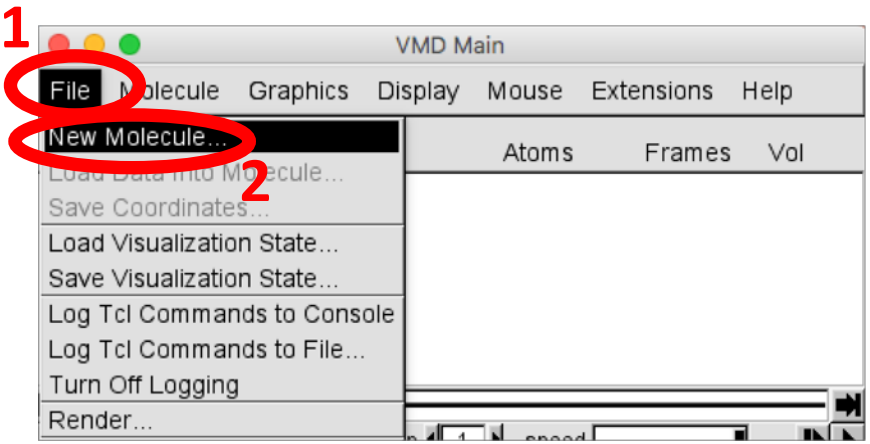
Analyze Simulation Result : Visualize Atom Trajectory

- A simple BASH script **cat_pdb.sh** is included in the tarball to concatenates all PDB files under **DAT** directory into one PDB file.
- Transfer the output file (**output.pdb**) to your computer and load it on VMD.

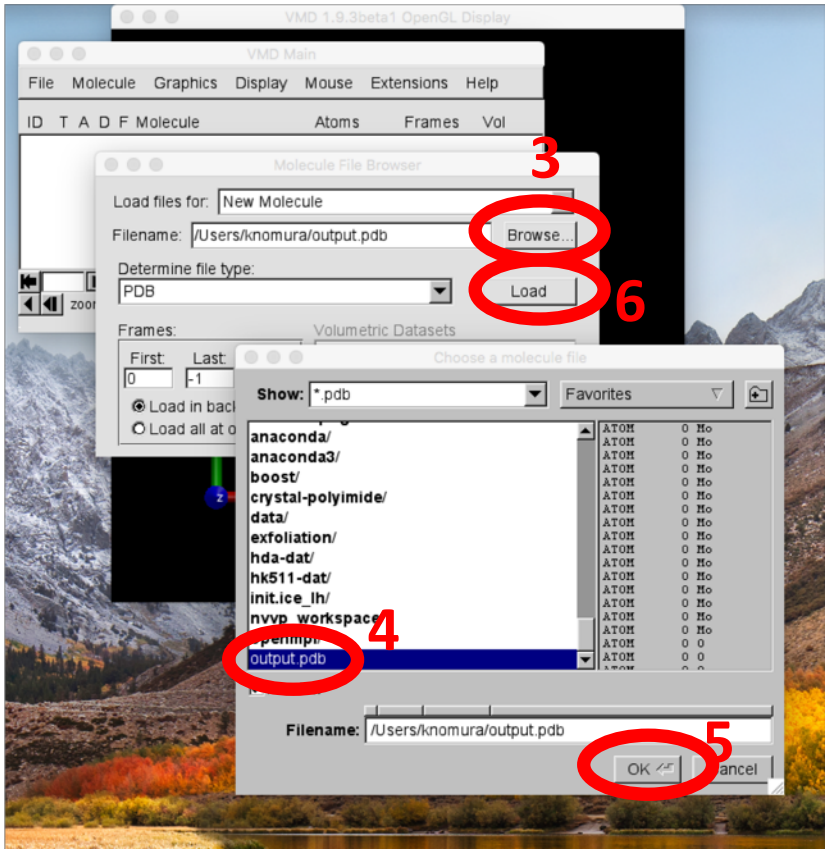
```
$ ./cat_pdb.sh
DAT/000000000.pdb
DAT/000000100.pdb
DAT/000000200.pdb
DAT/000000300.pdb
…
'all.pdb' -> 'output.pdb'
```



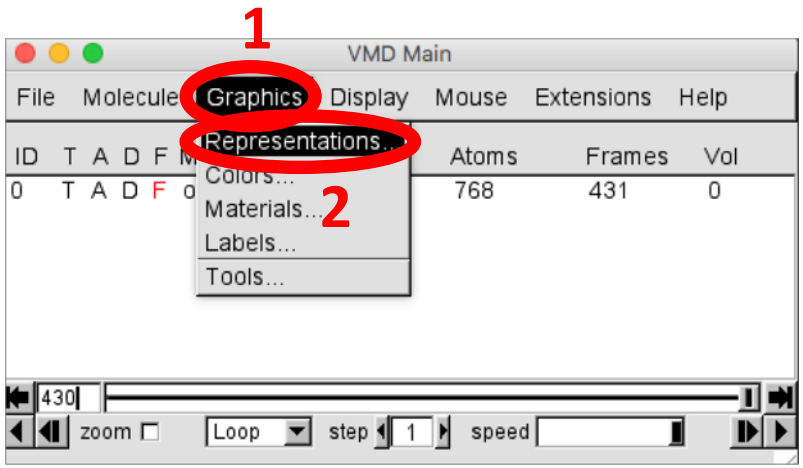
Analyze Simulation Result : Load output.pdb to VMD



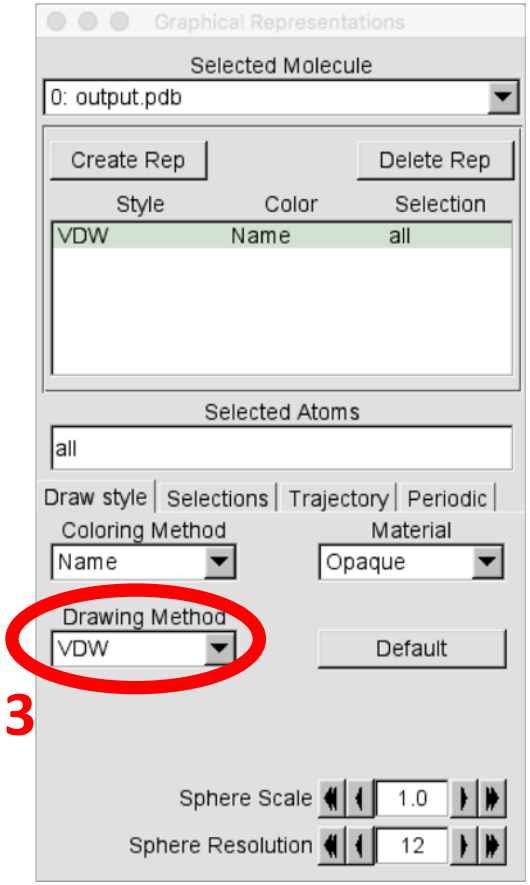
1. File
2. New Molecule
3. Browse
4. Select **output.pdb**
5. OK
6. Load



