|epcc|

**SS-98-06**

# Parallel Short-Ranged Molecular Dynamics Simulation with High Performance Fortran

**Giampietro Lea**

Department of Civil Engineering
University of Padua
Venetia

**Abstract**

We first take a quick look at the Molecular Dynamics method, by briefly introducing the mathematical and physical theory behind the motion of a particle and its interaction with the other particles present in the simulation. We also discuss briefly the assumptions we generally make in order to simplify the problem. We look at some practical fields where the Molecular Dynamics method can be used, both at a molecular/atomic and at a macroscopic level. We then discuss in some depth the general structure of a Molecular Dynamics code and the steps that need to be taken in order to parallelize it. Finally, I explain the direction of my project and how it differed from the original plan, and I shall also talk about the current status of my program and my future plans for it.

# Contents

# 1 Introduction

A sample of any solid or liquid material considered at atomic level, is an assembly of so many particles that it may be inconceivable to simulate it exactly, even though the properties of an isolated molecule and the laws which govern molecular interaction are well established.

However we can develop a mathematical model of the physical system, with a relatively low number of artificially simple molecules interacting in an artificially simple way, using the fact that in an assembly of particles the macroscopic or bulk properties of the whole system may be rather simple and dependent on the average properties of the particles.

## 1.1 Quick overview of the method

The Molecular Dynamics method, as formulated by Alder and Wainwright in 1956, is now a standard technique for simulating solids, liquids and gases at the atomic or molecular level; however, it can also be used in some cases at macroscopic level. It treats real-time motion of the particles and can be used to study both equilibrium and time dependent properties.

The MD method proceeds from the solution of the Newtonian equations of motion of the particles in a many-particles system, starting from given initial positions and velocities. In other words, it simply evaluates the force on each particle at any instant of time, and then computes and traces the trajectories of the particles.

The inter-particle forces we consider might be

- long range, eg. gravitational as in the case of planets, or electrostatic as in the case of charged particles.
- short range, eg. contact forces as in the case of billiard balls or wheat grains, or inter-molecular as in the case of a crystal structure.

We shall see that the bulk of the time is spent in computing the pair-wise forces between the $N$ interacting particles, rather than in updating their position and velocity. In fact, if no special devices or simplifying assumptions are introduced, we shall have to compute $1/2(N(N-1))$ terms at each time step.

## 1.2 Brief introduction to the mathematical theory behind the motion of particles

Two methods are basically employed when studying the motion of multi-particle systems.

The first method, due to Euler, is applied to studying systems such as the motion of fluids. Rather than concentrating its attention on the single particle, it takes into consideration the change in velocity and density of a certain volume of fluid over a certain time interval, as it flows past a pre-set area in space. In addition, instead of tracking the motions of individual particles we simplify the problem by studying averaged quantities such as pressure, density etc.

The second one, due to Lagrange, is used when studying the motion of a single particle or body. In this method we focus our attention on the particle and trace its trajectory by solving for its velocity and its position at each time step. Until the advent of computers this second method was never used to study the motion of multi-particle systems, because it is computationally very expensive; in fact it forces us to solve the equations of motion for each single particle of the system.

Now thanks to the computational power given by computers we can successfully apply Lagrange method to studying a multi-particle system. We prefer in fact to use this method because it allows us to develop a simple model of the physical system based on the fundamental properties of each particle.

### 1.2.1 Single particle

As we know, the position of a particle is given by its position vector, with either 2 or 3 components depending on the dimensionality of the problem. Newton's first law states that a particle will remain at rest, or continue to move with uniform velocity if it is not acted upon by a force. — we must remember that velocity is a vector and therefore uniform velocity means constant value and direction.

Now, if we assume the particle to be initially at rest, we shall have a change both in the position vector and in the velocity vector as soon as a force acts upon the particle. Knowing the force acting on the particle, and given some initial conditions (such as the initial position and velocity) we can work out by successive integration the velocity and position at any successive instant of time t.

### 1.2.2 $N$ particles

In an $N$-particle system, the force on each particle arises from its interaction with all the other particles. Therefore we shall first have to work out the resultant force acting on each particle, and once we know this resultant we can then, just as in the case of a single particle, work out its trajectory. Therefore for particle 1 we shall have to calculate $(N-1)$ forces; for particle 2 there will be $(N-2)$ forces, and so on. The total number of forces to calculate will be $1/2(N(N-1))$.

