

Molecular Dynamics

Aiichiro Nakano

*Collaboratory for Advanced Computing & Simulations
Department of Computer Science*

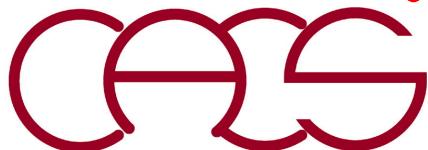
Department of Physics & Astronomy

*Department of Quantitative & Computational Biology
University of Southern California*

Email: anakano@usc.edu

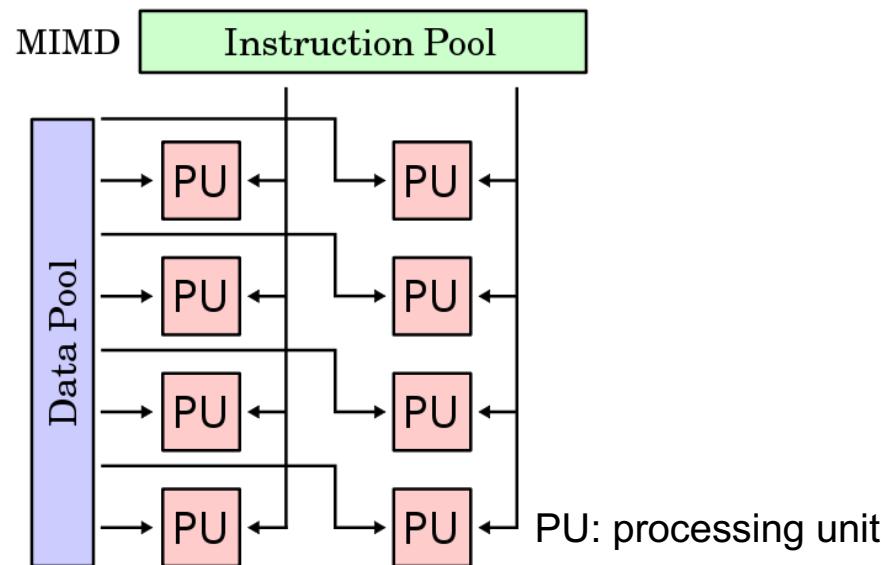
Objective: Basics of particle simulation

- Calculus (math) → simulation (science)
- Minimal knowledge to understand pmd.c



What to Learn with MD

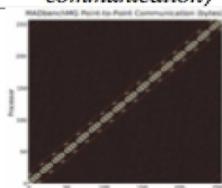
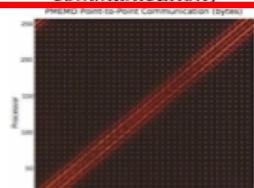
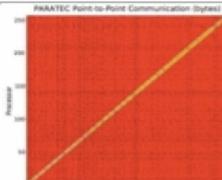
- An archetype of multiple-instruction multiple-data (MIMD) parallel applications: Discrete particle dynamics based on ordinary differential equation (ODE); interaction
- Data locality & scalable data structure: Linked-list cells
- Multiresolution algorithm: Divide-&-conquer



MIMD: Multiple autonomous processors concurrently execute different instructions on different data

Why MD: Dynamic Irregular Dwarf

HPC 7 dwarfs

Dwarf	Description	Communication Pattern (Figure axes show processors 1 to 256, with black meaning no communication)	NAS Benchmark / Example HW	Dwarf	Description	Communication Pattern (Figure axes show processors 1 to 256, with black meaning no communication)	NAS Benchmark / Example HW
1. Dense Linear Algebra (e.g., BLAS [Blackford et al 2002], ScaLAPACK [Blackford et al 1996], or MATLAB [MathWorks 2006])	Data are dense matrices or vectors. (BLAS Level 1 = vector-vector; Level 2 = matrix-vector; and Level 3 = matrix-matrix.) Generally, such applications use unit-stride memory accesses to read data from rows, and strided accesses to read data from columns.	 The communication pattern of MadBench, which makes heavy use of ScaLAPACK for parallel dense linear algebra, is typical of a much broader class of numerical algorithms	Block Triadiagonal Matrix, Lower Upper Symmetric Gauss-Seidel / Vector computers, Array computers	4. N-Body Methods (e.g., Barnes-Hut [Barnes and Hut 1986], Fast Multipole Method [Greengard and Rokhlin 1987])	Depends on interactions between many discrete points. Variations include particle-particle methods, where every point depends on all others, leading to an $O(N^3)$ calculation, and hierarchical particle methods, which combine forces or potentials from multiple points to reduce the computational complexity to $O(N \log N)$ or $O(N)$.	 PMEMD's communication pattern is that of a particle mesh Ewald calculation	(no benchmark) / GRAPE [Tokyo 2006], MD-GRAPE [IBM 2006]
2. Sparse Linear Algebra (e.g., SpMV, OSKI [OSKI 2006], or SuperLU [Demmel et al 1999])	Data sets include many zero values. Data is usually stored in compressed matrices to reduce the storage and bandwidth requirements to access all of the nonzero values. One example is block compressed sparse row (BCSR). Because of the compressed formats, data is generally accessed with indexed loads and stores.	 SuperLU (communication pattern pictured above) uses the BCSR method for implementing sparse LU factorization.	Conjugate Gradient / Vector computers with gather/scatter	5. Structured Grids (e.g., Cactus [Goodale et al 2003] or Lattice-Boltzmann Magnetohydrodynamics [LBMHD 2005])	Represented by a regular grid; points on grid are conceptually updated together. It has high spatial locality. Updates may be in place or between 2 versions of the grid. The grid may be subdivided into finer grids in areas of interest ("Adaptive Mesh Refinement"); and the transition between granularities may happen dynamically.	 Communication pattern for Cactus, a PDE solver using 7-point stencil on 3D block-structured grids.	Multi-Grid, Scalar Penta-diagonal / QCDOC [Edinburg 2006], BlueGeneL
3. Spectral Methods (e.g., FFT [Cooley and Tukey 1965])	Data are in the frequency domain, as opposed to time or spatial domains. Typically, spectral methods use multiple butterfly stages, which combine multiply-add operations and a specific pattern of data permutation, with all-to-all communication for some stages and strictly local for others.	 PARATEC: The 3D FFT requires an all-to-all communication to implement a 3D transpose, which requires communication between every link. The diagonal stripe describes BLAS-3 dominated linear-algebra step required for orthogonalization.	Fourier Transform / DSPs, Zalink PDSP [Zarlink 2006] or FIDAP [FLUENT 2006]	6. Unstructured Grids (e.g., ABAQUS [ABAQUS 2006] or FIDAP [FLUENT 2006])	An irregular grid where data locations are selected, usually by underlying characteristics of the application. Data point location and connectivity of neighboring points must be explicit. The points on the grid are conceptually updated together. Updates typically involve multiple levels of memory reference indirection, as an update to any point requires first determining a list of neighboring points, and then loading values from those neighboring points.		Unstructured Adaptive / Vector computers with gather/scatter, Tera Multi Threaded Architecture [Berry et al 2006]
				7. Monte Carlo (e.g., Quantum Monte Carlo [Aspuru-Guzik et al 2005])	Calculations depend on statistical results of repeated random trials. Considered embarrassingly parallel.	Communication is typically not dominant in Monte Carlo methods.	Embarrassingly Parallel / NSF Teragrid

