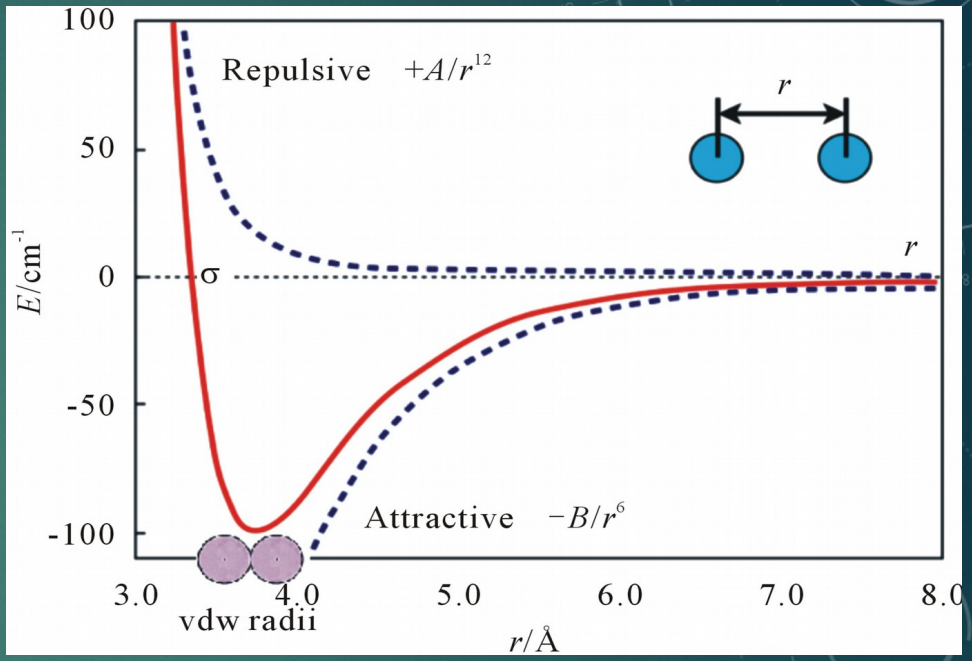
Potenziale Lennard-Jones con dipendenza radiale di σ

di:
Giorgio Guerini Rocco
Tommaso Forni

Corso Metodi Computazionali della Fisica 2016/2017

Potenziale LJ standard nella forma 12-6:

$$V_{LJ}(r) = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$



Lo scopo del progetto è implementare una modifica al potenziale LJ della forma:

$$\sigma \rightarrow \sigma(R) = \sigma_0\left(1 - \frac{\alpha_{max}}{\Delta t} tR\right)$$

R: distanza radiale dall'origine.
α: parametro di contrazione.
Δt: intervallo totale di tempo.

La classe *pair_mylj:*

- PairStyle(mylj/cut,PairMyLJCut)
- void PairMyLJCut::compute(int eflag, int vflag)
- void PairMyLJCut::allocate()
- void PairMyLJCut::coeff(int narg, char **arg)
- void *PairMyLJCut::extract(const char *str, int &dim)

Il metodo compute:

```
shrink = (1 - alpha[itype][jtype]*rm );
shrink = shrink * shrink;
if (rsq < cutsq[itype][jtype] * shrink) {

        shrink = shrink * shrink * shrink;
        //shrink = pow(shrink,3.0);
        r2inv = 1.0/rsq;
        r6inv = r2inv*r2inv*r2inv;
        forcelj = shrink * r6inv * (lj1[itype][jtype]*r6inv*shrink -
lj2[itype][jtype]);
        fpair = factor_lj*forcelj*r2inv;
        f[i][0] += delx*fpair;
        f[i][1] += dely*fpair;
        f[i][2] += delz*fpair;
}
```

Test: script in python

- class particle:
- class simulation:
    - def force(self,a, b):
    - def compute(self,d):
    - def update(self):
    - def energydump(self):
    - def run(self,dumps):

