Molecular Dynamics

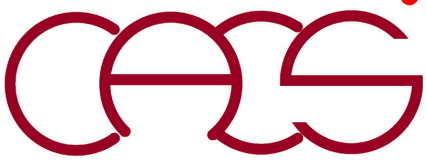
Aiichiro Nakano

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

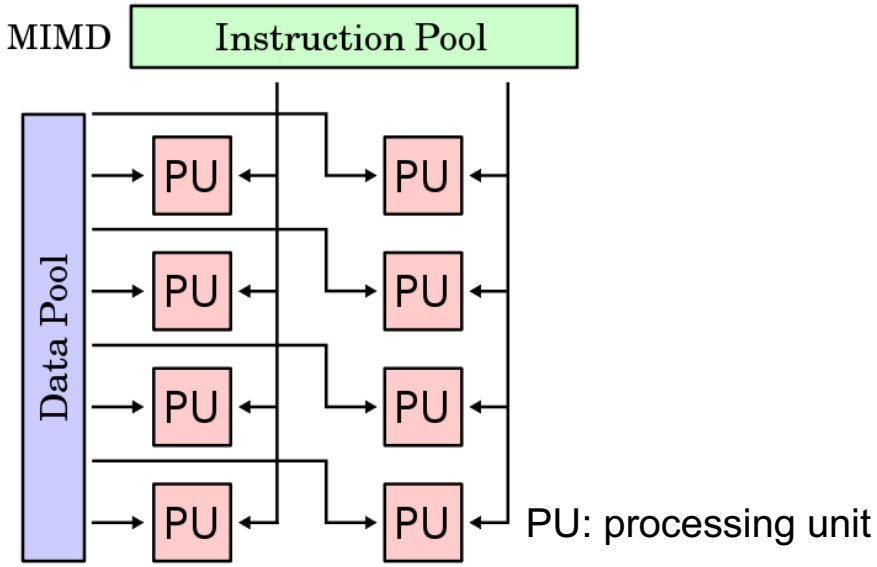
Objective: Basics of particle simulation

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



What to Learn with MD

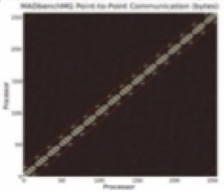
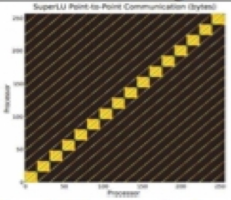
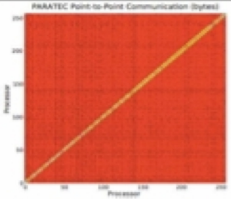
- **An archetype of multiple-instruction multiple-data (MIMD) parallel applications: Discrete particle dynamics based on ordinary differential equation (ODE); interaction**
- **Data locality & scalable data structure: Linked-list cells**
- **Multiresolution algorithm: Divide-&-conquer**

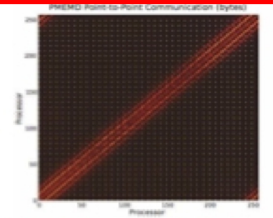
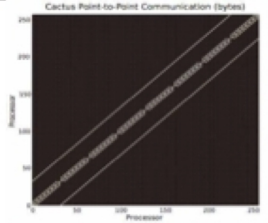


MIMD: Multiple autonomous processors concurrently execute different instructions on different data

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*
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
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
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]

