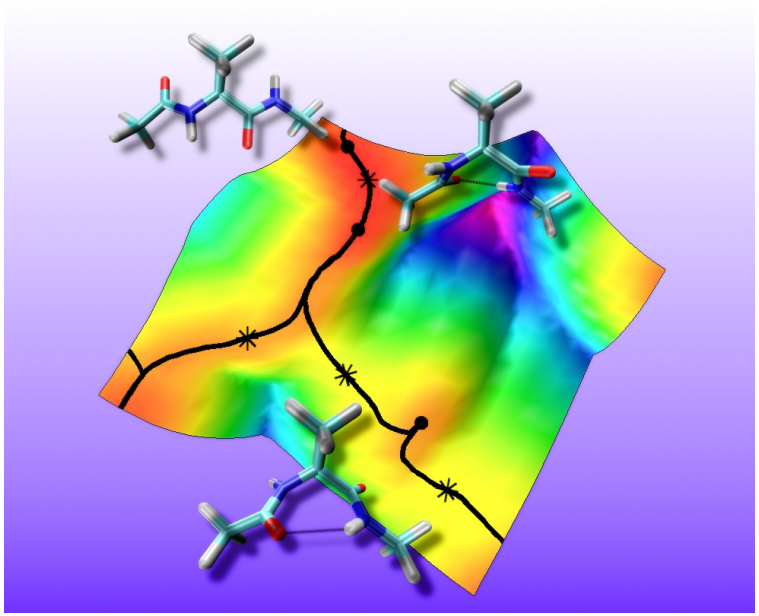
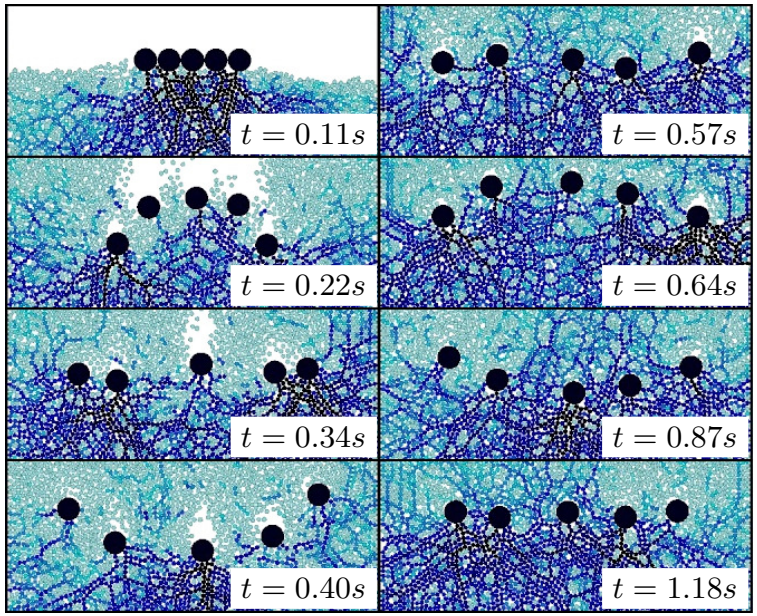
Simulaciones de Dinámica Molecular
Detalles de implementación

J. Manuel Solano Altamirano
Facultad de Ciencias Químicas
Benemérita Universidad Autónoma de Puebla

Ideas generales

- Imitar lo que las partículas hacen en la realidad, suponiendo que su dinámica está governada por alguna función potencial.
- Usamos las leyes de Newton para calcular las fuerzas que actúan sobre cada partícula del sistema.





Ideas generales

test

Ecuaciones de movimiento

$$\vec{F}_i(t_n) = m_i \ddot{\vec{x}}_i(t_n), \quad i = 1, 2 \ldots, N \tag{1}$$

$$\vec{F} = -\nabla U(\vec{x}) \tag{2}$$

$$\vec{v}_i = \frac{d\vec{x}_i}{dt} \tag{3}$$

$$\frac{d\vec{v}_i}{dt} = \frac{\vec{F}_i}{m} \tag{4}$$

Ecuaciones de movimiento

$$\vec{v}_i = \frac{d\vec{x}_i}{dt}; \qquad \frac{d\vec{v}_i}{dt} = \frac{\vec{F}_i}{m} \tag{5}$$

- Sistema de $6N$ ecuaciones diferenciales ordinarias, generalmente acopladas ($U(\{\vec{x}_i\})$).
- Solución analítica imposible, pero solución numérica directa*:

$$\vec{x}_{i,n+1} = \vec{x}_{i,n} + \delta t \cdot \vec{v}_{i,n} \tag{6}$$

$$\vec{v}_{i,n+1} = \vec{v}_{i,n} + \delta t \cdot \vec{F}_{i,n}(\vec{x}_{i,n})/m. \tag{7}$$

δt se conoce como "paso de tiempo".

