Molecular Dynamics

Aiichiro Nakano

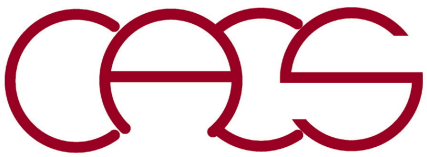
Collaboratory for Advanced Computing & Simulations
Department of Computer Science
Department of Physics & Astronomy
Department of Quantitative & Computational Biology
University of Southern California

Email: anakano@usc.edu

Molecular dynamics (MD) is an archetype of scientific computing; use it to learn parallel computing & visualization

Objectives: Understand

1. `md.c` line by line
2. simple MD principles well
3. computational complexity & flop/s



Father of Molecular Dynamics

PHYSICAL REVIEW VOLUME 136, NUMBER 2A 19 OCTOBER 1964

Correlations in the Motion of Atoms in Liquid Argon*

A. RAHMAN
Argonne National Laboratory, Argonne, Illinois
(Received 6 May 1964)

A system of 864 particles interacting with a Lennard-Jones potential and obeying classical equations of motion has been studied on a digital computer (CDC 3600) to simulate molecular dynamics in liquid argon at 94.4°K and a density of 1.374 g cm^{-3}.

Aneesur Rahman—Father of molecular dynamics

Argonne physicist Aneesur Rahman, known worldwide as the "father of molecular dynamics," pioneered the application of computer science to physical systems.

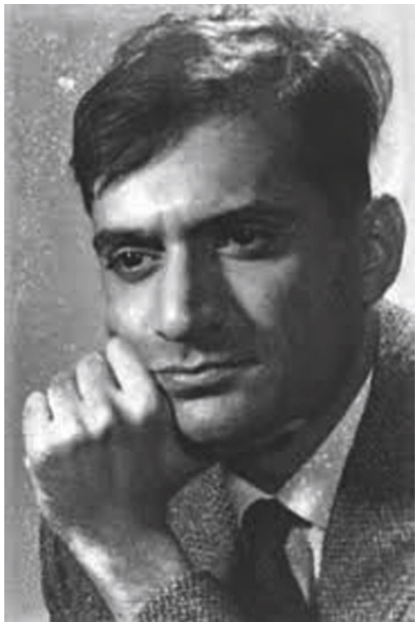
In 1960, Rahman successfully modeled the behavior of a cluster of 864 argon atoms on a computer that could perform only 150,000 calculations per second. **150 Kflop/s!**

While Argonne's new IBM Blue Gene ® /P supercomputer runs nearly 3 million times faster than Rahman's CDC 3600, today's scientists still base the code for their models on Rahman's algorithms.

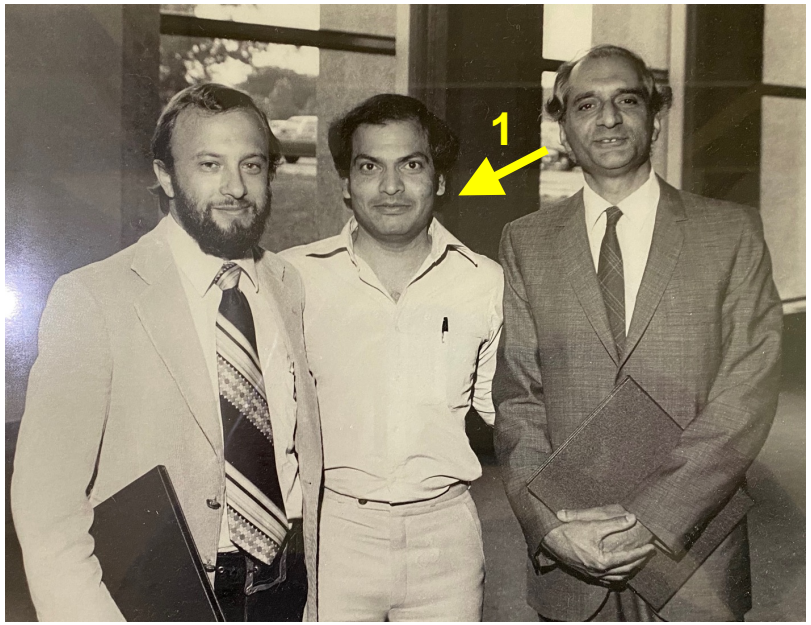
Since 1993, the American Physical Society has annually awarded the Aneesur Rahman Prize for outstanding achievement in computational physics research.

See the Nobel lecture by Michael Levitt [https://aiichironakano.github.io/phys516/levitt-lecture-slides.pdf]
See Berkeley CS 267 HW2 [https://sites.google.com/lbl.gov/cs267-spr2024]

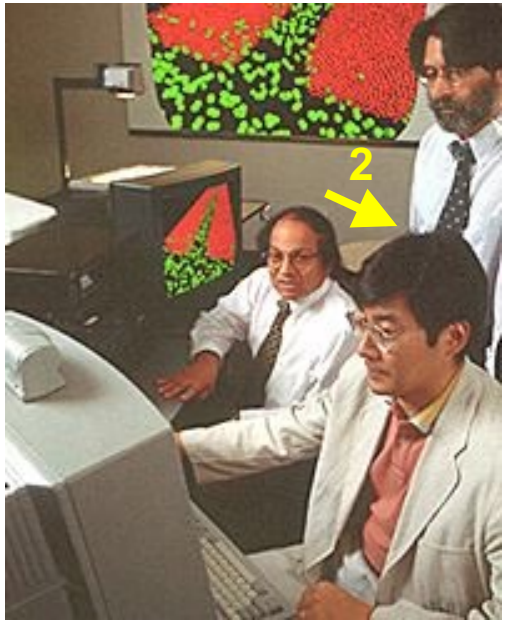
Your Rahman Number?



Anees Rahman ('67)



Anees Rahman & Priya Vashishta
at Argonne National Lab ('81)



https://aiichironakano.github.io/cs596/Battimelli-ComputerMeetsPhysics-Springer20.pdf, pp. 58 & 128

Priya Vashishta,
Rajiv Kalia
and AN ('02)

Your Rahman number is 3

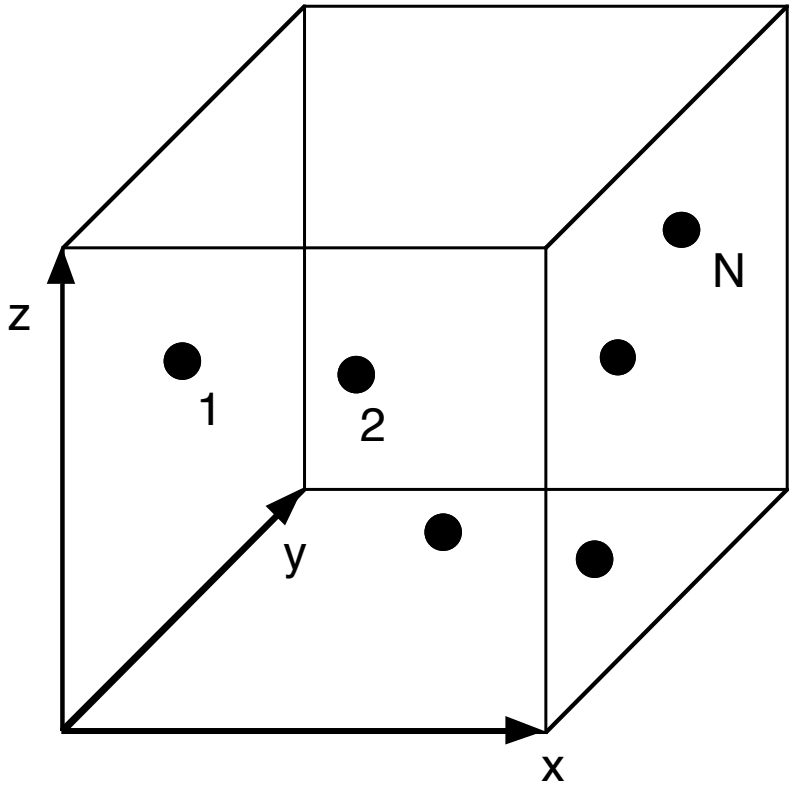
System: A Set of Point Atoms

$$\{\vec{r}_i = (x_i, y_i, z_i) \mid x_i, y_i, z_i \in \Re, i = 0, \ldots, N-1\}$$

`int nAtom`: N, # of atoms.

md.c `NMAX`: Max # of atoms.

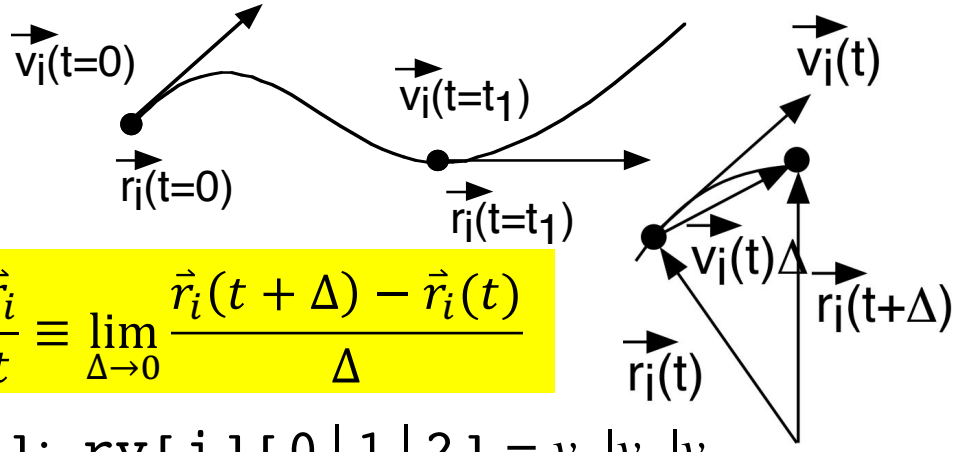
`double r[NMAX][3]`: `r[i][0|1|2]` = $x_i|y_i|z_i$.



See lecture on basic MD algorithm for math symbol *vs*. C variable correspondence

Trajectory

Trace of atom positions



Velocity

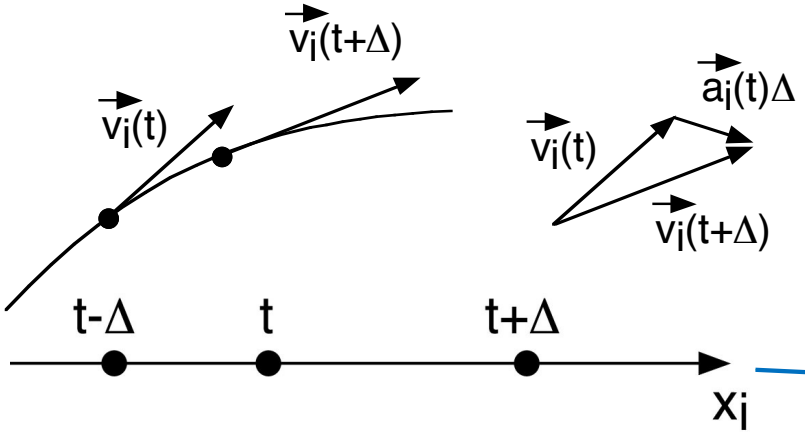
$$\vec{v}_i(t) = \dot{\vec{r}}_i(t) = \frac{d\vec{r}_i}{dt} \equiv \lim_{\Delta\to 0} \frac{\vec{r}_i(t+\Delta) - \vec{r}_i(t)}{\Delta}$$

`double rv[NMAX][3]: rv[i][0|1|2]` = $v_{ix}|v_{iy}|v_{iz}$

Acceleration

$$\vec{a}_i(t) = \ddot{\vec{r}}_i(t) = \frac{d^2\vec{r}_i}{dt^2} = \frac{d\vec{v}_i}{dt} \equiv \lim_{\Delta\to 0} \frac{\vec{v}_i(t+\Delta) - \vec{v}_i(t)}{\Delta}$$

`double ra[NMAX][3]: ra[i][0|1|2]` = $a_{ix}|a_{iy}|a_{iz}$



$$\vec{a}_i = \lim_{\Delta\to 0} \frac{\vec{v}_i(t+\Delta/2) - \vec{v}_i(t-\Delta/2)}{\Delta}$$

$$= \lim_{\Delta\to 0} \frac{\dfrac{\vec{r}_i(t+\Delta) - \vec{r}_i(t)}{\Delta} - \dfrac{\vec{r}_i(t) - \vec{r}_i(t-\Delta)}{\Delta}}{\Delta}$$

$$= \lim_{\Delta\to 0} \frac{\vec{r}_i(t+\Delta) - 2\vec{r}_i(t) + \vec{r}_i(t-\Delta)}{\Delta^2}$$

3-point difference

Newton's Equation of Motion

Trajectory is determined by Newton's 2nd law:

$$m\ddot{\vec{r}}_i(t) = \vec{F}_i(t)$$

Initial value problem: Given initial particle positions & velocities, $\{(\vec{r}_i(0), \vec{v}_i(0))\}$
Obtain those at later times $\{(\vec{r}_i(t), \vec{v}_i(t)); t > 0\}$