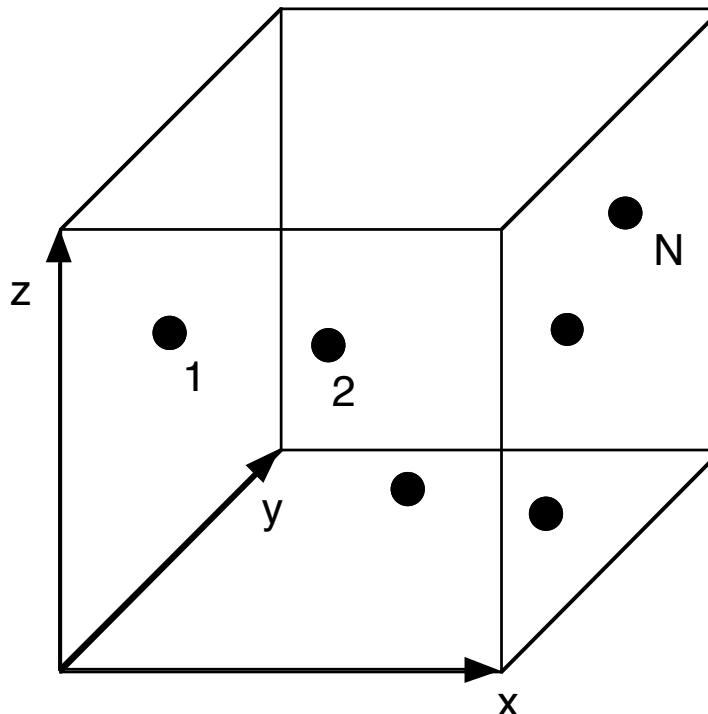
System: A Set of Point Atoms

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

int nAtom: N , # of atoms.

NMAX: Max # of atoms.

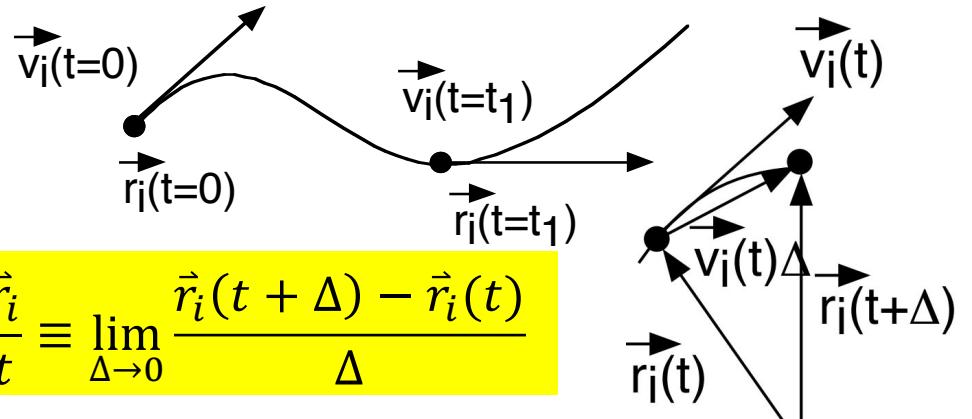
double r[NMAX][3]: $r[i][0|1|2] = x_i|y_i|z_i$.



See [pmd.h](#) on the class code page
<https://aiichironakano.github.io/cs653/src/parMD>

Key Concept: Trajectory via Snapshots

Trace of atom positions



Velocity

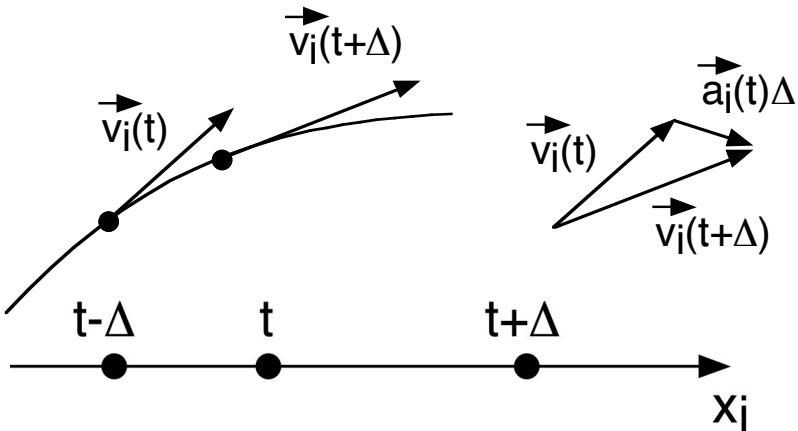
$$\vec{v}_i(t) = \dot{\vec{r}}_i(t) = \frac{d\vec{r}_i}{dt} \equiv \lim_{\Delta \rightarrow 0} \frac{\vec{r}_i(t + \Delta) - \vec{r}_i(t)}{\Delta}$$

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

Acceleration

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

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



$$\begin{aligned} \vec{a}_i &= \lim_{\Delta \rightarrow 0} \frac{\vec{v}_i(t + \Delta/2) - \vec{v}_i(t - \Delta/2)}{\Delta} \\ &= \lim_{\Delta \rightarrow 0} \frac{\frac{\vec{r}_i(t + \Delta) - \vec{r}_i(t)}{\Delta} - \frac{\vec{r}_i(t) - \vec{r}_i(t - \Delta)}{\Delta}}{\Delta} \\ &= \lim_{\Delta \rightarrow 0} \frac{\vec{r}_i(t + \Delta) - 2\vec{r}_i(t) + \vec{r}_i(t - \Delta)}{\Delta^2} \end{aligned}$$

Newton's Equation of Motion



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

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

Pair potential:

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

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

Example:

Lennard-Jones potential (normalized)

