IUMD User Guide

Introduction

IUMD stands for 'Indiana University Molecular Dynamics'. It is a FORTRAN 90 software package, meant to be run in parallel, which performs molecular dynamics simulations of matter found in neutron star crusts. Molecular dynamics simulations evolve a set of particles through some length of time by reading in positions and velocities and then integrating the equations of motion in order to find the updated positions and velocities. It is thus a deterministic simulation, unlike Monte Carlo methods. MD simulations are capable of providing thermodynamic information of the simulated material and is thus useful for finding macroscopic properties. The potentials used in IUMD are described by Horowitz et al., Phys. Rev. C 69, 045804 (2004).

Installation

Before intsalliation, you will need to obtain a copy the packages via the github.

`~> git clone https://github.com/zvacanti/Indiana-University-Molecular-Dynamics.git`

This needs to be pushed to the supercomputing platform of choice. It is designed to run on Big Red II at Indiana Universiy. To install, follow these steps.

1. Check that you have the correct modules. The list of required modules may be located under `md_version` → `src` → `MODULE_LIST.txt`
2. Swap and replace any modules needed. To check which modules are currently installed, type `~> module list`. Modules may be obtained by `~> module add MODULE_NAME` and swapped with `~> module swap MODULE_NAME_OLD MODULE_NAME_NEW.`
3. Navigate to the src sub-directory.
4. Run `~> make clean`
5. Run `~> make MDEF=BR2-pgi-cuda iumd_mpi_omp_cuda`

You are now ready to begin using IUMD.

Using IUMD

Once compiled, IUMD is simple to run. Begin by creating a directory to perform the simulation in. Although this step is optional, multiple sequential configuration files will be generated during the run and the output may quickly become difficult to manage. From the installation directory, copy the executable `iumd_mpi_open_cuda` as well as the submission script `BR2.pbs` and spec list `runmd.in`. Alternatively, one may leave `iumd_mpi_open_cuda` in the home directory and instead change the path in the `BR2.pbs` file.

To begin your first run you will need to edit `runmd.in` and `BR2.pbs` to reflect the specifics of the simulation. Pay attention that the number of requested nodes in the `BR2.pbs` match in lines 11 and 29.

Example Run:

runmd.in

```
&run_parameters
   sim_type = 'nucleon', !type of simulation    Options are 'nucleon' and 'ion'
   start = 'md.00013300000.xv8b',   Starting configuration.  'random' and 'nuclear' are other options.
!  tstart = 0.00, !start time
   dt = 2.00, !time step (fm/c)
!
!For warmup steps:
   nwgroup = 1, !groups
   nwsteps = 0, !steps per group
!
!For measurement steps:   ngroup·ntot·nind = number of timesteps; timesteps·dt = simulation length
   ngroup   = 100, !groups
   ntot     = 100, !measurements per group
   nind     = 100, !steps between measurements
   nptensor =  10, !steps between pressure tensor outputs
   ntnorm   = 100, !steps between temperature normalization
   ncom     = 100, !steps between C.O.M. motion cancellations
!
!Parameters controlling configuration output:
   nckpt    = 5000, !checkpoint frequency (number of timesteps)
   nout     = 5000, !configuration output frequency
   ftype    = 'x4b', !positions and velocities, real*4.  Rev.B format.
   append   = .true., !each configuration goes to a single md.out file
!
!System description:
  spec_list =
    30720, 0.0, 1.0, !neutrons   This represents a sim.  with N = 51,200 and Y_P = 0.4
    20480, 1.0, 1.0, !protons
   xmass    = 939.00, !nucleon mass (MeV)
   rho0     = 5.0E-02,!number density
   kT       = 1.00, !kT (MeV)
!  deltakT  = 1.e-4,
!  ndeltakT = 10000,
!
   aspect0  = 1.0d0, 1.0d0, 1.0d0,
   epsdot   = , , ,    Rate of change of sim.  volume in x,y,z.  If only one dim.  is specified,
!                      the simulation will attempt to preserve the sim.  volume.
!Screened Coulomb interaction parameters:
    coulomb = 'screened-coulomb', !type of coulomb interaction
    xlambda = 10.000, !coulomb screening length (fm)
!   rccut   = 4.0, 'lambda', !  coulomb cut-off (fm)
!
!Nuclear interaction parameters:
   nuclear  = 'HPP', !type of nuclear interaction
   aa       = 110.000, !parameter a (MeV)
   bb       = -26.000, !parameter b (MeV)
   cc       = 24.000, !parameter c (MeV)
   xpacket  = 1.250, !gaussian range parameter (fm^2)
!
!Random number generator:
   irnd     = 4, !grnd
   iseed    = 11,
!MPI and CUDA parameters
!  procdim  = 2, 2, !MPI process grid dimensions
   nschunk  = 2, !number of source chunks
thrdBlkDim = 64, 4 !CUDA thread block and block grid dimensions
/
```