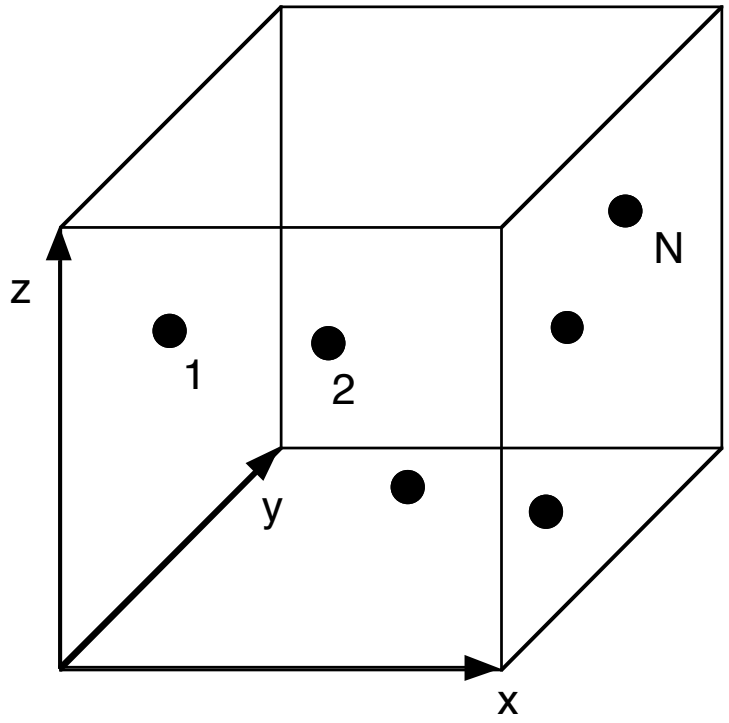
Dwarf	*Description*	*Communication Pattern (Figure axes show processors 1 to 256, with black meaning no communication)*	*NAS Benchmark / Example HW*
4. N-Body Methods (e.g., Barnes-Hut [Barnes and Hut 1986], Fast Multipole Method [Greengard and Rokhlin 1987])	Depends on interactions between many discrete points. Variations include particle-particle methods, where every point depends on all others, leading to an $O(N^2)$ 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]
5. Structured Grids (e.g., Cactus [Goodale et al 2003] or Lattice-Boltzmann Magneto-hydrodynamics [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
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

https://www2.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html

System: A Set of Point Atoms

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

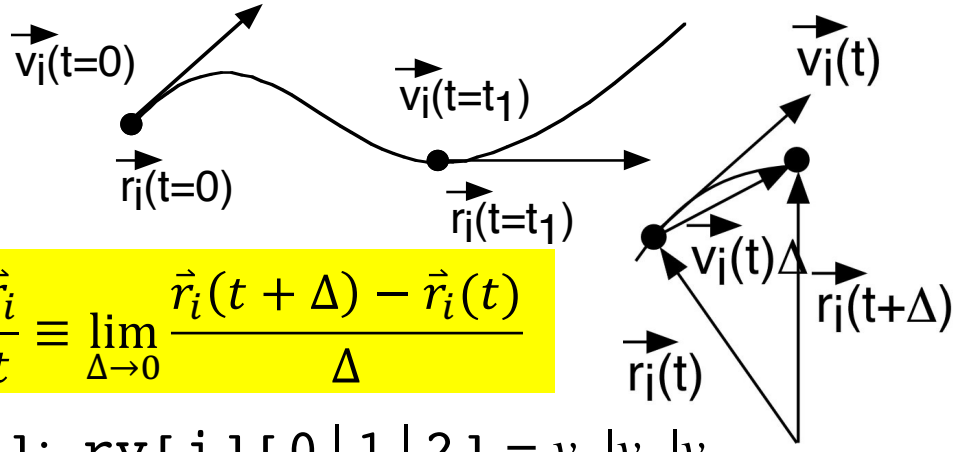
`int nAtom`: N, # of atoms.
`NMAX`: Max # of atoms.
`double r[NMAX][3]`: `r[i][0|1|2]` = $x_i|y_i|z_i$.



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

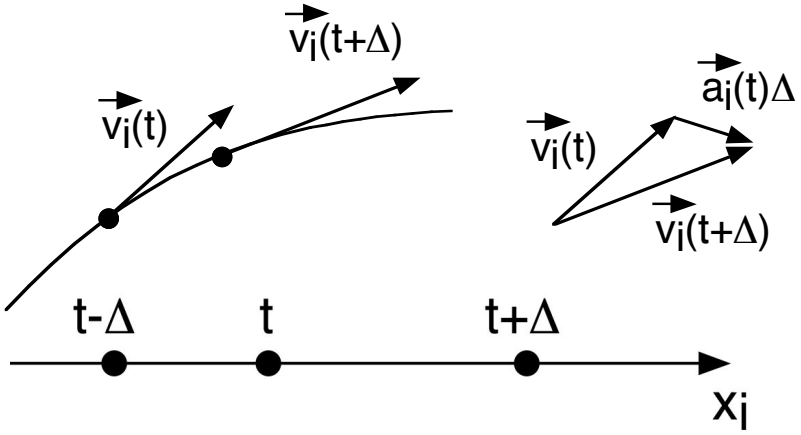
$$\vec{v}_i(t) = \dot{\vec{r}}_i(t) = \frac{d\vec{r}_i}{dt} \equiv \lim_{\Delta\to 0} \frac{\vec{r}_i(t+\Delta) - \vec{r}_i(t)}{\Delta}$$