Molecular Dynamics Problem

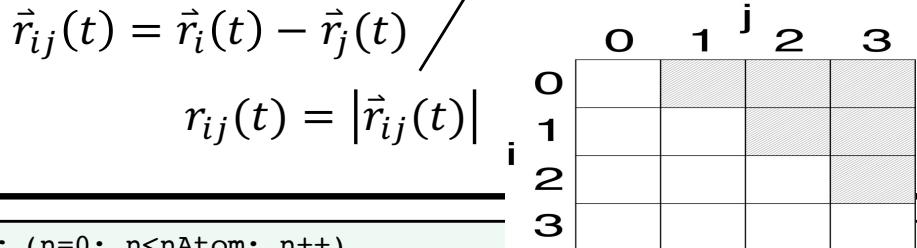
Given initial atomic positions & velocities, $\{(\vec{r}_i(0), \vec{v}_i(0)) | i = 0, \dots, N - 1\}$,
 obtain those at later times, $\{(\vec{r}_i(t), \vec{v}_i(t)) | i = 0, \dots, N - 1; t > 0\}$,
 by integrating the ordinary differential equation,

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

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

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;
                }
            }
        }
    }
}

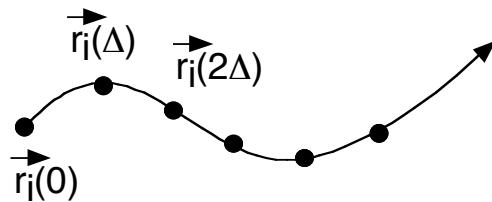
```

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

Time Discretization

Snapshots with time interval Δ : double DeltaT

$$(\vec{r}_i(0), \vec{v}_i(0)) \mapsto (\vec{r}_i(\Delta), \vec{v}_i(\Delta)) \mapsto (\vec{r}_i(2\Delta), \vec{v}_i(2\Delta)) \mapsto \dots$$



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

Solution: Taylor expansion

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

cf. <https://aiichironakano.github.io/phys516/TaylorExpansion.pdf>

Verlet Discretization

Let's predict the next step using Taylor expansion

Position:

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

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

Velocity:

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

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

Verlet Algorithm

Verlet discretization:

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



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. $\vec{v}_i(t) \leftarrow [\vec{r}_i(t + \Delta) - \vec{r}_i(t - \Delta)]/2\Delta$

Loup Verlet

in out
ComputeAccel(): r[][] → ra[][]

```
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

Solution: 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.

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

For proof, see <https://aiichironakano.github.io/phys516/02MD.pdf>

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

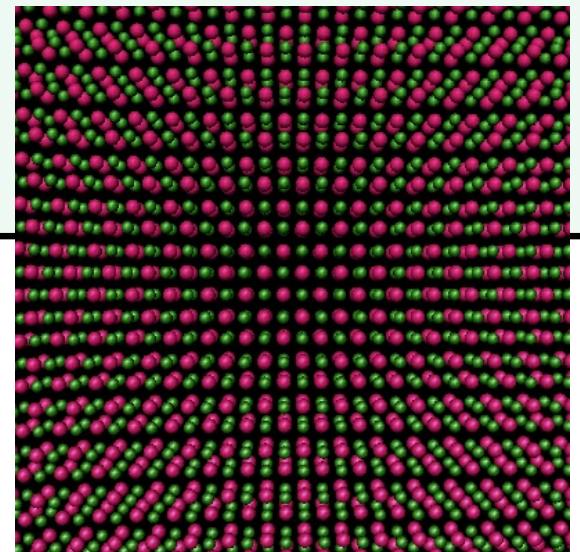
$\vec{r}_i \leftarrow \vec{r}_i + \vec{v}_i \Delta$ for all i

 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

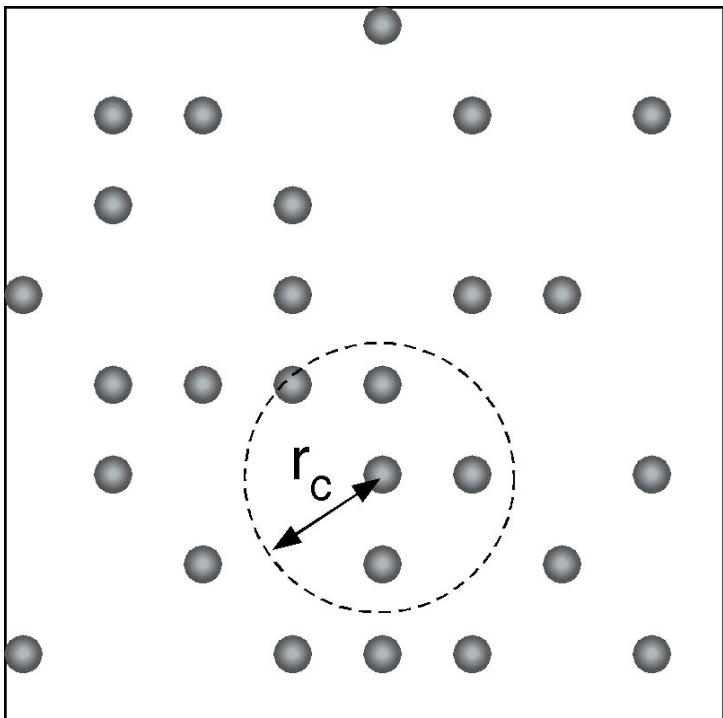
stepLimit+1 calls to function ComputeAccel()



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

Data-local migration path to parallel MD



Prune the search space!

Cell Data Structures

Data-local spatial indexing scheme = cell + linked list

- Cell size

$$L_{c\alpha} = \lfloor L_\alpha / r_c \rfloor \text{ (int lc[3])}$$

$$r_{c\alpha} = L_\alpha / L_{c\alpha} \text{ (\alpha = x, y, z) (double rc[3])}$$

where

L_α : simulation box length

(double Region[3])

r_c : Cut-off length (RCUT)

- Vector cell index, $0 \leq c_\alpha \leq L_{cx}-1$

- Serial cell index:

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

or

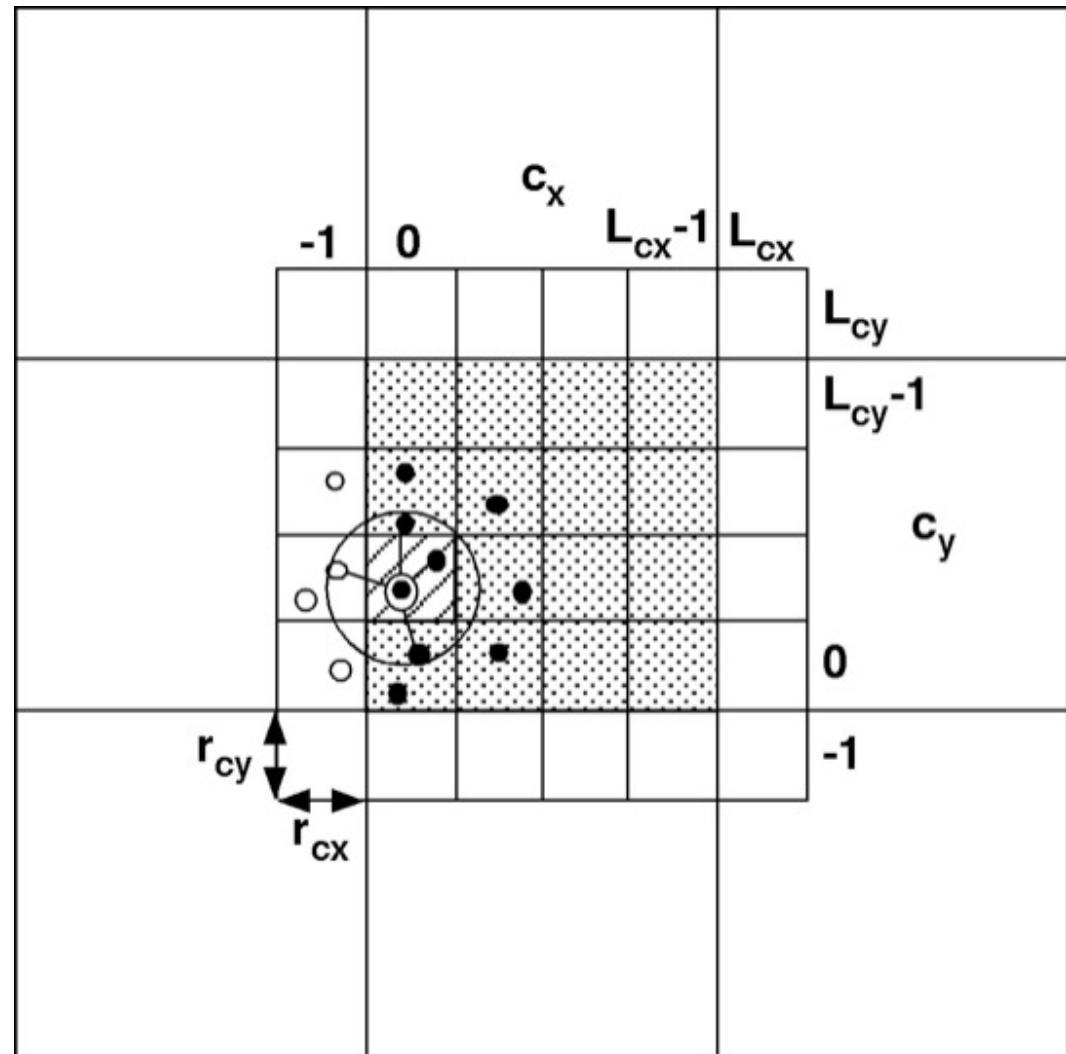
$$c_x = \lfloor c / (L_{cy} L_{cz}) \rfloor$$

$$c_y = \lfloor c / L_{cz} \rfloor \bmod L_{cy}$$

$$c_z = c \bmod L_{cz}$$

- Atom-to-cell mapping:

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

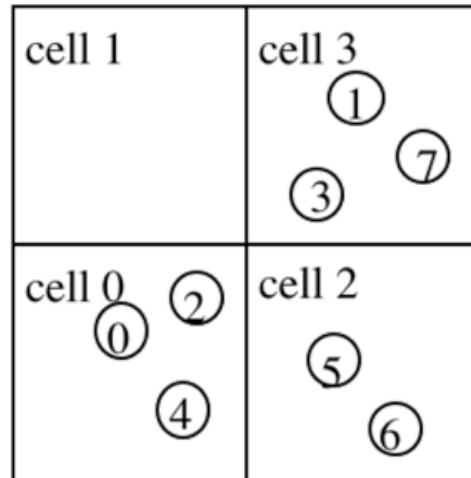


Linked Lists

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.



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

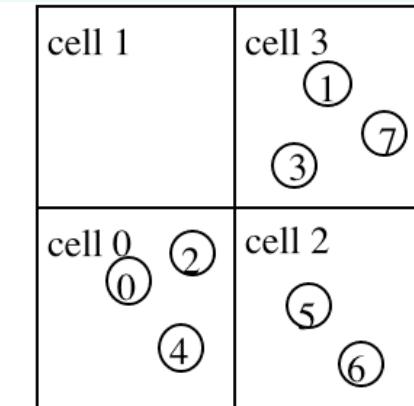
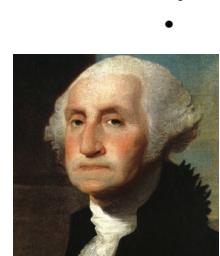


Linked List Construction Algorithm

```
/* Reset the headers, head */
for (c=0; c<lxyz; c++) head[c] = EMPTY;
/* Scan atoms to construct headers, head, & linked lists, lscl */
for (i=0; i<nAtom; i++) {O(N)
    /* 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;
}
```

