POST-PROMD: an open-source POST-PROcessing software for Molecular Dynamics simulations

Ashkan Shekaari[1,*]

[1] *Department of Physics, K. N. Toosi University of Technology, Tehran 15875-4416, Iran*

(Dated: June 11, 2025)

In this work, a new software, named POST-PROMD, has been developed, which is capable of computing three main phase transition quantities including the Lindemann index, mean square displacement (MSD) of atoms, and radial distribution function (RDF) of systems of particles, only via post-processing the output file containing atomic positions generated during molecular dynamics (MD) simulations. POST-PROMD runs independently (is not a module patched to any other software), is length-scale-independent (consistent with both quantum mechanical and classical simulations), is released for the first time under the GNU GPL (Version 3) as a free and open-source package, has originally been written to post-process the atomic-positions file generated during Car-Parrinello MD simulations of Quantum ESPRESSO, and is software-independent nevertheless, capable of post-processing the MD output of any other software.

PACS numbers: 02.70.Ns, 05.70.Fh, 07.05.Tp
Keywords: POST-PROMD, Software Development, Molecular Dynamics Simulation, Phase Transition, Open Source

I. INTRODUCTION

One of the most important capabilities of Quantum ESPRESSO [1–3] is to perform density functional molecular dynamics (DFMD) simulations within both Car-Parrinello (CP) [4] and Born-Oppenheimer (BO) [5, 6] frameworks. However, there is still a lack of required codes/modules to post-process raw outputs of such MD simulations generated by the CPV package. As an illustration, the output file `"cp.pos"` of CPMD simulations—including atomic positions of the system at every time-step—contains important information about phase transition/finite-temperature behavior/dynamics of that system, extracting information from which is left to the user.

To this end, i.e., to extract such important information contained in `"cp.pos"`, we have written a post-processing package for MD simulations, called POST-PROMD (acronym for "a POST-PROcessor for Molecular Dynamics simulations"), which computes three main phase transition indicators, including (i) the Lindemann index of the whole system, (ii) mean square displacement (MSD) of every atom/particle of the system, and (iii) radial distribution function (RDF) of the whole system.

The present package has been written both in Fortran 95 [7] and Linux Bash [8, 9], and is release for the first time ever, as a free and open-source package, with assigned version `v.1.0.0`, under the GNU General Public License (Version 3, 29 June 2007) [10]. The logo of POST-PROMD has also been illustrated in Fig. 1.

FIG. 1. Logo of POST-PROMD, designed by the author using `https://app.logo.com`.

II. PHASE TRANSITION IN SOLIDS

Investigating phase transition/finite-temperature behavior of the condensed matter at the computational level of research using MD simulations is carried out via calculating a number of indicators the three important of which are the Lindemann index, MSD, and RDF, as mentioned before. These quantities are prevalently used to probe phase transition in solid materials, namely solid-to-liquid transition; however, they can also be applied to so-called few-body systems of atoms/molecules whose bulk phases are not solid. The point at which phase transition takes place is the same as the melting temperature (T_m) of the solid; at this point, temperature dependence of the three phase transition indicators aforementioned undergo a sudden, substantial increase compared to their former behaviors. The facts aforementioned have clearly been demonstrated in an investigation on phase transition of B_{36} nanocluster [11].

In the following, we describe the three phase transition quantities of interest, and illustrate their associated computer codes.

III. LINDEMANN INDEX

The Lindemann index Δ_{rms}, also known as global Lindemann index, or root-mean-square bond length

* shekaari.theory@gmail.com; shekaari@email.kntu.ac.ir

fluctuation [12], is indeed the extent of fluctuations in bond lengths between atoms/particles of the system, averaged both over all those atoms, and the total time span of the MD simulation. This quantity is defined as

$$\Delta_{rms} = \frac{2}{N(N-1)} \sum_{i>j} \frac{\left(\langle r_{ij}^2 \rangle_t - \langle r_{ij} \rangle_t^2 \right)^{1/2}}{\langle r_{ij} \rangle_t}, \quad (1)$$

where N is total number of atoms, r_{ij} is distance between atoms i and j, and $\langle ... \rangle_t$ is time (t) average over the entire trajectory/simulation. $2/N(N-1)$ also means average over total number of atomic pairs/bonds $N(N-1)/2$. The two modules of POST-PROMD, namely `"lind.sh"` and `"ln.source"` are used to compute the Lindemann index of the system according to Eq. 1, being described as follows. Here, files suffixed by `".sh"` are purely Bash scripts, while those with suffix `".source"` have mostly been written in Fortran 95.