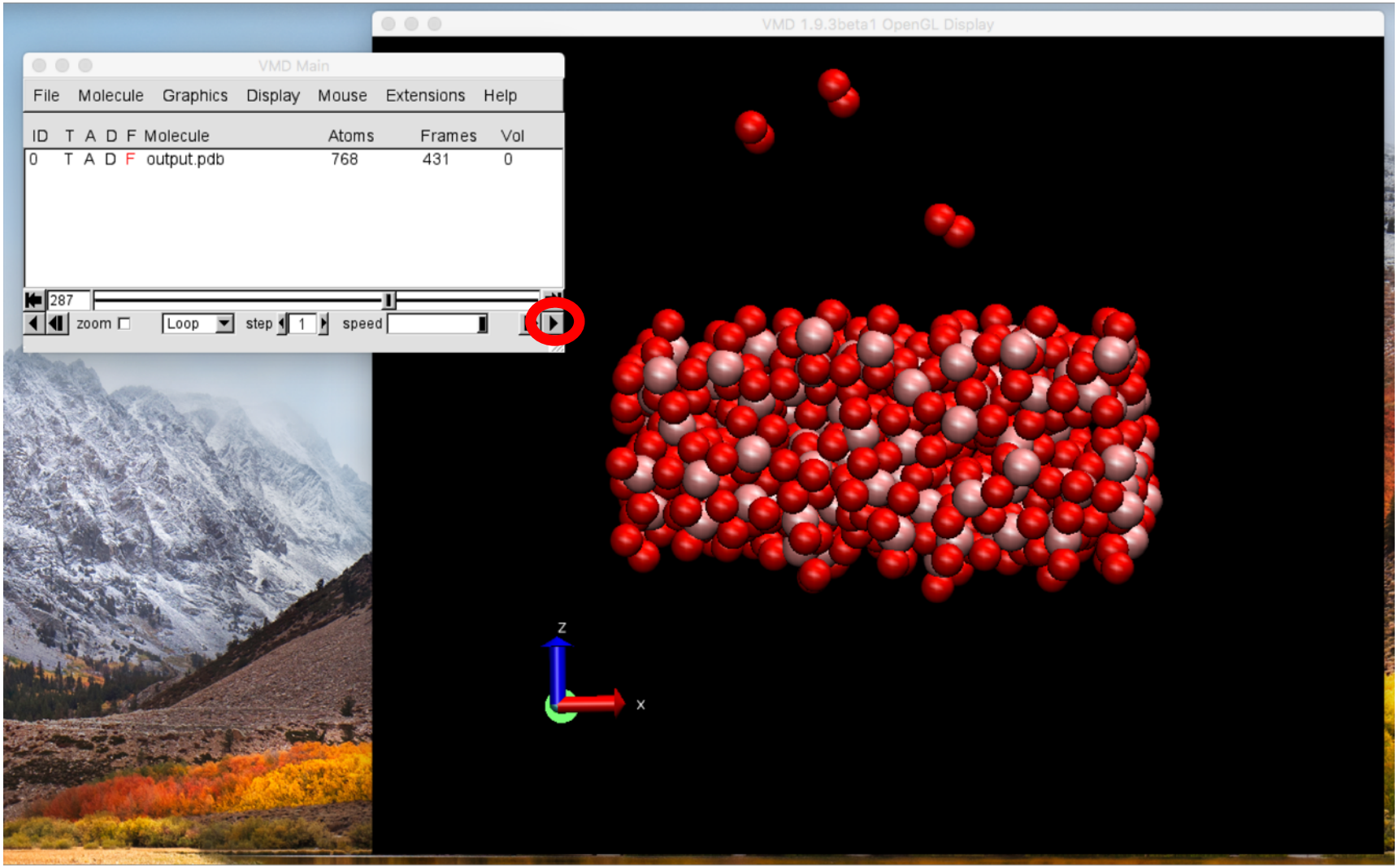
Analyze Simulation Result : Change Atom Representation



1. Graphics
2. Representations..
3. Drawing Method
   Choose **VDW**
4. Adjust viewing angle with mouse



Analyze Simulation Result : Play Animation



Analyze Simulation Result : Bond Analysis

- A simple Python script **count_bond.py** is included in the tarball.
- **count_bond.py** counts the number of bonds of each bond type.
- No argument is necessary, just run **count_bond.py** from your working directory that has **DAT** directory.

```
$ ./count_bond.py | tee bond.txt
```

- You will see output below.

```
...
./DAT/000080000.bnd  :  1-1   22   1-3   2092   3-3   42
./DAT/000080100.bnd  :  1-1   22   1-3   2124   3-3   36
./DAT/000080200.bnd  :  1-1   22   1-3   2132   3-3   42
./DAT/000080300.bnd  :  1-1   22   1-3   2120   3-3   34
./DAT/000080400.bnd  :  1-1   22   1-3   2154   3-3   36
...
```