`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]` = $a_{ix}|a_{iy}|a_{iz}$



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

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

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

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

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

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, $\{(\vec{r}_i(0), \vec{v}_i(0)) | i = 0, \ldots, N-1\}$,

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,

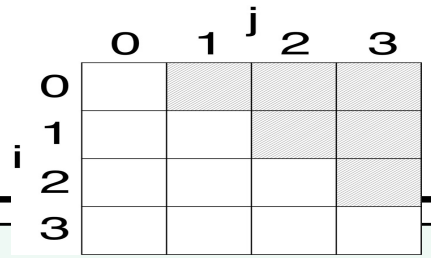
$$\ddot{\vec{r}}_k(t) = \vec{a}_k(t) = -\frac{\partial}{\partial \vec{r}_k} \sum_{i<j} u(r_{ij}) = \sum_{i<j} \vec{r}_{ij}(t) \left(-\frac{1}{r}\frac{du}{dr}\right)_{r=r_{ij}(t)} (\delta_{ik} - \delta_{jk})$$

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

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

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

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

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

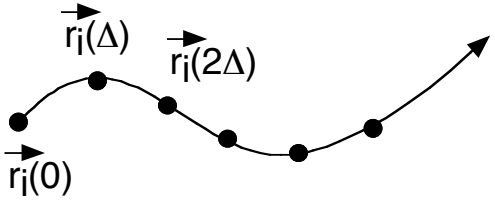
$\vec{a}_j$ −= $\vec{a}$

```
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;
         }
      }
   }
}
```