Campos externos

- Fuerzas originadas por algún campo externo:

$$\vec{F}_i = -\nabla U_e(\vec{x}_i) \tag{8}$$

Generalmente $t_{cpu} \propto N$, donde N es el número de partículas del sistema.

Interacciones partícula-partícula

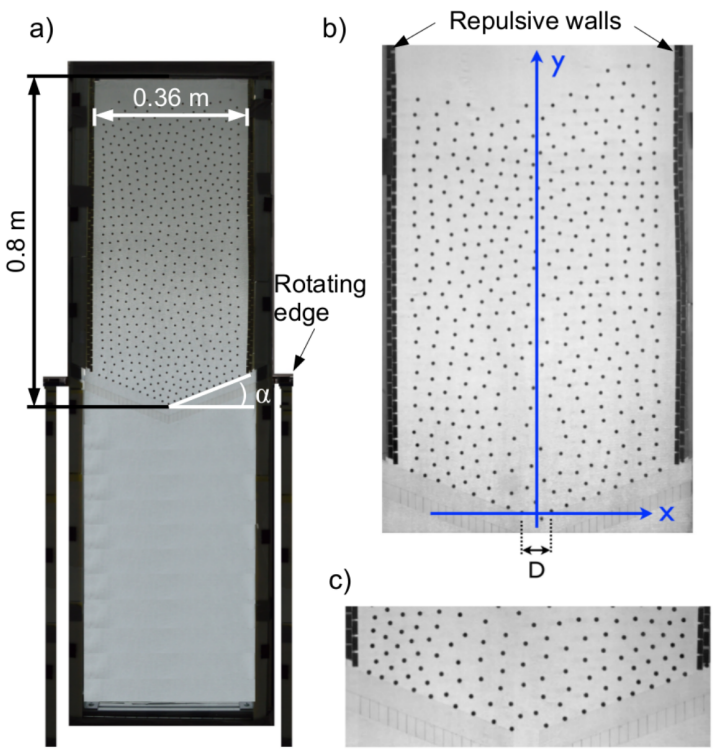
- Fuerzas originadas por alguna interacción entre partículas. A primera aproximación, estas interacciones son llamadas *por pares*.

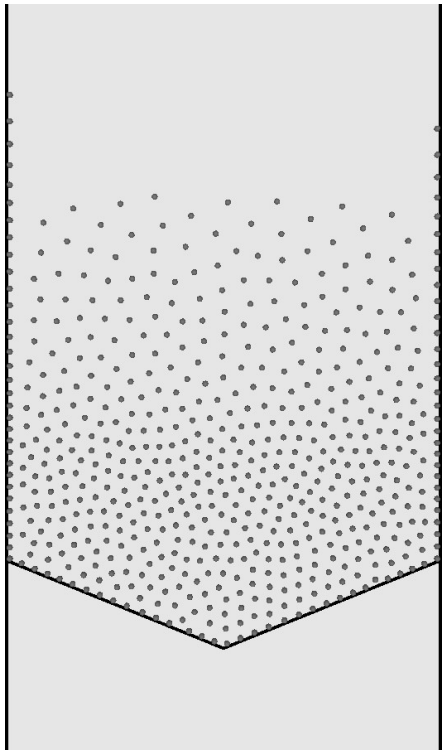
$$\vec{F}_i = \sum_j \vec{F}_{j \to i}. \tag{9}$$

  Generalmente $t_{cpu} \propto N^2$.
- Ejemplos: Fuerzas de Van der Waals (y en general enlaces químicos), colisiones, interacción dipolo-dipolo, etc.
- Algunos esquemas reducen el tiempo de cómputo.

