Accelerating Force Calculation in Molecular Dynamics with Transition to LAMMPS

Cuong Nguyen

MIT PSAAP-3 Team

05 February 2021

Molecular Dynamics

- **MD simulation integrates Newton's equations of motion in time:**

$$m_n \frac{d^2 \boldsymbol{r}_n}{dt^2} = \boldsymbol{f}_n = -\frac{\partial U(\boldsymbol{r}_1, \ldots \boldsymbol{r}_N)}{\partial \boldsymbol{r}_n}$$

- **MD simulation requires calculation of a force field from a given potential energy surface (PSE)**
  - Force calculation is the most expensive operation in MD simulations
  - Force calculation often takes up more than 50% of the overall complexity
  - For complex potentials, it may even cost 90% of the overall complexity.

- **Making force calculation fast is key to accelerating MD simulations**

Techniques to Accelerate Force Calculation

- **Exploit Newton's third law to reduce the cost by a factor of 2**
  - It is easily implemented on single core, but get more complicated on many cores.
- **Exploit additional symmetry properties of a potential to reduce the complexity**
  - SNAP potential exploit additional symmetries to reduce the number of terms by 3x and the complexity from $O(J^7)$ to $O(J^5)$.
- **Implement force calculation routines on accelerators**
  - LAMMPS-CUDA, LAMMPS-KOKKOS, LAMMPS-OMP, and LAMMPS-INTEL packages
- **Optimize serial code through inlining, unrolling, scheduling, cse**
- **Optimize parallel algorithms on a single node and across many nodes**

Thompson, A. P., Swiler, L. P., Trott, C. R., Foiles, S. M., & Tucker, G. J. (2015). ***Spectral neighbor analysis method for automated generation of quantum-accurate interatomic potentials***. *Journal of Computational Physics*, *285*, 316-330.

Zhang, Xiaohua, et al. "***ddcMD: A fully GPU-accelerated molecular dynamics program for the Martini force field***." *The Journal of Chemical Physics* 153.4 (2020): 045103.

Our Ideas to Accelerate Force Calculation

- **Consider five different parallel algorithms for force calculation**
  - Force decomposition algorithm (implemented in LAMMPS CPU packages)
  - Atom decomposition algorithm (implemented in LAMMPS-GPU and LAMMPS-KOKKOS)
  - **Spatial decomposition algorithm** (implemented in LAMMPS-MPI package)
  - **Hybrid decomposition algorithm** (not implemented in LAMMPS)
  - **Multi-level decomposition algorithm** (not implemented in LAMMPS)
- **Deploy our CS tools to optimize the code on various levels**
  - Use **Enzyme** to calculate the potential derivatives
  - Employ **Tiramisu** to optimize data/memory layouts and computation loops
  - Leverage **OpenCilk** compiler to optimize code at compiling time
- **Provide portable implementations on different hardware architectures**
  - A single package targets different types of processors including CPU and GPU
- **Extend our force calculation code to a wide variety of empirical and ML potentials**
  - Allow users to define a potential “on the fly”
- **Transition our code base to LAMMPS**

Force Calculation Algorithms

- **Implement and auto-tune parallel algorithms for force calculation**
  - Force decomposition algorithm (in several LAMMPS-CPU packages)
  - Atom decomposition algorithm (in LAMMPS-GPU and LAMMPS-KOKKOS)
  - **Spatial decomposition algorithm** (in distributed memory LAMMPS-MPI)
  - **Hybrid decomposition algorithm** (not implemented in LAMMPS)
  - **Multi-level decomposition algorithm** (not implemented in LAMMPS)
- **Performance of each algorithm depends on many different factors**
  - Processor types (CPU, AMD GPU, NVIDIA GPU)
  - Number of threads, number of block threads, shared memory, main memory
  - Number of atoms, number of neighbors
  - The form of the potential
- **Try all these algorithms on a quick MD simulation and choose the best performance among them to run the actual MD simulation**

Plimpton, S. (1995). **Fast parallel algorithms for short-range molecular dynamics**. *JCP*, *117*(1), 1-19.

