Latin American Introductory School on Parallel Programming
and Parallel Architecture for High-Performance Computing

LAMMPS

Dr. Richard Berger

High-Performance Computing Group
College of Science and Technology
Temple University
Philadelphia, USA

richard.berger@temple.edu

The Abdus Salam International Centre for Theoretical Physics

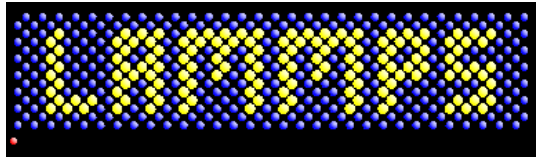
TEMPLE UNIVERSITY

Outline

Outline

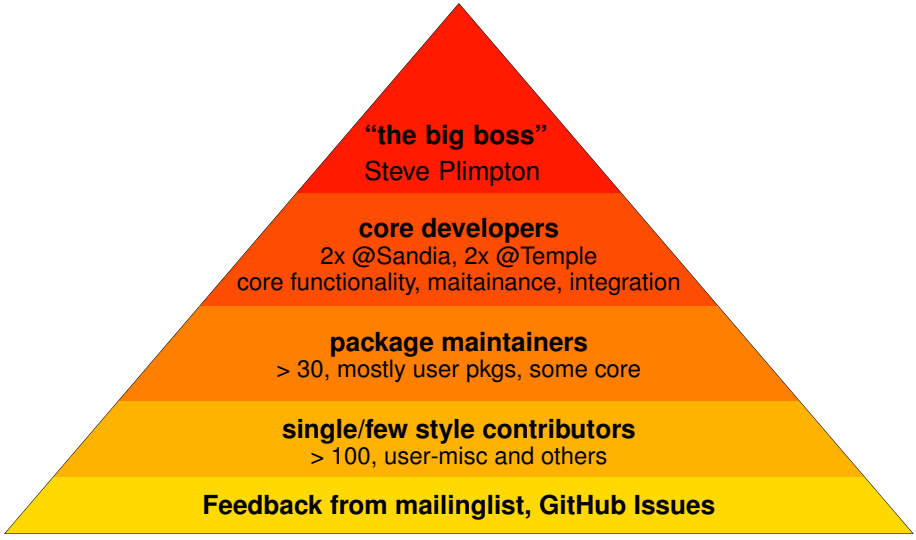
What is LAMMPS?

- Classical Molecular-Dynamics Code
- Open-Source, highly portable C++
- Freely available for download under GPL
- Easy to download, install, and run
- Well documented
- Easy to modify and extend with new features and functionality
- Active user's email list with over 650 subscribers
- More than 1000 citations/year

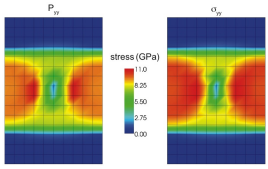


- Atomistic, mesoscale, and coarse-grain simulations
- Variety of potentials (including many-body and coarse-grain)
- Variety of boundary conditions, constraints, etc.
- Developed by Sandia National Laboratories and many collaborators, such as Temple University