Potential energy:

$$\vec{F}_i = -\frac{\partial}{\partial \vec{r}_i} V(\vec{r}^N) = -\left(\frac{\partial V}{\partial x_i}, \frac{\partial V}{\partial y_i}, \frac{\partial V}{\partial z_i}\right)$$

where the partial derivative is

$$\frac{\partial V}{\partial x_k} = \lim_{h \to 0} \frac{V(x_0, y_0, z_0, \ldots, x_k + h, y_k, z_k, \ldots x_{N-1}, y_{N-1}, z_{N-1}) - V(x_0, y_0, z_0, \ldots, x_k, y_k, z_k, \ldots x_{N-1}, y_{N-1}, z_{N-1})}{h}$$

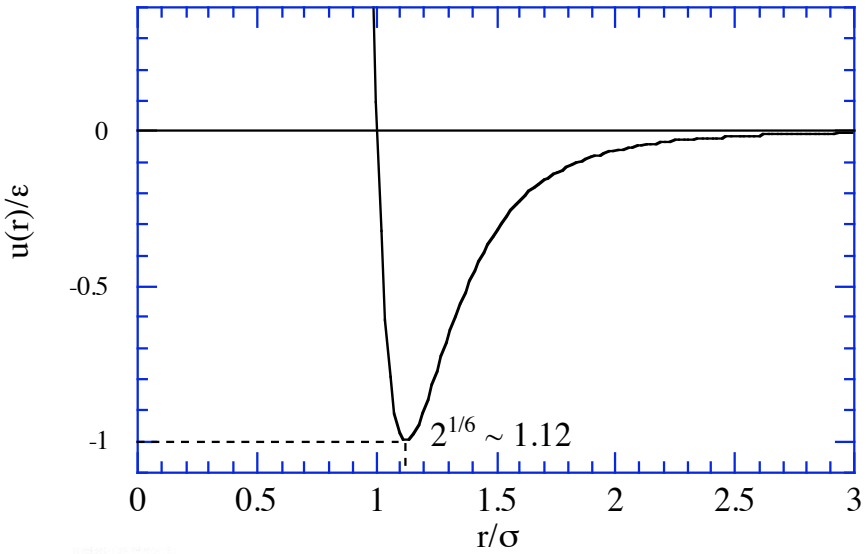
Lennard-Jones Potential

Sum over ∀distinct pairs

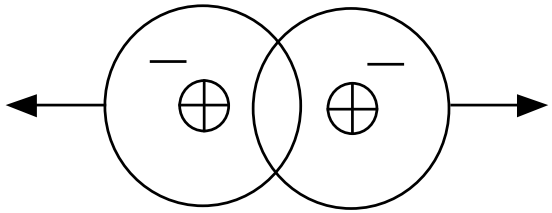
$$V(\vec{r}^N) = \sum_{i<j} u(r_{ij}) = \sum_{i=0}^{N-2} \sum_{j=i+1}^{N-1} u(r_{ij})$$

where $\vec{r}_{ij} = \vec{r}_i - \vec{r}_j; r_{ij} = |\vec{r}_{ij}|$ **and**

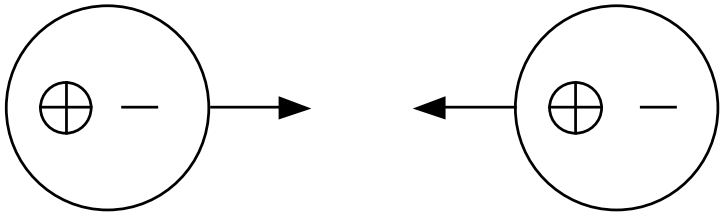
$$u(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$



Short-range repulsion by Pauli exclusion between electrons



Long-range attraction by polarization



Johanes D. van der Waals
Nobel Physics Prize (1910)

Wolfgang Pauli
Nobel Physics Prize (1945)

John E. Lennard-Jones
PPS **43**, 461 (1931)

Normalization

For Argon atoms: *Atoms are tiny & fast*

- $m = 6.6 \times 10^{-23}$ gram
- $\varepsilon = 1.66 \times 10^{-14}$ erg (erg = gram•cm^2/second^2)
- $\sigma = 3.4 \times 10^{-8}$ cm

Define length, energy & time units as

$$\begin{cases} \vec{r}_i = \vec{r}_i' \sigma = 3.4\times10^{-8}[\text{cm}]\times\vec{r}_i' \\ V = V'\varepsilon = 1.66\times10^{-14}[\text{erg}]\times V' \\ t = \sigma\sqrt{m/\varepsilon}\,t' = 2.2\times10^{-12}[\text{sec}]\times t' \end{cases}$$

The equation of motion in these units:

$$\frac{d^2\vec{r}_i}{dt^2} = -\frac{\partial V}{\partial \vec{r}_i} = \vec{a}_i$$

$$V(\vec{r}^N) = \sum_{i<j} u(r_{ij})$$

$$u(r) = 4\left(\frac{1}{r^{12}} - \frac{1}{r^6}\right)$$

1.66×10^{-14} erg
$= 3.97 \times 10^{-25}$ kcal
$= 7.35 \times 10^{-28}$ Big Mac

540 kcal

Analytic Force Formula

Chain rule: $\vec{a}_k = -\frac{\partial}{\partial \vec{r}_k}\sum_{i<j} u(r_{ij}) = -\sum_{i<j}\frac{\partial r_{ij}}{\partial \vec{r}_k}\frac{du}{dr_{ij}}$

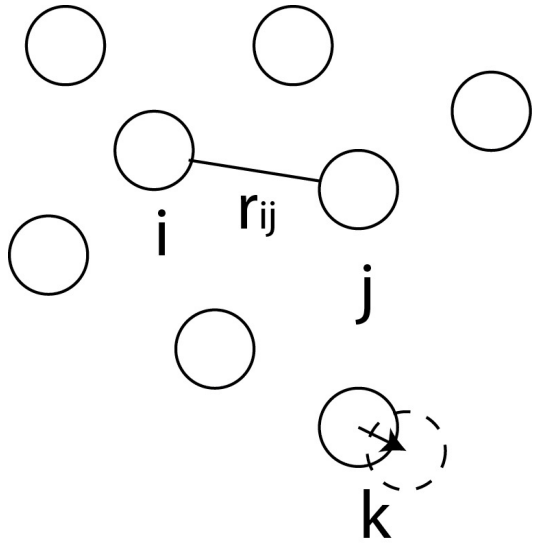
$$\frac{\partial r_{ij}}{\partial \vec{r}_k} = \left(\frac{\partial}{\partial x_k}, \frac{\partial}{\partial y_k}, \frac{\partial}{\partial z_k}\right)\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$$

$$= \frac{\left(2(x_i - x_j), 2(y_i - y_j), 2(z_i - z_j)\right)}{2\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}}(\delta_{ik} - \delta_{jk}) \quad \delta_{ij} = \begin{cases} 1, & i = k \\ 0, & i \neq k \end{cases}$$

$$\left(\because \frac{d}{dx}[f(x)]^{1/2} = \frac{1}{2}[f(x)]^{-1/2}\frac{df}{dx}\right)$$

$$\frac{du}{dr} = 4\left(-\frac{12}{r^{13}} + \frac{6}{r^7}\right) = -\frac{48}{r}\left(\frac{1}{r^{12}} - \frac{1}{2r^6}\right)$$

$$\left(\because \frac{d}{dr}r^{-n} = -nr^{-n-1}\right)$$



For chain rule, see Sec. 6.5.2 in "Deep Learning" (https://www.deeplearningbook.org)

Molecular Dynamics Problem

Given initial atomic positions & velocities, $\{(\vec{r}_i(0), \vec{v}_i(0)) | i = 0, \ldots, N-1\}$,

obtain those at later times, $\{(\vec{r}_i(t), \vec{v}_i(t)) | i = 0, \ldots, N-1; t > 0\}$,

by integrating the ordinary differential equation,

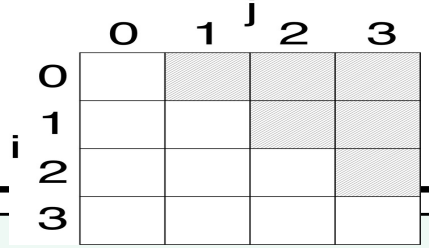
$$\ddot{\vec{r}}_k(t) = \vec{a}_k(t) = -\frac{\partial}{\partial \vec{r}_k} \sum_{i<j} u(r_{ij}) = \sum_{i<j} \vec{r}_{ij}(t) \left(-\frac{1}{r}\frac{du}{dr}\right)_{r=r_{ij}(t)} (\delta_{ik} - \delta_{jk})$$

$$\delta_{ik} = \begin{cases} 1 & i = k \\ 0 & i \neq k \end{cases}$$

$$\vec{r}_{ij}(t) = \vec{r}_i(t) - \vec{r}_j(t)$$

$$r_{ij}(t) = |\vec{r}_{ij}(t)|$$

where $$-\frac{1}{r}\frac{du}{dr} = \frac{48}{r^2}\left(\frac{1}{r^{12}} - \frac{1}{2r^6}\right)$$



Force calculation algorithm—$O(N^2)$:

for $k = 0$ to N-1, $\vec{a}_k = 0$

for $i = 0$ to N-2

for $j = i$+1 to N-1

compute $\vec{a} = \vec{r}_{ij}\left(-\frac{1}{r}\frac{du}{dr}\right)_{r=|\vec{r}_{ij}|}$

$\vec{a}_i$ += $\vec{a}$

$\vec{a}_j$ −= $\vec{a}$

Newton's 3rd law

```
for (n=0; n<nAtom; n++)
   for (k=0; k<3; k++) ra[n][k] = 0.0;
for (j1=0; j1<nAtom-1; j1++) {
   for (j2=j1+1; j2<nAtom; j2++) {
      for (rr=0.0, k=0; k<3; k++) {
         dr[k] = r[j1][k] - r[j2][k];
         dr[k] = dr[k] - SignR(RegionH[k],dr[k]-RegionH[k])
                       - SignR(RegionH[k],dr[k]+RegionH[k]);
         rr = rr + dr[k]*dr[k];
      }
      if (rr < rrCut) {
         ri2 = 1.0/rr; ri6 = ri2*ri2*ri2; r1 = sqrt(rr);
         fcVal = 48.0*ri2*ri6*(ri6-0.5) + Duc/r1;
         for (k=0; k<3; k++) {
            f = fcVal*dr[k];
            ra[j1][k] = ra[j1][k] + f;
            ra[j2][k] = ra[j2][k] - f;
         }
      }
   }
}
```

Iteration_count

$$= 1 + 2 + \cdots + (N-1)$$

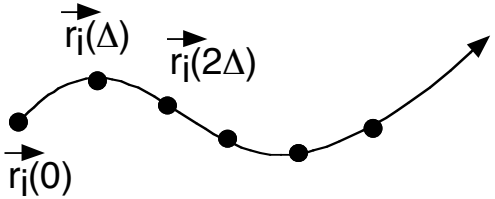
$$= \frac{(N-1)(1+N-1)}{2}$$

$$= \frac{(N-1)N}{2} = O(N^2)$$

Time Discretization

Snapshots with time interval Δ: `double DeltaT`

$$\left(\vec{r}_i(0), \vec{v}_i(0)\right) \mapsto \left(\vec{r}_i(\Delta), \vec{v}_i(\Delta)\right) \mapsto \left(\vec{r}_i(2\Delta), \vec{v}_i(2\Delta)\right) \mapsto \cdots$$



Question: How to predict the next state, $\left(\vec{r}_i(t+\Delta), \vec{v}_i(t+\Delta)\right)$**, from the current state,** $\left(\vec{r}_i(t), \vec{v}_i(t)\right)$**?**

Solution: Taylor expansion

$$f(x_0+h) = \sum_{n=0}^{\infty} \frac{h^n}{n!}\frac{d^n f}{dx^n} = f(x_0) + h\left.\frac{df}{dx}\right|_{x=x_0} + \left.\frac{h^2}{2}\frac{d^2 f}{dx^2}\right|_{x=x_0} + \left.\frac{h^3}{3!}\frac{d^3 f}{dx^3}\right|_{x=x_0} + \cdots$$

See the Taylor expansion note for a proof

Verlet Discretization

Position:

$$\vec{r}_i(t+\Delta)=\vec{r}_i(t)+\vec{v}_i(t)\Delta+\frac{1}{2}\vec{a}_i(t)\Delta^2+\frac{1}{6}\dddot{\vec{r}}_i(t)\Delta^3+O(\Delta^4)$$

$$+\ \vec{r}_i(t-\Delta)=\vec{r}_i(t)-\vec{v}_i(t)\Delta+\frac{1}{2}\vec{a}_i(t)\Delta^2-\frac{1}{6}\dddot{\vec{r}}_i(t)\Delta^3+O(\Delta^4)$$

$$\vec{r}_i(t+\Delta)+\vec{r}_i(t-\Delta)=2\vec{r}_i(t)+\vec{a}_i(t)\Delta^2+\frac{1}{6}\dddot{\vec{r}}_i(t)\Delta^3+O(\Delta^4)$$

$$\therefore \vec{r}_i(t+\Delta)=2\vec{r}_i(t)-\vec{r}_i(t-\Delta)+\vec{a}_i(t)\Delta^2+O(\Delta^4)$$