We can see that if number of particles is of order$(N)$ then the number of forces calculations is of order $(N^2)$.

To actually solve for the motion we first we divide our time interval into smaller sub-intervals of length $dt$. Then at each time step we compute the resultant of the forces acting on each particle and finally we update the velocity and the position.

For example, in a one-dimensional system, if the known force is $F$ then the velocity at a time $t + dt$ is given by:

$$v(t + dt) = v(t) + \frac{F(t)}{m}dt$$

and the position at the same instant is given by:

$$x(t + dt) = x(t) + v(t)dt$$

This is called the Euler integration scheme.

## 1.3 Assumptions used in this method in order to simplify the problem

As we have seen, the bulk of the time will be spent in computing the $1/2(N(N-1))$ force terms at each time step. We must therefore introduce some special device or simplifying assumption in order to reduce this computing time if we are to simulate any more than a few thousand particles.

Also, in some cases, we can also reduce the number of forces to be calculated by assuming the interaction potential between particles to become negligible beyond a certain cut-off distance.

Another simplifying assumption we usually make is to treat the particles as single points: their position can therefore be given by the coordinates of their center of gravity and we can disregard completely any rotational movement they may have.

Different algorithms differ therefore in the number and type of simplifications they make. In other words, more complicated and more accurate algorithms exist for solving the motion of the particles, but their better accuracy may involve an increase in computational time.

If we assume we are working in a conservative force field where $F = -\nabla V$ then we have energy conservation, i.e.

$$E_{tot} = E_{kin} + E_{pot} = \frac{1}{2}mv^2 + V = constant$$

At each time step we can calculate $E_{tot}$ in order to check the force calculations and integration step. The better the algorithm the more constant will be the value of $E_{tot}$.

# 2 Practical uses of the Molecular Dynamics method

## 2.1 At molecular/atomic level

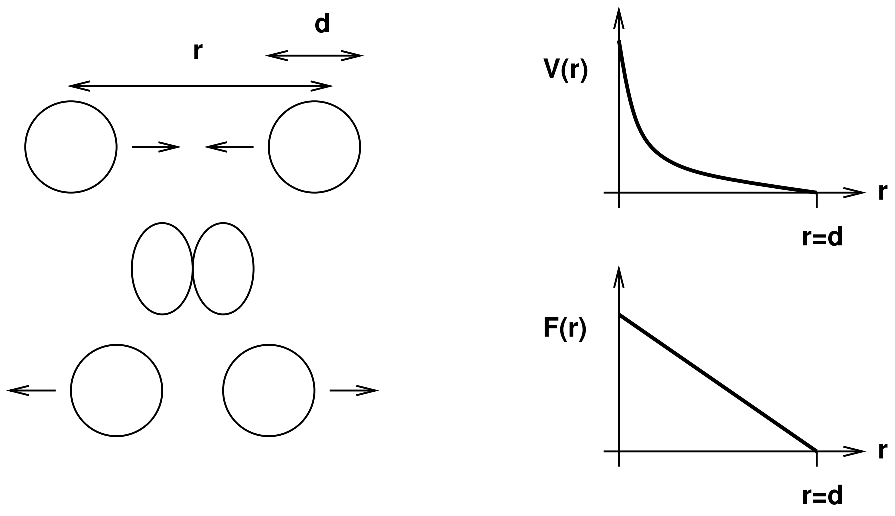
The Molecular Dynamics method can be used for studying fracture and crack propagation, at molecular level, of materials under load. In this case, since we are dealing with short-range forces, we could use a typical short range potential, such as the Lennard-Jones potential, modified in such a way that it goes to zero at a certain distance $r_{max}$. This will result in a considerable reduction of the force calculation, since the value of the interacting forces will become zero for particles placed at a relative distance greater than this cut-off radius $r_{max}$.

At molecular level the number of particles we deal with is quite huge, eg. a cube of $10^9$ atoms of material is only 0.35 micrometers on a side.

## 2.2 At macroscopic level

The Molecular Dynamics method can also be used when studying the motion of a multi-particle macroscopic system. All that differs from the microscopic case is the form of the potential used, i.e. the force law.