Brown, W. M., Wang, P., Plimpton, S. J., & Tharrington, A. N. (2011). **Implementing molecular dynamics on hybrid high performance computers–short range forces.** CPC, *182*(4), 898-911.

Force Decomposition Algorithm

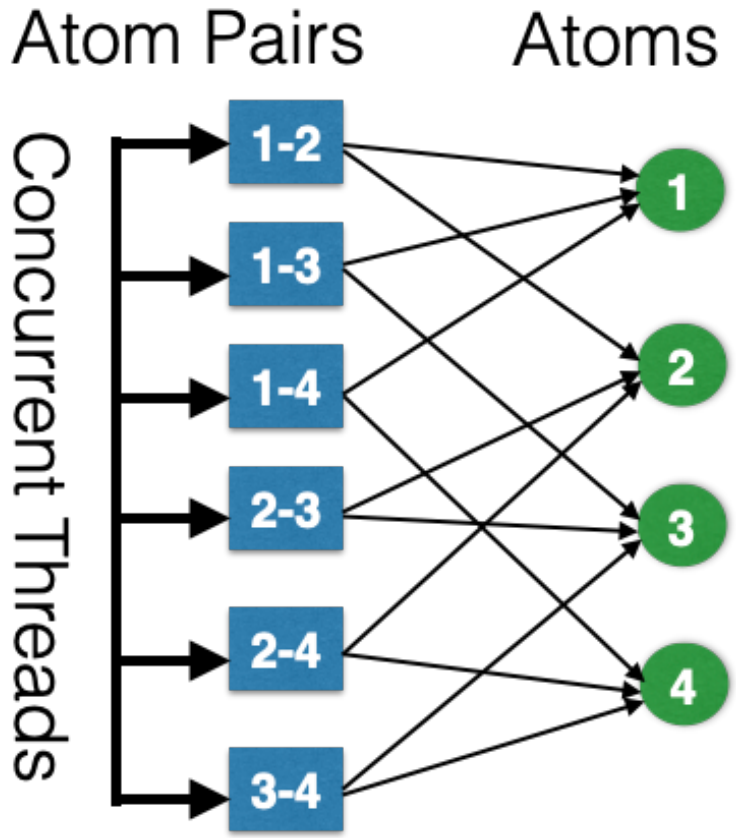
- **Assigning threads to atom pairs**
  - Each thread calculates and adds pair force F_{ij} to both atoms i and j

- **Use atomicAdd to avoid race condition (-)**

- **If the number of neighbors is large, atomicAdd can lead to large overhead (-)**

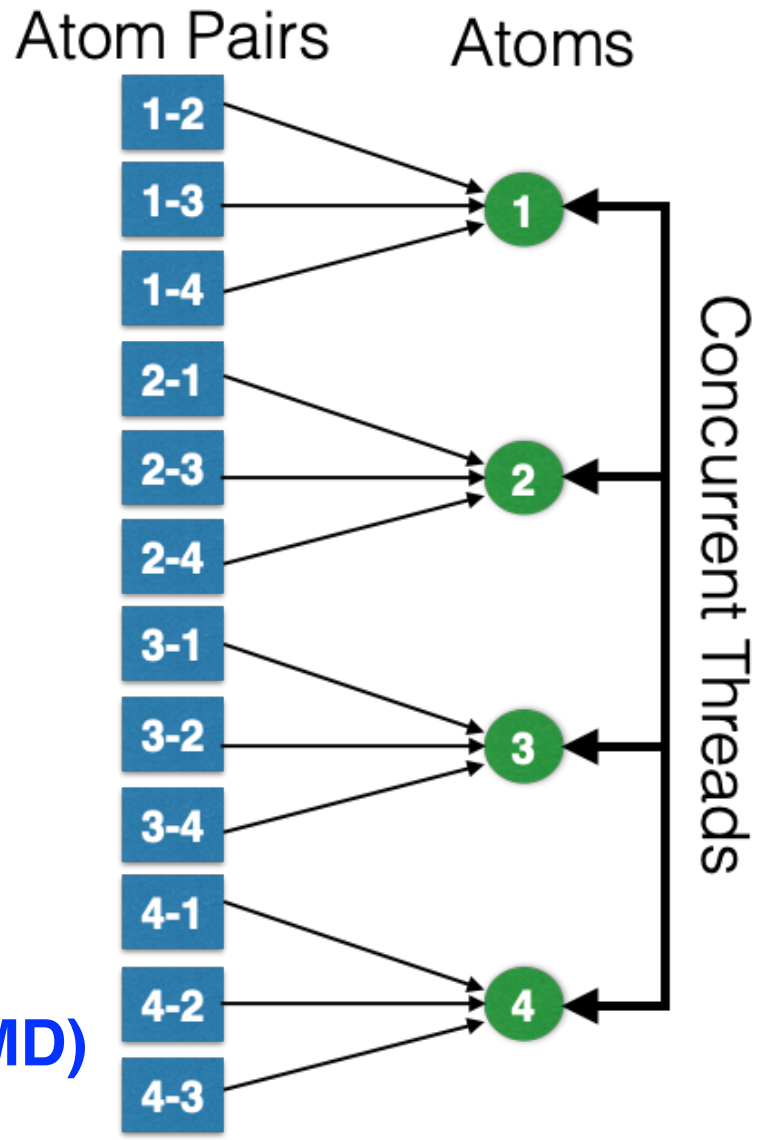
- **The algorithm makes use of Newton's third law to reduce the complexity by a factor of 2 (+)**