Velocity:

$$\vec{r}_i(t+\Delta)=\vec{r}_i(t)+\vec{v}_i(t)\Delta+\frac{1}{2}\vec{a}_i(t)\Delta^2+\frac{1}{6}\dddot{\vec{r}}_i(t)\Delta^3+O(\Delta^4)$$

$$-\ \vec{r}_i(t-\Delta)=\vec{r}_i(t)-\vec{v}_i(t)\Delta+\frac{1}{2}\vec{a}_i(t)\Delta^2-\frac{1}{6}\dddot{\vec{r}}_i(t)\Delta^3+O(\Delta^4)$$

$$\vec{r}_i(t+\Delta)-\vec{r}_i(t-\Delta)=2\vec{v}_i(t)\Delta+O(\Delta^3)$$

$$\therefore \vec{v}_i(t)=\frac{\vec{r}_i(t+\Delta)-\vec{r}_i(t-\Delta)}{2\Delta}+O(\Delta^2)$$

Verlet Algorithm

Verlet discretization:

$$\begin{cases} \vec{r}_i(t+\Delta) = 2\vec{r}_i(t) - \vec{r}_i(t-\Delta) + \vec{a}_i(t)\Delta^2 + O(\Delta^4) \\ \vec{v}_i(t) = \dfrac{\vec{r}_i(t+\Delta) - \vec{r}_i(t-\Delta)}{2\Delta} + O(\Delta^2) \end{cases}$$

Loup Verlet

Verlet algorithm:

Given $\vec{r}_i(t-\Delta)$ & $\vec{r}_i(t)$,

1. Compute $\vec{a}_i(t)$ as a function of $\{\vec{r}_i(t)\}$
2. $\vec{r}_i(t+\Delta) \leftarrow 2\vec{r}_i(t) - \vec{r}_i(t-\Delta) + \vec{a}_i(t)\Delta^2$
3. $\vec{v}_i(t) \leftarrow [\vec{r}_i(t+\Delta) - \vec{r}_i(t-\Delta)]/2\Delta$

ComputeAccel(): r[][] (in) → ra[][] (out)

```
for (n=0; n<nAtom; n++)
    for (k=0; k<3; k++) ra[n][k] = 0.0;
for (j1=0; j1<nAtom-1; j1++) {
    for (j2=j1+1; j2<nAtom; j2++) {
        for (rr=0.0, k=0; k<3; k++) {
            dr[k] = r[j1][k] - r[j2][k];
            dr[k] = dr[k] - SignR(RegionH[k],dr[k]-RegionH[k])
                          - SignR(RegionH[k],dr[k]+RegionH[k]);
            rr = rr + dr[k]*dr[k];
        }
        if (rr < rrCut) {
            ri2 = 1.0/rr; ri6 = ri2*ri2*ri2; r1 = sqrt(rr);
            fcVal = 48.0*ri2*ri6*(ri6-0.5) + Duc/r1;
            for (k=0; k<3; k++) {
                f = fcVal*dr[k];
                ra[j1][k] = ra[j1][k] + f;
                ra[j2][k] = ra[j2][k] - f;
            }
        }
    }
}
```

Drawback: Positions & velocities are not simultaneously updated for the same time step

Velocity Verlet Algorithm

Theorem: The following algebraic equation gives the same sequence of states, $(\vec{r}_i(n\Delta), \vec{v}_i(n\Delta))$, as that obtained by the Verlet discretization.

$$\begin{cases} \vec{r}_i(t+\Delta) = \vec{r}_i(t) + \vec{v}_i(t)\Delta + \frac{1}{2}\vec{a}_i(t)\Delta^2 \\ \vec{v}_i(t+\Delta) = \vec{v}_i(t) + \frac{\vec{a}_i(t) + \vec{a}_i(t+\Delta)}{2}\Delta \end{cases}$$

Velocity Verlet algorithm:

Given $(\vec{r}_i(t), \vec{v}_i(t))$,

1. Compute $\vec{a}_i(t)$ as a function of $\{\vec{r}_i(t)\}$
2. $\vec{v}_i\left(t+\frac{\Delta}{2}\right) \leftarrow \vec{v}_i(t) + \frac{\Delta}{2}\vec{a}_i(t)$
3. $\vec{r}_i(t+\Delta) \leftarrow \vec{r}_i(t) + \vec{v}_i\left(t+\frac{\Delta}{2}\right)\Delta$
4. Compute $\vec{a}_i(t+\Delta)$ as a function of $\{\vec{r}_i(t+\Delta)\}$
5. $\vec{v}_i(t+\Delta) \leftarrow \vec{v}_i\left(t+\frac{\Delta}{2}\right) + \frac{\Delta}{2}\vec{a}_i(t+\Delta)$

```
void SingleStep() {
  int n,k;
  HalfKick();
  for (n=0; n<nAtom; n++)
    for (k=0; k<3; k++)
      r[n][k] = r[n][k]
              + DeltaT*rv[n][k];
  ApplyBoundaryCond();
  ComputeAccel();
  HalfKick();
}

void HalfKick() {
  int n,k;
  for (n=0; n<nAtom; n++)
    for (k=0; k<3; k++)
      rv[n][k] = rv[n][k]
               + DeltaTH*ra[n][k];
}
```

Velocity Verlet Algorithm for `StepLimit` Steps

Initialize $(\vec{r}_i, \vec{v}_i)$ for all i

Compute $\vec{a}_i$ for all i as a function of $\{\vec{r}_i\}$ **function `ComputeAccel()`**

for `stepCount` = 1 to `StepLimit`

do the following **function `SingleStep()`**

$\vec{v}_i \leftarrow \vec{v}_i + \vec{a}_i \Delta/2$ for all i

$\vec{r}_i \leftarrow \vec{r}_i + \vec{v}_i \Delta$ for all i

Compute $\vec{a}_i$ for all i as a function of $\{\vec{r}_i\}$ **function `ComputeAccel()`**

$\vec{v}_i \leftarrow \vec{v}_i + \vec{a}_i \Delta/2$ for all i

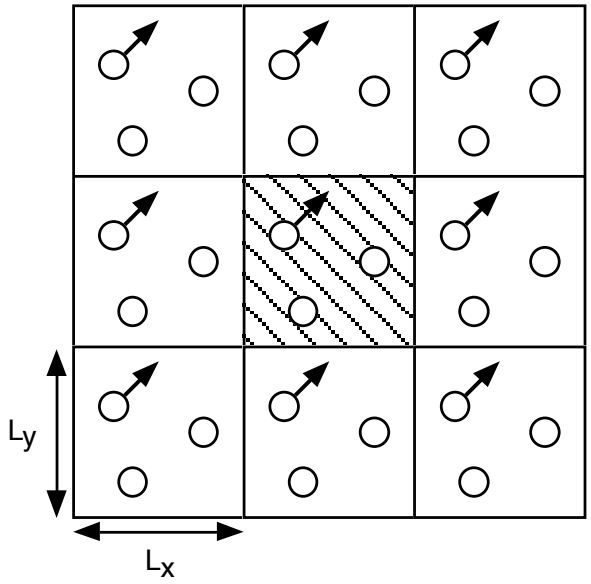
endfor

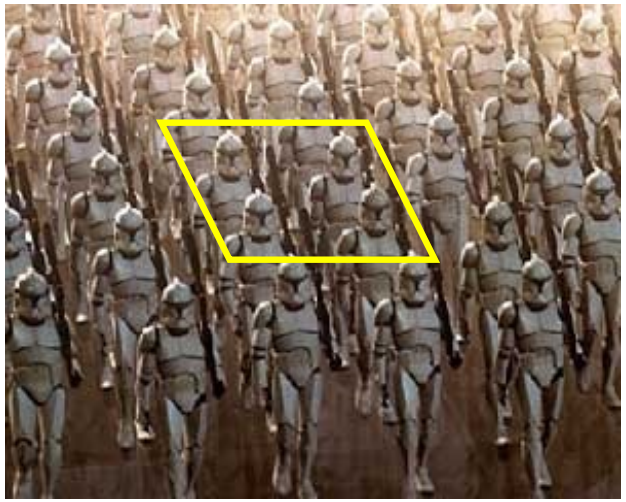
`stepLimit+1` calls to function `ComputeAccel()`

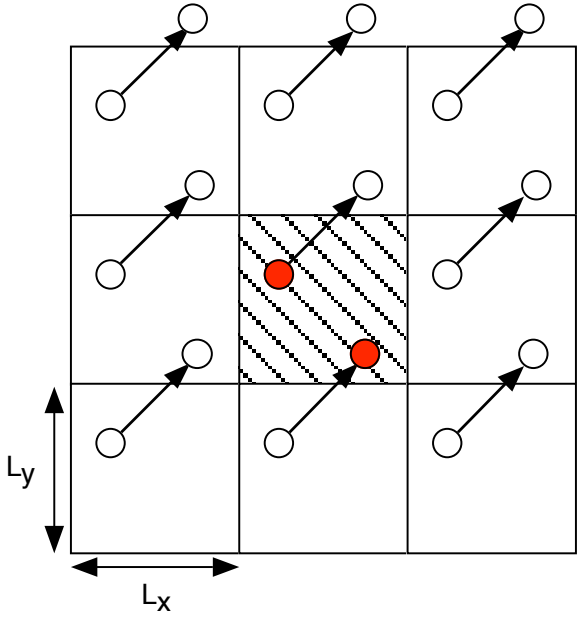
```
ComputeAccel();
for (stepCount=1; stepCount<=StepLimit; stepCount++) {
  SingleStep();
  if (stepCount%StepAvg == 0) EvalProps();
}
```

Periodic Boundary Condition

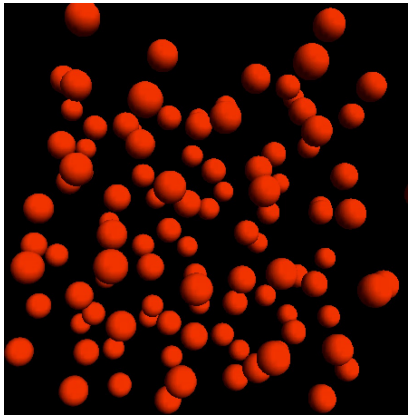
- Allows simulation of bulk properties
- Replicate a simulation box of size $L_x \times L_y \times L_z$ to form an infinite lattice
  `double Region[3]:Region[0|1|2]` $= L_x | L_y | L_z$
- All images of an atom in the original simulation box move in concert







- The representative atom of each image set is defined to reside in the central simulation box
  $0 \le x_i < L_x, 0 \le y_i < L_y, 0 \le z_i < L_z$
- A representative atom, which leaves the central box, returns to the central box from the other side of the box



Periodic Boundary Condition

Implementation:

$$\begin{cases} x_i \leftarrow x_i - \text{SignR}\left(\frac{L_x}{2}, x_i\right) - \text{SignR}\left(\frac{L_x}{2}, x_i - L_x\right) \\ y_i \leftarrow y_i - \text{SignR}\left(\frac{L_y}{2}, y_i\right) - \text{SignR}\left(\frac{L_y}{2}, y_i - L_y\right) \\ z_i \leftarrow z_i - \text{SignR}\left(\frac{L_z}{2}, z_i\right) - \text{SignR}\left(\frac{L_z}{2}, z_i - L_z\right) \end{cases}$$

$$\text{SignR}\left(\frac{L_x}{2}, x_i\right) = \begin{cases} \frac{L_x}{2} & x_i > 0 \\ -\frac{L_x}{2} & x_i \le 0 \end{cases}$$

```
void ApplyBoundaryCond() {
  int n,k;
  for (n=0; n<nAtom; n++)
    for (k=0; k<3; k++)
      r[n][k]=r[n][k]
              -SignR(RegionH[k],r[n][k])
              -SignR(RegionH[k],r[n][k]-Region[k]);
}
```

```
double SignR(double v,double x) {
  if (x > 0)
    return v;
  else
    return -v;
}
```

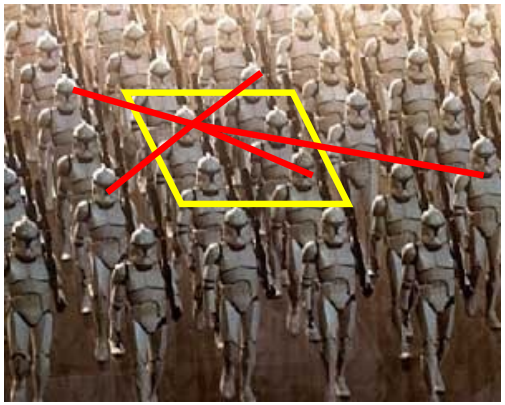
`double RegionH[3]`: An array $(L_x/2, L_y/2, L_z/2)$