For example, the Molecular Dynamics method can be used to study a silo full of grain, emptying. As a simplification we can assume the particles as deformable spheres of diameter $d$, which interact only with those neighbouring particles which actually touch them. This means we have a cut-off distance $r_{max} = d$. Therefore, instead of $(N-1)$ forces to calculate for each particle, we shall have perhaps only about 10 contact forces in a three-dimensional system. Assuming elastic behaviour, we can also use a simple force law where the potential reaches a maximum at zero distance, and will go to zero at distance $d$.



# 3 Structure of a Molecular Dynamics code

## 3.1 Interaction calculation

To keep things simple we look at the structure of a Molecular Dynamics code for a two-dimensional system.

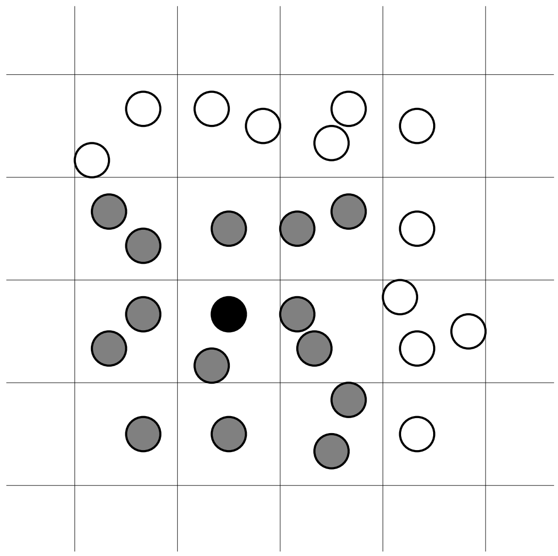
We can start by considering the working environment as a rectangular region. This region will typically be significantly larger than the interaction cut-off distance $r_{max}$; this means that a large number of particles will not interact, and it is obviously not necessary to explicitly calculate the force between them.

In order ensure that we do not perform unnecessary calculations, we simply divide the region into a large collection of small cells of size larger than $\mathrm{r}_{max}$. Each particle is assigned geometrically to the proper cell according to its coordinates.

Now with our cell structure in place we proceed to calculate the forces for all the particles in a cell. This will be a two-step process, which will be carried out for each single particle in the chosen cell:

- first, calculate all the forces between particles in the same cell,
- second, calculate all forces due to the interaction with particles in the neighbouring 8 cells (in 2D) or 26 cells (in 3D).

Because the cells are larger than the cutoff distance, we can guarantee that this procedure includes all possible interactions.



In the diagram above we illustrate how, by assuming a certain cut-off radius and defining a cell structure, we can minimize the number of force calculations. Only the interactions between the black particle and the other (shaded) particles in the same cell and in its eight neighbouring cells need to be calculated. The white particles in the more distant cells are all beyond the cutoff distance.

### 3.2 Redistribution of particles

After each time step, the particles in the system will be moved to their new positions. We call a redistribution procedure which checks the new coordinates of each particle, and if the particle is found to be in the wrong cell it will be assigned to the proper one.

### 3.3 Parallelisation

To parallelise the algorithm, we give an equal-sized sub-regions of the simulation region to each available processor. There are only a few changes needed to the procedure.

- For cells along the edge of each processor, i.e. the boundary cells, we must calculate the interactions with cells placed on different processors. In order to do so, each processor will send and receive the particle data for each boundary cell to and from the appropriate neighbouring processors.
- In the redistribution phase, particles in the boundary cells may move to sub-regions owned by a different processor. Again, we must add a communication step to allow transfer of particles between processors.

Due to the regular communications patterns involved, the above parallelisation steps can easily be achieved using High Performance Fortran (HPF).

# 4 Work done

## 4.1 Original Plan

I must say that fortunately my supervisor allowed me a complete free hand in deciding the direction along which to develop my project; this allowed me to move along a different direction from the one set out originally.

Initially I was given the choice to write my code in FORTRAN77 since the HPF compiler could parallelize a code written with this earlier version of Fortran with th inclusion of a few directives. Actually I had been taught to program using FORTRAN77 and it would have been much easier for me to write my code with it, but I decided that these two months at EPCC were the right time for moving on to Fortran90. Now, at the end of the two months, I must say I took the right decision because I have realized that much of FORTRAN77 has either been made obsolete or changed in some way by the new Fortran90 Standards. Some of the new features allowed by Fortran90, such as Dynamic Memory Allocation, have been applied to my code. In addition, many Fortran90 constructs can be automatically parallelised by an HPF compiler without the need to include extra directives.