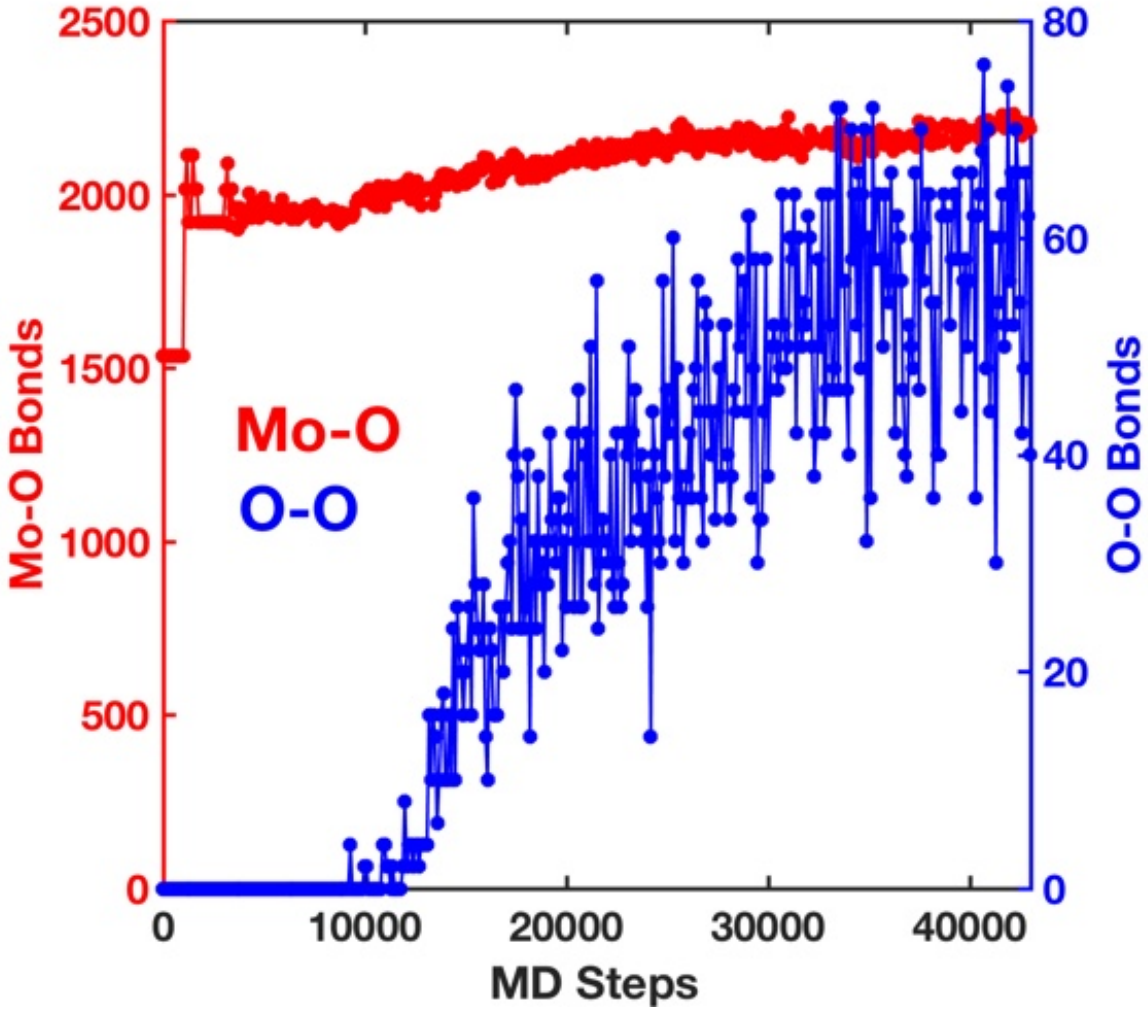
Analyze Simulation Result : Bond Analysis

- Blue columns are atom type combinations, e.g. 1-Mo and 3-O, and red columns are the number of bonds.

```
...
./DAT/000080000.bnd  :  1-1   22   1-3   2092   3-3   42
./DAT/000080100.bnd  :  1-1   22   1-3   2124   3-3   36
./DAT/000080200.bnd  :  1-1   22   1-3   2132   3-3   42
./DAT/000080300.bnd  :  1-1   22   1-3   2120   3-3   34
./DAT/000080400.bnd  :  1-1   22   1-3   2154   3-3   36
./DAT/000080500.bnd  :  1-1   22   1-3   2118   3-3   46
...
```

- Transfer **bond.txt** file to your laptop. Use any software to plot the number of bonds for each bond-type.

Analyze Simulation Result : Bond Analysis



If something went wrong..

- If something went wrong, we can start over by copying the original configuration file, **rxff.bin**, in **init** directory to **DAT**.
- Make sure you are in the working directory for this hands-on (**moo3** here), then type following.

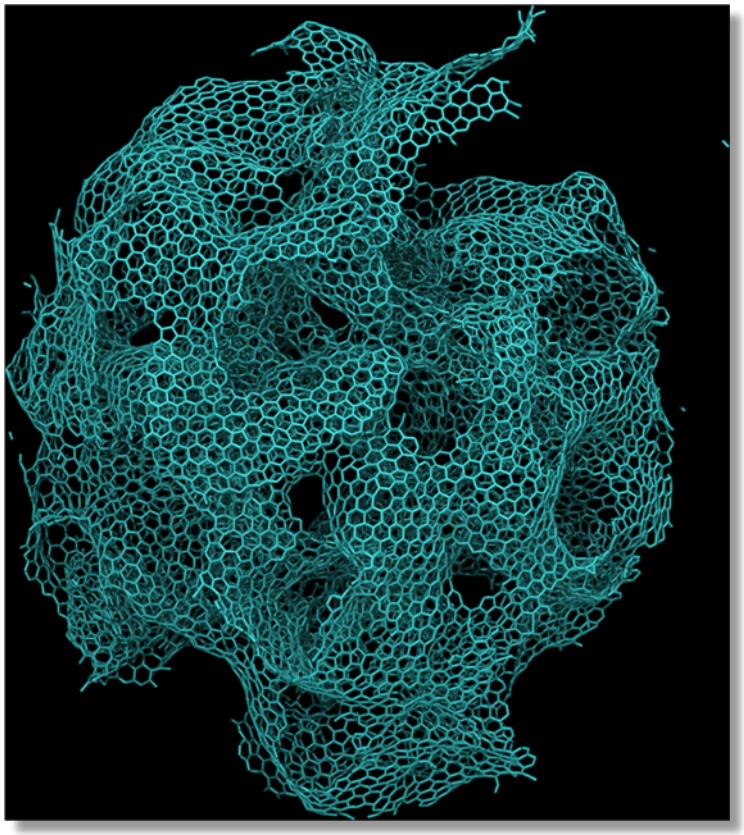
```
$ cp init.moo3/rxff.bin DAT/
```

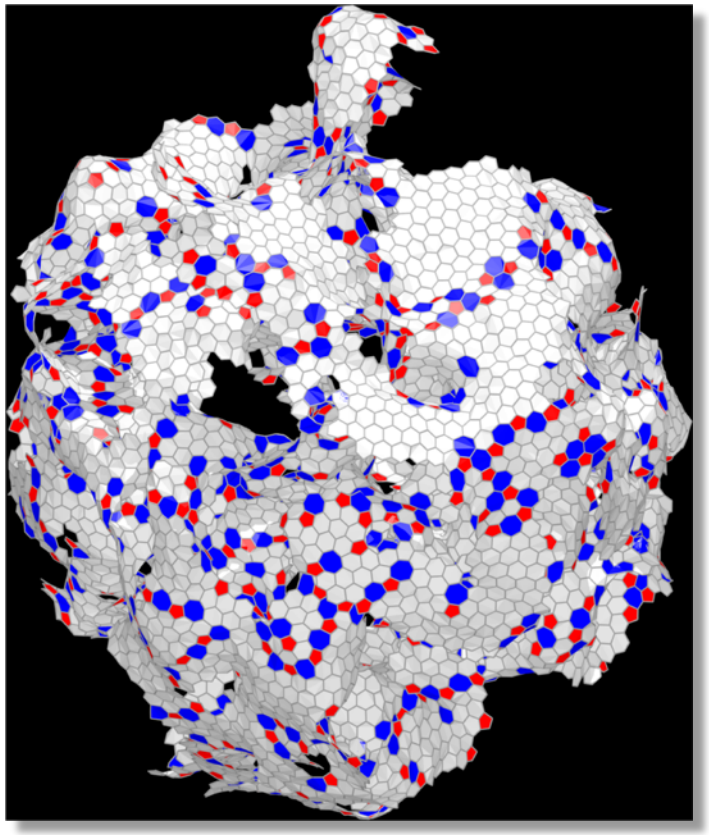
- We prepared all input parameter files in inputs directory, so you can use them as well.

```
$ cp inputs/rxmd.in-0* .
```

- Then, you can start submitting your job from **01-relax.sh**, **02-heatup.sh**, and **03-run.sh**.
- Be sure to submit these jobs one by one. Wait until the job status becomes **C** (complete), then submit next job.

