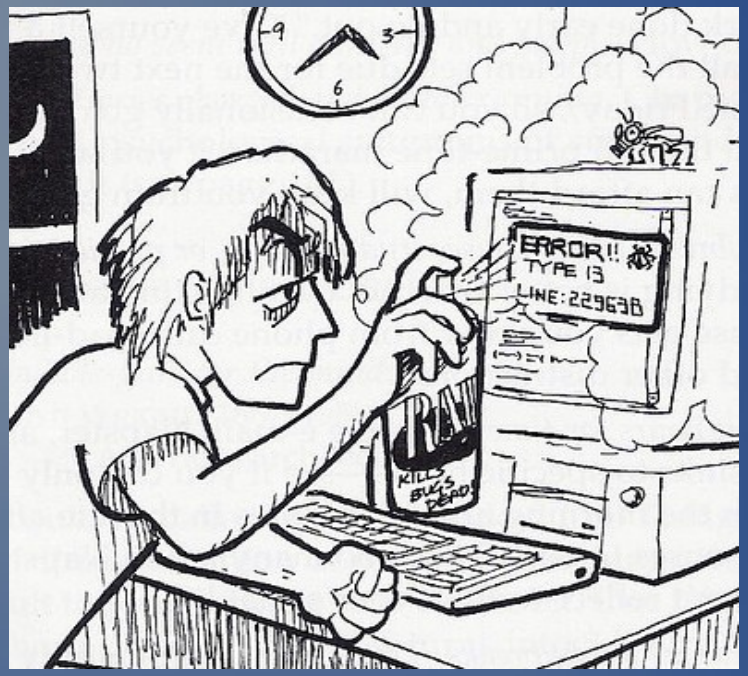
The Survival



Federico Muñoz, Valeria Martin, Maaly Ghariballah, Shaukat Ali and Chris Macdermaid (supervisor).

Workshop on Computer Programming and Advanced Tools for Scientific Research Work & Quantum ESPRESSO Developer Training

Outline

- Project Management & distribution of task
- Refactor the Code and provide python interface
- Project Documentation
- Changes in Simulation Infrastructure

Project Management & distribution of task

- Development (2)
- Changes in Simulation Infrastructure (1)
- Unit testing (1)
- Documentation (1)

Modules of Fortran program & Python Interface

Modules: cell.f90,force.f90,getekin.f90, f90, io.f90, kinds.f90, ljmd.f90,mdsys.f90, physconst.f90,utils.f90, velverlet.f90

Python Interface

```
import ljmd
import numpy as np
pyfname = np.array('argon_108')

ljmd.interface.callopen(pyfname)

#ljmd.interface.set_natoms(108)            # natoms
#ljmd.interface.set_mass(39.948)       # mass in AMU
#ljmd.interface.set_epsilon(0.2379)      # epsilon in kcal/mol
#ljmd.interface.set_sigma(3.405)       # sigma in angstrom
#ljmd.interface.set_rcut(8.5)             # rcut in angstrom
#ljmd.interface.set_box(17.1580)       # box length (in angstrom)
#ljmd.interface.set_nsteps(10000)       # nr MD steps
#ljmd.interface.set_dt(5.0)             # MD time step (in fs)
#ljmd.interface.set_nprint(100)       # output print frequency

ljmd.interface.run()
```

Modules of Fortran program & Python Interface

Fortran interface

```
MODULE interface
  USE ljmd
IMPLICIT NONE
 CONTAINS
  SUBROUTINE run
  IMPLICIT NONE
    CALL lljmd
  END SUBROUTINE

  SUBROUTINE callopen(pyfname)
  IMPLICIT NONE
  CHARACTER*(*) :: pyfname
    CALL openfiles(pyfname)
  END SUBROUTINE

  SUBROUTINE set_natoms(myatoms)
    IMPLICIT NONE
    INTEGER :: myatoms
    natoms = myatoms
  END SUBROUTINE set_natoms
```