Lo script implementa un integratore al primo ordine: usando un timestep della simulazione molto piccolo le energie e le forze del sistema (di poche particelle) sono in accordo con quelle calcolate da LAMMPS

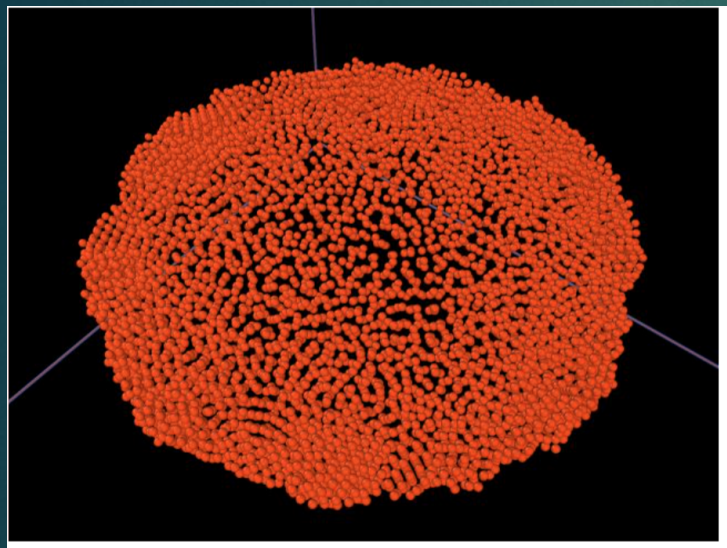
Script LAMMPS: quench.lmp

```
region          box1 block -20 20 -20 20 0 5
region          disco cylinder z 0 0 v_r 0 5
create_box      1 box1
create_atoms            1 random 8000 2452454 box1
mass            1 1.0
velocity        all create $T 87287
pair_style      lj/cut 1.3
pair_coeff      1 1 1.0 1.0 1.3
neighbor        0.3 bin
neigh_modify    every 20 delay 0 check no
min_style       cg
minimize        0.01 1.0e-8 1000 100000
fix             1 all nve
fix             3 all heat 1  -200
run             10000
```

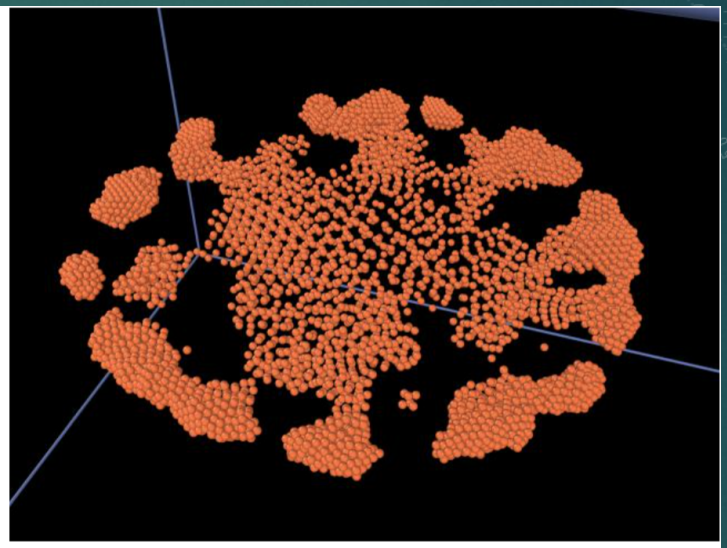
Script LAMMPS: rdisk.lmp

```
read_dump    slowdisk.txt 0 x y z box no add yes
pair_style   mylj/cut 1.3
pair_coeff   1 1 1.0 1.0 1.3  0.0
variable     shrink equal 0.7/v_r
variable     delta equal "ramp(0.0,v_shrink)"
fix          3 all adapt 1 pair mylj/cut alpha 1 1 v_delta
fix          1 all nve
fix          2 all viscous 5
dump         id all atom 50 dump.ov
thermo       50
run          10000
fix          3 all adapt 1 pair mylj/cut alpha 1 1 v_shrink
run          5000
```