Condiciones de frontera

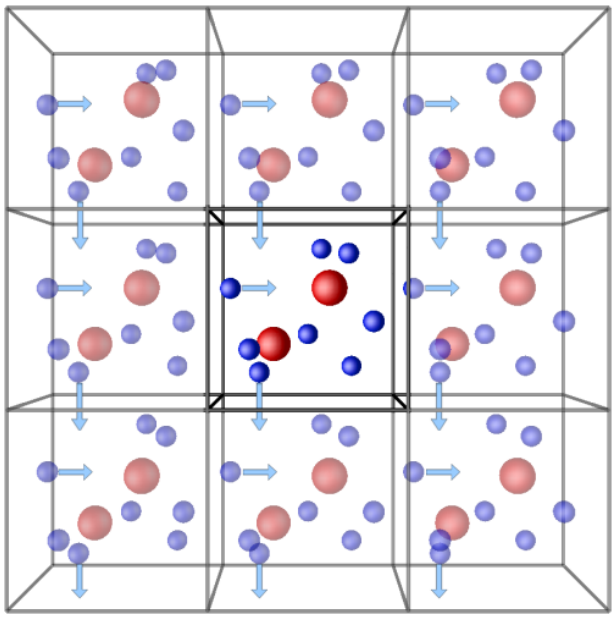
- Sistemas confinados: fronteras físicas.

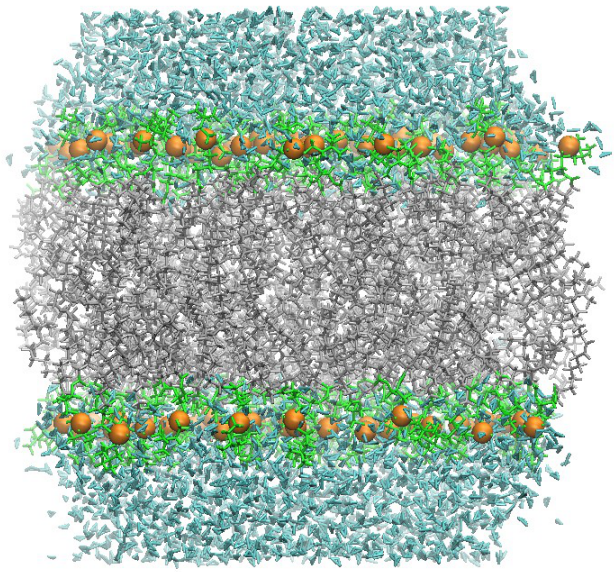




Condiciones de frontera

- Sistemas infinitos: Condiciones de frontera periódicas.





Open source

- LAMMPS (C++)
- GROMACS (C/C++)
- NAMD (C++)
- OpenMD (C++)

¿GPUs?

Open source

¿Cómo pasamos de las ecuaciones a un programa?

¿Dónde obtener el código?