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

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

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

Verlet Discretization

Let's predict the next step using Taylor expansion

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

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

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;
            }
        }
    }
}
```

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()`**

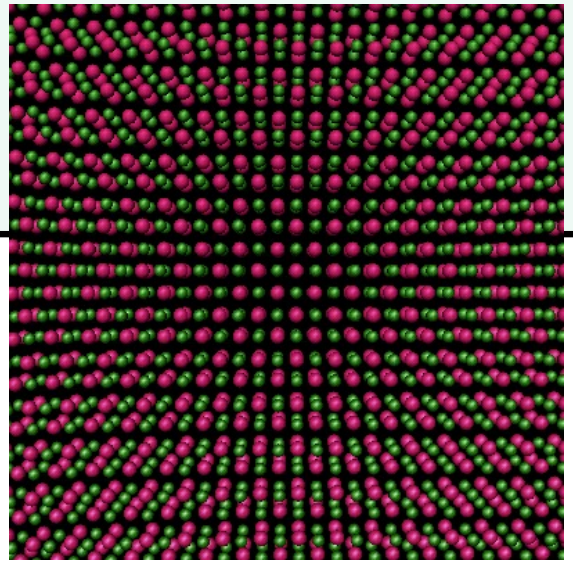
$\vec{v}_i \leftarrow \vec{v}_i + \vec{a}_i \Delta/2$ for all i

$\vec{r}_i \leftarrow \vec{r}_i + \vec{v}_i \Delta$ 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

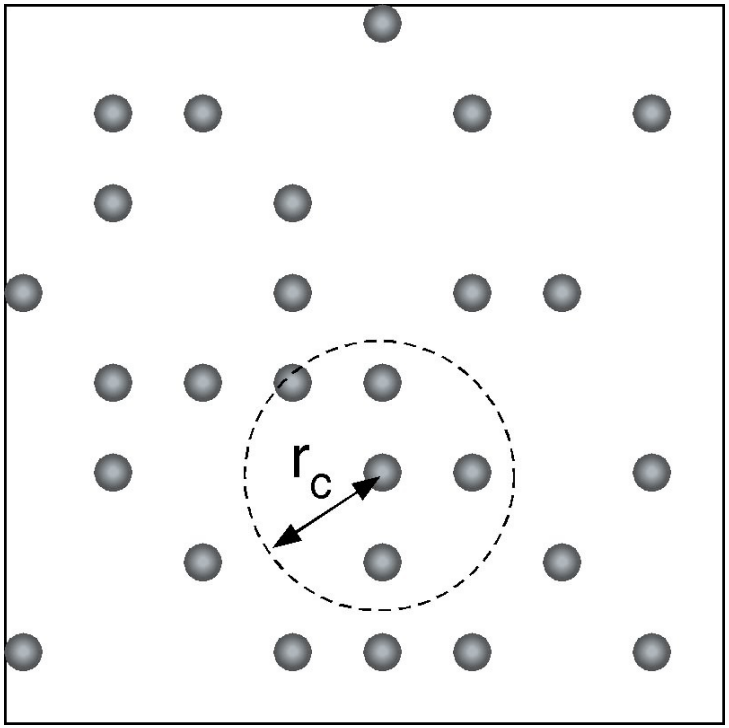