A. lind.sh

```
#!/bin/bash
# Copyright (C) 2024 Ashkan Shekaari/POST-PROMD.
# This file has been distributed under the terms of
# GNU General Public License (Version 3, 29 June 2007).
# PLEASE, take a look at the file 'License'
# in the root directory of the present distribution,
# or visit https://www.gnu.org/licenses/gpl-3.0.txt.
# For details: Please, take a look at 'README' file.
cp par.input par2.dat
cat ln.source >> par2.dat
mv par2.dat make_lind.sh
bash make_lind.sh
rm make_lind.sh
```

B. ln.source

```
# Copyright (C) 2024 Ashkan Shekaari/POST-PROMD.
# This file has been distributed under the terms of
# GNU General Public License (Version 3, 29 June 2007).
# PLEASE, take a look at the file 'License'
# in the root directory of the present distribution,
# or visit https://www.gnu.org/licenses/gpl-3.0.txt .
# For details: Please, take a look at 'README' file.
eql=$(((ign0/iprint0)*n0))
m=$((n0 + 1))
cat cp.pos | sed '1~'$m'd' > pos.pos2
cat pos.pos2 | sed '1,'$eql'd' > pos.pos
rm pos.pos2
cat > module.f90 << EOF
module initial_setup
implicit none
save
integer, parameter ::       n = $n0
integer, parameter :: int_num = n*(n - 1)/2
integer, parameter ::   nstep = $nstep0
integer, parameter ::  iprint = $iprint0
integer, parameter ::     ign = $ign0
integer, parameter ::     rat = (nstep - ign)/iprint
integer, parameter :: line_f1 = rat*n
integer, parameter :: line_f2 = rat*int_num
```

```
real, parameter ::    dt = $dt0
real, parameter :: t_tot = rat*dt
real, parameter ::     c = dt/t_tot
end module
EOF
ifort -c module.f90
cat > fluct1.f90 << EOF
program distance_pair
use initial_setup
implicit none
integer i, j, k
real*8, dimension(:, :), allocatable :: r
real*8, dimension(:, :), allocatable :: pos
allocate(pos(3, line_f1))
allocate(r(line_f1, line_f1))
open(1, file = 'pos.pos')  ! input file
open(2, file = 'rij.data') ! output file containing rij
do i = 1, line_f1
  read(1, *) pos(:, i)
end do
do i = 1, line_f1, n
  do k = 1, n - 1
    do j = i + k, i + n - 1
      r(j, i + k - 1) = sqrt((pos(1, j) - pos(1, i + k - 1))**2 &
      + (pos(2, j) - pos(2, i + k - 1))**2 &
      + (pos(3, j) - pos(3, i + k - 1))**2)
      write(2, *) r(j, i + k - 1)
    end do
  end do
end do
deallocate(pos)
deallocate(r)
end program
EOF
cat > fluct2.f90 << EOF
program rmsblf
use initial_setup
implicit none
integer i, k
real*8, dimension(:), allocatable :: rijsum, rij2sum, rij
real*8, dimension(:), allocatable :: rijtimeavg, rij2timeavg, term
real*8 s, delta_rms
allocate(rijsum(line_f2))
allocate(rij2sum(line_f2))
allocate(rij(line_f2))
allocate(rijtimeavg(line_f2))
allocate(rij2timeavg(line_f2))
allocate(term(line_f2))
open(1, file = 'rij.data') ! input file
do i = 1, line_f2
  read(1, *) rij(i)
end do
do k = 1, int_num
  rijsum(k) = 0.
  rij2sum(k) = 0.
  term(k) = 0.
end do
do k = 1, int_num
  do i = k, line_f2, int_num
```

```
      rijsum(k) = rijsum(k) + rij(i)
      rij2sum(k) = rij2sum(k) + rij(i)*rij(i)
    end do
    rijtimeavg(k) = rijsum(k)*c
    rij2timeavg(k) = rij2sum(k)*c
    term(k) = (sqrt(rij2timeavg(k) - &
    rijtimeavg(k)*rijtimeavg(k)))/rijtimeavg(k)
  end do
  s = sum(term)
  deallocate(rijsum)
  deallocate(rij2sum)
  deallocate(rij)
  deallocate(rijtimeavg)
  deallocate(rij2timeavg)
  deallocate(term)
  delta_rms = s/real(int_num)
  print*,'    >>> delta_rms =', delta_rms
  end program
  EOF
  ifort module.o fluct1.f90 && ./a.out
  ifort module.o fluct2.f90 && ./a.out
  rm *.f90 *.mod a.out pos.pos rij.data *.o
```