BR2.pbs

```
##
## MD_6.3.1 code using
## F90+MPI+OMP+CUDA
##
##
##
#PBS -S /bin/bash
#PBS -N run
#PBS -V
#PBS -q gpu
#PBS -l nodes=32:ppn=16,walltime=00:30:00    The node count here should match that on the final line. (-n ##)
#
# -n   Total number of MPI processes (default:  1)
# -N   Number of MPI processes per node.  (XE6:  1-32; XK7:  1-16)
# -S   Number of MPI processes per socket (XE6:  1-16; XK7:  1-16)
# -d   Number of cores per MPI process (for OpenMP: XE6:  1-32; XK7:  1-16)

export OMP_NUM_THREADS=16

cd $PBS_O_WORKDIR
echo -e "\n--------------------------------------------------------------------"
echo "Node file:  $PBS_NODEFILE"
echo "Nodes:"
cat -n $PBS_NODEFILE
echo -e "\n--------------------------------------------------------------------"
checkjob -v $PBS_JOBID
echo -e "\n--------------------------------------------------------------------"

aprun -n 32 -N 1 -d 16 -cc none ~/iumd_mpi_omp_cuda    The path to the executable goes here. In this example, the executable is left in the home directory for convienence.
```

Other than the above files, you should have an executable, either in the same directory or at a location specified in the BR2.pbs script, along with a configuration if you choose to use it. For this example, we will run from a configuration we already have. Configuration files come with the `.xv8b` or `.xv4b` extensions. These files are not human-readable, but consist of position ('x') and velocity ('v') information to 8-star or 4-star precision in FORTRAN. Another option is to read in a positon-only file (`.x4b, .x8b`) and then give them a boltzman distribution for velocities.

Once we've edited the `runmd.in` and `BR2.pbs` files as shown above, we check to make sure all the files are in our directory.

```
user@login1:  example> ls
BR2.pbs   md.00013300000.xv8b   runmd.in   iumd_mpi_omp_cuda
```

Everything is there and our initial configuration file matches the line in the runmd file, so we submit it.

```
user@login1:  example> qsub BR2.pbs
2261959
```

We come back later to take a look at the generated output.

```
user@login1:  example> ls
BR2.pbs                md.00013310000.xv8b    md.00013320000.xv8b      runmd.in
input.in               md.00013320000.log     md.traj.00013320000.x4b  run.o2250931
md.00013300000.xv8b    md.00013320000.meas    run.e2250931
```

These sequentially numbered files are new configurations which have been generated. The `run.e` file contains any error messages from the machine. The `run.o` file contains information about the codes performance, as well as the value of the pressure tensor and energy values of each step of the run. The `.xv8b` files are the generated configurations. Another simulation could be started from one of these configurations.

This is especially useful for long jobs which may take a long time to run. A job can be run for some number of steps, and then continued later to finish.

If starting from random, it will be necessary to equilibrate the file before it is possible to learn much from the configurations. This can be checked by graphing the energy terms of the `run.o` file.

```
$awk '{if ($1 == "e") print $0}' run.o > file.txt
$gnuplot
>plot "file.txt" using 3:6 with lines
```

Once this graph has leveled off, it is safe to work with the generated configurations. This `run.o` file also contains information on the utilities used and the output of the pressure tensor. Rows beginning with 'e' are step, time, Vtot, Ttot, E, and pressure. Rows beginning with 'p' are step, time, P_xx, P_xy, P_xz, P_yy, P_yz, and P_zz.

Parameter Values The following table represents parameters that a user may wish to change during a typical run. Parameters that are only relevant for the machine are mostly neglected.