- **Each thread has light workload and allows for making good use of shared memory (+)**



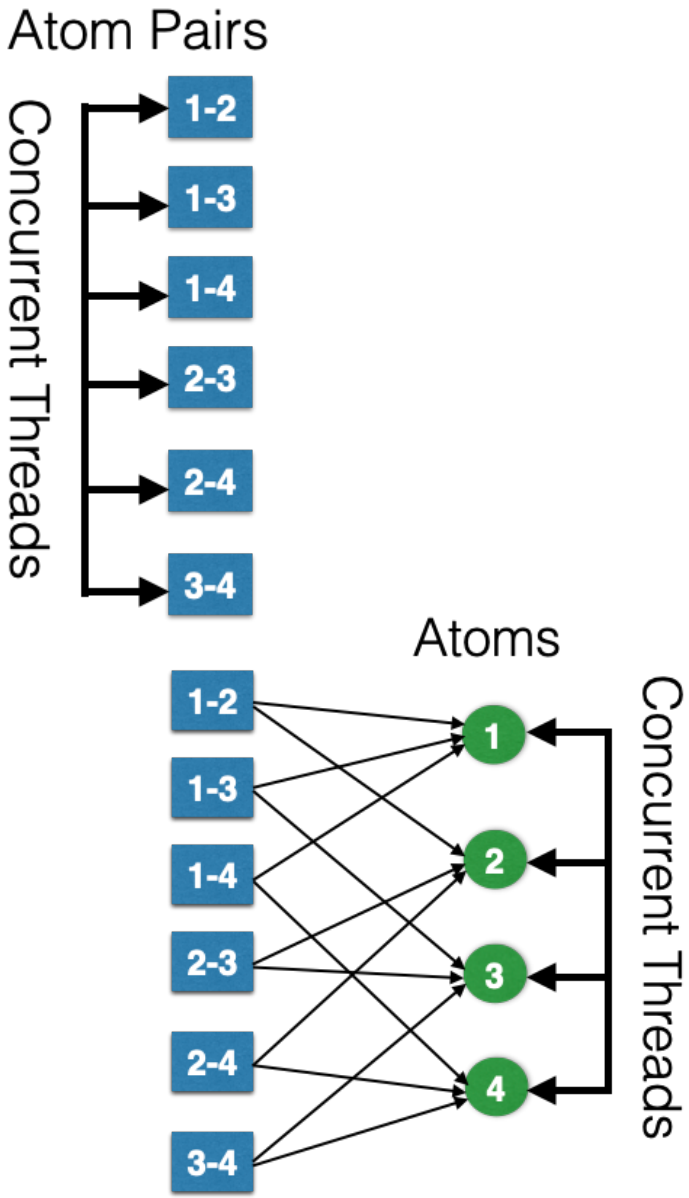
Atom Decomposition Algorithm

- **Assigning threads to atoms**
  - Each thread i calculates forces F_{ij} for all neighbors and adds them to atom i
- **Embarrassingly parallel, so avoid atomicAdd (+)**
- **It does not make use of Newton's third law, so its complexity is twice as the previous one (-)**
- **Each thread has heavy workload, which is difficult to make use of shared memory (-)**
- **It is implemented in LAMMPS's GPU and Kokkos packages, and ddcMD (https://github.com/LLNL/ddcMD)**