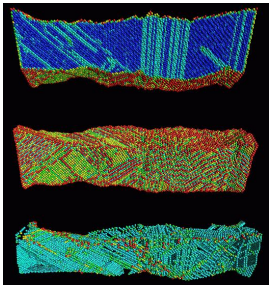
LAMMPS Development Pyramid



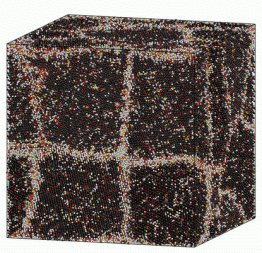
LAMMPS Use Cases



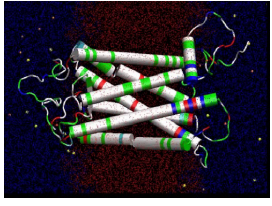
(a) Solid Mechanics



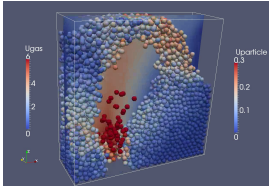
(b) Material Science



(c) Chemistry

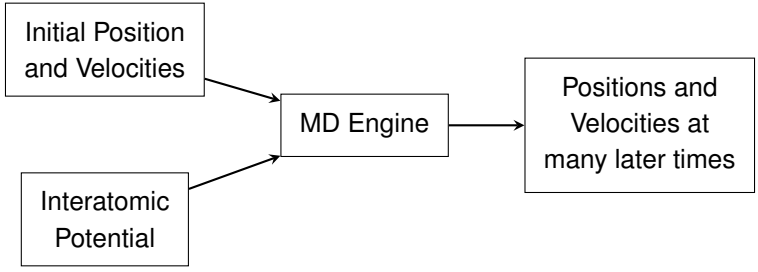


(d) Biophysics



(e) Granular Flow

What is Molecular Dynamics?



Mathematical Formulation

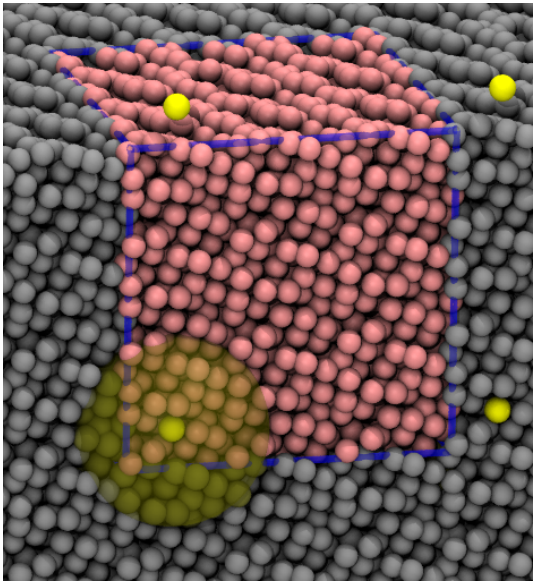
- classical mechanics
- atoms are point masses m_i
- positions, velocities, forces: $\mathbf{r}_i$, $\mathbf{v}_i$, $\mathbf{f}_i$
- Potential Energy Functions: $V(\mathbf{r}^N)$
- 6N coupled ODEs

$$\frac{d\mathbf{r}_i}{dt} = \mathbf{v}_i$$

$$\frac{d\mathbf{v}_i}{dt} = \frac{\mathbf{F}_i}{m_i}$$

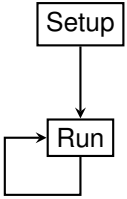
$$\mathbf{F}_i = -\frac{d}{d\mathbf{r}_i} V\left(\mathbf{r}^N\right)$$

Simulation of Liquid Argon with Periodic Boundary Conditions



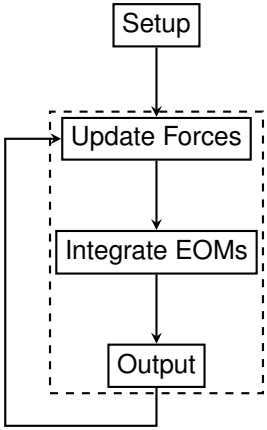
Outline

Basic Structure



- Setup domain & read in parameters and initial conditions
- Propagate system state over multiple time steps

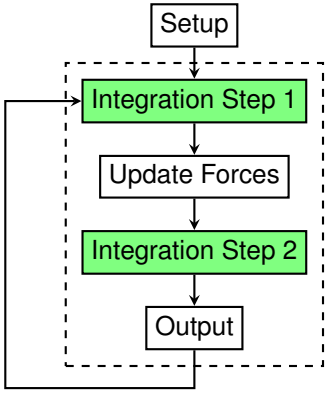
Basic Structure



Each time step consists of

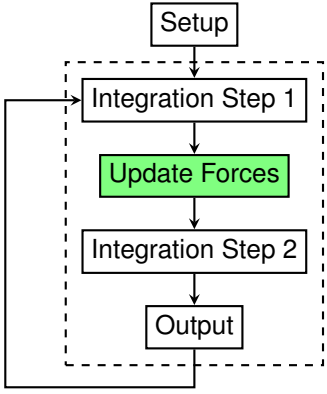
- Computing forces on all atoms
- Integrate equations of motion (EOMs)
- Output data to disk and/or screen