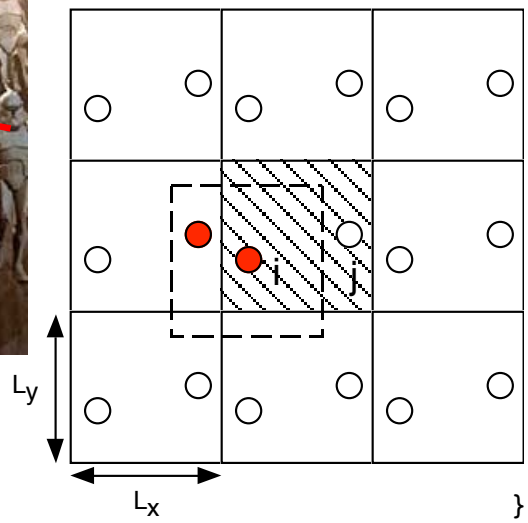
Minimum Image Convention

- **Each atom, *i*, interacts with only the image with the minimum distance among all the images of another atom, *j***
- **Exact, if the interatomic potential has a finite range,** $r_c \le \min(L_x, L_y, L_z)/2$

$$\begin{cases} -\frac{L_x}{2} \le x_{ij} < \frac{L_x}{2} \\ -\frac{L_y}{2} \le y_{ij} < \frac{L_y}{2} \\ -\frac{L_z}{2} \le z_{ij} < \frac{L_z}{2} \end{cases}$$

$$\begin{cases} x_{ij} \leftarrow x_{ij} - \mathrm{SignR}\left(\frac{L_x}{2}, x_{ij} + \frac{L_x}{2}\right) - \mathrm{SignR}\left(\frac{L_x}{2}, x_{ij} - \frac{L_x}{2}\right) \\ y_{ij} \leftarrow y_{ij} - \mathrm{SignR}\left(\frac{L_y}{2}, y_{ij} + \frac{L_y}{2}\right) - \mathrm{SignR}\left(\frac{L_x}{2}, y_{ij} - \frac{L_x}{2}\right) \\ z_{ij} \leftarrow z_{ij} - \mathrm{SignR}\left(\frac{L_z}{2}, z_{ij} + \frac{L_z}{2}\right) - \mathrm{SignR}\left(\frac{L_z}{2}, z_{ij} - \frac{L_z}{2}\right) \end{cases}$$



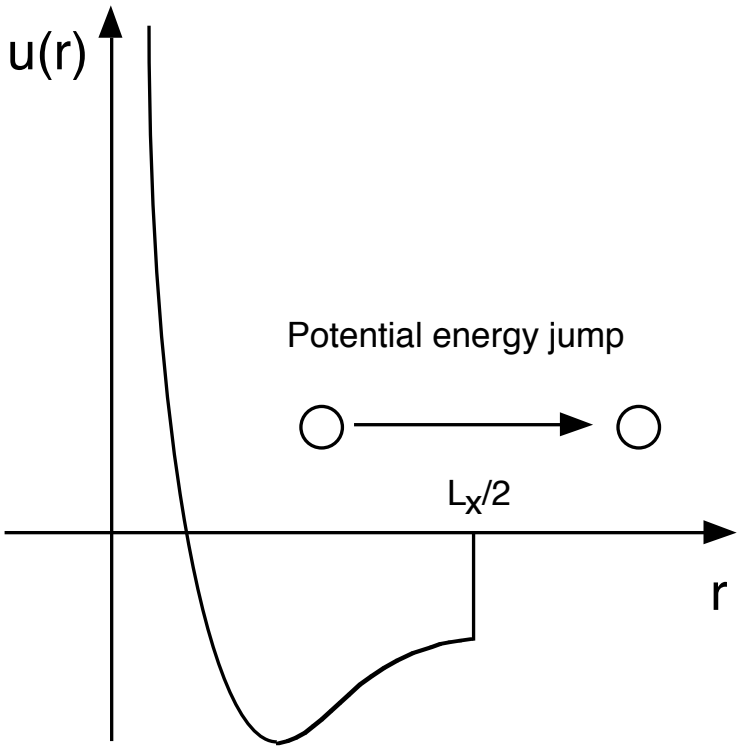


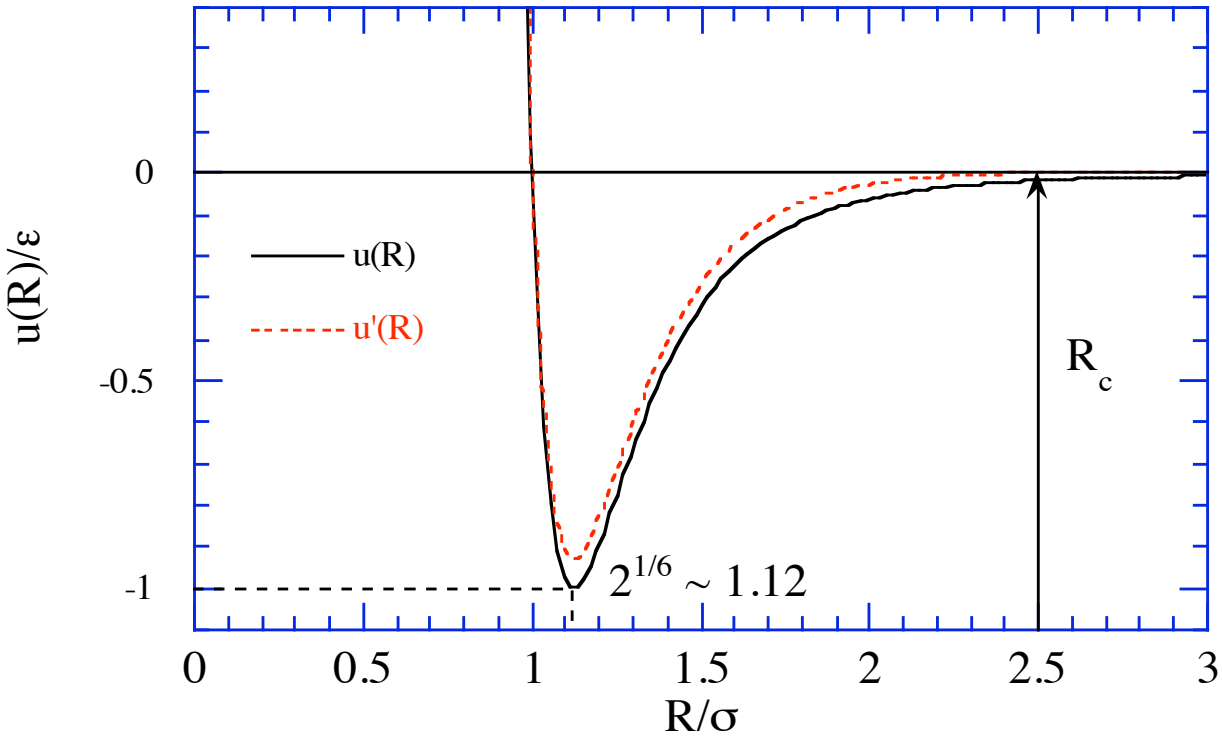

```
for (j1=0; j1<nAtom-1; j1++) {
   for (j2=j1+1; j2<nAtom; j2++) {
      for (rr=0.0, k=0; k<3; k++) {
         dr[k] = r[j1][k] - r[j2][k];
         dr[k] = dr[k] - SignR(RegionH[k],dr[k]-RegionH[k])
                       - SignR(RegionH[k],dr[k]+RegionH[k]);
         rr = rr + dr[k]*dr[k];
      }
      if (rr < rrCut) {
         ri2 = 1.0/rr; ri6 = ri2*ri2*ri2; r1 = sqrt(rr);
         fcVal = 48.0*ri2*ri6*(ri6-0.5) + Duc/r1;
         for (k=0; k<3; k++) {
            f = fcVal*dr[k];
            ra[j1][k] = ra[j1][k] + f;
            ra[j2][k] = ra[j2][k] - f;
         }
      }
   }
}
```

Shifted Potential

- **Make the minimum image convention exact**
- **Truncate $u(r)$ such that both it & its derivative du/dr are continuous at $r = r_c$**

$$u'(r) = \begin{cases} u(r) - u(r_c) - (r - r_c)\left.\dfrac{du}{dr}\right|_{r=r_c} & r < r_c \\ 0 & r \geq r_c \end{cases}$$






```
#define RCUT 2.5
```

Force Calculation in `md.c`

```
for (n=0; n<nAtom; n++)
  for (k=0; k<3; k++) ra[n][k] = 0.0;
for (j1=0; j1<nAtom-1; j1++) {
  for (j2=j1+1; j2<nAtom; j2++) {
    for (rr=0.0, k=0; k<3; k++) {
      dr[k] = r[j1][k] - r[j2][k];
      dr[k] = dr[k] - SignR(RegionH[k],dr[k]-RegionH[k])
                    - SignR(RegionH[k],dr[k]+RegionH[k]);
      rr = rr + dr[k]*dr[k];
    }
    if (rr < rrCut) {
      ri2 = 1.0/rr; ri6 = ri2*ri2*ri2; r1 = sqrt(rr);
      fcVal = 48.0*ri2*ri6*(ri6-0.5) + Duc/r1;
      for (k=0; k<3; k++) {
        f = fcVal*dr[k];
        ra[j1][k] = ra[j1][k] + f;
        ra[j2][k] = ra[j2][k] - f;
      }
    }
  }
}
```

minimum image convention

shifted potential

```
rr = RCUT*RCUT; ri2 = 1.0/rr; ri6 = ri2*ri2*ri2; r1 = sqrt(rr);
Uc = 4.0*ri6*(ri6-1.0);
Duc = -48.0*ri6*(ri6-0.5)/r1;
```

$$\vec{a}_k(t) = \sum_{i<j} \vec{r}_{ij}(t) \left(\frac{48}{r^2} \left(\frac{1}{r^{12}} - \frac{1}{2r^6} \right) \right)_{r=|\vec{r}_{ij}(t)|} (\delta_{ik} - \delta_{jk})$$

Energy Conservation

- **Total energy = kinetic energy + potential energy**

$$E = K + V = \sum_{i=0}^{N-1} \frac{m}{2} \left|\dot{\vec{r}}_i\right|^2 + \sum_{i<j} u(r_{ij})$$

- **Total energy is constant as a function of time, if no discretization error**

$$\dot{E} = \dot{K} + \dot{V}$$

$$= \sum_{i=0}^{N-1} \frac{m}{2} \frac{d}{dt} \left(\dot{x}_i^2 + \dot{y}_i^2 + \dot{z}_i^2\right) + \sum_{i=0}^{N-1} \left(\frac{dx_i}{dt}\frac{\partial V}{\partial x_i} + \frac{dy_i}{dt}\frac{\partial V}{\partial y_i} + \frac{dz_i}{dt}\frac{\partial V}{\partial z_i}\right)$$

$$= \sum_{i=0}^{N-1} \frac{m}{2} 2(\dot{x}_i\ddot{x}_i + \dot{y}_i\ddot{y}_i + \dot{z}_i\ddot{z}_i) + \sum_{i=0}^{N-1} \left(\dot{x}_i \frac{\partial V}{\partial x_i} + \dot{y}_i \frac{\partial V}{\partial y_i} + \dot{z}_i \frac{\partial V}{\partial z_i}\right)$$

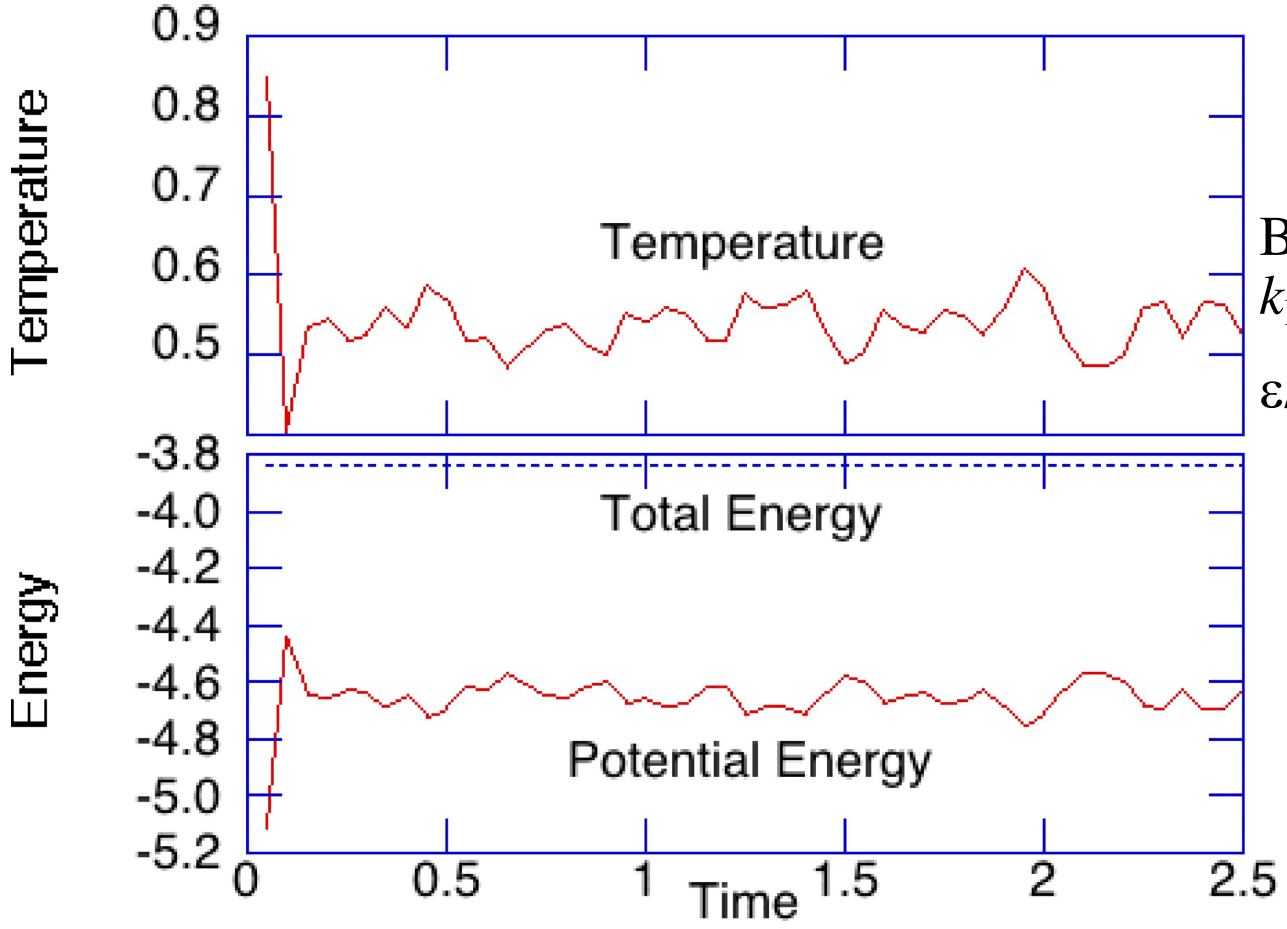
$$= \sum_{i=0}^{N-1} m\dot{\vec{r}}_i \bullet \ddot{\vec{r}}_i + \sum_{i=0}^{N-1} \dot{\vec{r}}_i \bullet \frac{\partial V}{\partial \vec{r}_i} = \sum_{i=0}^{N-1} \dot{\vec{r}}_i \bullet \left(m\ddot{\vec{r}}_i + \frac{\partial V}{\partial \vec{r}_i}\right) = 0$$