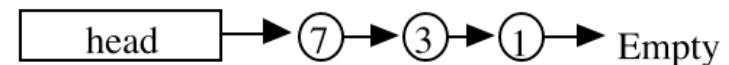
where

$$\begin{aligned} lcyz &= lc[1]*lc[2] \\ lxyz &= lcyz*lc[0] \end{aligned}$$



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

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

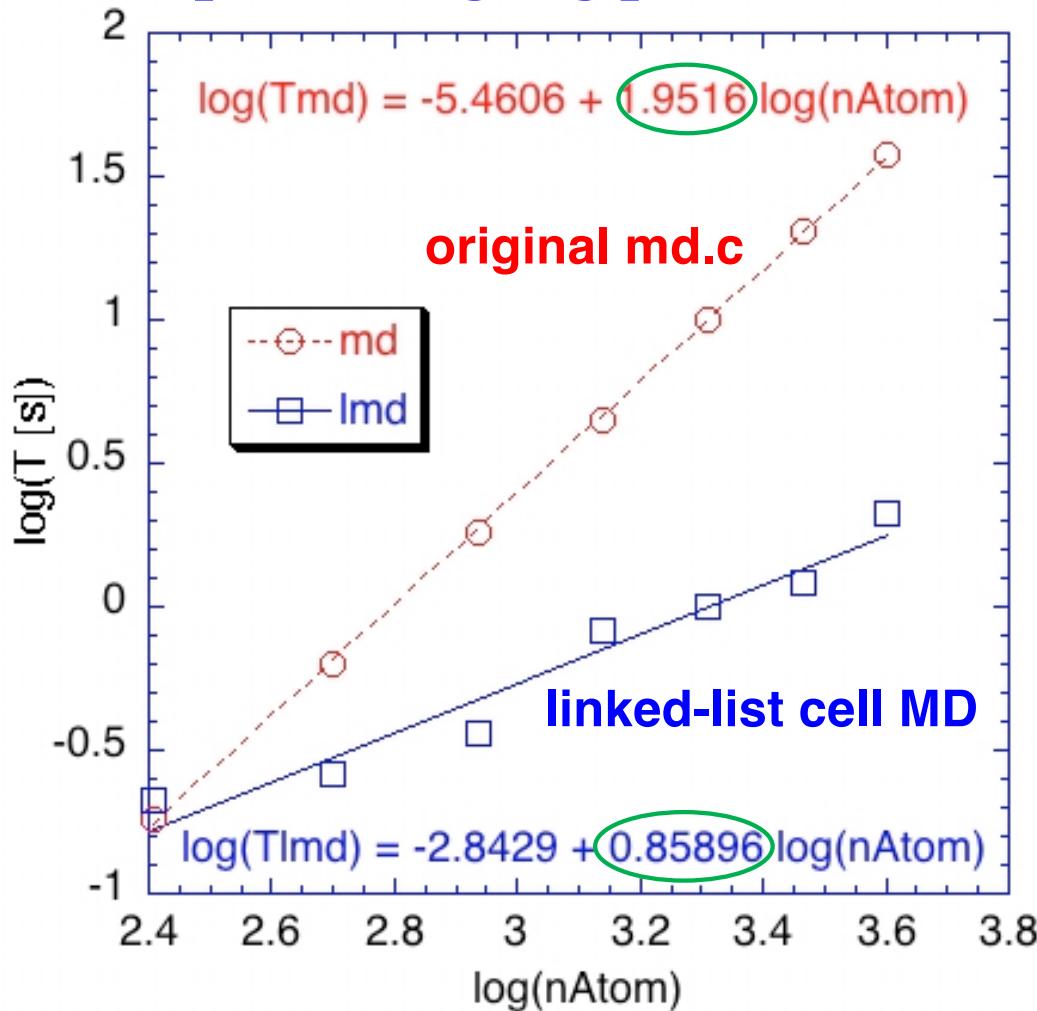


$O(N)$ Force Calculation Algorithm

```
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 inner cells */  
O(N)   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 */  
O(1)       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) */  
                   rij = ri-(rj+rshift); /* Image-corrected relative pair position */  
                   if (rij < rc2)  
                       Compute forces on pair (i, j)  
               }  
               j = lscl[j]; /* Follow the chain of linked atoms in neighbor cell */  
           }  
           i = lscl[i]; /* Follow the chain of linked atoms in central cell */  
       }  
   }  
}
```

Scaling of Linked-List Cell MD

Slope in a log-log plot reveals the complexity power

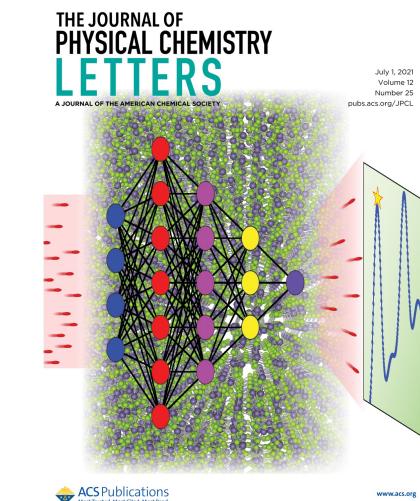
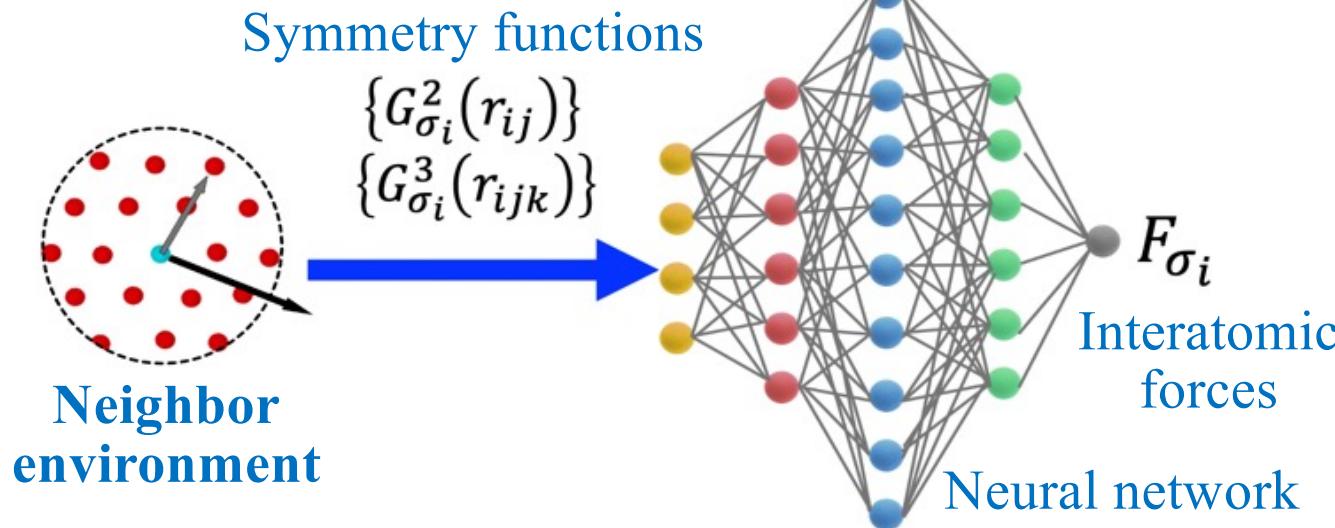


$$\begin{aligned} T &= CN^p \\ \Downarrow \\ \overbrace{\log T}^y &= \log C + \log N^p \\ &= \log C + p \overbrace{\log N}^x \end{aligned}$$

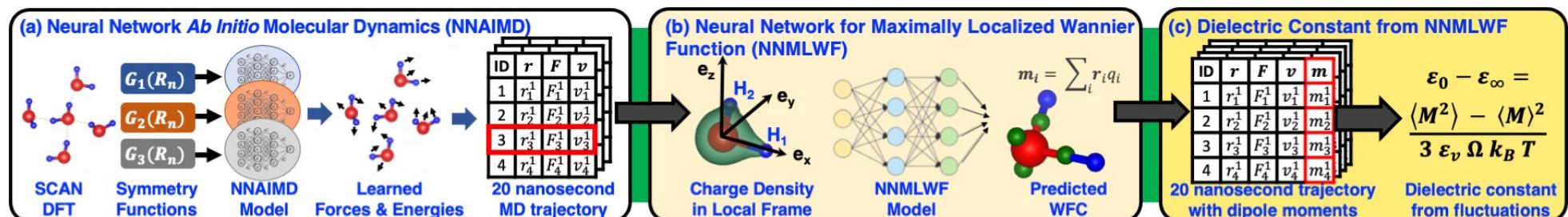
How to do $O(N)$ MD if there is no cut-off radius r_c ?
→ Use fast multipole method (FMM)!