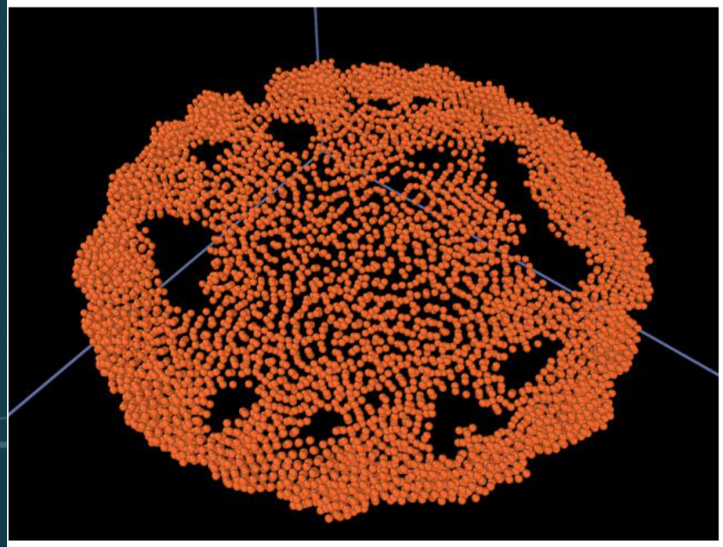
Risultati delle simulazioni: fasi osservate del sistema