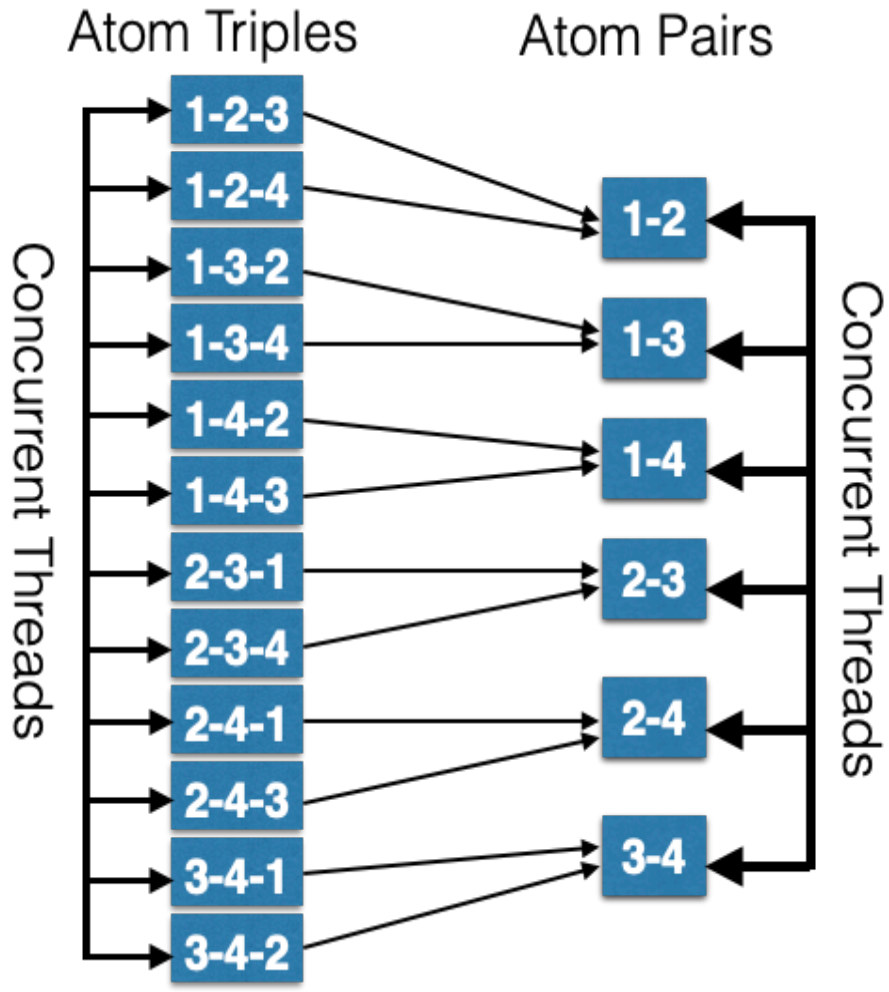
Hybrid Decomposition Algorithm

- **Assigning threads to atom pairs**
  - Each thread ij calculate pair force F_{ij}
- **Assigning threads to atoms**
  - Each thread i sums pair forces F_{ij} to atom i
- **It has the same complexity as the force decomposition and avoids using atomicAdd (+)**
- **Each thread has light workload, which allows it to make good use of shared memory (+)**
- **It extends to many-body and ML potentials (+)**