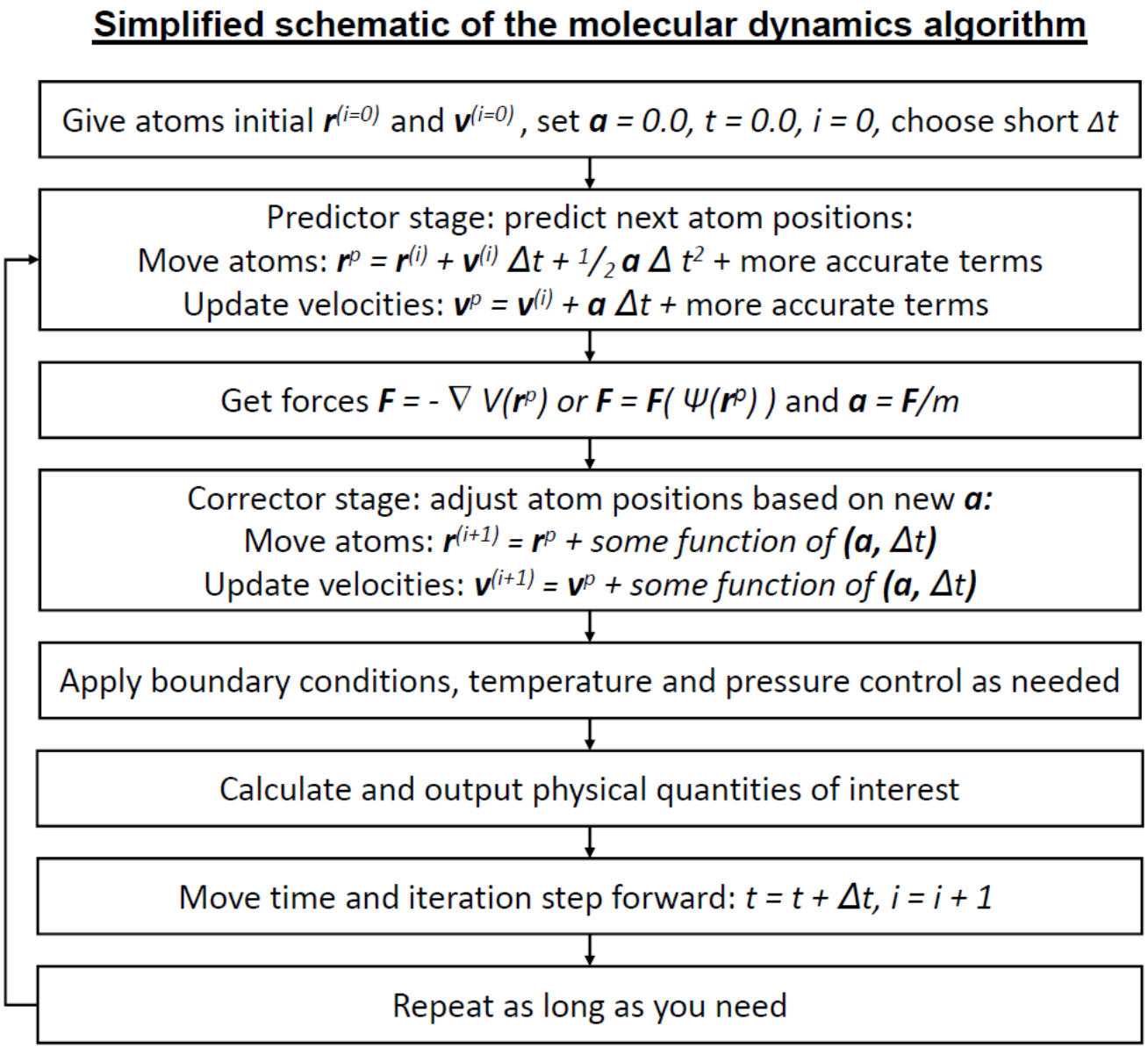
- Versión paralelizada (OpenMP, Cell-lists). Cortesía de Axel Kohlmeyer.

```
1 $ git clone git://github.com/akohlmey/ljmd-f.git
```

- Versión simple (basada en la versión de Axel).

```
1 $ git clone git://github.com/jmsolano/ljmd-teach.git
```

Diagrama básico de Dinámica Molecular



Módulos

- Lector de configuración
- Lector de posiciones iniciales (generador)
- Integrador
- Cálculo de fuerzas
- Módulo de guardado de información
- Módulo maestro o principal

Implementación de un programa simple de MD

```
1  PROGRAM LJMD
2      use MDSystem
3          ...
4      IMPLICIT NONE
5      CHARACTER(len=sln) :: inputFileName
6      INTEGER            :: i
7      CALL get_command_argument(1,inputFileName)
8      CALL LoadMDSysFromFile(inputFileName) !part of MDSystem
9      CALL LoadParticleDataFromFile(rstfile) !part of MDSystem
10     CALL InitIntegratorVerlet !part of IntegratorVerlet
11     CALL InitVanderWaalsForce !part o VanderWaalsForce
12     DO nfi=1,nsteps
13         IF (MOD(nfi,nprint) == 0) THEN
14             CALL AppendXYZData(trjfile)
15             CALL AppendProperties(ergfile)
16         END IF
17         CALL AddVanderWaalsForces
18         CALL AdvanceStepVerlet
19         CALL ApplyBoundaryConditions
20         CALL ApplyThermodynamicConditions
21     DONE
22     CALL SaveToRestFile(rstfile)
23     CALL CleanMDSys !part of MDSystem
24 END PROGRAM LJMD
```