`sim_type`	'nucleon', 'ion'
`start`	'random'
	Configuration files with extension .x4b, .x8b, .xv4b, .xv8b, .xva8b for version 6.2.
	Other versions may require other file types.
`tstart`	No real number will crash the program, but only positive reals are suggested.
`dt`	No real number will crash the program, but only positive reals are suggested.
`ngroup, ntot, nind`	Any integer. These three multiplied together give the number of timesteps.
`nptensor, ntnorm, ncom`	Any integer.
`nckpt, nout`	Any integer
`ftype`	.x4b, .x8b, .xv4b, .xv8b
`append`	.true., .false.
`spec_list =`	Number of neutrons, number of protons. Any fraction is allowed, however
	the total number of particles must conform to $800 \cdot 2n$, $n = 0, 1, 2, ...$ $n = 51200$ is common.
`rho0`	Any positive real number
`kT`	Any positive real number
`aspect0`	Any positive real number
`epsdot`	Any positive real number. However if the volume preservation is on,
	the simulation may crash if a dimension goes to zero.
`coulomb`	'screened-coulomb', '' (blank to turn it off)
`xlambda`	Any positive real number
`nuclear`	'HPP', '' (blank to turn it off)

The remaining parameters are either physical constants or parameters fed directly to the machine.

Common Runs

Equilibrating a Sample Should you wish to take a random configuration to a stable one, you will need to change the start value to 'random' and allow the configuration some time. The amount of required time is dependent on the parameters of the run, but a million time-steps is usually a good value. To check if the run has reached an equilibrium, graph the energy column. A data file of just the energy vs. time step may be obtained by running

```
awk '{if$1==''e'') print $0}' run.o > FNAME.txt
```

Stretching Stretching runs are relatively simple to run. The only change unique to this run is to set `epsdot` so that one axis is changing length. Note that the other axis should be left blank. If they are set to 0 instead of being left blank, the system will no longer preserve the volume of the piece, and thus the density is changing as the simulation runs. If the axis in question is being compressed, be careful that the

simulation does not run so long that that axis contracts to zero; the simulation will attempt to continue, but the output will not be reliable.

Ion Ion runs may be accomplished by turning off the nuclear force as described above, making all the particles into protons, commenting out the neutron line, and changing the sim_type to 'ion'.

Magnetic Field This kind of run is slightly more complicated than the others listed above. To accomplish a run with a uniform magnetic field, you will need to first change the source code and then recompile. This magnetic field is hard-coded in the z-direction, so you will need to rotate your configuration so that the z-direction coincides with the direction you wish the magnetic field to point in.

To change the magnetic field, locate in `md_globals.f` line 144. Edit the values appropriately, and then recompile. The new executable will be still be called `iumd_mpi_omp_cuda`, although you can use `mv` to rename it to distinguish it from other IUMD executables. Be sure to change the path in the `BR2.pbs` file to match this new executable.

IUMD File Formats & Custom Configurations If one wishes to generate custom configurations from scratch, such as for a specific lattice, it is necessary to know which file types IUMD can handle. IUMD 6.2.0 and above use 'revision B file formats', a custom format for IUMD. The type of file is encoded in the extension. Each includes the following header:

```
character(10)    code_name     !name of program which created file
character(8)     code_version  !program version
character(8)     xdate         !date configuration was written to file, from fortran date_and_time
character(10)    xdaytime      !time of day, from date_and_time
character(5)     xtimezone     !time zone, from date_and_time
character(20)    sim_type      !simulation type
real(dbl)        time          !simulation time stamp (REQUIRED)
real(dbl)        rho           !particle number density (REQUIRED)
real(dbl)        aspect(3)     !aspect ratio of box edge lengths (REQUIRED)
real(dbl)        ev            !average potential energy par particle
real(dbl)        ek            !average kinetic energy per particle
real(dbl)        px            !pressure
real(dbl)        pp(3,3)       !pressure tensor
integer          n             !number of particles (REQUIRED)
```

Here, the type 'dbl' refers to the fortran double precision
`integer, parameter :: dbl=kind(1.0d0)`.

Any header information that is not included should be set to blanks for characters or zero for numbers. Particle data are as follows:

```
real(dbl)        x(3,n)        !positions
real(dbl)        v(3,n)        !velocities
real(dbl)        a(3,n)        !accelerations
real(dbl)        zii(n)        !Z values
real(dbl)        zii(n)        !A values
character(6)     ctype(n)      !character string indicating particle type
```

There are several revision B unformatted file types, as seen below.

```
x4b                          - real*4 positions
xv4b                         - real*4 positions, velocities
x8b                          - real*8 positions
xv8b                         - real*8 positions, velocities
xva8b                        - real*8 positions, velocities, accelerations
```