Modules of Fortran program & Python Interface

ljmd

```
MODULE ljmd
  USE kinds
  USE io
  USE utils
  USE mdsys
  USE cell
  INTEGER :: nprint
 CONTAINS

SUBROUTINE lljmd
IMPLICIT NONE

  INTEGER :: i, j
  INTEGER, EXTERNAL :: omp_get_num_threads
  nthreads = 1
  !$OMP parallel shared(nthreads)
  !$OMP master
  !$   nthreads = omp_get_num_threads()
  !$   WRITE(stdout,'(A,I2,A)') 'Running OpenMP version
using ',nthreads,' thread(s).'
  !$OMP end master
  !$OMP end parallel

  ! allocate storage for simulation data.
  ALLOCATE(pos(natoms,3),vel(natoms,3),frc(natoms,3,nthreads))

  DO i=1,natoms
      READ(12,*) (pos(i,j),j=1,3)
  END DO
```

```
DO i=1,natoms
      READ(12,*) (vel(i,j),j=1,3)
  END DO
  CLOSE(12)

 ! set up cell list
  CALL mkcell
  CALL updcell

  ! initialize forces and energies
  nfi=0
  frc(:,:,:) = 0.0_dbl
  CALL force
  CALL getekin

  WRITE(stdout, *) 'Starting simulation with ', natoms,
' atoms for', nsteps, ' steps'
  WRITE(stdout, *) '       NFI                TEMP
EKIN                          EPOT&
          &                    ETOT'
  CALL output

! main MD loop
  DO nfi=1, nsteps

      ! write output, if requested
      IF (mod(nfi,nprint) == 0) THEN
          CALL output
      END IF
```

Modules of Fortran program & Python Interface

```
        ! propagate system and recompute energies
        CALL updcell
        CALL velverlet
        CALL getekin
    END DO

! clean up: close files, free memory
    WRITE(stdout,'(A)') 'Simulation Done.'
    CALL rmcell
    CALL ioclose
    DEALLOCATE(pos,vel,frc)
END SUBROUTINE ljmd

SUBROUTINE openfiles(pyfname)
    IMPLICIT NONE
    CHARACTER(LEN=30) :: ifname,restfname, trajfname,
ergfname, logfname,fname
    CHARACTER*(*) :: pyfname
    INTEGER :: iosI,iosR, length!, nprint
        fname = TRIM( pyfname )
       length = LEN_TRIM( fname )
       ifname = TRIM(fname)
       ifname( length+1:length+4 ) = ".inp"
       OPEN(UNIT=11,FILE=ifname,IOSTAT=iosI,STATUS='OLD')
       IF( iosI /= 0 ) THEN
          WRITE(6,*) " "
          WRITE(6,*) "No input file ", ifname, " found."
          WRITE(6,*) " "
       ELSE
            READ(11,*) natoms
            READ(11,*) mass
            READ(11,*) epsilon
            READ(11,*) sigma
READ(11,*) rcut
            READ(11,*) box
            READ(11,*) nsteps
            READ(11,*) dt
            READ(11,*) nprint
            rewind(11)
       END IF
     restfname = TRIM(fname)
       restfname( length+1:length+5 ) = ".rest"
       OPEN( UNIT=12, FILE=restfname, IOSTAT=iosR, STATUS="OLD" )
       IF( iosR /= 0 ) THEN
          WRITE(6,*) " "
          WRITE(6,*) "No restart file ", restfname, " found."
          WRITE(6,*) "Program STOP "
          WRITE(6,*) " "
          STOP
       END IF

       trajfname = TRIM(fname)
       trajfname( length+1:length+4 ) = ".xyz"
       OPEN( UNIT=13, FILE=trajfname, STATUS="UNKNOWN" )

       ergfname = TRIM(fname)
       ergfname( length+1:length+4 ) = ".dat"
       OPEN( UNIT=14, FILE=ergfname, STATUS="UNKNOWN" )

       logfname = TRIM(fname)
       logfname( length+1:length+4 ) = ".log"
       OPEN( UNIT=15, FILE=logfname, STATUS="UNKNOWN" )

  END SUBROUTINE openfiles

  END MODULE ljmd
```