Neural-Network Quantum Molecular Dynamics

- NNQMD@scale could revolutionize atomistic modeling of materials, providing quantum-mechanical accuracy at a fraction of computational cost



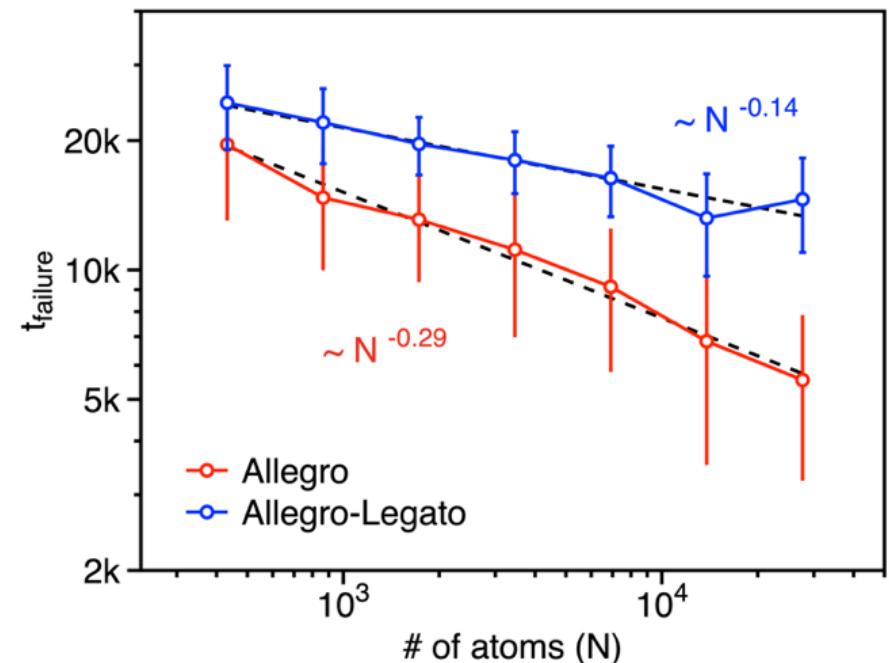
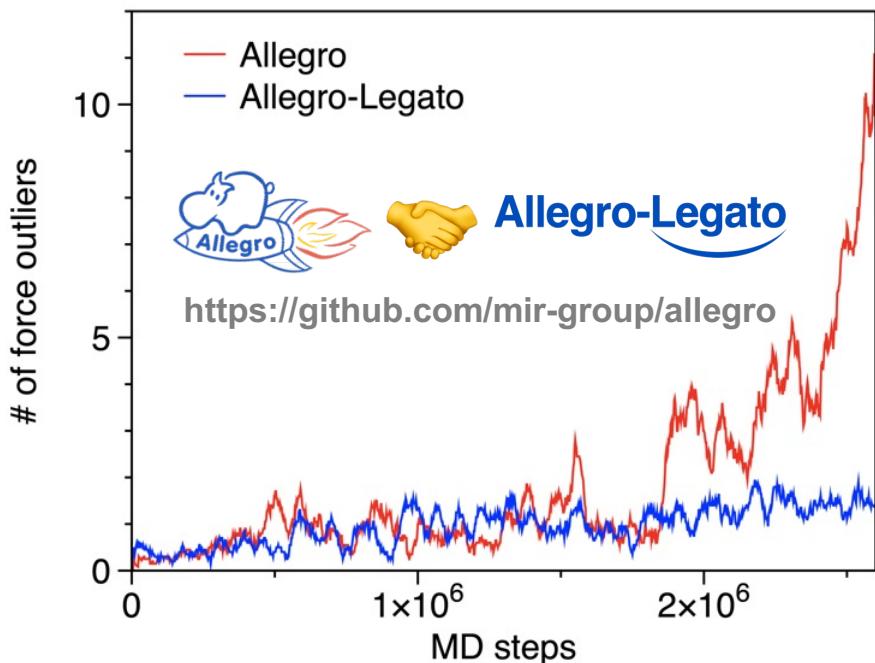
- Combine neural networks to predict: (1) atomic forces for performing MD simulations; and (2) maximally-localized Wannier-function (MLWF) centers for computing quantum properties like electronic dipoles



P. Rajak *et al.*, *J. Phys. Chem. Lett.* **12**, 6020 ('21)
A. Krishnamoorthy *et al.*, *Phys. Rev. Lett.* **126**, 216403 ('21)

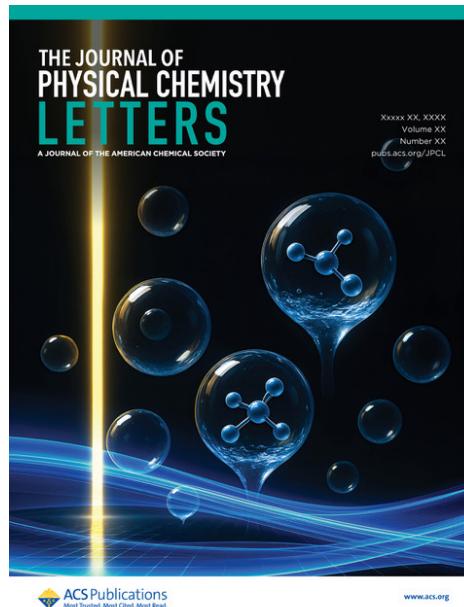
Fast & Robust NNQMD: Allegro-Legato

- **Allegro (fast) NNQMD:** State-of-the-art *accuracy & speed* founded on *group-theoretical equivariance* & local descriptors Musaelian et al., *Nat. Commun.* **14**, 579 ('23)
- **Fidelity-scaling problem:** On massively parallel computers, growing number of unphysical (adversarial) force predictions prohibits simulations involving larger numbers of atoms for longer times *It's symmetry & data locality!*
- **Allegro-Legato (fast and “smooth”):** *Sharpness aware minimization (SAM)* enhances the *robustness* of Allegro through improved smoothness of loss landscape
 $w_* = \operatorname{argmin}_w [L(w) + \max_{\|\epsilon\|_2 \leq \rho} \{L(w + \epsilon) - L(w)\}]$ (L : loss; w : model parameters)
- **Elongated time-to-failure scaling, $t_{\text{failure}} = O(N^{-\beta})$,** without sacrificing accuracy or speed, thereby achieving spectroscopically stable long-time Hamiltonian trajectory