IV. MEAN SQUARE DISPLACEMENT (MSD)

In contrast to the Lindemann index and radial distribution function, which are quantities related to the entire system, MSD is a single-particle quantity, and POST-PROMD consequently computes it for every individual atom of the system according to

$$\langle \mathbf{r}_I^2(t) \rangle = \frac{1}{n} \sum_{i=1}^{n} \Big[\mathbf{R}_I \left(t_{0i} + t \right) - \mathbf{R}_I \left(t_{0i} \right) \Big]^2, \qquad (2)$$

where we average over n different time origins t_{0i} distributed over the entire trajectory with the interval `dt0` (included as an input in the file `"par.input"`, described later on in section VI) between any two consecutive t_{0i} for the average. The I runs over number N of atoms, and $\mathbf{R}_I$ is the position vector of the Ith atom with respect to the center of mass (COM) of the system.

The two modules `"msd.sh"` and `"msd.source"`, computing MSD according to Eq. 2, are illustrated in the following.

A. msd.sh

```
#!/bin/bash
# Copyright (C) 2024 Ashkan Shekaari/POST-PROMD.
# This file has been distributed under the terms of
# GNU General Public License (Version 3, 29 June 2007).
# PLEASE, take a look at the file 'License'
# in the root directory of the present distribution,
# or visit https://www.gnu.org/licenses/gpl-3.0.txt.
# For details: Please, take a look at 'README' file.
cp par.input par2.dat
cat msd.source >> par2.dat
mv par2.dat make_msd.sh
bash make_msd.sh
rm make_msd.sh
```

B. msd.source

```
# Copyright (C) 2024 Ashkan Shekaari/POST-PROMD.
# This file has been distributed under the terms of
# GNU General Public License (Version 3, 29 June 2007).
# PLEASE, take a look at the file 'License'
# in the root directory of the present distribution,
# or visit https://www.gnu.org/licenses/gpl-3.0.txt.
# For details: Please, take a look at 'README' file.
eql=$(((ign0/iprint0)*n0))
m=$((n0 + 1))
cat cp.pos | sed '1~'$m'd' > pos.pos2
cat pos.pos2 | sed '1,'$eql'd' > pos.pos
rm pos.pos2
cat > msd.f90 << EOF
program msd_maker
implicit none
integer c, i, j
integer, parameter ::          n = $n0
integer, parameter ::        ign = $ign0
integer, parameter ::      nstep = $nstep0
integer, parameter ::     iprint = $iprint0
real, parameter ::            dt = $dt00
real, parameter ::            au = 2.4189
real, parameter ::           scl = 0.00001
integer, parameter ::       line = ((nstep - ign)/iprint)*n
integer, parameter ::        den = (nstep - ign)/iprint
real, parameter :: dt_cp_dot_pos = dt*iprint*au*scl
real*8, allocatable, dimension(:) :: s, x, y, z
character*12 :: filename
allocate (x(line))
allocate (y(line))
allocate (z(line))
allocate (s(line))
open(0,  file = 'pos.pos')
do i = 1, n
  write(filename,'("msd",i2,".out")') i
  open(unit=i,file=filename)
end do
do i = 1, line
  read(0, *) x(i), y(i), z(i)
end do
do i = 1, n
  c = 0
  s(i) = 0.
  do j = i, line, n
    c = c + 1
    s(i) = s(i) + ((x(j + n) - x(i))*(x(j + n) - x(i)) + &
    (y(j + n) - y(i))*(y(j + n) - y(i)) + &
    (z(j + n) - z(i))*(z(j + n) - z(i)))
    write(i, *) c*dt_cp_dot_pos, s(i)/real(den)
  end do
end do
deallocate (x)
deallocate (y)
deallocate (z)
deallocate (s)
end program
EOF
ifort msd.f90 && ./a.out
rm msd.f90 a.out pos.pos
mkdir msd_output_files
mv msd*.out msd_output_files
```