Velocity-Verlet Integration



- By default, Velocity-Verlet integration scheme is used in LAMMPS to propagate the positions of atoms
  1. Propagate all velocities for half a time step and all positions for a full time step
  2. Compute forces on all atoms to get accelerations
  3. Propagate all velocities for half a time step
  4. Output intermediate results if needed

Force Computation



Pairwise Interactions

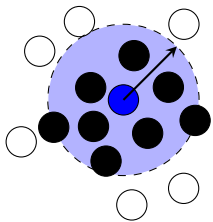
The total force acting on each atom i is the sum of all pairwise interactions with atoms j:

$$F_i = \sum_{j \neq i} F_{ij}$$

Cost

With n atoms the total cost of computing all forces F_{ij} would be $O(n^2)$

Force Computation



- cost of each individual force computation depends on selected interaction models
- many models operate using a cutoff distance r_c, beyond which the force contribution is zero

Lennard-Jones pairwise additive interaction:

$$F_{ij} = \begin{cases} 4\varepsilon \left[-12 \left(\frac{\sigma}{r_{ij}} \right)^{13} + 6 \left(\frac{\sigma}{r_{ij}} \right)^{7} \right] & r_{ij} < r_c \\ 0 & r_{ij} \geq r_c \end{cases}$$

Reducing the number of forces to compute

Verlet-Lists (aka. Neighbor Lists)

- each atom stores a list of neighboring atoms within a cutoff radius (larger than force cutoff)
- this list is valid for multiple time steps
- only forces between an atom and its neighbors are computed

Using Newton's Third Law of Motion

- Whenever a first body exerts a force **F** on a second body, the second body exerts a force $-\mathbf{F}$ on the first body.
- if we compute F_{ij}, we already know F_{ji}

$$F_{ji} = -F_{ij}$$

- $\Rightarrow$ **We can cut our force computations in half!**
- Neighbor lists only need to be half size

Reducing the number of forces to compute

Verlet-Lists (aka. Neighbor Lists)

- each atom stores a list of neighboring atoms within a cutoff radius (larger than force cutoff)
- this list is valid for multiple time steps
- only forces between an atom and its neighbors are computed

Note:
Finding neighbors is still an $O(n^2)$ operation!
But we can do better...

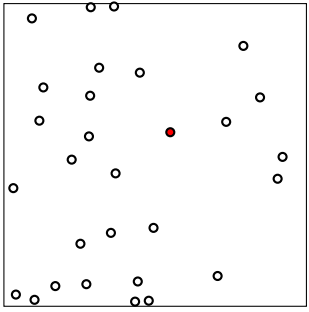
Using Newton's Third Law of Motion

- Whenever a first body exerts a force **F** on a second body, the second body exerts a force $-\mathbf{F}$ on the first body.
- if we compute F_{ij}, we already know F_{ji}

$$F_{ji} = -F_{ij}$$

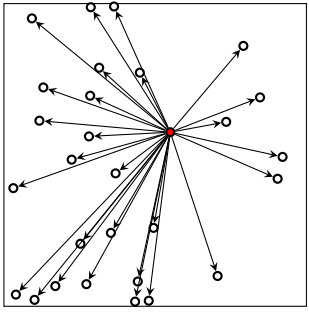
- $\Rightarrow$ **We can cut our force computations in half!**
- Neighbor lists only need to be half size

Cell List Algorithm



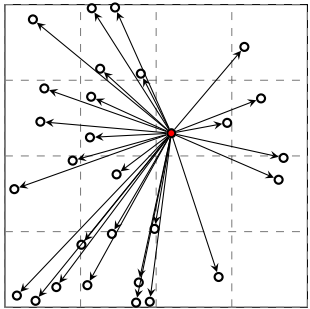
- We want to compute the forces acting on the red atom

Cell List Algorithm



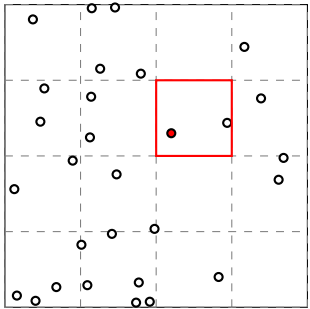
- Without any optimization, we would have look at all the atoms in the domain