Modules of Fortran program & Python Interface

```
# -*- Makefile -*-
SHELL=/bin/sh
FC=gfortran
FFLAGS=-Wall -g -std=f95 -O3 -ffast-math -fomit-frame-pointer
AR=ar

# list of source files
SRC=kinds.f90 physconst.f90 mdsys.f90 getekin.f90 utils.f90
 io.f90 cell.f90 force.f90 velverlet.f90 ljmd.f90 interface.f90
VPATH=../src
OBJ=$(SRC:%.f90=%.o)

default:   ljmd.so

clean:
	rm -f *.mod *.o *.a *.so *.pyf

# linker rule
../ljmd-serial.x: $(OBJ)
	$(FC) -o $@ $(FFLAGS) $^
```

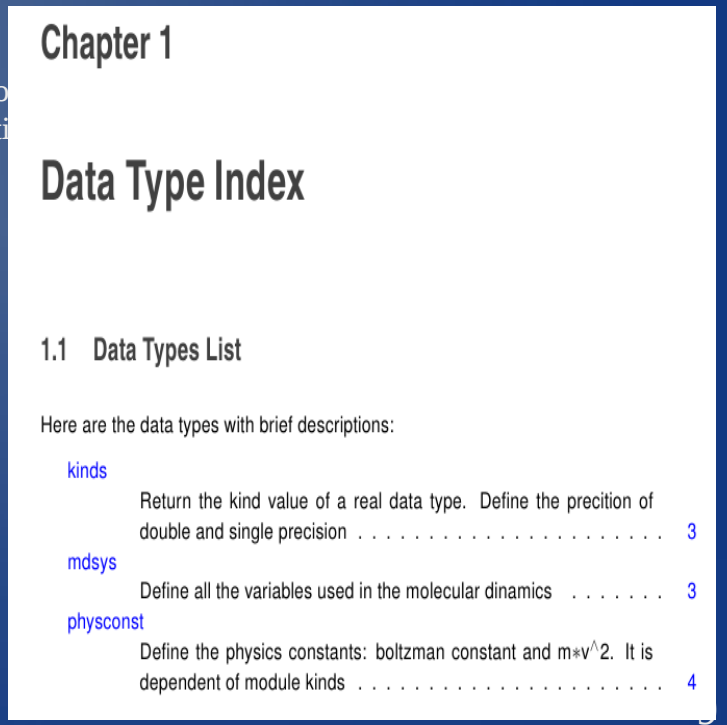
```
# compilation pattern rule for objects
%.o: %.f90
	$(FC) -c -fPIC $(FFLAGS) $<

# dependencies
ljmd.o: ljmd.f90

# signature file creation
ljmd.pyf : ../src/interface.f90
	f2py -m ljmd -h $@ $<

# concatenation of libraries
libljmd.a : $(OBJ)
	ar crvs $@ $^

# compilation of share library
ljmd.so : ljmd.pyf libljmd.a
	f2py -m ljmd --fcompiler=gfortran -c $^
```

Documentation

- 2 versions of documentation was created. Doxygen was used to extract structure from the code
- 1st version is based on original code
- 2nd version is based on our modified code

Kinds.f90

```
!> Return the kind value of a real data type. Define the precition of double and single precision.
MODULE kinds
  IMPLICIT NONE
  INTEGER, PARAMETER :: dbl = selected_real_kind(14,200)  ! double precision flo
  INTEGER, PARAMETER :: sgl = selected_real_kind(6,30)    ! single precision floati
  INTEGER, PARAMETER :: sln = 200                         ! length of I/O input line
  PRIVATE
  PUBLIC :: sgl, dbl, sln
END MODULE kinds
```



Chapter 1

Data Type Index

1.1 Data Types List

Here are the data types with brief descriptions:

Doc: Unit Testing

Test-case ID:

Test-case Statement:

Purpose (Prerequisite):

Test (Input) Data:

Steps to be executed:

Expected Result:

Actual Result:

Pass/Fail:

Comments:

Changes in Simulation Infrastructure

Extended the simulation infrastructure so it can handle N type of atoms instead of one atom
Tried to do it with minimum change to original code
And it work!

Demo

Link: https://github.com/chrismacdermaid/ljmd-f


Thank You!


Grazie!!!