$$\vec{a} \bullet \vec{b} = (a_x, a_y, a_z) \bullet (b_x, b_y, b_z) = a_x b_x + a_y b_y + a_z b_z \text{ and } \frac{d}{dt}(fg) = \dot{f}g + f\dot{g}$$

```
double kinEnergy, potEnergy, totEnergy;
```

Energy Conservation

md.c: Δ = 0.005



Temperature, *T*

$$\frac{3Nk_BT}{2} = \sum_{i=0}^{N-1} \frac{m}{2} \left|\dot{\vec{r}}_i\right|^2$$

Boltzmann constant:
$k_B = 1.38062 \times 10^{-16}$ erg/Kelvin

$\varepsilon/k_B = 120$ Kelvin

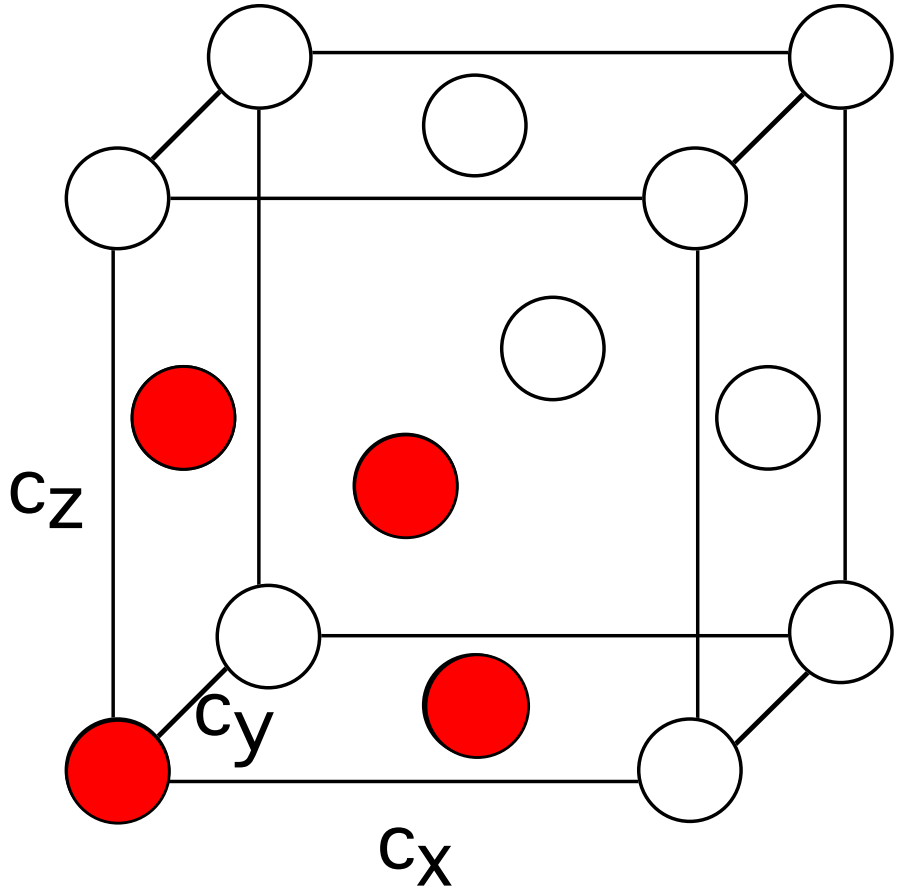
Normalized form

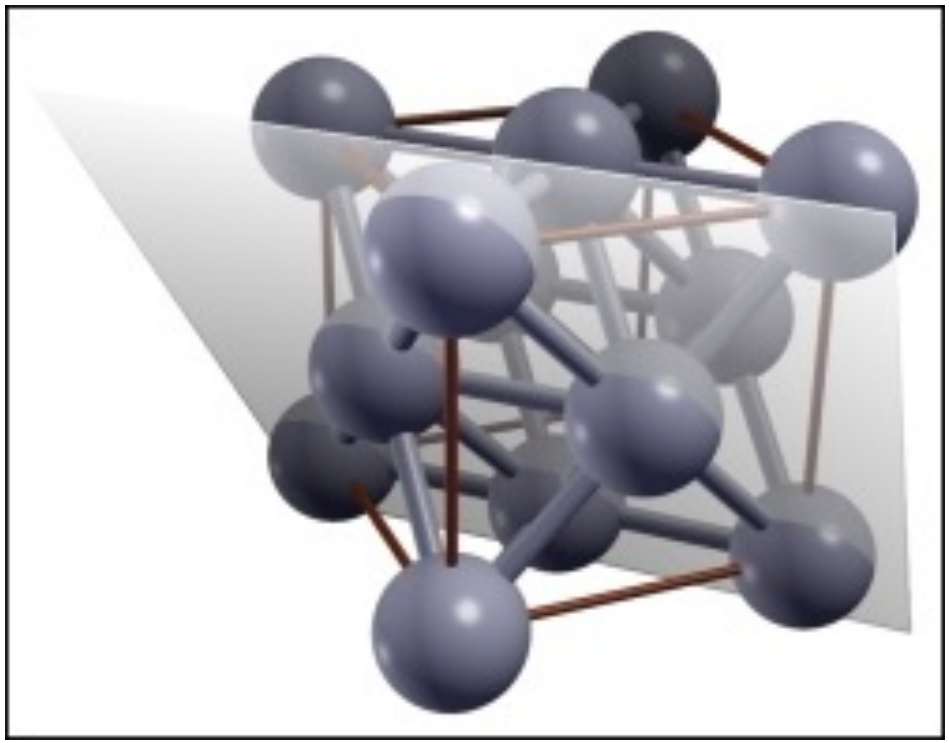
$$\frac{T}{\varepsilon/k_B} = \frac{1}{3N} \sum_{i=0}^{N-1} \left|\dot{\vec{r}}_i{}'\right|^2$$

```
double temperature;
```

Initial Configuration

- **Face centered cubic (FCC) lattice: 4 atoms per unit cell, c^3**
- **Density** $\rho = 4/c^3$





`double gap[3]`: (c_x, c_y, c_z), the edge lengths of the unit cell
`double Density`: Input parameter, input
`int InitUcell[3]`: # of unit cells in the x, y, and z directions, input
nAtom = 4×`InitUcell[0]`×`InitUcell[1]`×`InitUcell[2]`

Initial Velocities

- **Generate random velocities of magnitude, v_0**

$$\frac{v_0^2}{3} = T_{init} \Rightarrow v_0 = \sqrt{3T_{init}}$$

- **Random velocity**

$$\vec{v}_i = v_0(\xi_0, \xi_1, \xi_2) = v_0\vec{\xi}$$

where the randomly oriented vector of unit length is $\vec{\xi}$

- **Algorithm**

Let rand() be a uniform random-number generator in the range [0, 1]

1. $\zeta_i = 2\times\text{rand}()-1$ $(i = 0, 1)$ so that $-1 \le \zeta_i < 1$
2. $s^2 = \zeta_0^2 + \zeta_1^2$
3. if $s^2 < 1$, accept $(\xi_0, \xi_1, \xi_2) = \left(2\sqrt{1-s^2}\zeta_0, 2\sqrt{1-s^2}\zeta_1, 1-2s^2\right)$
4. else reject and go to 1

`double InitTemp`: Initial temperature, input

MD Input Parameters

md.in

```
3 3 3
0.8
1.0
0.005
10000
10
```

InitUcell[3]

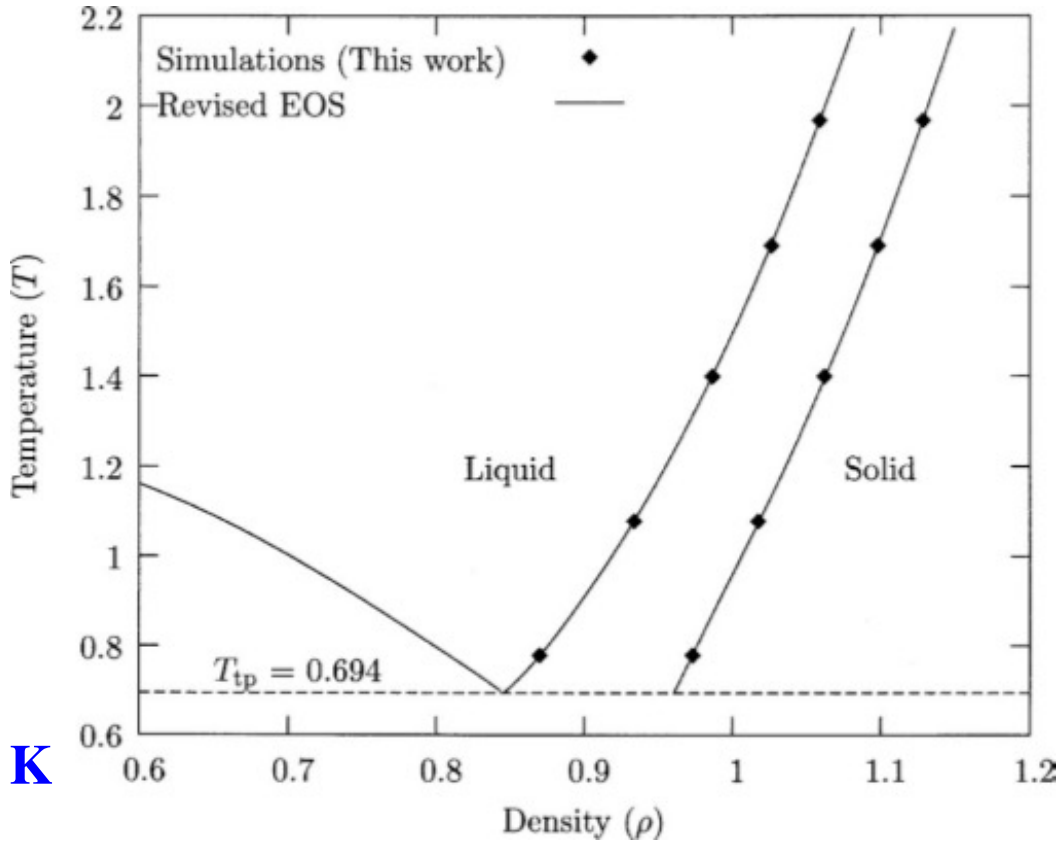
Density in 0.0254 Å^{-3} for Ar

InitTemp

DeltaT

StepLimit

StepAvg



in 120 K

in 2.2 ps

Compare runtime of `md.c` for `InitUcell[] = {3,3,3}` *vs.* `{10,10,10}`

108 atoms 4000 atoms

Recap: MD Computation

$\{\vec{a}_i | i = 0, \cdots, N-1\} \leftarrow ComputeAccel(\{\vec{r}_i\})$

Repeat *StepLimit* times

1. $\vec{v}_i \leftarrow \vec{v}_i + \frac{\Delta}{2}\vec{a}_i$ → *DeltaT*
2. $\vec{r}_i \leftarrow \vec{r}_i + \vec{v}_i\Delta$
3. $\{\vec{a}_i\} \leftarrow ComputeAccel(\{\vec{r}_i\})$
4. $\vec{v}_i \leftarrow \vec{v}_i + \frac{\Delta}{2}\vec{a}_i$