Cell List Algorithm



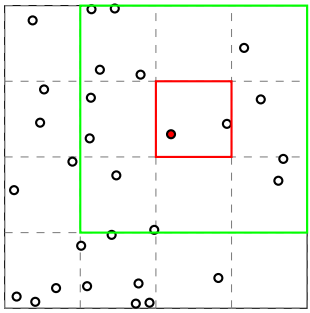
- When using Cell Lists we divide our domain into equal-size cells
- The cell size is proportional to the force cut-off

Cell List Algorithm



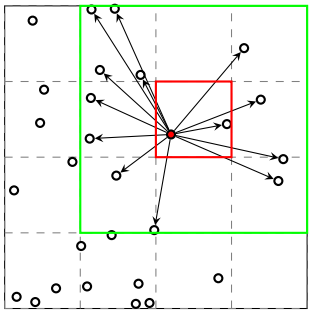
- Each atom is part of one cell

Cell List Algorithm

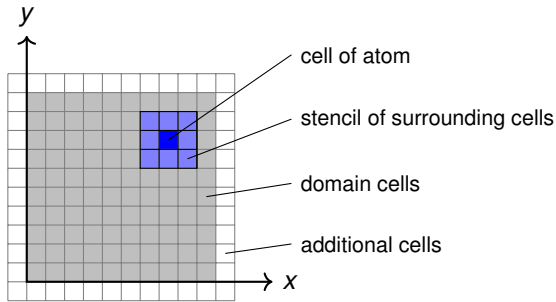


- Because of the size of each cell, we can assume any neighbor must be within the surrounding cells of an atom's parent cell

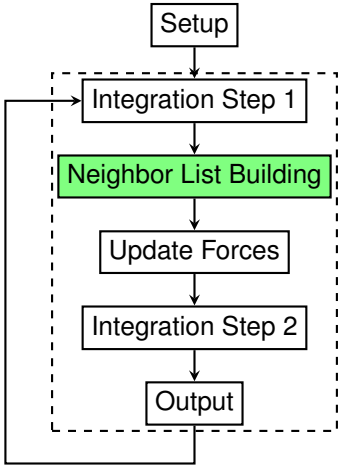
Cell List Algorithm



- Only a stencil of neighboring cells is searched when building an atom's neighbor list:
  - 9 cells in 2D
  - 27 cells in 3D
- To avoid corner cases additional cells are added to the data structure which allows using the same stencil for all cells.

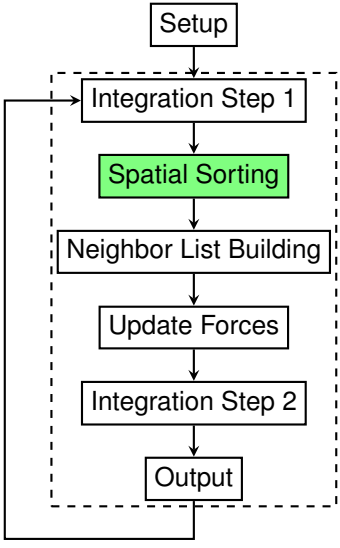


Finding Neighbors



- Combination of Cell-List and Verlet-List algorithm
- Reduces the number of atom pairs which have to be traversed

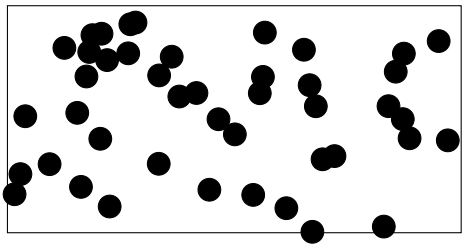
Improving caching efficiency



- atom data is periodically sorted
- atoms close to each other are placed in nearby memory blocks
- this can be efficently implemented by sorting by cells
- this improves cache efficiency during traversal