Ejemplo de archivo de configuración

```
108                 # natoms
39.948              # mass in AMU
0.2379              # epsilon in kcal/mol
3.405               # sigma in angstrom
8.5                 # rcut in angstrom
17.1580             # box length (in angstrom)
argon_108.rest      # restart
argon_108.xyz       # trajectory
argon_108.dat       # energies
10000               # nr MD steps
5.0                 # MD time step (in fs)
100                 # output print frequency
```

Definiciones básicas

```
1  #ifndef _KINDS_FNF_
2  #define _KINDS_FNF_
3  MODULE kinds
4    IMPLICIT NONE
5    INTEGER, PARAMETER :: dbl = selected_real_kind(14,200)   ! double precision
         floating point
6    INTEGER, PARAMETER :: sgl = selected_real_kind(6,30)     ! single precision
         floating point
7    INTEGER, PARAMETER :: sln = 200                          ! length of I/O input
         line
8    INTEGER, PARAMETER :: stdin=5
9    INTEGER, PARAMETER :: stdout=6
10   INTEGER, PARAMETER :: inunit = 30                        ! input(mdsys) unit
         mask
11   INTEGER, PARAMETER :: rstunit = 31                       ! restitution unit
         mask
12   INTEGER, PARAMETER :: trjunit = 32                       ! trajectory unit mask
13   PRIVATE
14   PUBLIC :: sgl, dbl, sln, stdin, stdout, inunit,rstunit, trjunit
15 END MODULE kinds
16 #endif
17 !#endif _KINDS_FNF_
```

Variables y parámetros globales

```
1  #include "kinds.F95"
2  #include "utils.F95"
3
4  #ifndef _MDSYSTEM_FNF_
5  #define _MDSYSTEM_FNF_
6  MODULE MDSystem
7      use kinds
8      IMPLICIT NONE
9      INTEGER :: natoms,nfi,nsteps,nprint
10     REAL(kind=dbl) :: dt, mass, epsilon, sigma
11     REAL(kind=dbl) :: rcut, boxlength
12     REAL(kind=dbl) :: ekin, epot, temp
13     REAL(kind=dbl), POINTER, DIMENSION (:,:) :: x, v, f
14     REAL(kind=dbl), POINTER, DIMENSION (:,:) :: xp  !needed for Verlet
           Integrator!
15     CHARACTER(len=sln) :: rstfile, trjfile, ergfile
16 CONTAINS
```

Variables y parámetros globales

```
1   SUBROUTINE LoadMDSysFromFile(inName)
2       use IOUtils
3       IMPLICIT NONE
4       CHARACTER(LEN=sln), intent(in)          ::inName
5       INTEGER kk
6       OPEN(UNIT=inunit, FILE=TRIM(inName), STATUS='UNKNOWN', FORM='FORMATTED')
7       READ(inunit,*) natoms
8       READ(inunit,*) mass
9       READ(inunit,*) epsilon
10      READ(inunit,*) sigma
11      READ(inunit,*) rcut
12      READ(inunit,*) boxlength
13      CALL getline(inunit,rstfile)
14      CALL getline(inunit,trjfile)
15      CALL getline(inunit,ergfile)
16      READ(inunit,*) nsteps
17      READ(inunit,*) dt
18      READ(inunit,*) nprint
19      CLOSE(UNIT=inunit)
20      !Allocate memory...
21      ALLOCATE(x(natoms,3),xp(natoms,3),v(natoms,3),f(natoms,3))
22      x=0.0_dbl
23      xp=0.0_dbl
24      v=0.0_dbl
25      f=0.0_dbl
26  END SUBROUTINE LoadMDSysFromFile
```

Variables y parámetros globales

```
1     SUBROUTINE LoadParticleDataFromFile(inName)
2         CHARACTER(LEN=sln), INTENT(in) :: inName
3         INTEGER  kk
4         OPEN(UNIT=rstunit,FILE=TRIM(inName),STATUS='UNKNOWN',FORM='FORMATTED')
5         IF(natoms==0) RETURN
6         DO kk=1,natoms
7             read(rstunit,*) x(kk,1),x(kk,2),x(kk,3)
8         END DO
9         CLOSE(rstunit)
10    END SUBROUTINE
```

Variables y parámetros globales

```
1       SUBROUTINE CleanMDSys
2           IMPLICIT NONE
3           DEALLOCATE(x,v,f)
4           DEALLOCATE(xp)
5       END SUBROUTINE CleanMDSys
6   END MODULE MDSystem
7   #endif
```