The following code is given to write this information to an unformatted file.

```
inquire(14,open=xopnd)
if(.not.xopnd) then
   if(xappend) then
   open(14,FILE=file,STATUS='UNKNOWN',FORM='UNFORMATTED',POSITION='APPEND')
   else
      open(14,FILE=file,STATUS='UNKNOWN',FORM='UNFORMATTED')
   endif
endif

call date_and_time(xdate,xdaytime,xtimezone)
write(14) xftype
write(14) code_name,code_version
write(14) xdate,xdaytrime,xtimezone
write(14) sim_type
write(14) time, rho, aspect, ev, ek, px, pp, n
select case (trim(xftype))
   case('x4b')
      write(14) (real(x(1,i)), real(x,2,i)), real(x,3)), i=0, n-1)    case('xv4b')
      write(14) (real(x(1,i)), real(x,2,i)), real(x,3)) &
            real(v(1,i)),real(v(2,i)),real(v(3,i)), i=0, n-1)
case('x8b')
      write(14) (x(1,i), x(2,i), x(3,i), i=0,n-1)
   case('xv8b')
      write(14) (x(1,i), x(2,i), x(3,i), v(1,i), v(2,i), v(3,i), i=o,n-1)
   case('xva8b')
      write(14) (x(1,i), x(2,i), x(3,i), v(1,i), v(2,i), v(3,i), &
            a(1,i), a(2,i), a(3,i), i=0,n-1)
end select

if(xclose) close(14)
```

Similiarly, formatted file types are:

```
xfb                         - positions
xvfb                        - positions, velocities
xZAfb                       - positions, charges, masses
xvZAfb                      - positions, velocities, charges, masses
xvaZAf                      - positions, velocities, accelerations, charges, masses
XYZ                         - Standard XYZ format for VMD
ZAfb                        - list of numbers of ions of each charge and mass
```

The following code is given to write this information to a formatted file.

```
inquire(14,opened=xopnd)
if(.not.xopnd) then
   if(xappend) then
      open(14,FILE=file,STATUS='UNKNOWN',FORM='FORMATTED',POSITION='APPEND')
   else
      open(14,FILE=file,STATUS='UNKNOWN',FORM='FORMATTED')
   endif
endif

call date_and_time(xdate,xdaytime,xtimezone)
if(trim(xftype).ne.'XYZ') then
   write(14,10010) trim(xftype)
   write(14,10010) trim(code_name), trim(code_version)
   write(14,10010) trim(xdate),trim(xdaytime),trim(xtimezone)
   write(14,10010) trim(sim_type)
   write(14,10040) time
   write(14,10044) rho
   write(14,10050) aspect(1),aspect(2),aspect(3)
   write(14,10054) ev,ek,px
   write(14,10058) pp
   write(14,10060) n
endif

10010 format('# ',a,2x,a,2x,a)
10040 format('# ',f14.2)
10044 format('# ',es19.12)
10050 format('# ',3(1x,f16.12))
10054 format('# ',3(1x,es19.12))
10058 format('# ',/,2('# ',3(1x,es19.12),/,'# ',3(1x,es19.12)))
10060 format(' ',i10)

   select case(trim(xftype))
      case('xfb')
         write(14,100114) (x(1,i), x(2,i), x(3,i), i=0,n-1)
10114    format(es24.16,es24.16,es24.16)
      case('xvfb')
         write(14,10124) (x(1,i),x(2,i),x(3,i),v(1,i),v(2,i),v(3,i),i=0,n-1)
10124    format(es24.16,es24.16,es24.16,es24.16,es24.16,es24.16)
      case('xZAfb')
         write(14,10134) (x(1,i), x(2,i), x(3,i), zii(i), aii(i),i=0,n-1)
10134    format(es24.16,es24.16,es24.16,1x,f8.3,1x,f8.3)
      case('xvZAfb')
         write(14,10144) (x(1,i),x(2,i),x(3,i),v(1,i),v(2,i),v(3,i) &
            ,zii(i),aii(i), i=0,n-1)
10144    format(es24.16,es24.16,es24.16,es24.16,es24.16,es24.16, 1x,f8.3,1x,f8.3)
      case('xvaZAf')
         write(14,10154) (x(1,i),x(2,i),x(3,i),v(1,i),v(2,i),v(3,i) &
            ,a(1,i), a(2,i),a(3,i),zii(i),aii(i), i=0,n-1)
10154    format(9(es24.16),1x,f8.3,1x,f8.3)
      case('XYZ')
         write(14,10160) n
10160    format(i10)
         write(14,10164) (ctype(i),x(1,i),x(2,i),x(3,i), i=0,n-1)
10062    format('# ',3(1x,f12.6))
```

```
10164    format(1x,a6,1x,f12.6,1x,f12.6,1x,f12.6)
   end select
   if(xclose) close(14)
```

Similar code exists to write for formatted and unformatted revision A file formats. However, it is not recommended that these be used, as they are intended for an earlier version of IUMD.