V. RADIAL DISTRIBUTION FUNCTION (RDF)

The quantity RDF $[g(r)]$, being calculated both in two and three spatial dimensions, computes the average number $\langle N\rangle$ of atoms within the shell volume V_s ($= 2\pi r dr$ in two, and $4\pi r^2 dr$ in three dimensions) of radius r around COM of the system, between $r - dr/2$ and $r + dr/2$, according to

$$g(r) = \frac{\langle N\rangle}{N}\Big/ V_s, \tag{3}$$

where dr is shell width. Indeed, POST-PROMD computes the average of atoms within each shell over the entire simulation, and then repeats the same for the next shell until reaching the outermost that encompasses the entire system. This is why RDF calculation takes a longer time compared to the Lindemann index or MSD. In the following, the two modules "`rdf.sh`" and "`rdf.source`", computing RDF according to Eq. 3, are represented.

A. rdf.sh

```
#!/bin/bash
# Copyright (C) 2024 Ashkan Shekaari/POST-PROMD.
# This file has been distributed under the terms of
# GNU General Public License (Version 3, 29 June 2007).
# PLEASE, take a look at the file 'License'
# in the root directory of the present distribution,
# or visit https://www.gnu.org/licenses/gpl-3.0.txt.
# For details: Please, take a look at 'README' file.
cp par.input par2.dat
cat rdf.source >> par2.dat
mv par2.dat make_rdf.sh
bash make_rdf.sh
rm make_rdf.sh
```