`stepLimit`+1 calls to function `ComputeAccel()`

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

  where

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

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

- **Serial cell index:**

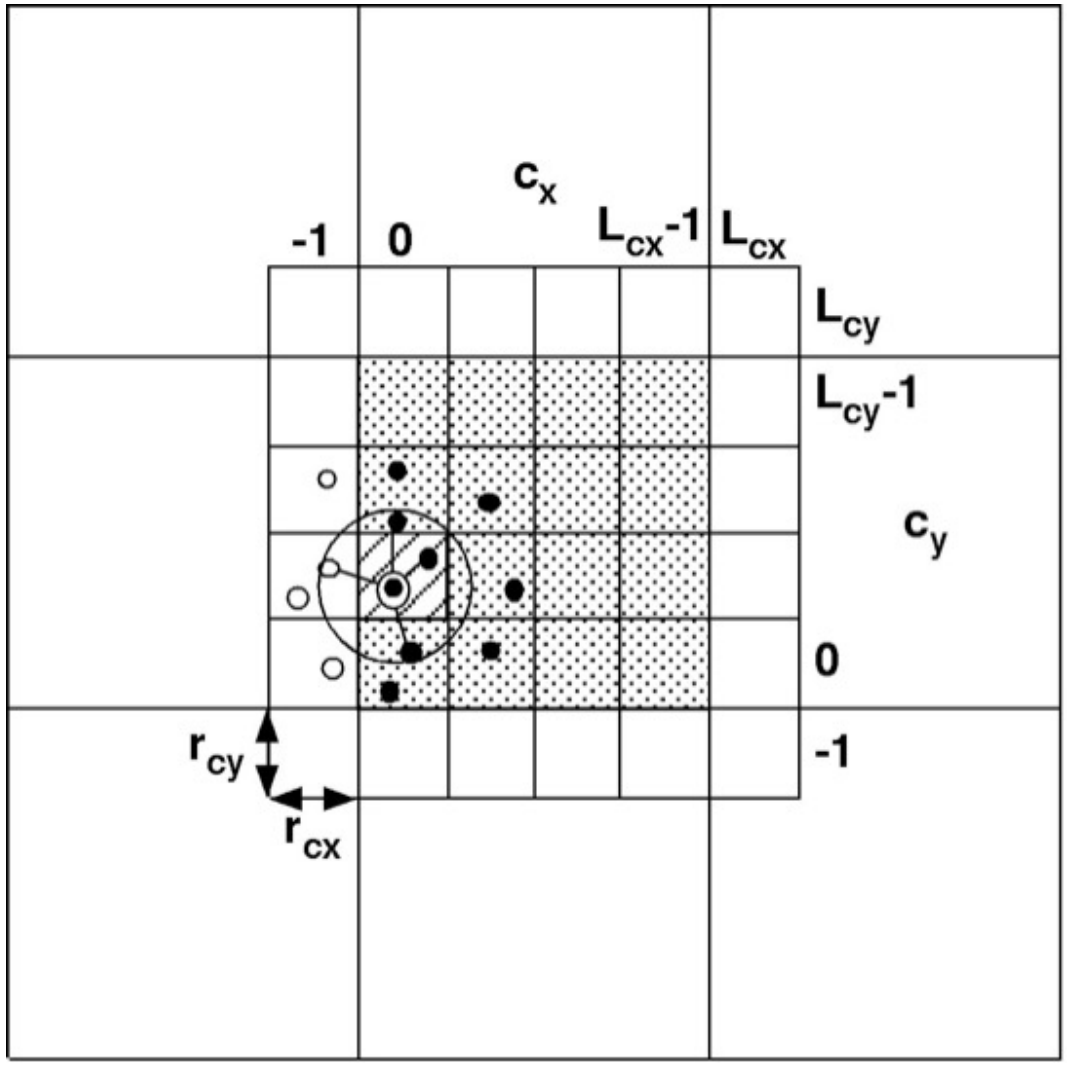
$$c = c_x L_{cy} L_{cz} + c_y L_{cz} + c_z$$

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:**

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

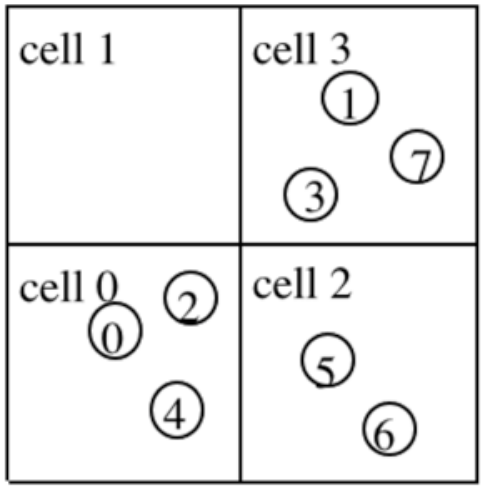


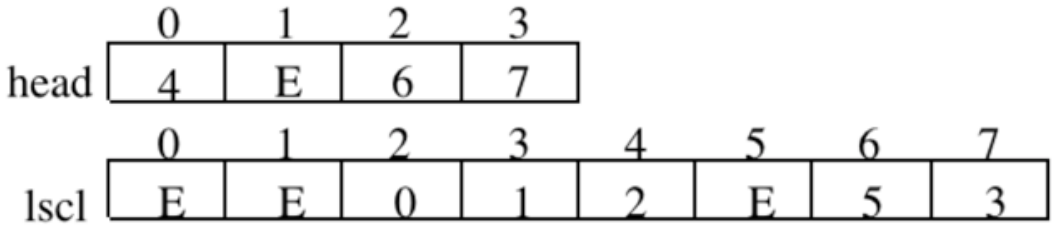
Linked Lists

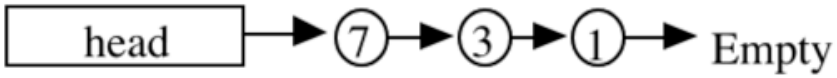
Data Structures:

`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.



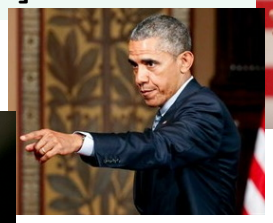