Foundation Model: Allegro-FM

- Foundation models are a paradigm shift in AI, where a single universal model acquires sufficient generalizability for diverse downstream tasks
- *Allegro-FM* describes many material properties & processes covering 89 elements in the periodic table, exhibiting *emergent capabilities* for which the model was not trained
- Universality was achieved by unifying large databases of multiple fidelity through affine (shift & scale) transformations in a metamodel-space: *total energy alignment (TEA)* Shiota *et al.*, arXiv:2412.13088 ('24)



K. Nomura *et al.*, *J. Phys. Chem. Lett.* **16**, 6637 ('25)

Nobel Prize in Physics(-Computer Science)

The Nobel Prize in Physics 2024



Ill. Niklas Elmehed © Nobel Prize Outreach

John J. Hopfield

Prize share: 1/2



Ill. Niklas Elmehed © Nobel Prize Outreach

Geoffrey E. Hinton

Prize share: 1/2

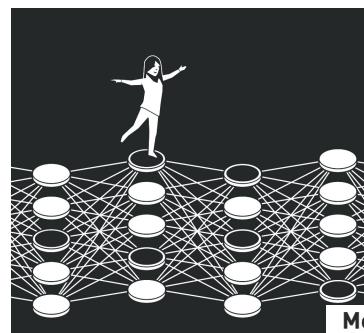
<https://www.nobelprize.org/prizes/physics/2024>

The Nobel Prize in Physics 2024 was awarded to John J. Hopfield and Geoffrey E. Hinton "for foundational discoveries and inventions that enable machine learning with artificial neural networks"

"Now what" by John Hopfield (*Princeton*)

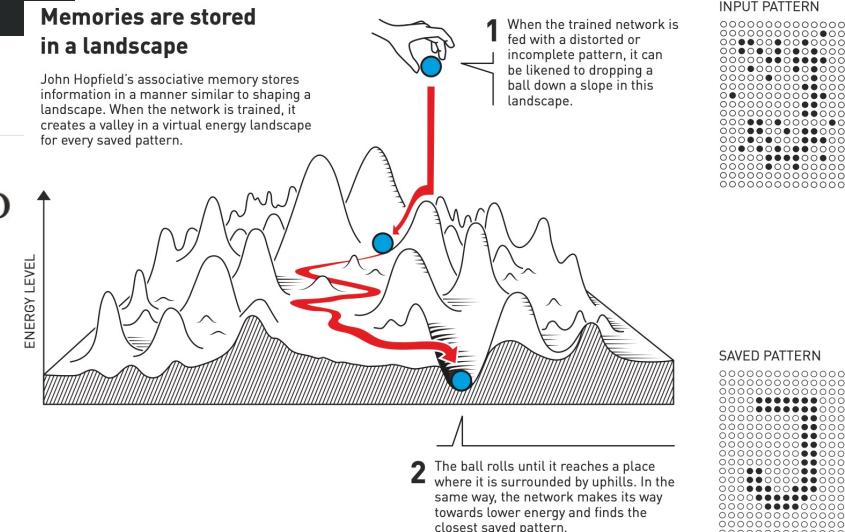
This year's laureates used tools from physics to construct methods that helped lay the foundation for today's powerful machine learning.

Because physics has contributed tools for the development of machine learning, it is interesting to see how physics, as a research field, is also benefitting from artificial neural networks.



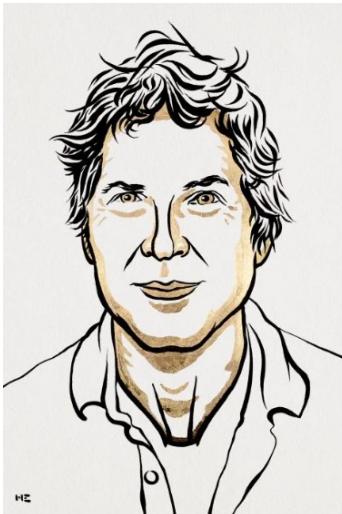
Memories are stored in a landscape

John Hopfield's associative memory stores information in a manner similar to shaping a landscape. When the network is trained, it creates a valley in a virtual energy landscape for every saved pattern.



Nobel Prize in Chemistry(-Computer Science)

The Nobel Prize in Chemistry 2024



Ill. Niklas Elmehed © Nobel Prize Outreach

David Baker

Prize share: 1/2



Ill. Niklas Elmehed © Nobel Prize Outreach

Demis Hassabis

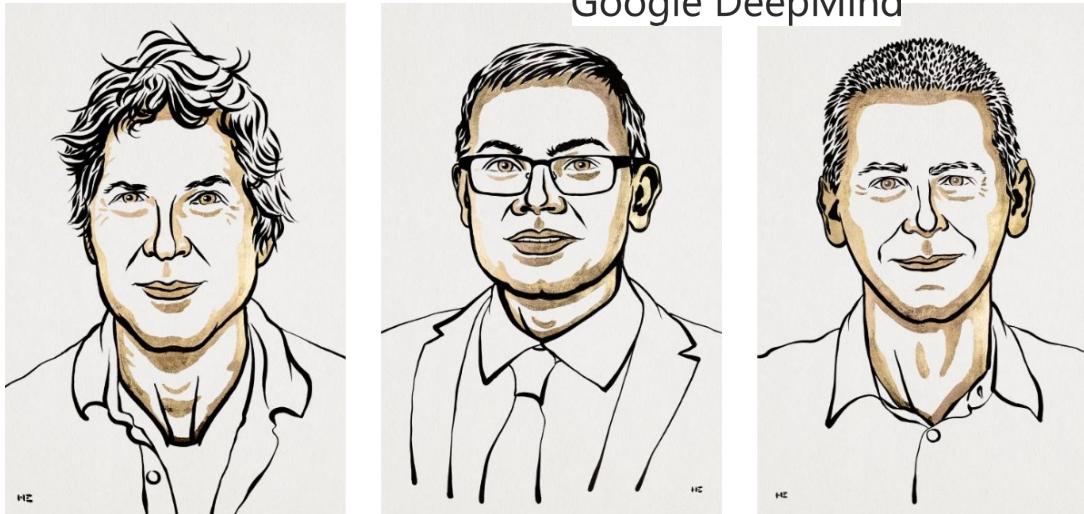
Prize share: 1/4



Ill. Niklas Elmehed © Nobel Prize Outreach

John M. Jumper

Prize share: 1/4



<https://www.nobelprize.org/prizes/chemistry/2024>

The Nobel Prize in Chemistry 2024 was divided, one half awarded to David Baker "for computational protein design", the other half jointly to Demis Hassabis and John M. Jumper "for protein structure prediction"

Levinthal paradox (1968): How the Nature folds an amino-acid sequence into a global energy minimum 3D structure (which is known to be NP complete) within microseconds (~ billion molecular-dynamics steps)?