RXMD Hands-on : SiC Nanoparticle Oxidation





Nanocarbon synthesis by high-temperature oxidation of nanoparticles,
K. Nomura, et al., *Scientific Reports* **6**, 24109 (2016)

RXMD Hands-on : SiC Nanoparticle Oxidation

- First, copy a tar file **rxmd.sicnp.tar.gz** for this hands-on session to your working directory.

```
$ cp ~magics60/magics/rxmd.sicnp.tar.gz .
$ tar xvfz rxmd.sicnp.tar.gz
```

- The tar command will create a directory called **sicnp**. **sicnp** is going to be your working directory in this hands-on. Change directory to **sicnp**, then copy executable **rxmd** to **sicnp**.

```
$ cd sicnp
$ cp ../rxmd .
```

- You should have following files and directories.

```
01-relax.sh  02-heatup.sh  03-run.sh  cat_pdb.sh*
count_bond.py  rxmd.in
DAT/  ffield  init.sicnp/  inputs/
```

RXMD Hands-on : SiC Nanoparticle Oxidation

- Change directory to **init.sicnp** and type **make** to create initial config.

```
$ cd init.sicnp/
$ make
```

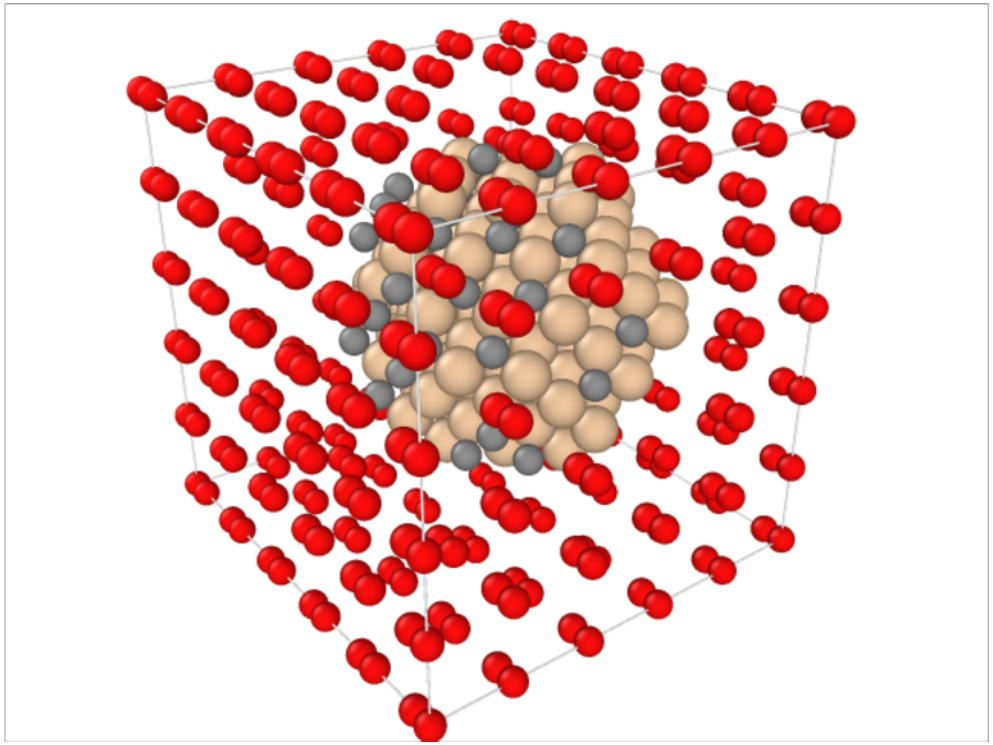
```
ifort -c   geninit.F90
ifort -o geninit geninit.o
./geninit
 input file:   input.xyz
 ffield file:   ../ffield
 nprocs,vprocs       2     2     1     1
 mctot,mc       1     1     1     1
  1-C   2-H   3-O   4-N   5-S   6-Si  7-X
     547   no
rmin(1:3),rmax(1:3):    1.00000E-09    1.00000E-09    1.00000E-09      8.11762E-01    7.71762E-01    7.71762E-01
 sum(lnatoms), lnatoms:         547        338        209
L1, L2, L3, Lalpha, Lbeta, Lgamma:        30.000     30.000     30.000     90.000     90.000     90.000
cp -v rxff.bin ../DAT
'rxff.bin' -> '../DAT/rxff.bin'
```

- Move back to working directory.

```
$ cd ..
```

RXMD Hands-on : SiC Nanoparticle Oxidation

- Initial structure looks like this,



- Number of Atoms : 547
116 Si + 125 C + 306 O
- Lattice Parameters:
30.0(Å) x 30.0(Å) x 30.0(Å)
90.0 90.0 90.0
- Relax free surface and heatup the system upto 2400(K)

Simulation Schedule

- First we relax the free surfaces by quenching, then increase the system temperature up to 2400K by velocity scaling.

1. Surface Relaxation :

rxmd.in-00 : for 2000 MD steps
rxmd.in-01 : for 2000 MD steps
rxmd.in-02 : for 2000 MD steps

2. Heatup:

rxmd.in-03 : to 300K for 5000 MD steps
rxmd.in-04 : to 900K for 5000 MD steps
rxmd.in-05 : to 1500K for 5000 MD steps
rxmd.in-06 : to 1800K for 5000 MD steps
rxmd.in-07 : to 2100K for 5000 MD steps
rxmd.in-08 : to 2400K for 5000 MD steps

3. Measurement :

Keep temperature at 2400K and run.