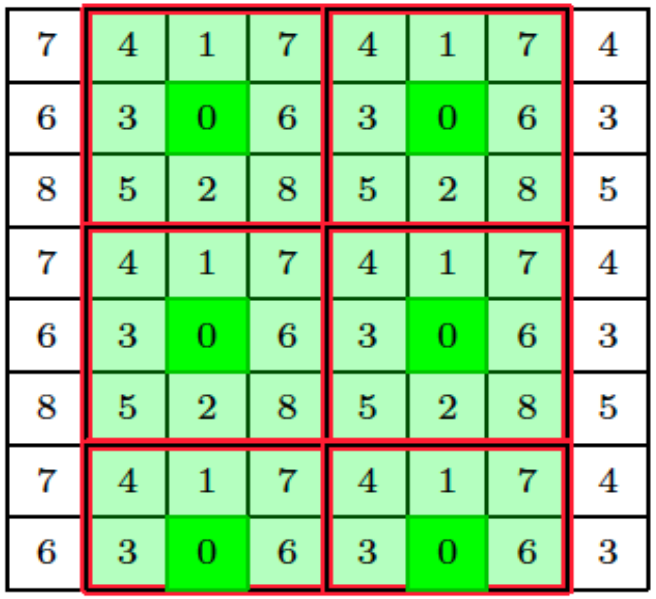
Multi-Level Decomposition for Many-Body Potentials

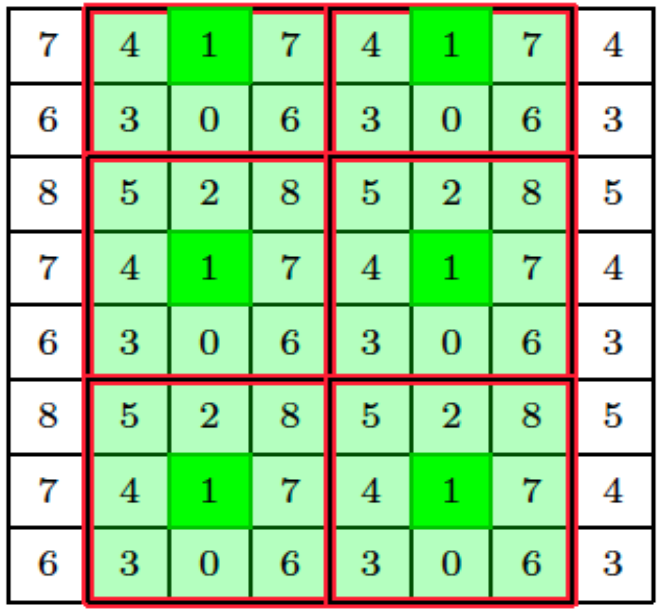
- **Assigning threads to atom triplets**
  - Each thread ijk calculates triplet interaction S_{ijk}
- **Assigning threads to atom pairs**
  - Each thread ij calculates pair force F_{ij} by summing S_{ijk} over index k
- **Assigning threads to atoms**
  - Each thread i sums pair forces F_{ij} to atom i
- **For ML potentials, the index k represents the kth spectrum component of the descriptors.**