***ComputeAccel*(): *r*[][] → *ra*[][]**

```
for (i=0; i<N; i++)
  for (k=0; k<3; k++) ra[i][k] = 0.0;
for (i=0; i<N-1; i++) {
  for (j=i+1; j<N; j++) {
    ...
    for (k=0; k<3; k++) {
      ...
      ra[i][k] = ra[i][k] + fk;
      ra[j][k] = ra[j][k] - fk;
    }
  }
}
```

$O(N^2)$ floating-point operations

MD bottom line: Keep moving atoms ← kicked by neighbor atoms

Linked-List Cell MD

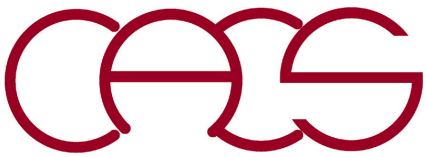
Aiichiro Nakano

Collaboratory for Advanced Computing & Simulations
Department of Computer Science
Department of Physics & Astronomy
Department of Quantitative & Computational Biology
University of Southern California

Email: anakano@usc.edu

Objectives:

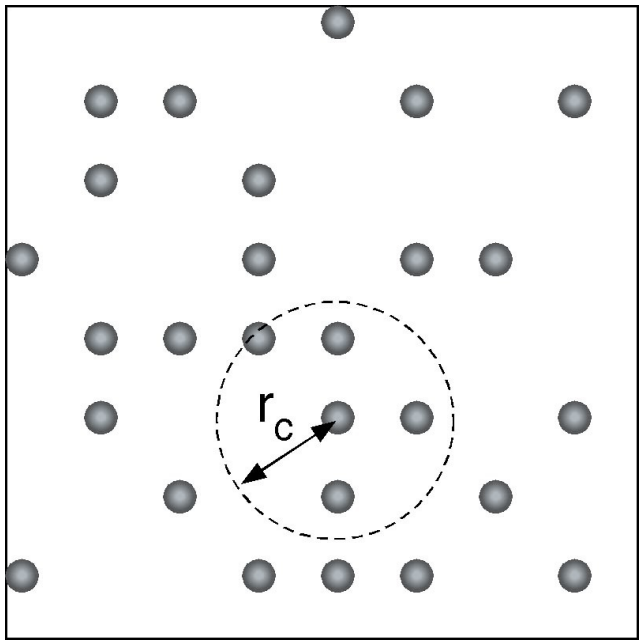
1. **Reduced complexity, $O(N^2) \rightarrow O(N)$, for large problems**
2. **Migration path to parallel MD — "data locality"!**



See program lmd.c

Linked-List Cell Molecular Dynamics

- **Computational complexity of ComputeAccel() in md.c: $\propto N(N-1)/2 = O(N^2)$**
- **Data locality (cut-off length, r_c) reduces the complexity to $O(N)$: $N\times(4\pi/3)r_c^3\times(N/V)$**
- **$O(N)$ algorithm uses: (1) spatially localized cells; & (2) linked lists to keep track of atoms' cell membership**




```
for (j1=0; j1<nAtom-1; j1++) {
 for (j2=j1+1; j2<nAtom; j2++) {
  for (rr=0.0, k=0; k<3; k++) {
   dr[k] = r[j1][k]-r[j2][k];
   dr[k] = dr[k]-SignR(RegionH[k],dr[k]-RegionH[k])
                -SignR(RegionH[k],dr[k]+RegionH[k]);
   rr = rr+dr[k]*dr[k];
  }
  if (rr < rrCut) {
   ri2 = 1.0/rr; ri6 = ri2*ri2*ri2; r1 = sqrt(rr);
   fcVal = 48.0*ri2*ri6*(ri6-0.5)+Duc/r1;
   for (k=0; k<3; k++) {
    f = fcVal*dr[k];
    ra[j1][k] = ra[j1][k]+f;
    ra[j2][k] = ra[j2][k]-f;
   }
  }
 }
}
```

$nAtom = N$

$rrcut = r_c$

DIY: Compare the runtime of md.c & lmd.c!

Cell Data Structures

- **Cell size**
  $L_{c\alpha} = \lfloor L_\alpha / r_c \rfloor$ (α = x, y, z) (`int lc[3]`)
  $r_{c\alpha} = L_\alpha / L_{c\alpha}$ (`double rc[3]`) Cell size

  How many cutoff lengths fit in the box? (note the floor operation)
  This guarantees that all neighbor atoms reside in the nearest-neighbor cells

  where

  L_α: simulation box length
  (`double Region[3]`)
  r_c: Cut-off length (`RCUT`)

- **Vector cell index, $0 \le c_\alpha \le L_{c\alpha} - 1$**

- **Serial cell index:**

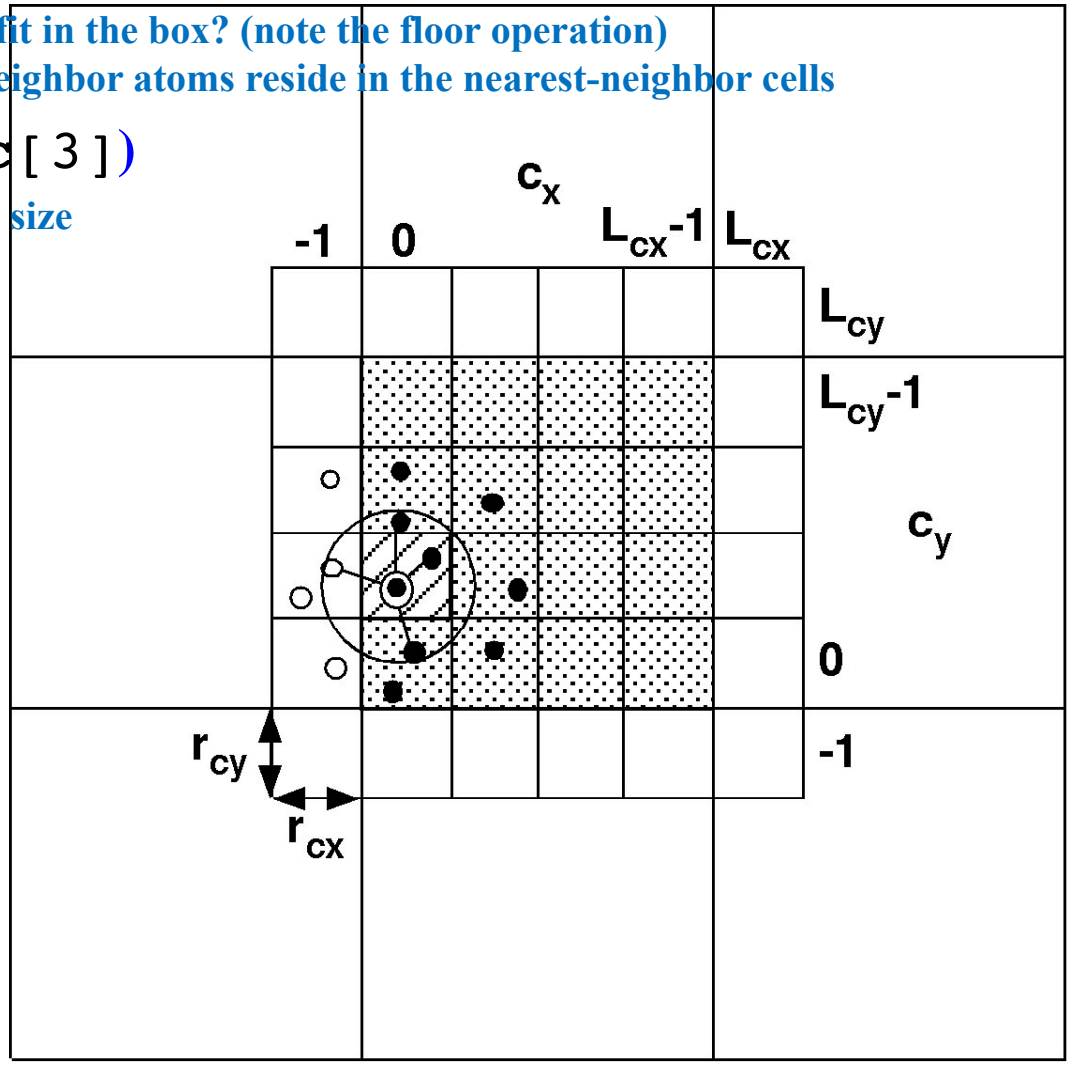
$$c = c_x L_{cy} L_{cz} + c_y L_{cz} + c_z$$

  or

$$c_x = \lfloor c/(L_{cy} L_{cz}) \rfloor$$
$$c_y = \lfloor c/L_{cz} \rfloor \bmod L_{cy}$$
$$c_z = c \bmod L_{cz}$$

- **Atom-to-cell mapping:**

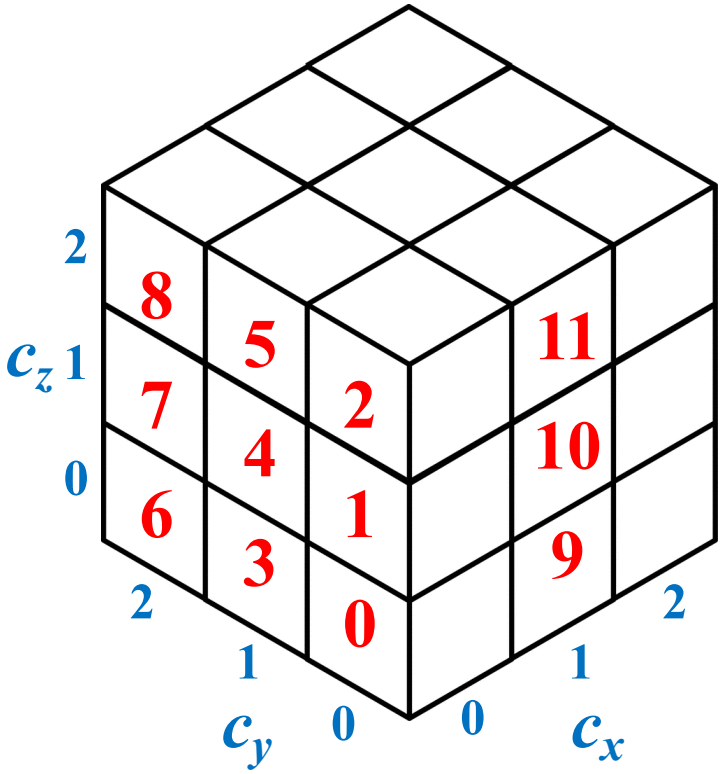
$$c_\alpha = \lfloor r_\alpha / r_{c\alpha} \rfloor$$



$2\times$  $+ 3\times$  $+ 4\times$ 

$= 234¢$

Cell Index Example



How many cells are before me?

$$c = c_x L_{cy} L_{cz} + c_y L_{cz} + c_z$$

or

$$c_x = \lfloor c/(L_{cy} L_{cz}) \rfloor$$
$$c_y = \lfloor c/L_{cz} \rfloor \bmod L_{cy}$$
$$c_z = c \bmod L_{cz}$$

Which column, row, & height?

$$L_{cx} = L_{cy} = L_{cz} = 3$$

c	cx	cy	cz
0	0	0	0
1	0	0	1
2	0	0	2
3	0	1	0
4	0	1	1
5	0	1	2
6	0	2	0
7	0	2	1
8	0	2	2
9	1	0	0
10	1	0	1
11	1	0	2
...			

reset

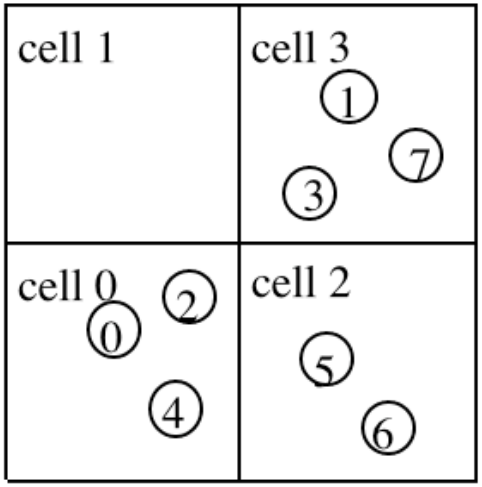
slow ← fast

Linked Lists

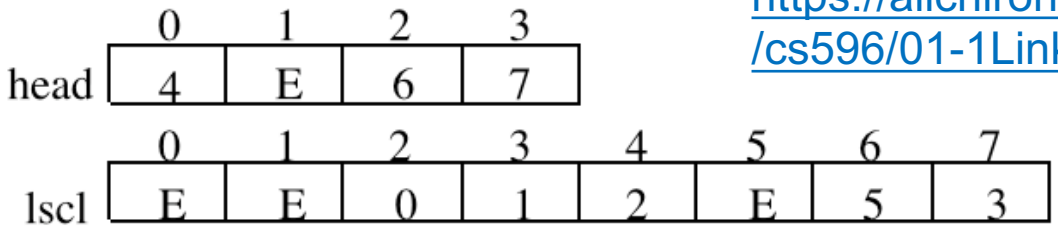
Data Structures:

`lscl[NMAX]`: Linked lists; `lscl[i]` holds the atom index to which the *i*-th atom points.