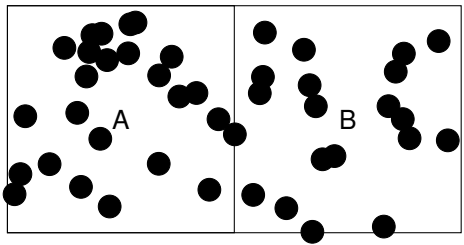
Outline

Geometric/Spatial Domain Decomposition



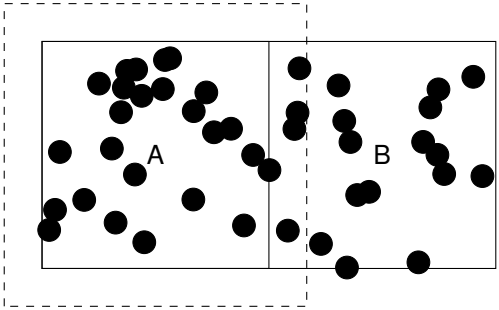
- LAMMPS uses spatial decomposition to scale over many thousands of cores

Geometric/Spatial Domain Decomposition



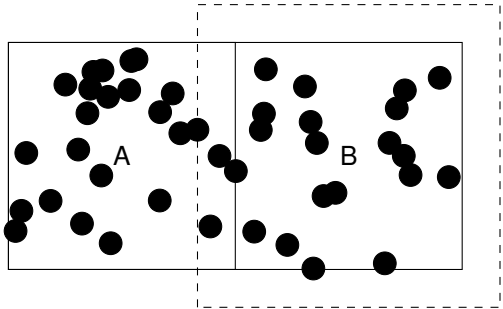
- the simulation box is split into multiple parts across available dimensions

Geometric/Spatial Domain Decomposition



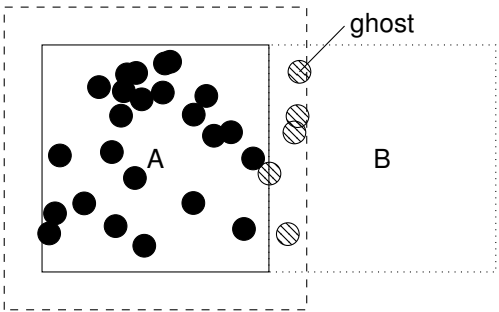
- each MPI process is responsible for computations on atoms within its subdomain
- each subdomain is extended with **halo regions** which duplicates information from adjacent subdomains

Geometric/Spatial Domain Decomposition



- each MPI process is responsible for computations on atoms within its subdomain
- each subdomain is extended with **halo regions** which duplicates information from adjacent subdomains

Geometric/Spatial Domain Decomposition

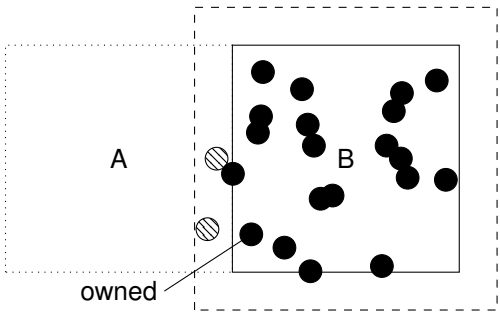


- each process only stores **owned atoms** and **ghost atoms**

owned atom: process is responsible for computation and update of atom properties

ghost atom: atom information comes from another process and is synchronized before each time step