Another decision I took was to spend the initial weeks on gaining some background knowledge on computer languages, machine architecture and massively parallel computers. All this knowledge I lacked because of my background as a Civil Engineer rather than a Computer Scientist. I also spent some time in gaining some information on Molecular Dynamics and Chemical Kinetics in general.

The last decision I made was to write a kind of “Template code” usable for any Molecular Dynamics simulation, instead of a code which could be applied only to the particular problem at hand. This, being pedantic, forced me to write a program which would, at all times, make continuous double-checks both on the data introduced by the user and on the specific operations carried out by the machine.

## 4.2 Structure of my program

My program is intended to simulate both the movement of particles subjected to molecular-type interactions or of particles subjected to macroscopic level forces, such as wheat grains or pool balls.

A choice is given to the user for a 2-D or 3-D system whose actual dimensions he can then set himself. He can also choose both the number and the size of particles; and in case he decides for different size particles he can set their dimension individually.

Macroscopic deformable particles are taken into consideration and simulated by using a 3-D ensemble of ’linear elastic’ springs, where the coefficient of elasticity can be varied by the user.

At all times, I have continuous checks on the data introduced by the user, for mistakes done both involuntarily and on purpose. The user is given two chances to rectify these mistakes; if he doesn’t comply the program is terminated.

In case of clashes between, for example, the size of the environment and the number of particles chosen, the user is allowed to go back and re-type the input of any particular data he wants. This can be seen in the following piece of code, which originally was inserted within a DO WHILE loop in order to allow the user to return onto his decisions.

In the example below, we can also see being used the new CYCLE statement allowed by Fortran90.

```
! ### STEP 6/c/1 ### : check the fitting in the case of a common
! radius, whose value we asked for in set_radius.

  IF (answer_for_radius == 1) THEN

    CALL check_diameter_case_1 (radius, ndim, length, clash_&
                               &with_environment, repeat_&
                               &first_question, resetting&
                               &_dimensions, resetting_nop&
                               &_and_size)

    IF (clash_with_environment .EQV. .TRUE.) CYCLE outer_1

  ELSE

! ### STEP 6/c/2 ### :  check the fitting in the case of
! different radii, whose value we asked for in set_radii.

    CALL check_diameters_case_2 (radii, ndim, length, clash_&
                                &with_environment, repeat_first&
                                &_question, resetting_&
                                &dimensions, resetting_nop_and_&
                                &size)

    IF (clash_with_environment .EQV. .TRUE.) CYCLE outer_1

  END IF
```

Throughout the program I make continuous use of the new Dynamic Memory Allocation features allowed by Fortran90, in order to minimize the use of memory. A double-check is also carried on the actual correct allocation of memory. This can be seen in the following piece of code, which originally was inserted within another DO WHILE loop in order to allow the user to return on his decisions. But being inside a DO loop I must first check if the array length has already been allocated and then, in case it hasn't, I proceed to physically allocate it.

```
  IF (.NOT. ALLOCATED (length)) THEN

    DO i = 1,3

      alloc_error = 0
      ALLOCATE (length(ndim), STAT = alloc_error)

      IF (alloc_error == 0) EXIT

! If error during allocation, i.e. alloc_error /= 0 then ....
```

```
      PRINT *," "
      PRINT *,"Program could not allocate space for array &
              &length,"
      PRINT *," "
      PRINT *,"but we are going to try again !"
      PRINT *," "

    END DO

! If after three tries we still are not successful we terminate
! the program.

    IF (.NOT. ALLOCATED (length)) THEN

      PRINT *," "
      PRINT *,"Three attempts to allocate with no success,"
      PRINT *," "
      PRINT *,"therefore we are going to terminate the &
              &program !"
      PRINT *," "
      STOP

    END IF

  END IF

  length(:) = 0.0
```