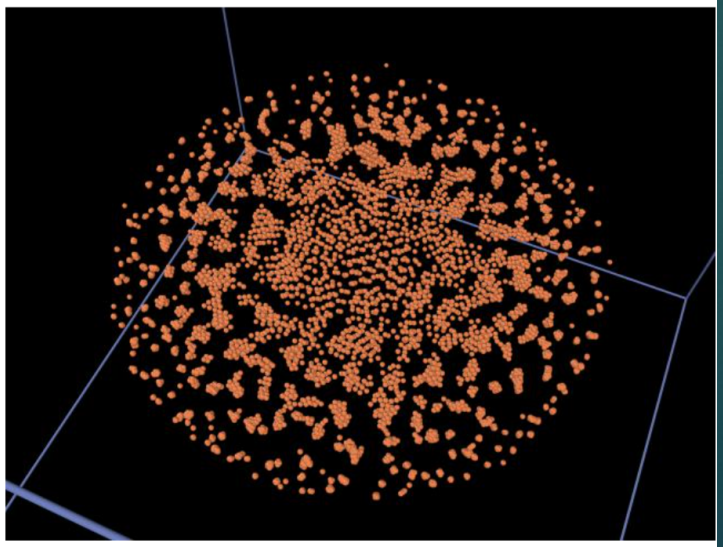
(a) *Restringimento.*



(b) *Cracking.*

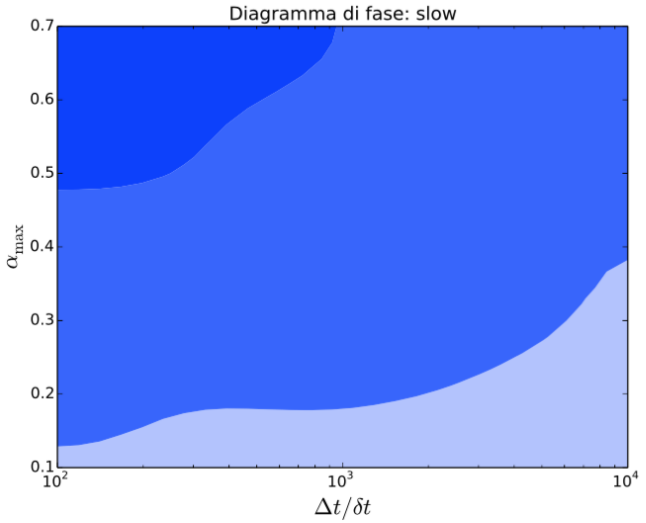


(c) *Formazioen di buchi.*

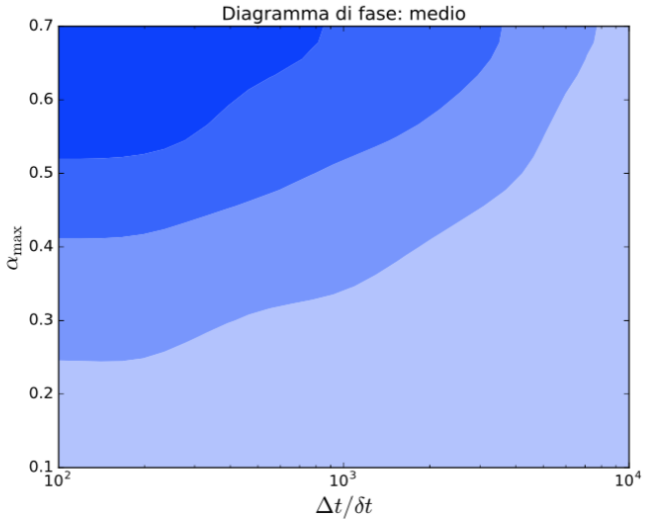


(d) *Frammentazione.*

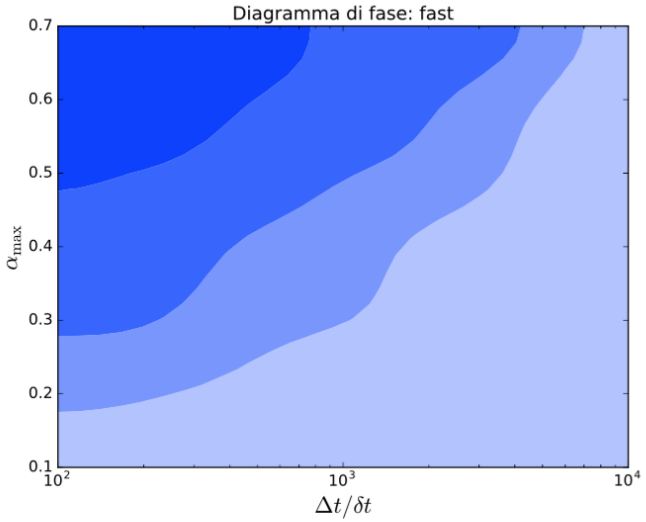
Risultati delle simulazioni: diagramma di fase del sistema



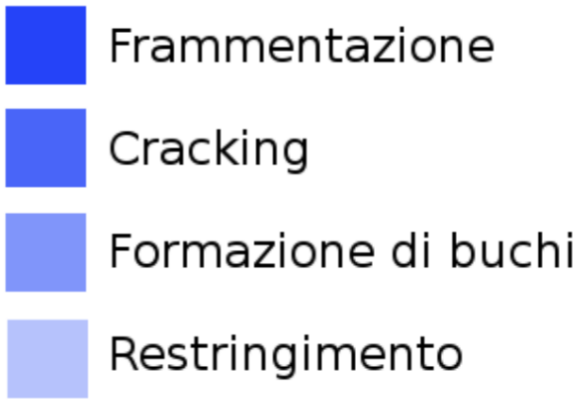
(a) *Quenching lento.*



(b) *Quenching medio.*



(c) *Quenching veloce.*



(d) *Legenda.*

Benchmark: confronto delle velocità di esecuzione delle simulazioni di 15000 timesteps

LJ (s)	MyLJ (s)	MyLJ ottimizzato (s)
13.3	62.4	25.7

Da test eseguiti l'istruzione in mylj.cpp responsabile del peggioramento delle prestazioni rispetto ad lj.cpp è:

```
forcelj = shrink6 * r6inv * (lj1[itype][jtype]*r6inv*shrink6 -
          lj2[itype][jtype]);
```