Simulation Schedule : Surface Relaxation

rxmd.in-00

```
4                    <mdmod>
0.01    2000     <dt> <ntime_step>
100  0.5   100      <treq> <vsfact> <sstep>
100    100          <fstep>  <pstep>
2 1 1              <vprocs>
1  500  1.d-6  10     <isQEq> <NMAXQEq> <QEq_tol> <qstep>
1.0  180           <Lex_fqs> <Lex_k>
.true. .true. .true. <isBinary> <isBondFile> <isPDB>
1.d-8              <ftol>
```

rxmd.in-01

```
4
0.1 2000
100 0.5 100
100 100
2 1 1
1 500 1.d-6 10
1.0 180
.true. .true. .true.
1.d-8
```

rxmd.in-02

```
4
0.25 2000
100 0.9 100
100 100
2 1 1
1 500 1.d-6 10
1.0 180
.true. .true. .true.
1.d-8
```

- **YOUR TASK** : create the three RXMD input parameter files, **rxmd.in-00, rxmd.in-01 rxmd.in-02**, shown on left.
- We provide a template file (**rxmd.in**) that you can copy to **rxmd.in-00**, then edit the RXMD parameters that need to be updated.
- Copy **rxmd.in-00** to **rxmd.in-01**, then edit it.
- Repeat this for **rxmd.in-02**.

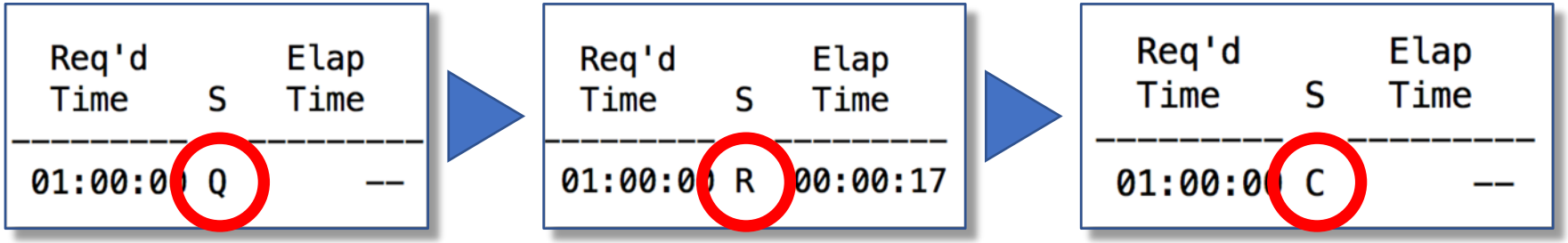
Submit job to USC HPC cluster

- Use **sbatch** to submit your job.

```
$ sbatch 01-relax.sh
```

- Monitor your job status using **squeue**. Your job status goes from **Q** (waiting in queue), **R** (running), and **C** (complete).

```
$ squeue -u ${USER}
```



- While the surface relaxation job is running, we will work on the input parameters for other steps.

Simulation Schedule : Heatup and Measurement Steps

rxmd.in-03

```
7                          <mdmode>
0.25    2000               <dt> <ntime_step>
300  0.9   100             <treq> <vsfact> <sstep>
100    100                 <fstep>  <pstep>
2 1 1                      <vprocs>
1  500  1.d-6  10    <isQEq> <NMAXQEq> <QEq_tol> <qstep>
1.0  180                   <Lex_fqs> <Lex_k>
.true. .true. .true. <isBinary> <isBondFile> <isPDB>
1.d-8                      <ftol>
```

- **YOUR TASK** : create the six input files from **rxmd.in-03** to **rxmd.in-08**.
- Red highlighted parameters are the difference from the previous step.

rxmd.in-04

```
7
0.25 2000
900 0.9 100
100 100
2 1 1
1 500 1.d-6 10
1.0 180
.true. .true. .true.
1.d-8
```

rxmd.in-05

```
7
0.25 2000
1500 0.9 100
100 100
2 1 1
1 500 1.d-6 10
1.0 180
.true. .true. .true.
1.d-8
```

rxmd.in-06

```
7
0.25 5000
1800 0.9 100
100 100
2 1 1
1 500 1.d-6 10
1.0 180
.true. .true. .true.
1.d-8
```

rxmd.in-07

```
5
0.25 5000
2100 0.9 100
100 100
2 1 1
1 500 1.d-6 10
1.0 180
.true. .true. .true.
1.d-8
```

rxmd.in-08

```
5
0.25 5000
2400 0.9 100
100 100
2 1 1
1 500 1.d-6 10
1.0 180
.true. .true. .true.
1.d-8
```

Submit job to USC HPC cluster

- After you have created all input parameter files, use **sbatch** to submit the heatup step.

```
$ sbatch 02-heatup.sh
```

- Monitor your job status using **squeue**.

```
$ squeue –u ${USER}
```

- When the heatup step finishes, i.e. job status becomes **C** (complete) status, submit the measurement step job using **03-run.sh**.

```
$ sbatch 03-run.sh
```

Analyze Simulation Result : Visualize Atom Trajectory

- While your job is running, checkpoint data (.bin) atom trajectory (.pdb files) and connectivity information (.bnd) will be saved into **DAT** directory.

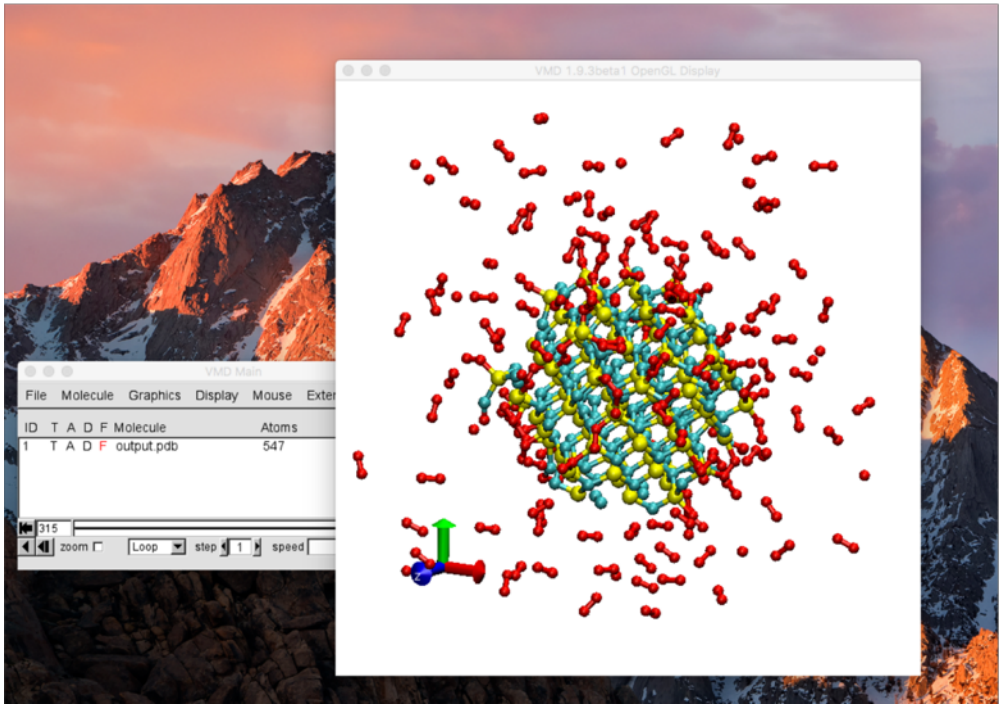
```
$ ls DAT/
000000000.bin
000000000.bnd
000000000.pdb
000000100.bin
000000100.bnd
000000100.pdb
...
```

- To visualize atom trajectory with VMD, we need to concatenate PDB files from different MD steps into one PDB file with proper separator keyword.
- Also, every line must have the same atom through all MD frames.