Linked List Construction Algorithm

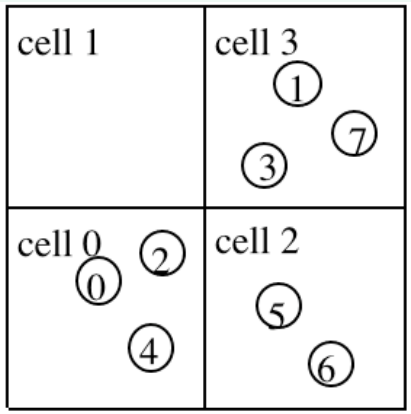
```
/* Reset the headers, head */
for (c=0; c<lcxyz; c++) head[c] = EMPTY;
/* Scan atoms to construct headers, head, & linked lists, lscl */
for (i=0; i<nAtom; i++) {
  /* Vector cell index to which this atom belongs */
  for (a=0; a<3; a++) mc[a] = r[i][a]/rc[a];
  /* Translate the vector cell index, 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]
```

$O(N)$



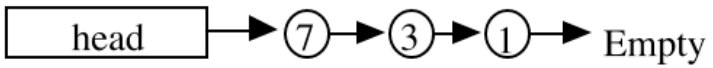
∴

Single pass over atoms → $O(N)$!



	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

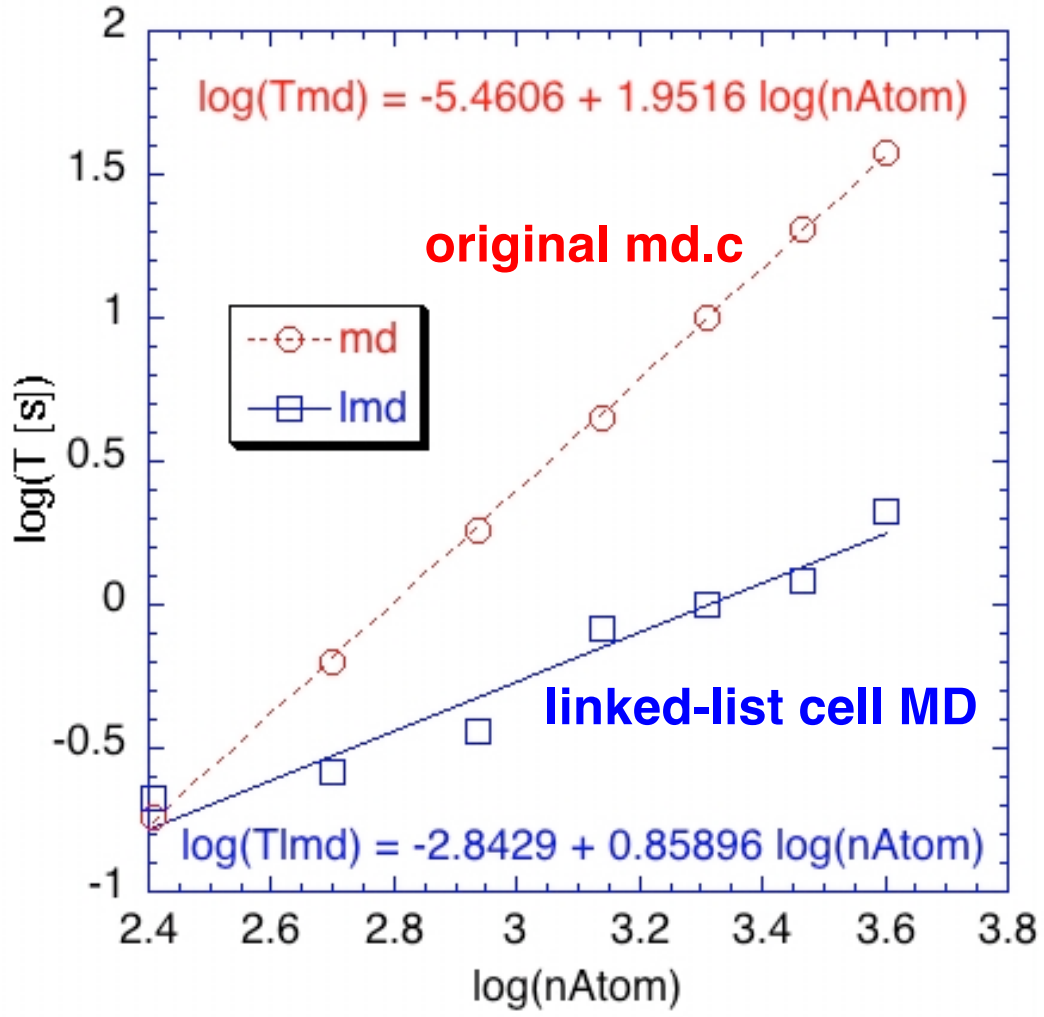
```
for (mc[0]=0; mc[0]<lc[0]; (mc[0])++)
for (mc[1]=0; mc[1]<lc[1]; (mc[1])++)
for (mc[2]=0; mc[2]<lc[2]; (mc[2])++) { /* Scan inner cells */