Geometric/Spatial Domain Decomposition

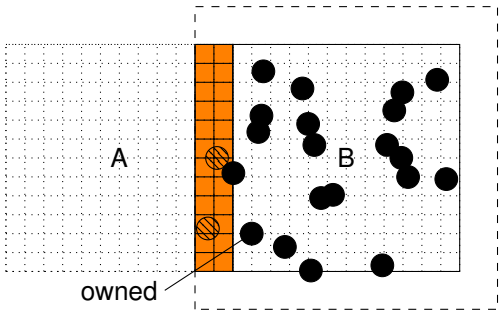


- each process only stores **owned atoms** and **ghost atoms**

owned atom: process is responsible for computation and update of atom properties

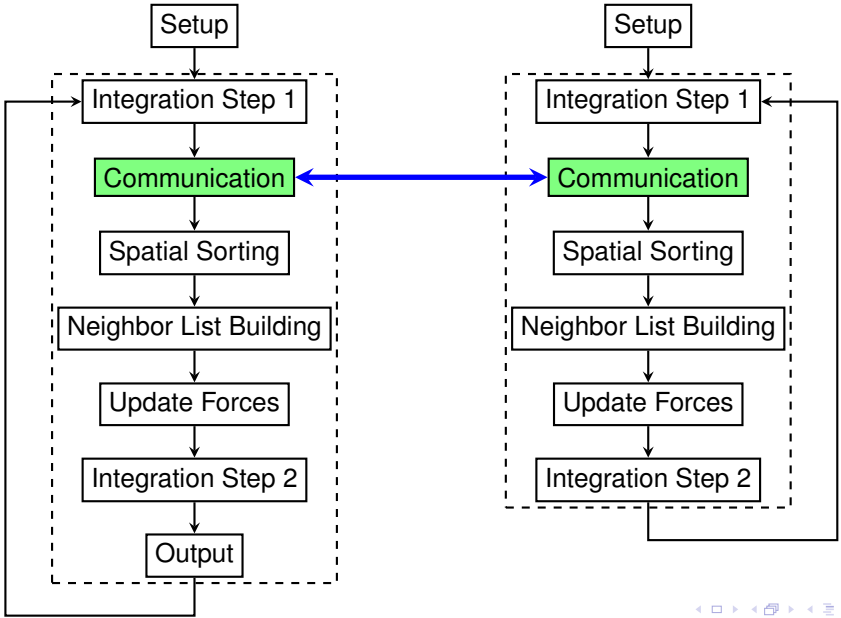
ghost atom: atom information comes from another process and is synchronized before each time step

Geometric/Spatial Domain Decomposition



- cell lists are used to determine which atoms need to be communicated

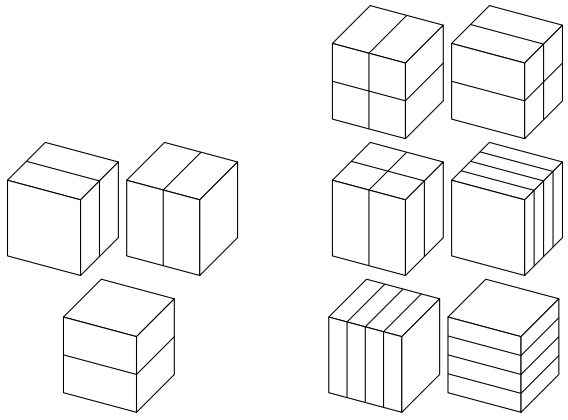
MPI Communication



MPI Communication

- communication happens after first integration step
- this is when atom positions have been updated
- atoms are migrated to another process if necessary
- positions (and other properties) of ghosts are updated
- Each process can have up to 6 communication partners in 3D
- With periodic boundary conditions it can also be its own communication partner (in this case it will simply do a copy)
- Both send and receive happen at the same time (`MPI_Irecv` & `MPI_Send`)

Decompositions

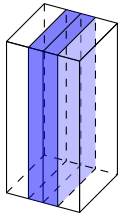


(a) $P = 2$ (b) $P = 4$

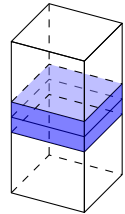
Figure: Possible domain decompositions with 2 and 4 processes

Communication volume

- The intersection of two adjacent halo regions determines the communication volume in that direction
- If you let LAMMPS determine your decomposition, it will try to minimize this volume



(a) *xz* halo region

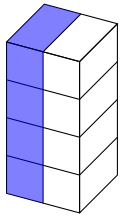


(b) *xy* halo region

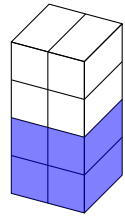
Figure: Halo regions of two different decompositions of a domain with an extent of 1x1x2.

Influence of Process Mapping

- The mapping of processes to physical hardware determines the amount of intra-node and inter-node communication
- (a) four processes must communicate with another node
- (b) two processes must communicate with another node



(a)

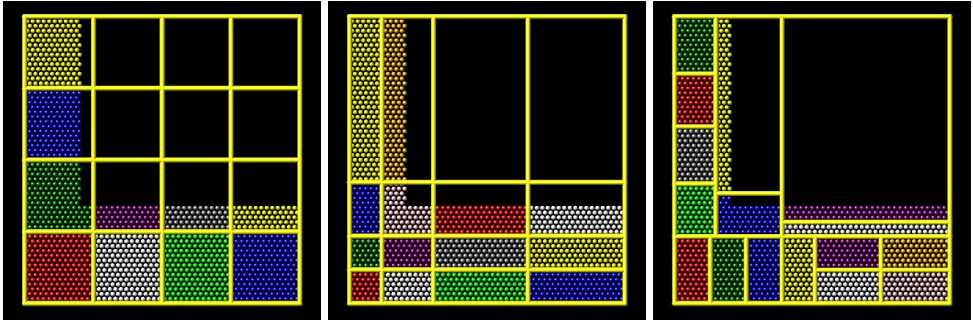


(b)

Figure: Two process mappings of a 1x2x4 decomposed domain.