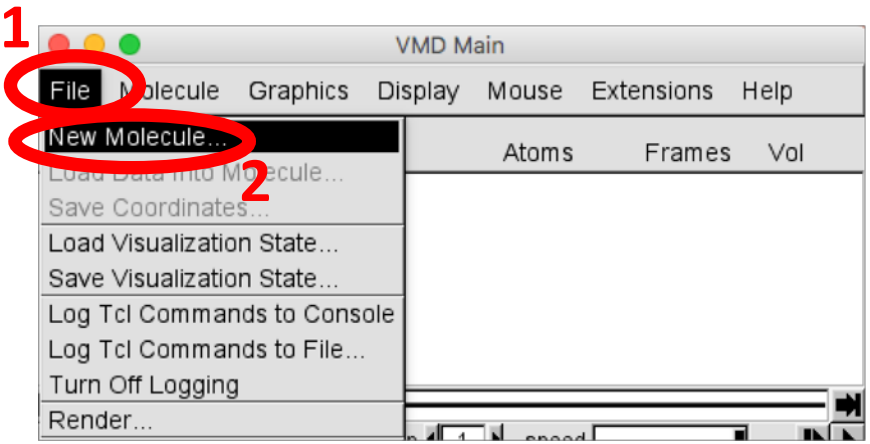
Analyze Simulation Result : Visualize Atom Trajectory

- A simple BASH script **cat_pdb.sh** is included in the tarball.
- **cat_pdb.sh** concatenates all PDB files under **DAT** directory into one PDB file.
- Transfer the output file (**output.pdb**) to your computer and load it on VMD.

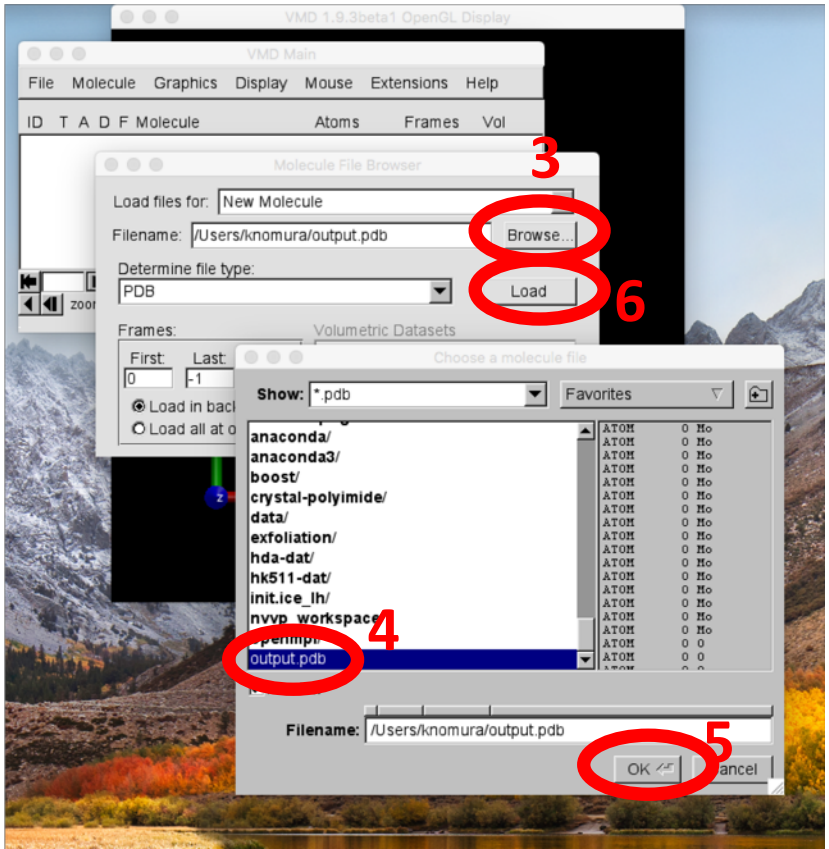
```
$ ./cat_pdb.sh
DAT/000000000.pdb
DAT/000000100.pdb
DAT/000000200.pdb
DAT/000000300.pdb
…
'all.pdb' -> 'output.pdb'
```



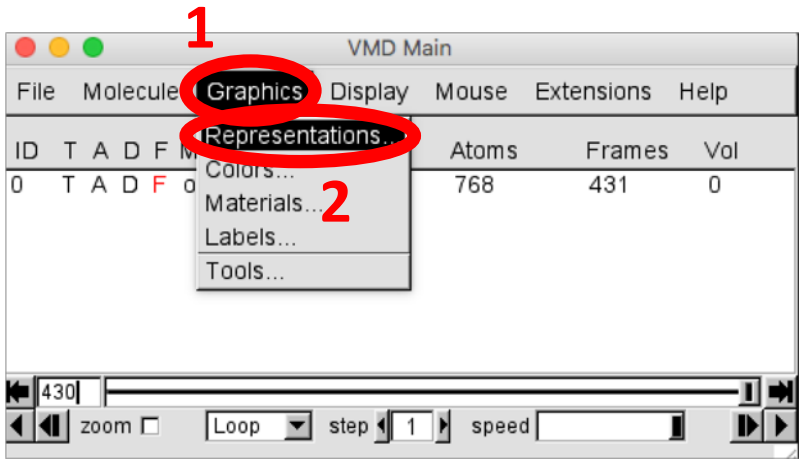
Analyze Simulation Result : Load output.pdb to VMD