How does AlphaFold2 work?

As part of AlphaFold2's development, the AI model has been trained on all the known amino acid sequences and determined protein structures.

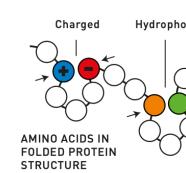
1. DATA ENTRY AND DATABASE SEARCHES

An amino acid sequence with unknown structure is fed into AlphaFold2, which searches databases for similar amino acid sequences and protein structures.

2. SEQUENCE ANALYSIS

The AI model aligns all the similar amino acid sequences – often from different species – and investigates which parts have been preserved during evolution.

In the next step, AlphaFold2 explores which amino acids could interact with each other in the three-dimensional protein structure. Interacting amino acids co-evolve. If one is charged, the other has the opposite charge, so they are attracted to each other. If one is replaced by a water-repellent (hydrophobic) amino acid, the other also becomes hydrophobic.

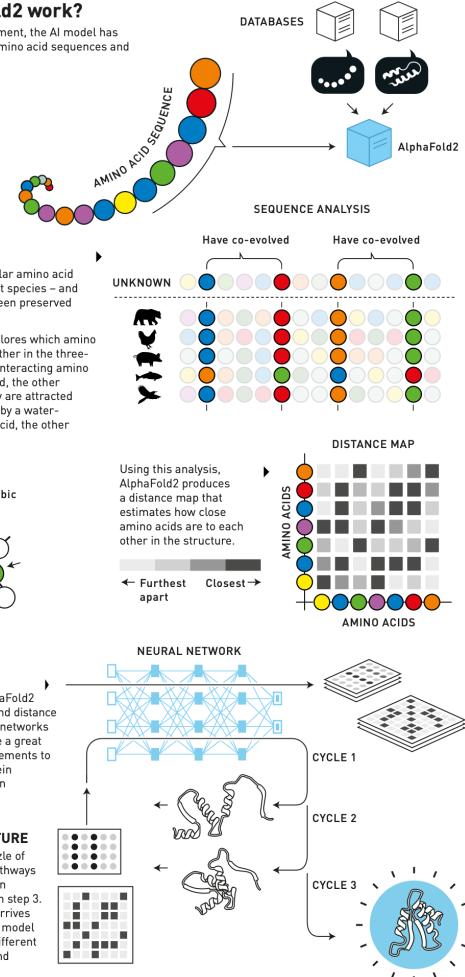


3. AI ANALYSIS

Using an iterative process, AlphaFold2 refines the sequence analysis and distance map. The AI model uses neural networks called transformers, which have a great capacity to identify important elements to focus on. Data about other protein structures – if they were found in step 1 – is also utilized.

4. HYPOTHETICAL STRUCTURE

AlphaFold2 puts together a puzzle of all the amino acids and tests pathways to produce a hypothetical protein structure. This is re-run through step 3. After three cycles, AlphaFold2 arrives at a particular structure. The AI model calculates the probability that different parts of this structure correspond to reality.



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.

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.



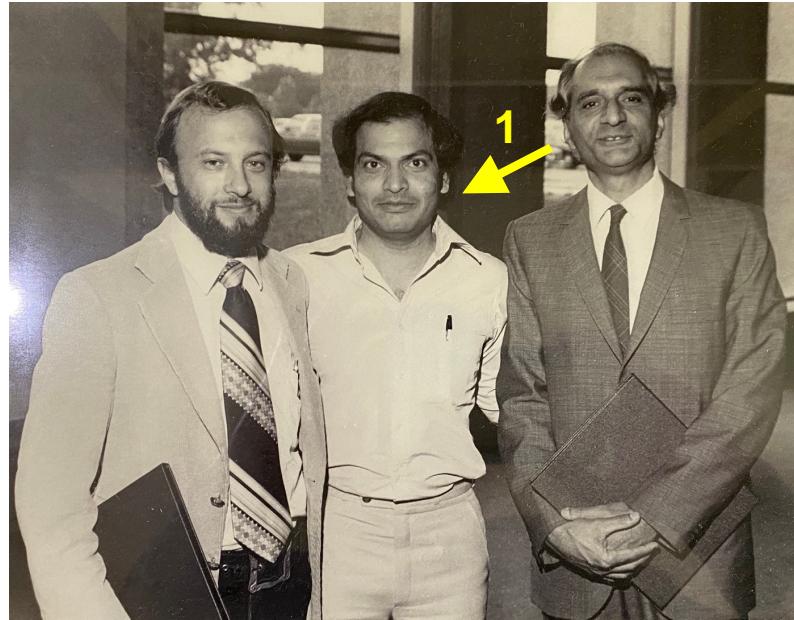
**Anees Rahman
(1927-1987)**

See the [Nobel lecture](#) by Michael Levitt

Your Rahman Number?



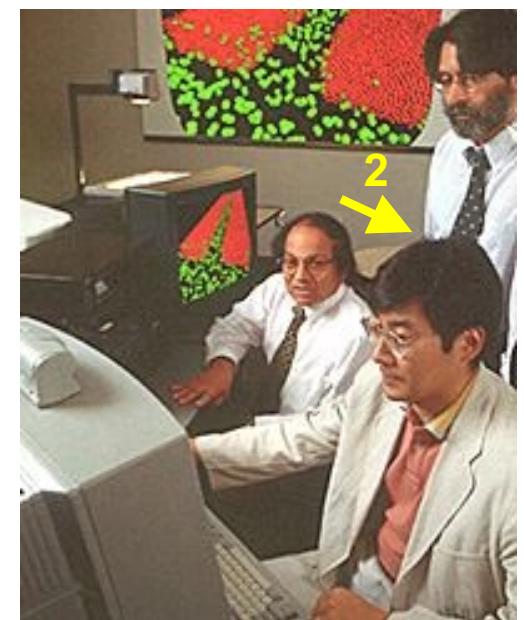
Anees Rahman ('67)



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

<https://aiichironakano.github.io/phys516/Battimelli-ComputerMeetsPhysics-Springer20.pdf>, pp. 58 & 128

Priya Vashishta,
Rajiv Kalia
and AN ('02)



Your Rahman number is 3

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*