`head[NCLMAX]`: `head[c]` holds the index of the first atom in the *c*-th cell, or `head[c]` = `EMPTY` (= –1) if there is no atom in the cell.



For details, see the lecture note on "linked-list cell MD algorithm"
https://aiichironakano.github.io/cs596/01-1LinkedListCell.pdf



Example of cell 3

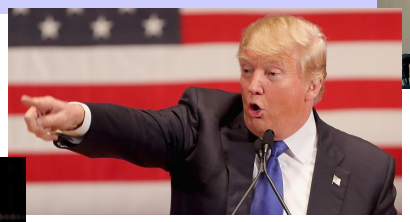
Linked List Construction Algorithm

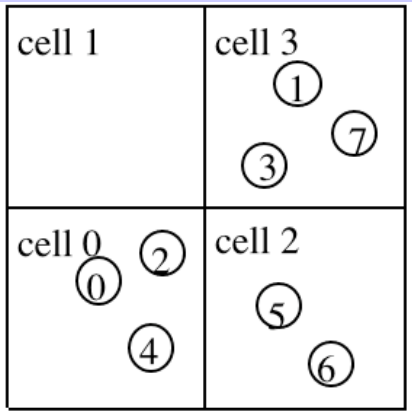
```
/* Reset the headers, head */
for (c=0; c<lcxyz; c++) head[c] = EMPTY;
/* Scan atoms to construct headers, head, & linked lists, lscl */
for (i=0; i<nAtom; i++) {
  /* Vector cell index to which this atom belongs */
  for (a=0; a<3; a++) mc[a] = r[i][a]/rc[a];
  /* Translate the vector cell index, mc, to a scalar cell index */
  c = mc[0]*lcyz+mc[1]*lc[2]+mc[2];
  /* Link to the previous occupant (or EMPTY if you're the 1st) */
  lscl[i] = head[c];
  /* The last one goes to the header */
  head[c] = i;
}

where
  lcyz = lc[1]*lc[2]
  lcxyz = lcyz*lc[0]
```

$O(N)$

$c_\alpha = \lfloor r_\alpha / r_{c\alpha} \rfloor$

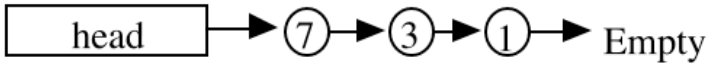
$c = c_x L_{cy} L_{cz} + c_y L_{cz} + c_z$





	0	1	2	3
head	4	E	6	7

	0	1	2	3	4	5	6	7
lscl	E	E	0	1	2	E	5	3



$O(N)$ Force Calculation Algorithm

```
for (mc[0]=0; mc[0]<lc[0]; (mc[0])++)
for (mc[1]=0; mc[1]<lc[1]; (mc[1])++)
for (mc[2]=0; mc[2]<lc[2]; (mc[2])++) { /* Scan all cells */
  c = mc[0]*lcyz+mc[1]*lc[2]+mc[2]; /* Calculate a scalar cell index */
  for (mc1[0]=mc[0]-1; mc1[0]<=mc[0]+1; (mc1[0])++)
  for (mc1[1]=mc[1]-1; mc1[1]<=mc[1]+1; (mc1[1])++)
  for (mc1[2]=mc[2]-1; mc1[2]<=mc[2]+1; (mc1[2])++) { /* Scan neighbor cells */
    for (a=0; a<3; a++) {  /* Unwrapping the periodic boundary condition */
      if (mc1[a] < 0)
        rshift[a] = -Region[a];
      else if (mc1[a]>=lc[a])
        rshift[a] = Region[a];
      else
        rshift[a] = 0.0;
    }
    c1 = ((mc1[0]+lc[0])%lc[0])*lcyz
       +((mc1[1]+lc[1])%lc[1])*lc[2]
       +((mc1[2]+lc[2])%lc[2]); /* Scalar cell index of the neighbor cell */
    i = head[c]; /* Scan atom i in cell c */
    while (i != EMPTY) {
      j = head[c1]; /* Scan atom j in cell c1 */
      while (j != EMPTY) {
        if (i < j) { /* Avoid double counting of pair (i, j) */
          r_ij = r_i-(r_j+r_shift); /* Image-corrected relative pair position */
          if (r_ij < r_c)
            Compute forces on pair (i, j)
        }
        j = lscl[j];
      }
      i = lscl[i];
    }
  }
}
```

$O(N)$

$3^3 = 27$

Taking care of periodic boundary condition

Complexity of Linked-List Cell MD

Learn computational complexity analysis by runtime measurement

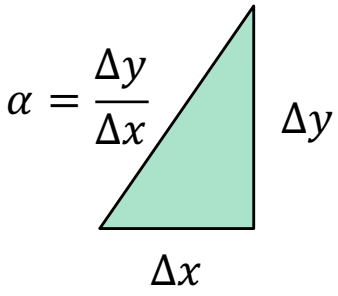
See lecture note on asymptotic analysis of functions

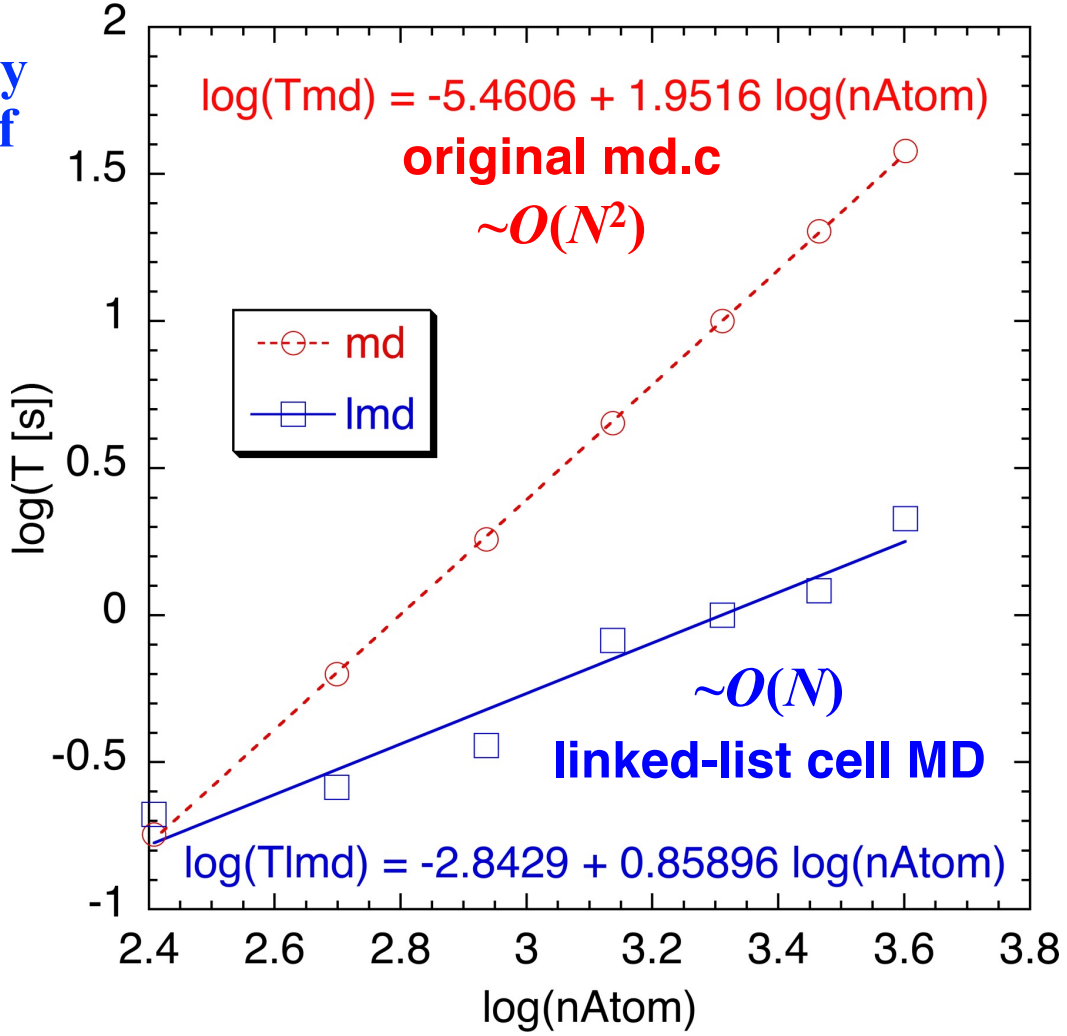
- **Power of polynomial complexity can be estimated by the slope of linear fit in log-log plot of runtime *T vs.* problem size *N***

$$T = cN^{\alpha}$$

$$\underbrace{\log T}_{y} = \underbrace{\log c}_{\beta} + \alpha \underbrace{\log N}_{x}$$

$$\alpha = \frac{\Delta y}{\Delta x}$$

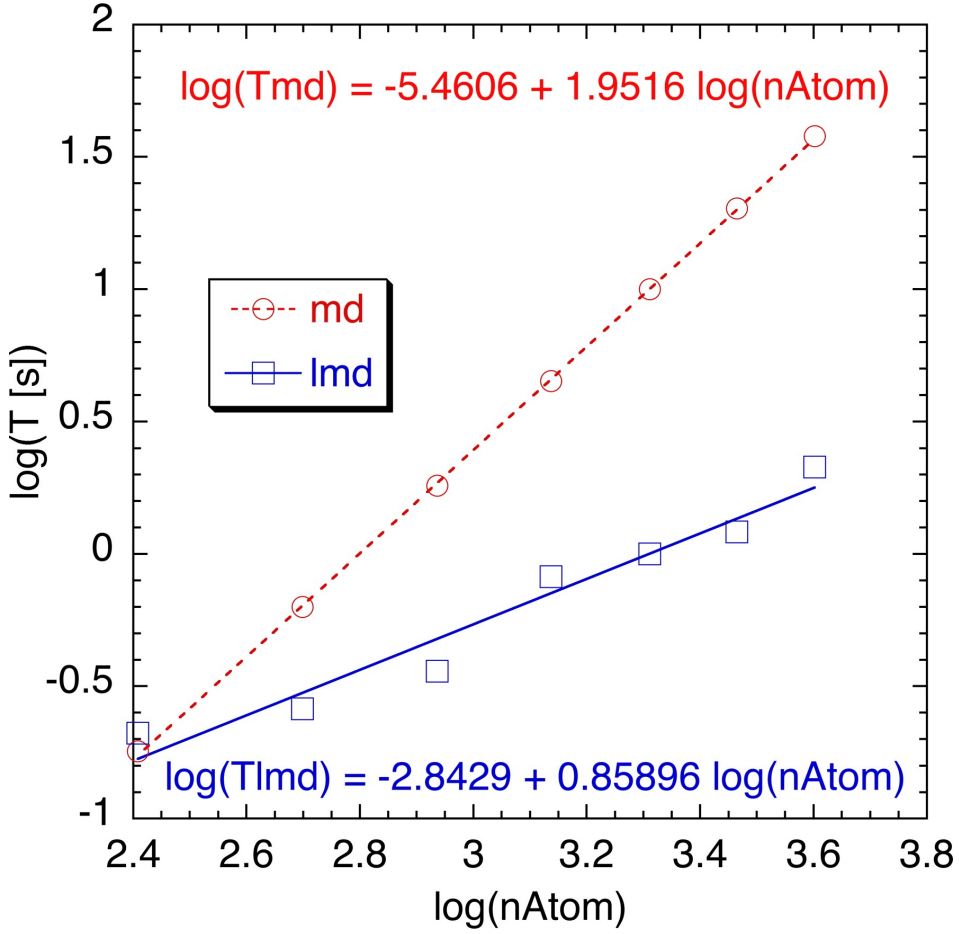




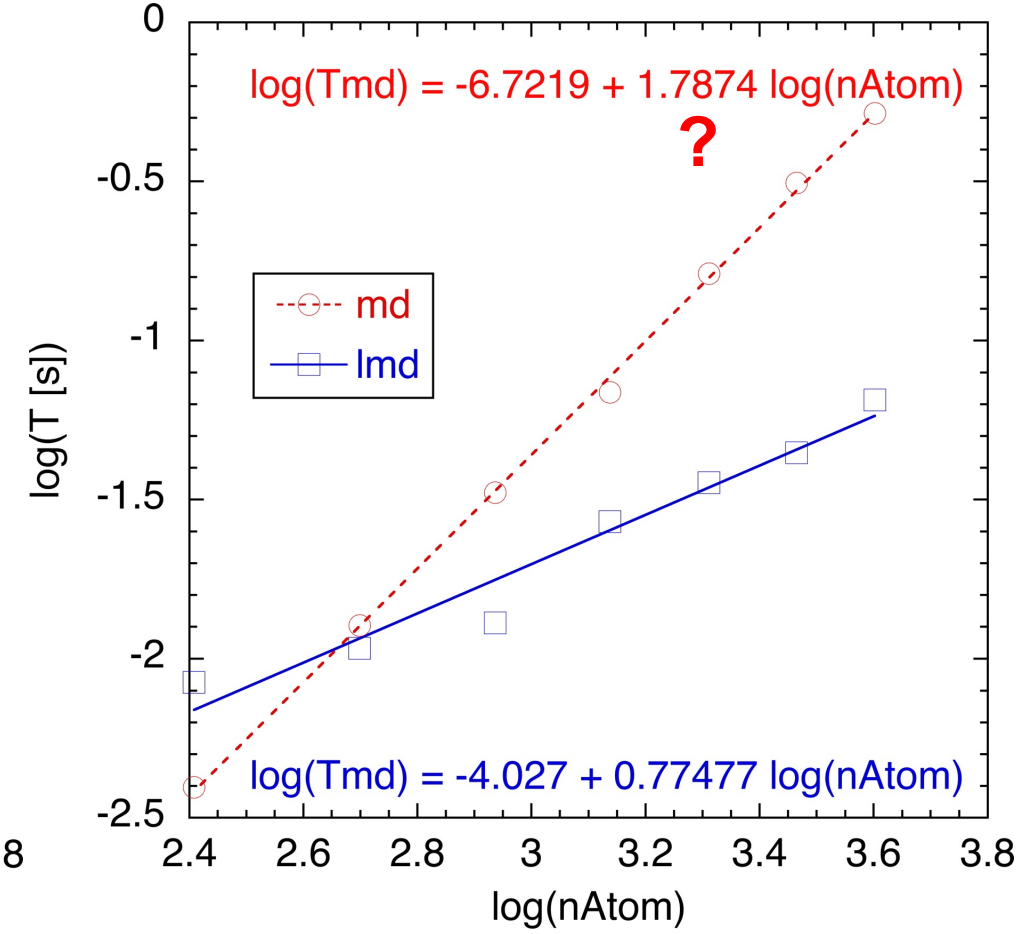
See lecture note on Least square fit of a line

Machine Dependence?