Spatial Decomposition Algorithm

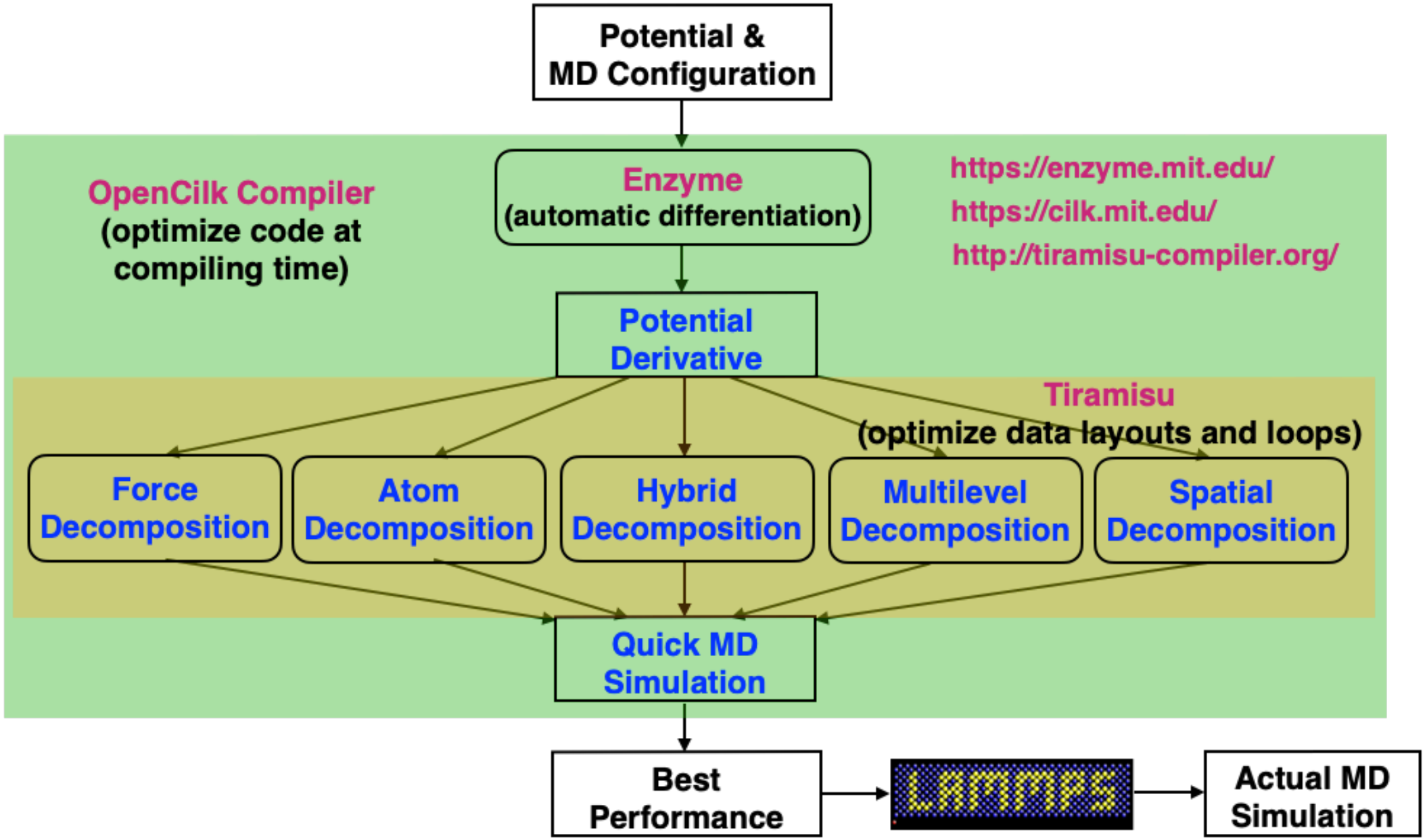
- **Divide the simulation box into cells**
  - The size of each cell is equal to the cut-off radius
- **Color the cells so that the same-colored cells are far apart by 2 times the cut-off radius**
  - There are 9 colors in 2D and 27 colors in 3D are needed
  - Each cell has 8 neighboring cells in 2D and 26 neighbors in 3D
- **For all cells with same color i**
  - Assign **a block of threads** to each i-colored cell
  - The number of threads in a cell is equal to the number of force pairs in that cell
  - Perform the hybrid decomposition to the i-colored cell