B. rdf.source

```
# Copyright (C) 2024 Ashkan Shekaari/POST-PROMD.
# This file has been distributed under the terms of
# GNU General Public License (Version 3, 29 June 2007).
# PLEASE, take a look at the file 'License'
# in the root directory of the present distribution,
# or visit https://www.gnu.org/licenses/gpl-3.0.txt.
# For details: Please, take a look at 'README' file.
eql=$(((ign0/iprint0)*n0))
m=$((n0 + 1))
cat cp.pos | sed '1~'$m'd' > pos.pos2
cat pos.pos2 | sed '1,'$eql'd' > pos.pos
rm pos.pos2
cat > module.f90 << EOF
module shared_data
implicit none
save
integer, parameter ::     n = $n0
real, parameter ::     rmax = $rmax0
integer, parameter :: nstep = $nstep0, iprint = $iprint0
integer, parameter ::   ign = $ign0
integer, parameter ::  snap = (nstep - ign)/iprint
```

```
real, parameter ::          cf = 0.529177010059357
real, parameter ::     delta_r = $delta_r0
real, parameter ::          pi = 4.*atan(1.)
integer, parameter ::       n2 = nint(rmax/delta_r)
integer, parameter :: line_num = n2*snap
end module
EOF
ifort -c module.f90
cat > rdf.f90 << EOF
program radial_df
use shared_data
implicit none
integer :: i, j, cxy, cxz, cyz, c2
real :: r
real, allocatable, dimension(:) :: x, y, z, d2xy, d2xz, d2yz, d3
real xcom, ycom, zcom
open(1, file = 'pos.pos')
open(2, file = 'rdf2.data')
allocate(x(n*snap))
allocate(y(n*snap))
allocate(z(n*snap))
allocate(d2xy(n*snap))
allocate(d2xz(n*snap))
allocate(d2yz(n*snap))
allocate(d3(n*snap))
do j = 1, snap
  xcom = 0.; ycom = 0.; zcom = 0.
  do i = (j - 1)*n + 1, j*n
    read(1, *) x(i), y(i), z(i)
    xcom = xcom + x(i)
    ycom = ycom + y(i)
    zcom = zcom + z(i)
  end do
  xcom = xcom/real(n)
  ycom = ycom/real(n)
  zcom = zcom/real(n)
  do i = (j - 1)*n + 1, j*n
    x(i) = x(i) - xcom
    y(i) = y(i) - ycom
    z(i) = z(i) - zcom
    d2xy(i) = sqrt(x(i)*x(i) + y(i)*y(i))*cf
    d2xz(i) = sqrt(x(i)*x(i) + z(i)*z(i))*cf
    d2yz(i) = sqrt(y(i)*y(i) + z(i)*z(i))*cf
    d3(i) = sqrt(x(i)*x(i) + y(i)*y(i) + z(i)*z(i))*cf
  end do
  r = 0.
  do while (r <= rmax)
    r = r + delta_r
    cxy = 0; cxz = 0; cyz = 0; c2 = 0
    do i = (j - 1)*n + 1, j*n
      if(d2xy(i) < r + 0.5*delta_r .and. d2xy(i) >= r - 0.5*delta_r) then
        cxy = cxy + 1
      end if
      if(d2xz(i) < r + 0.5*delta_r .and. d2xz(i) >= r - 0.5*delta_r) then
        cxz = cxz + 1
      end if
      if(d2yz(i) < r + 0.5*delta_r .and. d2yz(i) >= r - 0.5*delta_r) then
        cyz = cyz + 1
      end if
```

```
      if(d3(i) < r + 0.5*delta_r .and. d3(i) >= r - 0.5*delta_r) then
        c2 = c2 + 1
      end if
    end do
    write(2, *) r, cxy/(2.*pi*r*delta_r*n), &
    cxz/(2.*pi*r*delta_r*n), cyz/(2.*pi*r*delta_r*n), &
    c2/(4.*pi*r*r*delta_r*n)
  end do
end do
deallocate(x)
deallocate(y)
deallocate(z)
deallocate(d2xy)
deallocate(d2xz)
deallocate(d2yz)
deallocate(d3)
end program
EOF
ifort module.o rdf.f90 && ./a.out
cat > rdf2.f90 << EOF
program radial_df2
use shared_data
implicit none
integer i, j
real, allocatable, dimension(:) :: r, rdf2xy, rdf2xz, rdf2yz, rdf3
real :: s1xy, s1xz, s1yz, s2
open(1, file = 'rdf2.data')
open(2, file = 'rdf5.out')
allocate(r(line_num))
allocate(rdf2xy(line_num))
allocate(rdf2xz(line_num))
allocate(rdf2yz(line_num))
allocate(rdf3(line_num))
do i = 1, line_num
  read(1, *) r(i), rdf2xy(i), rdf2xz(i), rdf2yz(i), rdf3(i)
end do
do j = 1, n2
  s1xy = 0.; s1xz = 0.; s1yz = 0.; s2 = 0.
  do i = j, line_num, n2
    s1xy = s1xy + rdf2xy(i)
    s1xz = s1xz + rdf2xz(i)
    s1yz = s1yz + rdf2yz(i)
    s2 = s2 + rdf3(i)
  end do
  write(2, *) r(j), s1xy/snap, s1xz/snap, s1yz/snap, s2/snap
end do
deallocate(r)
deallocate(rdf2xy)
deallocate(rdf2xz)
deallocate(rdf2yz)
deallocate(rdf3)
end program
EOF
ifort module.o rdf2.f90 && ./a.out
rm *.f90 a.out *.o *.mod rdf2.data pos.pos
mkdir rdf_output_file
mv rdf5.out rdf_output_file
```

VI. THE INPUT

So far, we have demonstrated POST-PROMD modules that perform calculation (placed in <POST-PROMD_ROOT_DIRECTORY/src>). The Bash scripts suffixed with ".sh" use the same input files "cp.pos" and "par.input" to do calculation. The former ("cp.pos") is indeed an output file, formerly generated by the CPV package of QUANTUM ESPRESSO; the latter ("par.input"), on the other hand, is completely user-defined, containing a number of numerical parameters some of which have already been defined in MD input file of QUANTUM ESPRESSO. These input parameters are described in detail, as follows:

- n0 = Total number of particles (also included in MD input file of QUANTUM ESPRESSO).
- nstep0 = Total number of steps (also included in MD input file of QUANTUM ESPRESSO).
- iprint0 = Number of steps between successive output writings (also included in MD input file of QUANTUM ESPRESSO).
- ign0 = Number of steps for thermalization/thermal equilibration, to have reliable statistical averages. In fact, whether or not the initial atomic positions are random, it takes a time for the entire system to reach thermal equilibrium according to the target temperature defined in the simulation. Over this time interval, statistical averages, such as total energy, are not valid because they are only meaningful at thermal equilibrium according to theory (thermodynamics/statistical mechanics). Usually, integer values between 50 and 500 steps are suitable. Evidently, systems with larger numbers of particles take longer times/steps to reach thermal equilibrium (to be more accurate, the user has to check the diagram of total energy vs. simulation step/index to determine the time/step at which total energy begins to get converged).
- dt0 = Time-step, included in (1st row, 2nd column) of the MD output file "cp.pos".
- dt00 = Time-step in MD input file of QUANTUM ESPRESSO.
- delta_r0 = Increment value in radius r ranging from zero (COM of the system) to the outermost shell encompassing the whole system. It is a very sensitive parameter; the value 0.0005 Å works well.
- rmax0 = Distance from COM to the outermost shell of the system. For example, the value 5 Å is suitable for B_{36} nanocluster [11].