Static and Dynamic load-balancing

- With the default geometic decomposition, balancing happens by shifting boundaries along each axis and creating a non-uniform grid
- Recursive Coordinate Bisectioning (RCB) gradually partitions space by inserting cut planes
- Each method tries to balance the number of atoms based on either their number or a weight function



(a) Uniform grid (b) Non-Uniform grid (c) RCB

Dynamic Load-Balancing in action

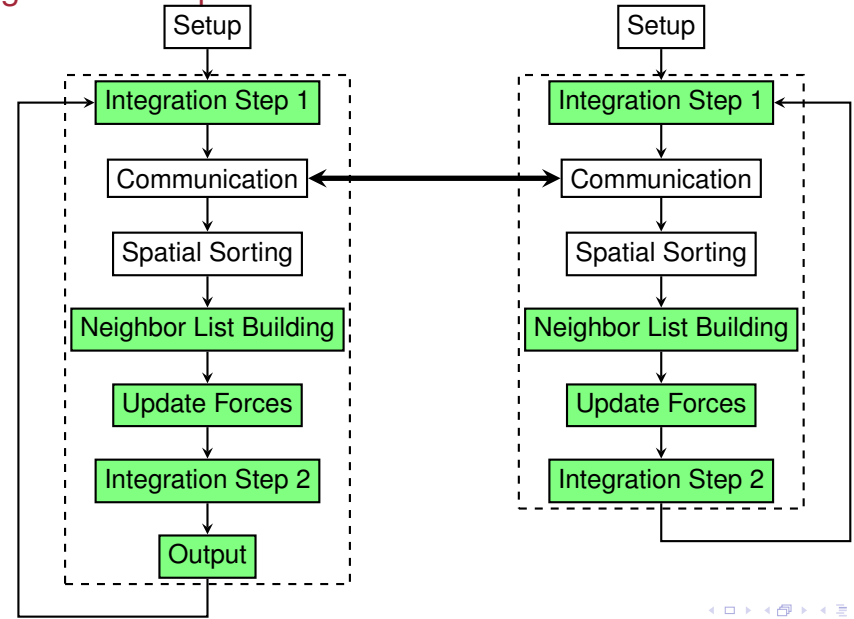
http://lammps.sandia.gov/movies/balance.mov

Outline

Combining MPI with OpenMP

- LAMMPS has a variety of accelerator packages (USER-OMP, KOKKOS, GPU, INTEL)
- USER-OMP package enables OpenMP threading within many simulation steps
- Multiple parts of the simulation loop be parallelized within a node
- Why would you want to do this?
  - Better utilization of node resources in some cases
  - Reduce MPI communication overhead, use less bandwidth
- For best performance we have to minimize synchronization points inside steps
- Parallelization of integration steps is trivial (simple loops)

Combining MPI with OpenMP



Newton's 3rd Law: Data Conflict

- Using Newton's 3rd law introduces a conflict
- Each force computation updates forces of two atoms: force F_{ij} and F_{ji}

```
#pragma omp parallel for
for(int i = 0; i < nlocal; i++) {
    // for each neighbor j
    {
        sys->fx[i] += fx;
        sys->fy[i] += fy;
        sys->fz[i] += fz;
        sys->fx[j] -= fx; // multiple threads could
        sys->fy[j] -= fy; // be writing j!
        sys->fz[j] -= fz;
    }
}
```

Avoiding conflict in force computation

Solutions:

- Disable Newton's 3rd law (Factor 2x!)
- Critical sections (bad)
- Atomics (better)
- Array reduction: each thread works on its own force array, which is later combined to the global force array (limited scalability per node)

Note

Array reduction is what is currently used in the USER-OMP package in LAMMPS. If multiple force computations are active, only the last one will perform the final array reduction.