Integrador

- Existen varios esquemas. Algunos no son simplécticos, es decir no conservan el volumen del espacio fase.
- Los más populares son Störmer-Verlet, Velocity-Verlet y Leapfrog.
- En el código usaremos Störmer-Verlet:

$$\vec{x}_{n+1} = 2\vec{x}_n - \vec{x}_{n-1} + \delta t^2 \vec{F}_n/m \quad (10)$$

$$\vec{v}_{n+1} = \frac{1}{2\delta t}\left(\vec{x}_{n+1} - \vec{x}_{n-1}\right) \quad (11)$$

Integrador

```
1  #ifndef _INTEGRATOR_FNF_
2  #define _INTEGRATOR_FNF_
3  #include "kinds.F95"
4  #include "mdsys.F95"
5  #include "physconst.F95"
6  MODULE IntegratorVerlet
7      use kinds
8      use MDSystem
9      IMPLICIT NONE
10     REAL(kind=dbl)        :: h,h2,oneOver2h
11     LOGICAL               :: isFirstIntegration
12     PRIVATE :: h,h2,oneOver2h,isFirstIntegration
13 CONTAINS
14     SUBROUTINE InitIntegratorVerlet
15         IMPLICIT NONE
16         isFirstIntegration=.TRUE.
17         h=dt
18         h2=h*h
19         oneOver2h=0.5_dbl/h
20     END SUBROUTINE InitIntegratorVerlet
21
22     SUBROUTINE IntegrateFirstVerlet
23         IMPLICIT NONE
24         INTEGER    :: i,j
25         DO i=1,natoms
26             DO j=1,3
27                 xp(i,j)=x(i,j)-v(i,j)*h
28             END DO
29         END DO
30     END SUBROUTINE IntegrateFirstVerlet
```

Integrador

```
1
2       SUBROUTINE AdvanceStepVerlet
3           use PhysicalConstants
4           IMPLICIT NONE
5           INTEGER             :: i
6           REAL(kind=dbl)      :: xtmp(3), vfac
7           vfac = 1.0_dbl / mvsq2e / mass
8           IF(isFirstIntegration.EQV..TRUE.) THEN
9               CALL IntegrateFirstVerlet
10              isFirstIntegration=.FALSE.
11          END IF
12          DO i=1,natoms
13                  xtmp=x(i,:)
14                  x(i,:)=(2.0_dbl*x(i,:))-xp(i,:)+(h2*vfac*f(i,:))
15                  v(i,:)=oneOver2h*(x(i,:)-xp(i,:))
16                  xp(i,:)=xtmp
17          END DO
18      END SUBROUTINE AdvanceStepVerlet
19  END MODULE IntegratorVerlet
```

Fuerzas

- En general, las interacciones por pares requieren $t_{cpu} \propto N^2$.
- Tercera ley de Newton*.

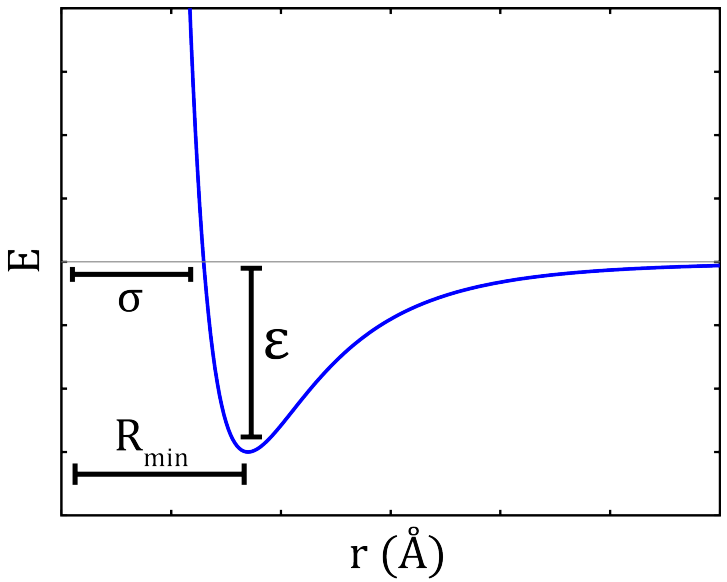
$$\vec{F}_{i \to j} = -\vec{F}_{j \to i} \tag{12}$$

- Paralelización del código (OpenMP, MPI, GPUs).