Another example of my double-checking can be seen below, where I make sure that when generating the initial coordinates, we have a continuous check on eventual clashes and penetrations between the particles already “safely” in place and the ones placed successively.

```
! Now I am going to work on the other particles.

  clash_with_origin_others = .TRUE.
  outside_working_space_others = .TRUE.
  clash_with_friend = .TRUE.

  DO WHILE ((clash_with_origin_others .EQV. .TRUE.) .OR. &
           &(outside_working_space_others .EQV. .TRUE.) .OR. &
           &(clash_with_friend .EQV. .TRUE.))

    clash_with_origin_others = .FALSE.
    outside_working_space_others = .FALSE.
    clash_with_friend = .FALSE.

   .............................................................
   .............................................................
   .............................................................

! Now I must check if I have a clash with the particles already
```

```
! present.

    DO j = 1,i-1

      separation_with_friend (j)= 0.0

      DO idim = 1, ndim

        separation_with_friend (j) = separation_with_friend &
       &(j) + ((coord (i,idim) - coord (j,idim)) * (coord &
       &(i,idim) - coord (j,idim)))

      END DO

      separation_with_friend (j) = SQRT (separation_with_&
                                  &friend (j))

      PRINT *," "
      PRINT *,"The distance between particle",i," ,i.e. the &
              &one we are placing,"
      PRINT *," "
      PRINT *,"and particle",j," is:",separation_with_friend (j)
      PRINT *," "

      IF (separation_with_friend (j) < (radii(i) + radii(j))) &
     &THEN

        clash_with_friend = .TRUE.
        PRINT *," "
        PRINT *,"I have a clash between particle",i, &
               &" and particle",j
        PRINT *," "
        EXIT

      END IF

    END DO

    IF (clash_with_friend .EQV. .TRUE.) EXIT

  END DO
```

# 5 Conclusions

At the moment my program has almost reached 4000 lines of code, but unfortunately is not yet complete. I do not expect the final code to be huge, but I predict that once finished the program will be about 20000 lines long. My plans are to keep on coding it in my spare time and then, once finished, run it on parallel machines that I will have access to at my own University.

# 6 Acknowledgements

I would like to thank my supervisor, Dr. David Henty, for the excellent support and freedom received during my work on the project. I would also like to thanks all the EPCC staff, and especially Catherine and Tracy, for all the help they gave us even on the most insignificant (but nonetheless troublesome) matters during our stay in Edinburgh.

I must say this experience has led me to actually like sitting in front of a screen typing away on the keyboard and wrecking my brain, till the early hours of the morning, in order to get a program to work. Before now I only thought of computers as machines good only for running Microsoft Office, AutocadLT and various computer games.

# 7 Bibliography


- Fortran90 Programming ; T.M.R.Ellis - Addison Wesley, 1997 Edition
- Optimizing Fortran90 - Reference, Version 02-00 for HI-UX/MPP; Hitachi, 1996 Edition
- EPCC Interactive Courseware (EPIC) on High Performance Fortran
- Elementi di Calcolo Numerico ; G.Gambolati - Libreria Cortina, Padova, 1991 Edition
- Past, Present, Parallel - A Survey of Available Parallel Computing Systems ; A.Trew, G.Wilson - Springer, 1991 Edition
- Dynamic aspects of molecular energy states ; T.L.Cottrell, 1965 Edition
- Introduction to molecular dynamics and chemical kinetics ; G.D. Billing, K.V.Mikkelsen, 1996 Edition
- Message-Passing Multi-Cell Molecular Dynamics on the Connection Machine 5 ; D.M.Beazley, P.S.Lomdahl - Los Alamos National Laboratory - USA, 1993 Edition
- Multi-Million Particle Molecular Dynamics on MPPs ; D.M.Beazley, P.S.Lomdahl - Los Alamos National Laboratory - USA, 1997 Edition
- A computer simulation in Newtonian and non-Newtonian fluid dynamics using an extension of a model due to Greenspan ; C.K.Lee - M.Sc. Thesis, University of Wales - Swansea, 1981
- Numerical study of three-dimensional flow using fast parallel particle algorithms ; G.J.Pringle - Ph.D. Thesis, Napier University, Edinburgh, 1994
- Parallel Computational Fluid Dynamics - Implementation and Results ; Horst.D.Simon - MIT Press, 1992 Edition

# 8 whoami

I am currently a student of Civil Engineering, specializing in Structures, at the University of Padua, in Venetia. At the moment, however, I am studying at the Civil Engineering Department of the University of Wales - Swansea, thanks to the Socrates exchange program set up between the two Universities. My goal is then to go on to take a Ph.D. in Civil Engineering, hopefully in the field of off-shore structures.

My hobbies include parachuting, mountaineering, skiing and shooting; sports which I picked up during the years spent as an Officer in the Italian Military Police Force - the Carabinieri.

# 9 Final note by the Supervisor