Note that in "par.input", there must not be any blank space before and after the equal signs (=). The sample input files ("cp.pos" and "par.input") have also been placed in <POST-PROMD_ROOT_DIRECTORY/input>.

VII. RELEASE INFORMATION

POST-PROMD has been uploaded to a number of repositories including GitHub, and GitLab with name post-promd-v.1.0.0.tar.gz, and to ZENODO with name POSTPROMD-v.1.0.0.zip and the doi:10.5281/zenodo.10982023. All the material included in this distribution is free and open-source software, meaning that one is allowed to redistribute/modify it under the GNU GPL. This package is a new feature; without any warranty, even the implied warranty, of merchantability/fitness for a particular purpose; runs independently; and is not a module patched to any other software such as QUANTUM ESPRESSO; however, the author's main motivation to write the package has been to fill the gap in post-processing the CPMD outputs of QUANTUM ESPRESSO, and this is why the specific format of the file "cp.pos" has been considered as a benchmark (to post-process the MD outputs of other computational packages, the user has to apply only minor changes to come up with desired input-file-formatting).

VIII. HOW TO RUN

No need to install, POST-PROMD would be ready to run only via unzipping the source file post-promd-v.1.0.0.tar.gz. To this end, the user needs to pursue the following simple steps:

1. Unzip the source file.
2. Place "cp.pos" in <POST-PROMD_ROOT_DIRECTORY/src>.
3. Edit "par.input" according to your MD setup, then place it in <POST-PROMD_ROOT_DIRECTORY/src>.

4. Open a terminal in `<POST-PROMD_ROOT_DIRECTORY/src>`.

5. To compute the Lindemann index, run in terminal: `bash lind.sh`.

6. To compute MSD, run in terminal: `bash msd.sh`.

7. To compute RDF, run in terminal: `bash rdf.sh`.

Note that the Fortran part of POST-PROMD, namely the files suffixed by `".source"`, are compatible with the Intel Fortran Compiler ifort; therefore, the user either has to have ifort installed, or modify the `".source"` files simply via replacing the word `"ifort"` with any other available Fortran compiler, such as `gfortran`, `mpif90`, etc. The output files generated by POST-PROMD are also described as follows:

1. The output of `bash lind.sh` is displayed in terminal.

2. Those of `bash msd.sh` will be in the directory `<msd_output_files>`, each of which belongs to one atom of the system. Each of these files, named with the template `"msd***.out"`, contain two columns of numerical data, which, in order, are: time-step, and MSD. The user has to plot these files.

3. The output of `bash rdf.sh`, namely `"rdf5.out"`, will be in the directory `<rdf_output_file>`; the number `'5'` stands for the five columns of numerical data contained in that file, being, in order: r (radius), rdf (RDF values) on xy plane, rdf on xz plane, rdf on yz plane, and rdf in three dimensions. The user also has to plot `"rdf5.out"`.

IX. EXAMPLE FILES

The present distribution of POST-PROMD also contains example files, placed in the directory `<POST-PROMD_ROOT_DIRECTORY/example>`, which are related to the CPMD simulation of the unit cell of SLSiN (single-layer silicon nitride; chemical formula: Si_3N_4; containing 14 atoms) [13], at $T = 5$ K. The related outputs have also been placed in the directory `<POST-PROMD_ROOT_DIRECTORY/example/reference>`.