Potencial de Lennard-Jones

- En el código de ejemplo usaremos el potencial de Lennard-Jones:

$$U_{LJ} = 3\varepsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right] \tag{13}$$



Fuerzas

```
1   #ifndef _VANDERWAALSFORCE_FNF_
2   #define _VANDERWAALSFORCE_FNF_
3   #include "kinds.F95"
4   #include "mdsys.F95"
5   MODULE VanderWaalsForce
6       use kinds
7       use MDSystem
8       IMPLICIT NONE
9       REAL(KIND=dbl)  :: c12,c6
10  CONTAINS
11      SUBROUTINE InitVanderWaalsForce
12          c12 = 4.0_dbl*epsilon*sigma**12
13          c6  = 4.0_dbl*epsilon*sigma**6
14      END SUBROUTINE InitVanderWaalsForce
```

Fuerzas

```
1     SUBROUTINE AddVanderWaalsForces
2         use MDSystem
3         IMPLICIT NONE
4         REAL(kind=dbl) :: rsq, rcutsq, rinv, delta(3)
5         REAL(kind=dbl) :: r6, ffac
6         INTEGER :: i, j
7
8         epot=0.0_dbl
9         f=0.0_dbl
10        rcutsq=rcut*rcut
11        DO i=1,natoms
12            DO j=i+1,natoms
13                delta=x(i,:)-x(j,:)
14                rsq = dot_product(delta,delta)
15                IF (rsq < rcutsq) THEN
16                    rinv = 1.0_dbl/rsq
17                    r6 = rinv*rinv*rinv
18                    ffac = (12.0_dbl*c12*r6 - 6.0_dbl*c6)*r6*rinv
19                    epot = epot + r6*(c12*r6 - c6)
20
21                    f(i,:) = f(i,:) + delta*ffac
22                    f(j,:) = f(j,:) - delta*ffac
23                END IF
24            END DO
25        END DO
26
27    END SUBROUTINE AddVanderWaalsForces
28 END MODULE VanderWaalsForce
29 #endif
30 !#endif _VANDERWAALSFORCE_FNF_
```

Taller

Latin American Introductory School to Parallel Programming and Parallel Architecture for High Performance Computing

12 to 23 February 2018
CINVESTAV and ININ, Ocoyoacac, Mexico

Further information:
http://indico.ictp.it/event/8344/
smr3187@ictp.it

The School has the goal of teaching participating scientists about modern computer hardware and programming to provide a foundation for future computational research using High Performance Computing (HPC). Participants will go through an intensive programme with a focus on practical skills.

Description:

School participants will learn to improve the efficiency of their research codes, and to parallelize them. Lectures on a selection of technical aspects of modern HPC hardware will be mixed with introductions to widely used parallel programming tools and libraries. The hands-on sessions will allow participants to practice on small example problems of general scientific interest. Example topics will cover numerical methods and parallel strategies, as well as data management.
The programme specifically addresses the needs of scientists using, writing, or modifying HPC applications. It will be mainly based on fundamental HPC-relevant features in widely used scientific software for high-performance computing:

- Computer architectures for HPC and how to optimize for them
- Parallel programming tools (MPI & OpenMP)
- Portable, flexible and parallel I/O (HDF5)
- Parallel programming best pactices
- Floating-point math
- High-performance libraries for the solution of common math problems

How to apply:

Online application:
http://indico.ictp.it/event/8344

Female scientists are encouraged to apply.

Grants:

A limited number of grants are available to support the attendance of selected participants from developing countries, with priority given to young scientists from Latin America. Accommodation and meals are covered at the campus for all selected participants. There is no registration fee.

Directors:

J. Klapp, ININ and Cinvestav-Abacus
I. Gitler, Cinvestav-Abacus
L. Sigalotti, UAM-Azcapotzalco
E. Santos Rodríguez, MCTP
M. Cruchaga, Universidad de Santiago de Chile

Lecturers:

R. Berger, Temple University
G. Pringle, EPCC
I. Girotto, ICTP

ICTP Scientific Contact:

I. Girotto, ICTP

Deadline:

26 November 2017