  c = mc[0]*lcyz+mc[1]*lc[2]+mc[2]; /* Calculate a scalar cell index */
  for (mc1[0]=mc[0]-1; mc1[0]<=mc[0]+1; (mc1[0])++)
  for (mc1[1]=mc[1]-1; mc1[1]<=mc[1]+1; (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^2)
            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 */
    }
  }
}
```

$O(N)$

$O(1)$

Scaling of Linked-List Cell MD

Slope in a log-log plot reveals the complexity power

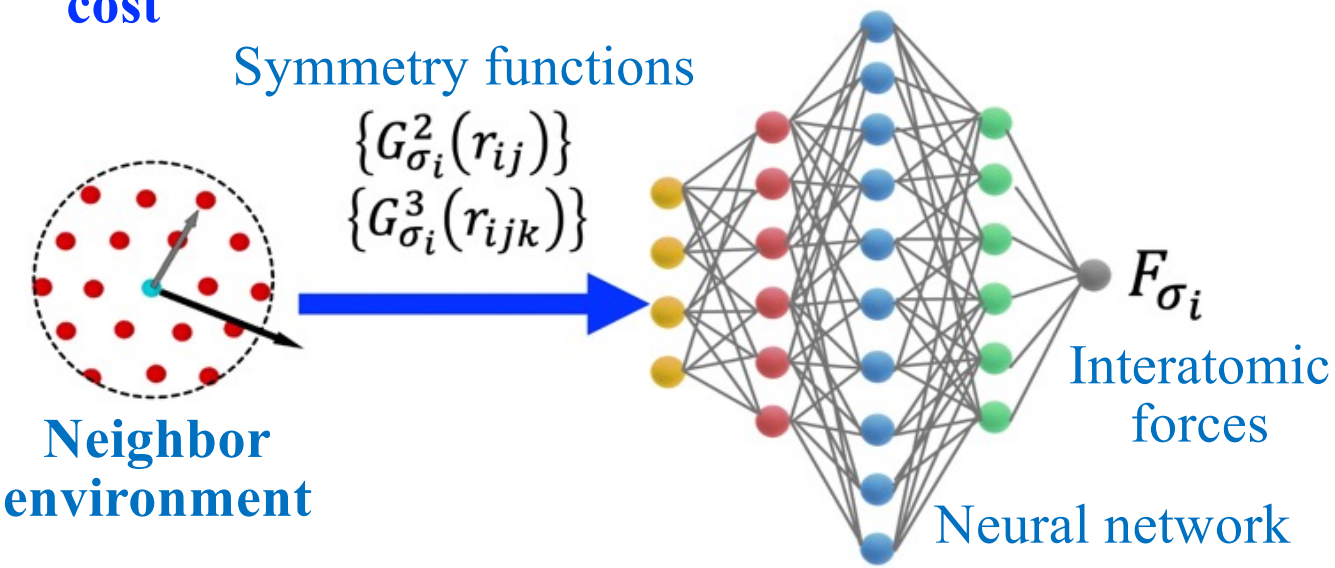


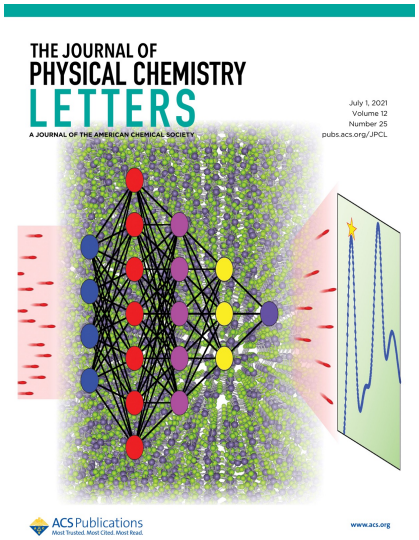
How to do $O(N)$ MD if there is no cut-off radius r_c?

→ Use fast multipole method (FMM)!

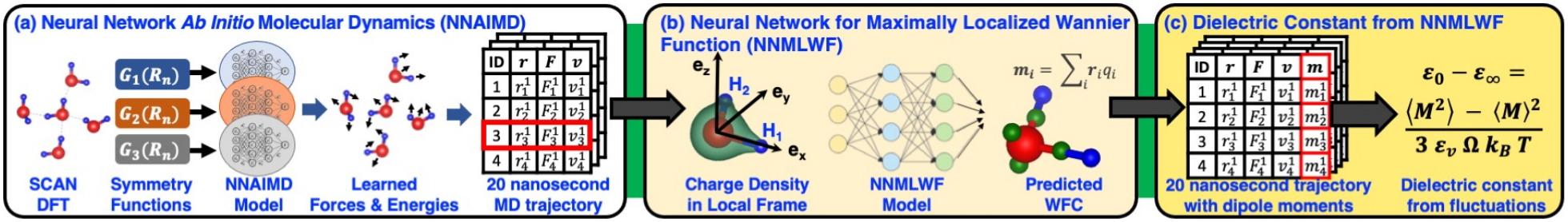
Neural-Network Quantum Molecular Dynamics

- **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)