2.53 GHz Intel Core2



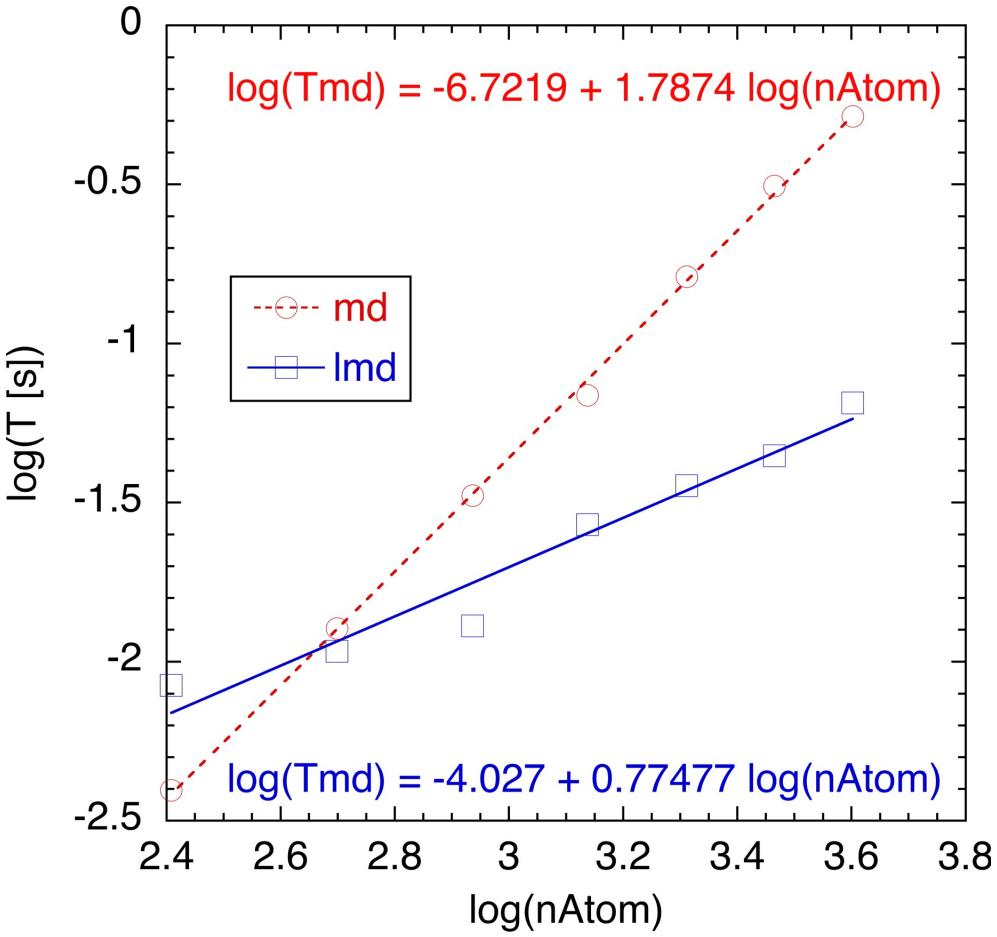
2.7 GHz Intel Core i7

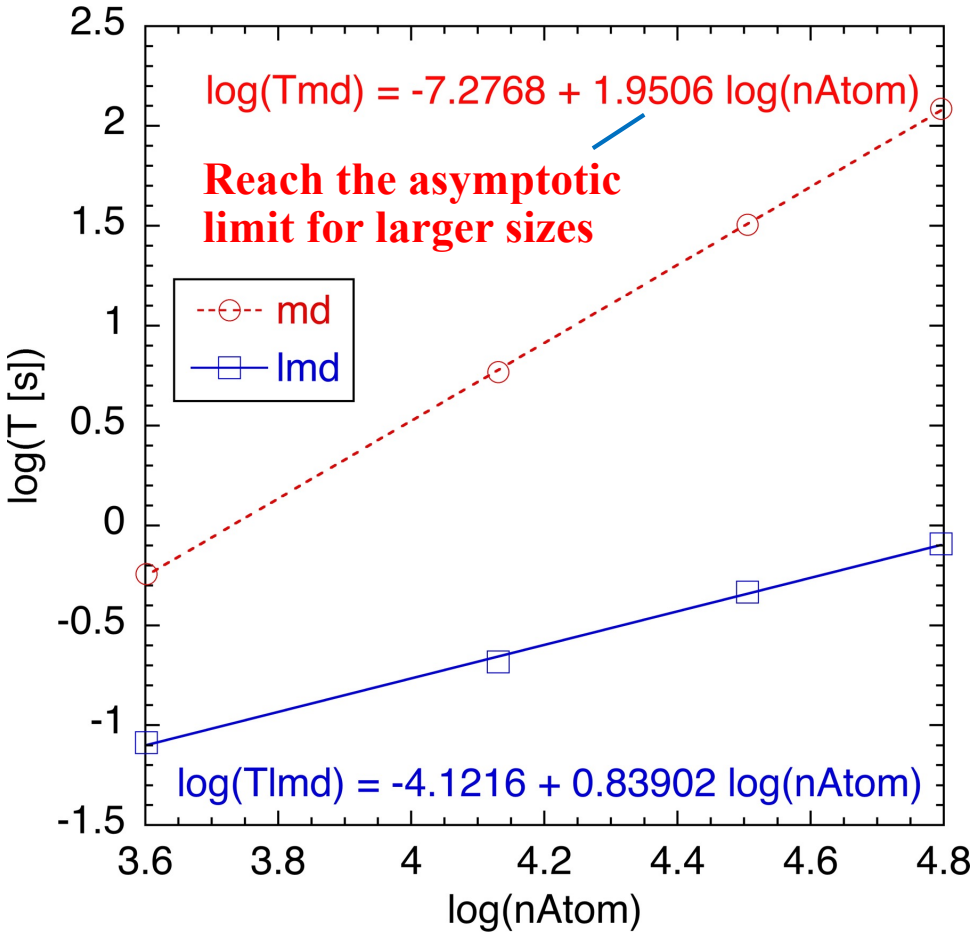


nAtom = 256, ..., 4000

Finite-Size Effect?

2.7 GHz Intel Core i7

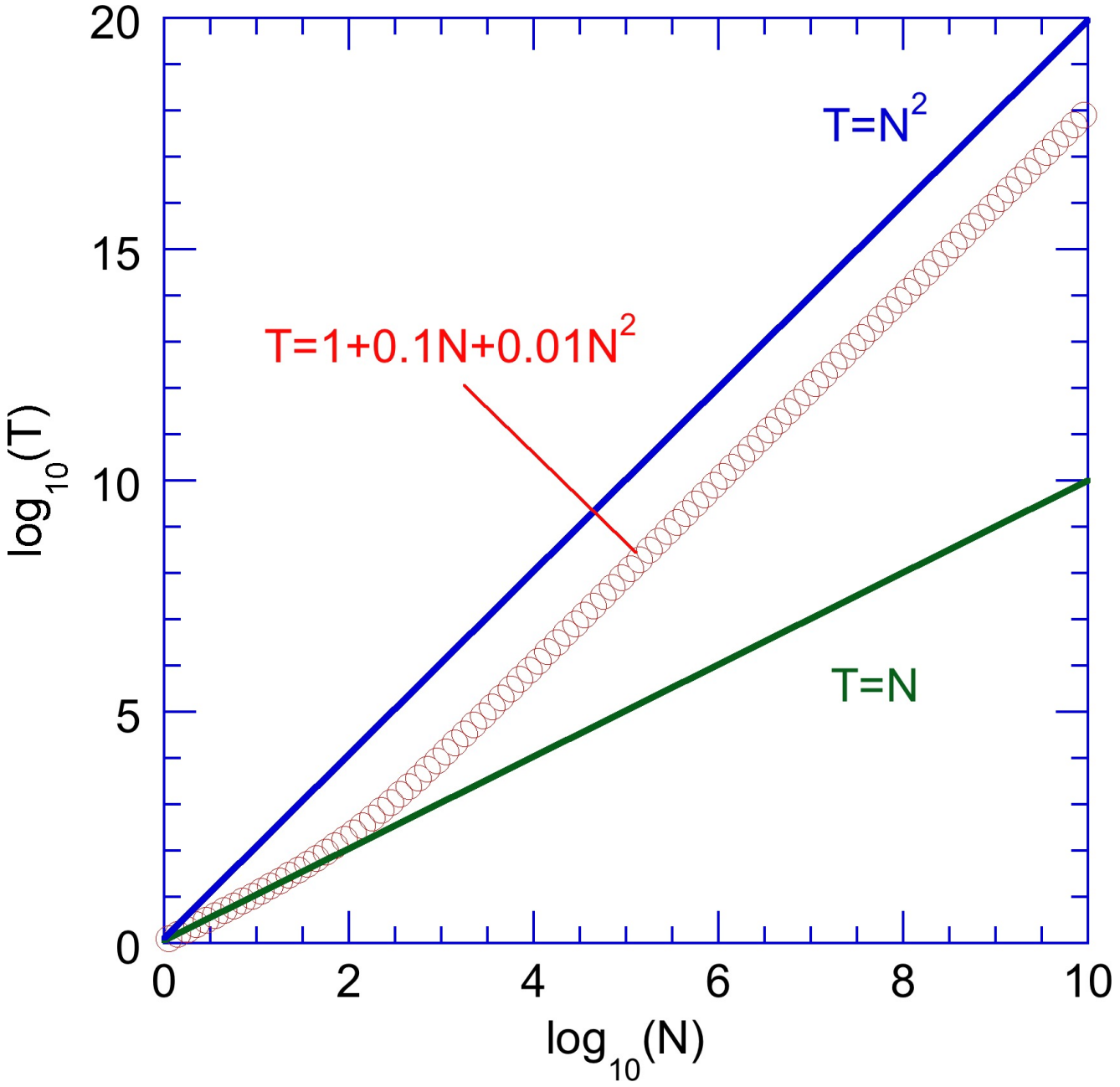




nAtom = 256, ..., 4000

nAtom = 4000, ..., 62500

Asymptotic Complexity?



Flop/s Performance of lmd.c

Flop/s = floating-point operations/second

M (mega)	$= 10^6$
G (giga)	$= 10^9$
T (Tera)	$= 10^{12}$
P (Peta)	$= 10^{15}$
X (Exa)	$= 10^{18}$

Program: lmd_sqrt_flop.c
Compile option: -O (default optimization)

```
cc -O –o lmd_sqrt_flop lmd_sqrt_flop.c -lm
```

```
===== Discovery cluster =====
Processor: 2.1 GHz Intel Xeon
Execution time (s) = 1.500000e-01
Number of FP operations = 1.609487e+08
MFlops rate = 1.072992e+03 (1.07 Gflop/s)

===== MacBook Pro =====
Processor: 2.3 GHz Intel Core i9
Execution time (s) = 6.417400e-02
Number of FP operations = 1.609487e+08
MFlops rate = 2.508005e+03 (2.51 Gflop/s)
```

Where to Go from Here

- **Mathematical details skipped in the slides can be found in the lecture notes:**

  https://aiichironakano.github.io/cs596/01MD.pdf

  https://aiichironakano.github.io/cs596/01-1LinkedListCell.pdf

- **Practical application of molecular dynamics simulations in materials science is covered by the following courses:**

  MASC 575: ***Basics of Atomistic Simulation of Materials***

  MASC 576: ***Molecular Dynamics Simulations of Materials & Processes***