X. CONCLUDING REMARKS

We have developed a new software named POST-PROMD, capable of post-processing the atomic-positions output file "cp.pos", generated by Car-Parrinello molecular dynamics (CPMD) simulations of Quantum ESPRESSO, in order to compute three main phase transition indicators including the Lindemann index, per-atom mean square displacement (MSD), and radial distribution function (RDF) of the system. Analyzing the temperature dependence of such quantities is widely used to investigate the finite-temperature behaviors, and to estimate the melting points of systems of atoms/particles. POST-PROMD is an independent package, not a module patched to any other software, and can then independently run to post-process any file containing atomic positions of the system stored at all time-steps as an input file generated by any MD software. Nevertheless, the author's main motivation has been to post-process the atomic-positions file (`"cp.pos"`) generated by Quantum ESPRESSO; this is why the specific format of this file has been considered as a benchmark in POST-PROMD. This package is free and open-source, released for the first time under the GNU GPL (Version 3), being technically independent of whether the MD simulation is performed classically or quantum mechanically, so is also applicable to classical simulations as well.

REFERENCES

[1] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G.L. Chiarotti, M. Cococcioni, I. Dabo, A.D. Corso, S. de Gironcoli, S. Fabris, G. Fratesi, R. Gebauer, U. Gerstmann, C. Gougoussis, A. Kokalj, M. Lazzeri, L. Martin-Samos, N. Marzari, F. Mauri, R. Mazzarello, S. Paolini, A. Pasquarello, L. Paulatto, C. Sbraccia, S. Scandolo, G. Sclauzero, A.P. Seitsonen, A. Smogunov, P. Umari, R.M. Wentzcovitch, Quantum ESPRESSO: a modular and open-source software project for quantum simulations of materials, J. Phys.: Cond. Matter 21 (2009) 395502.

[2] P. Giannozzi, O. Andreussi, T. Brumme, O. Bunau, M. Buongiorno Nardelli, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, M. Cococcioni, N. Colonna, I. Carnimeo, A. Dal Corso, S. de Gironcoli, P. Delugas, R.A. DiStasio Jr, A. Ferretti, A. Floris, G. Fratesi, G. Fugallo, R. Gebauer, U. Gerstmann, F. Giustino, T. Gorni, J. Jia, M. Kawamura, H.-Y. Ko, A Kokalj,

E Küçükbenli10, M. Lazzeri, M. Marsili, N. Marzari, F. Mauri, N.L. Nguyen, H.-V. Nguyen, A. Otero-de-la-Roza, L. Paulatto, S. Poncé, D. Rocca, R. Sabatini, B. Santra, M. Schlipf, A.P. Seitsonen, A. Smogunov, I. Timrov, T. Thonhauser, P. Umari, N. Vast, X. Wu, S. Baroni, Advanced capabilities for materials modelling with Quantum ESPRESSO, J. Phys.: Condens. Matter 29 (2017) 465901.

[3] P. Giannozzi, O. Baseggio, P. Bonfá, D. Brunato, R. Car, I. Carnimeo, C. Cavazzoni, S. de Gironcoli, P. Delugas, F. Ferrari Ruffino, A. Ferretti, N. Marzari, I. Timrov, A. Urru, S. Baroni, Quantum ESPRESSO toward the exascale, J. Chem. Phys. 152 (2020) 154105.

[4] R. Car, M. Parrinello, The unified approach to density functional and molecular dynamics in real space, Solid State Commun. 62 (1987) 403.

[5] D. Marx, J. Hutter, Ab Initio Molecular Dynamics: Basic Theory and Advanced Methods, 1st edn., Cambridge University Press, Cambridge, 2009.

[6] M. Born, R. Oppenheimer, Zur quantentheorie der molekeln, Ann. Phys. 389 (1927) 457.

[7] S.J. Chapman, Fortran for Scientists and Engineers, 4th edn., McGraw-Hill Education, 2 Penn Plaza, New York, NY 10121, 2018.

[8] L. Torvalds, The Linux edge, Commun. ACM 42 (1999) 38.

[9] `https://www.gnu.org/software/bash/`.

[10] `https://www.gnu.org/licenses/gpl-3.0.txt`.

[11] A. Shekaari, M. Jafari, Finite temperature properties and phase transition behavior of quasi-planar B_{36} nanocluster from first principles, Mater. Res. Express 6 (2019) 025014.

[12] F.A. Lindemann, The calculation of molecular vibration frequencies, Phys. Z. 11 (1910) 609.

[13] A. Shekaari, M. Jafari, Unveiling the first post-graphene member of silicon nitrides: A novel 2D material, Comput. Mater. Sci. 180 (2020) 109693.