http://web.mit.edu/neboat/www/6.172-fa12/chromatic.pdf

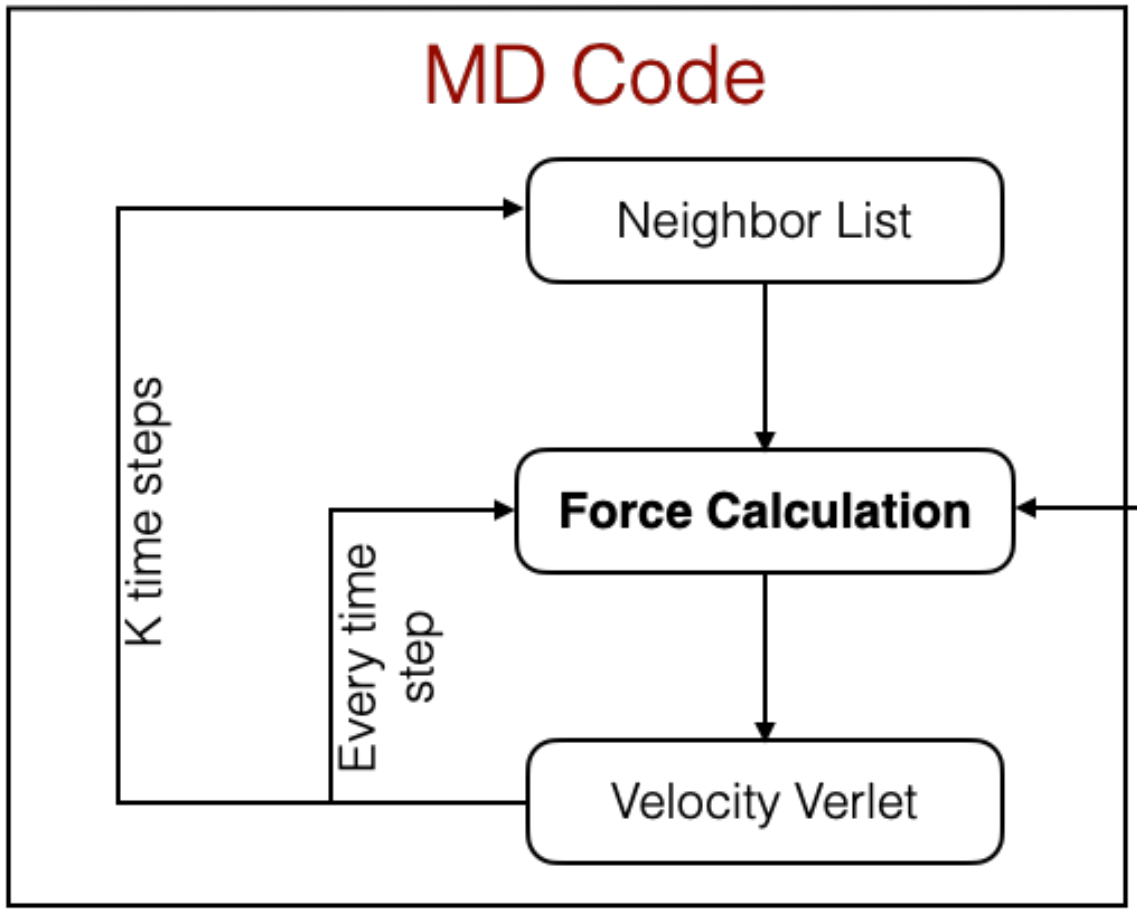
Force Calculation Framework



“On The Fly” Code Generation

- Leverage Exasim’s code generator (https://github.com/exapde/Exasim)
- The framework will support a wide range of empirical potentials and ML potentials

$$U(r_{ij}) = 4\varepsilon\left[\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right]$$



Code Generator

```
template <typename T>  __global__
void  kernelPairs(T *U, T *rij, T *param,
                  int *elemi, int *elemj, int npairs)
{
   int t = threadIdx.x + blockIdx.x * blockDim.x;
   while (t < npairs) {
      T r  = rij[t];
      int ei = elemi[t];
      int ej = elemj[t];
      T r2 = r*r;
      T r4 = r2*r2;
      T r6 = r2*r4;
      T r12 = r6*r6;
      U[t] = param[0]*(param[1]/r12 - param[2]/r6);
      t += blockDim.x * gridDim.x;
   }
}
```

Questions/Comments

- **Main features of our force calculation package:**
  - **Auto-tuned parallel algorithms:** Implement five parallel algorithms and choose the best one to perform MD simulations at run time. The proposed algorithms are better suited to ML potentials on many-core processors than existing algorithms.
  - **Code optimization:** Leverage our CS tools (OpenCilk and Tiramisu) to optimize serial and parallel code at different levels
  - **Code generation:** Employ Exasim's code generator to produce kernel code for potentials instead of making users to write code
  - **Automatic differentiation:** Use Enzyme to compute potential derivatives
  - **Portability:** Develop a single package to target different types of processors (Intel CPU, AMD GPU, NVIDIA GPU)

- **Transition our package to LAMMPS**
  - Leverage LAMMPS-KOKKOS package for fixes, computes, etc.