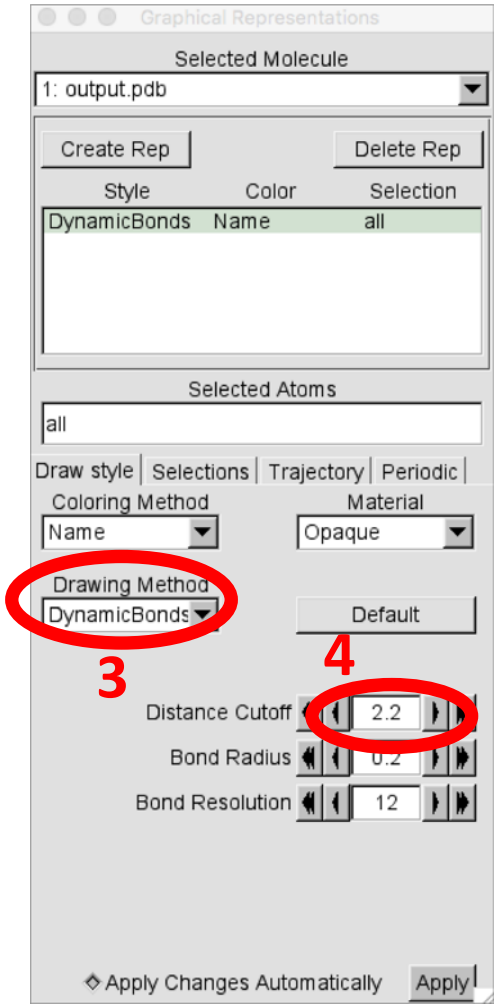


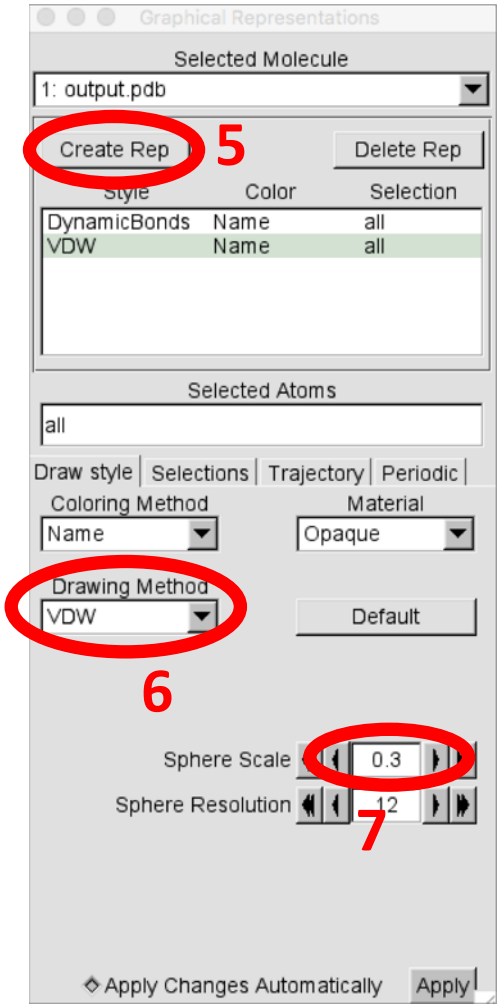
1. File
2. New Molecule
3. Browse
4. Select **output.pdb**
5. OK
6. Load



Analyze Simulation Result : Change Atom Representation

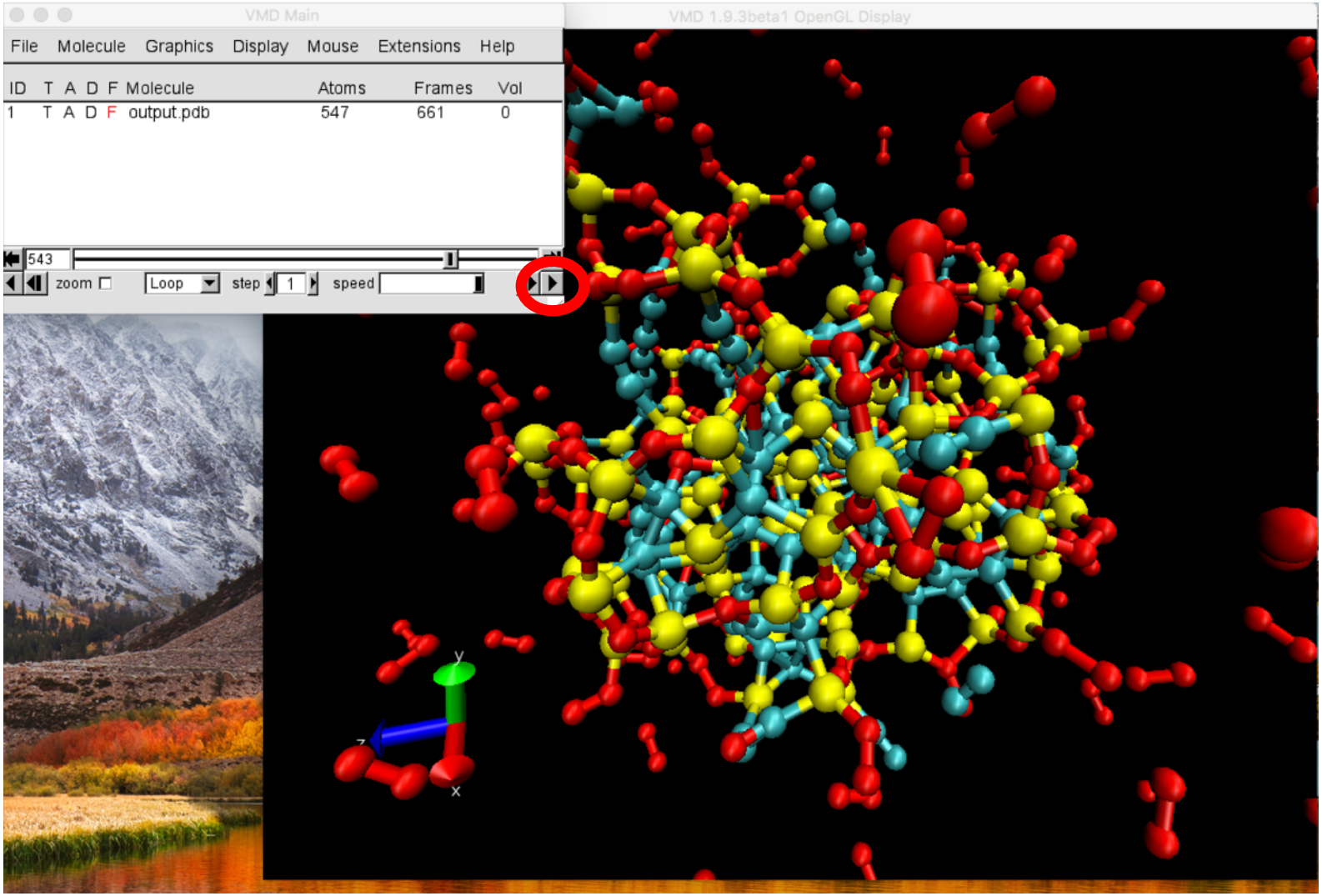






1. Graphics
2. Representations..
3. Drawing Method : Select **DynamicBonds**
4. Change Distance Cutoff to be **2.2**
5. Create Rep
6. Select **VDW**
7. Sphere Scale **0.3**
8. Adjust viewing angle with mouse