Serial Neighbor List generation

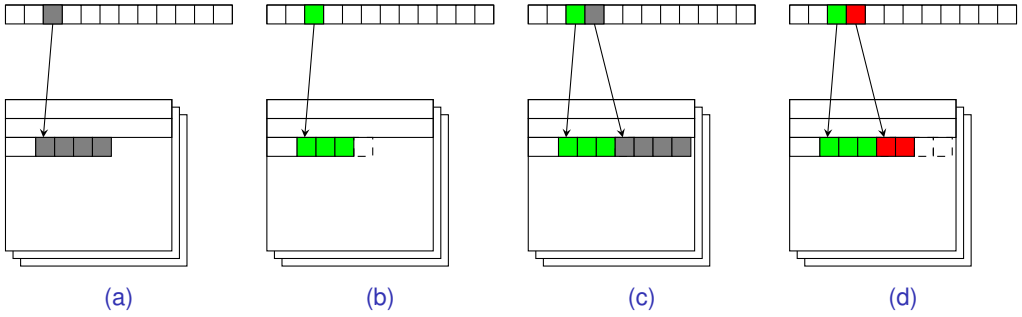


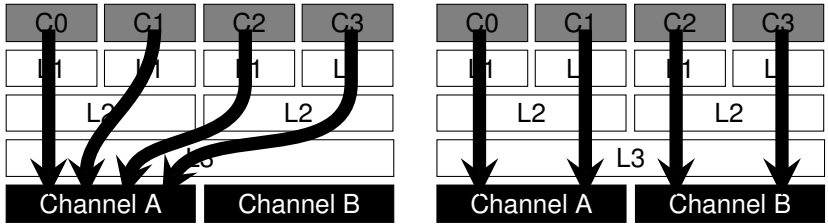
Figure: Memory allocation during neighbor list building using memory paging. (a) each atom gets enough memory space to store the maximum amount of neighbors. (b) the actual neighbors are stored and used space reported back to the allocator. (c) the allocation of the next neighbor list then starts at the beginning of the previous list. (d) this is repeated until all neighbor lists are generated.

If multiple threads work on this neighbor list generation, allocation of memory needs to be a critical section and has to be serialized.

First touch policy

- another problem is introduced due to the first-touch policy which is used by Linux
- `malloc` reserves a memory block from the OS
- however, the actual mapping of allocated memory to physical memory happens when you try to access **any bit in that memory block for the first time**
- the kernel will place this memory block on physical memory which is near the CPU core executing the accessing thread
- **that means the thread that first accesses a memory location determines where a block of memory is placed**

Memory contention



(a) global memory access (b) thread-local memory access

Figure: Memory contention can limit scalability of threaded code. If all threads access memory which is located closer to a single core, the effective available bandwidth is reduced.

Multi-threaded neighbor list generation

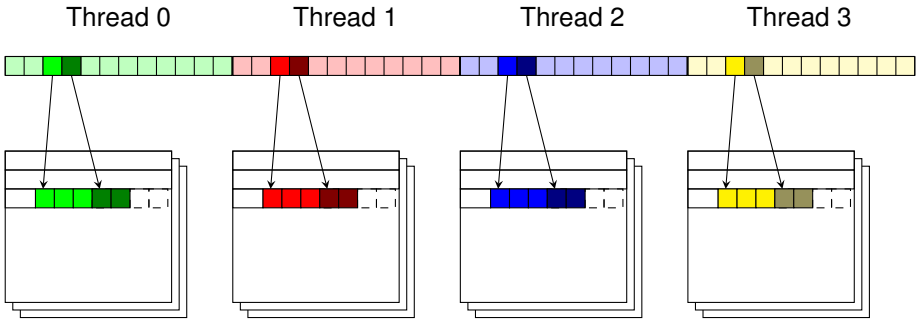


Figure: Multi-threaded generation of neighbor list using a page data structure for each thread. Each thread works on a sub-sequence of the atom list using its own memory pages to allocate neighbor lists.

General performance recommendations for hybrid codes

- Use one MPI process per socket / memory channel
- Maximize utilization per MPI process by using OpenMP threads
- Bind threads to cores and ensure data locality
- Minimize load-imbalance and synchronization