Analyze Simulation Result : Play Animation



Analyze Simulation Result : Bond Analysis

- A simple Python script **count_bond.py** is included in the tarball.
- **count_bond.py** counts the number of bonds of each bond type.
- No argument is necessary, just run **count_bond.py** from your working directory that has **DAT** directory.

```
$ ./count_bond.py | tee bond.txt
```

- You will see output below.

```
...
./DAT/000040000.bnd : 1-1  30  3-6  162  1-3  18  3-3  290  1-6  744  6-6  0
./DAT/000040100.bnd : 1-1  32  3-6  158  1-3  16  3-3  290  1-6  744  6-6  0
./DAT/000040200.bnd : 1-1  32  3-6  166  1-3  18  3-3  290  1-6  742  6-6  0
./DAT/000040300.bnd : 1-1  32  3-6  158  1-3  18  3-3  288  1-6  744  6-6  0
./DAT/000040400.bnd : 1-1  32  3-6  170  1-3  18  3-3  288  1-6  740  6-6  0
...
```

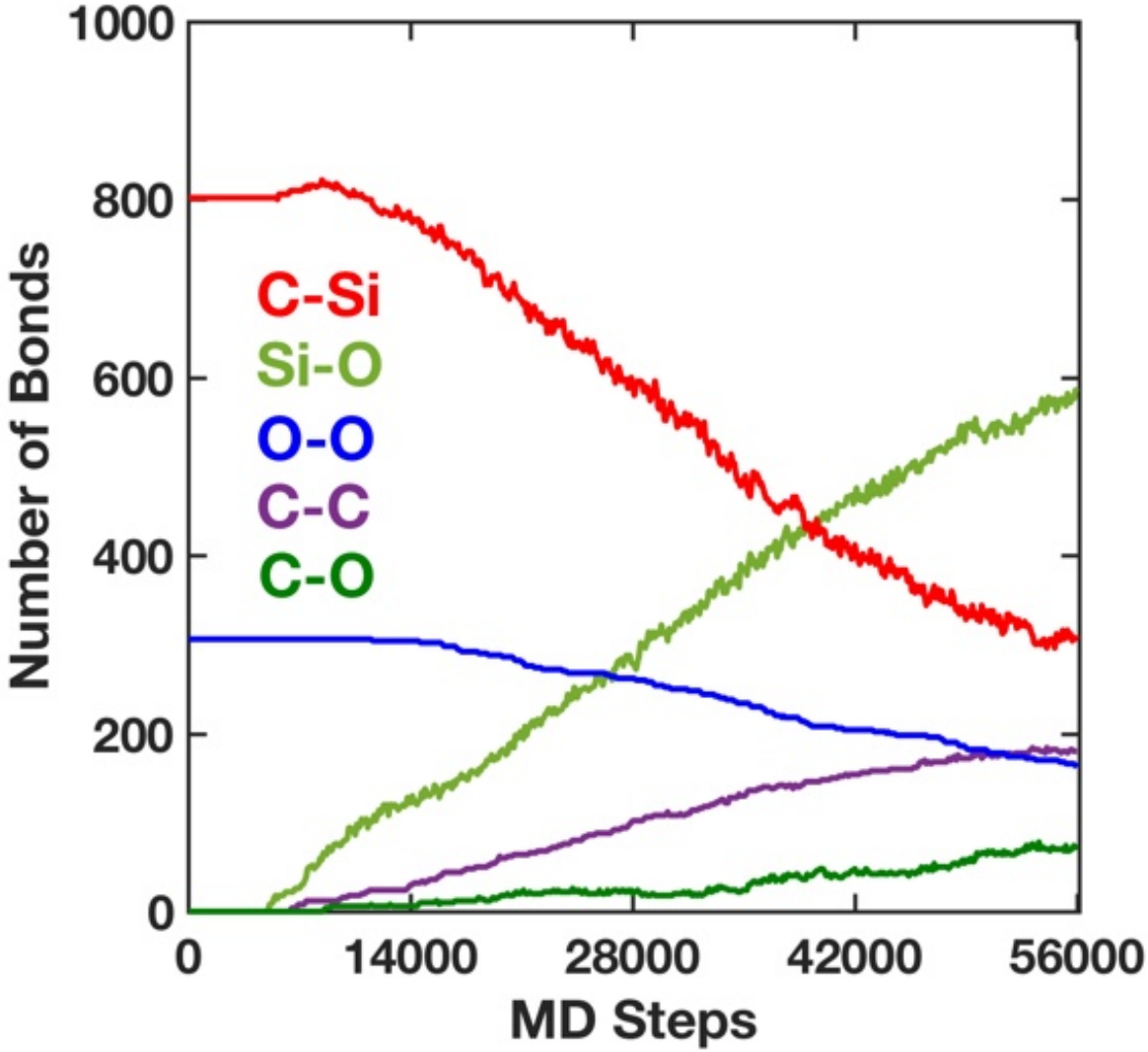
Analyze Simulation Result : Bond Analysis

- Blue columns are atom type combinations and red columns are the number of bonds.
- 1-C, 3-O and 6-Si, e.g. 1-3 means C-O bond.

```
...
./DAT/000040000.bnd  : 1-1  30  3-6  162  1-3  18  3-3  290  1-6  744  6-6  0
./DAT/000040100.bnd  : 1-1  32  3-6  158  1-3  16  3-3  290  1-6  744  6-6  0
./DAT/000040200.bnd  : 1-1  32  3-6  166  1-3  18  3-3  290  1-6  742  6-6  0
./DAT/000040300.bnd  : 1-1  32  3-6  158  1-3  18  3-3  288  1-6  744  6-6  0
./DAT/000040400.bnd  : 1-1  32  3-6  170  1-3  18  3-3  288  1-6  740  6-6  0
...
```

- Transfer **bond.txt** file to your laptop. Use any software to plot the number of bonds for each bond-type.

Analyze Simulation Result : Bond Analysis



If something went wrong..

- If something went wrong, we can start over by copying the original configuration file, **rxff.bin**, in **init** directory to **DAT**.
- Make sure you are in the working directory for this hands-on (**scinp** here), then type following.

```
$ cp init.sicnp/rxff.bin DAT/
```

- We prepared all input parameter files in inputs directory, so you can use them as well.

```
$ cp inputs/rxmd.in-0* .
```

- Then, you can start submitting your job from **01-relax.sh**, **02-heatup.sh**, and **03-run.sh**.
- Be sure to submit these jobs one by one. Wait until the job status becomes **C** (complete), then submit next job.