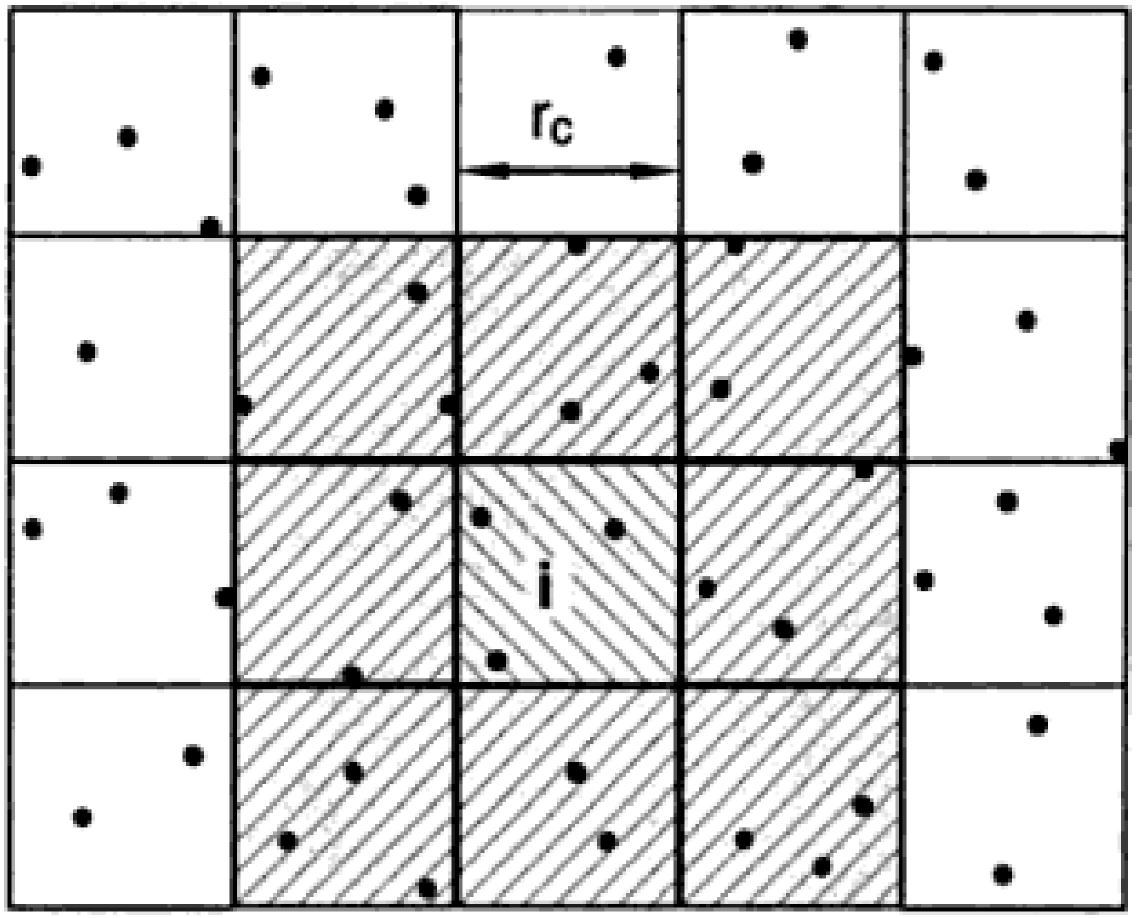
Particle Simulations

- Examples
  - Molecular dynamics (particles are atoms)
  - Astrophysical simulations (particles are stars or galaxies)
- Two main steps:
  - Computing the forces on each particle (using a model)
  - Updating the position of each particle (integration)

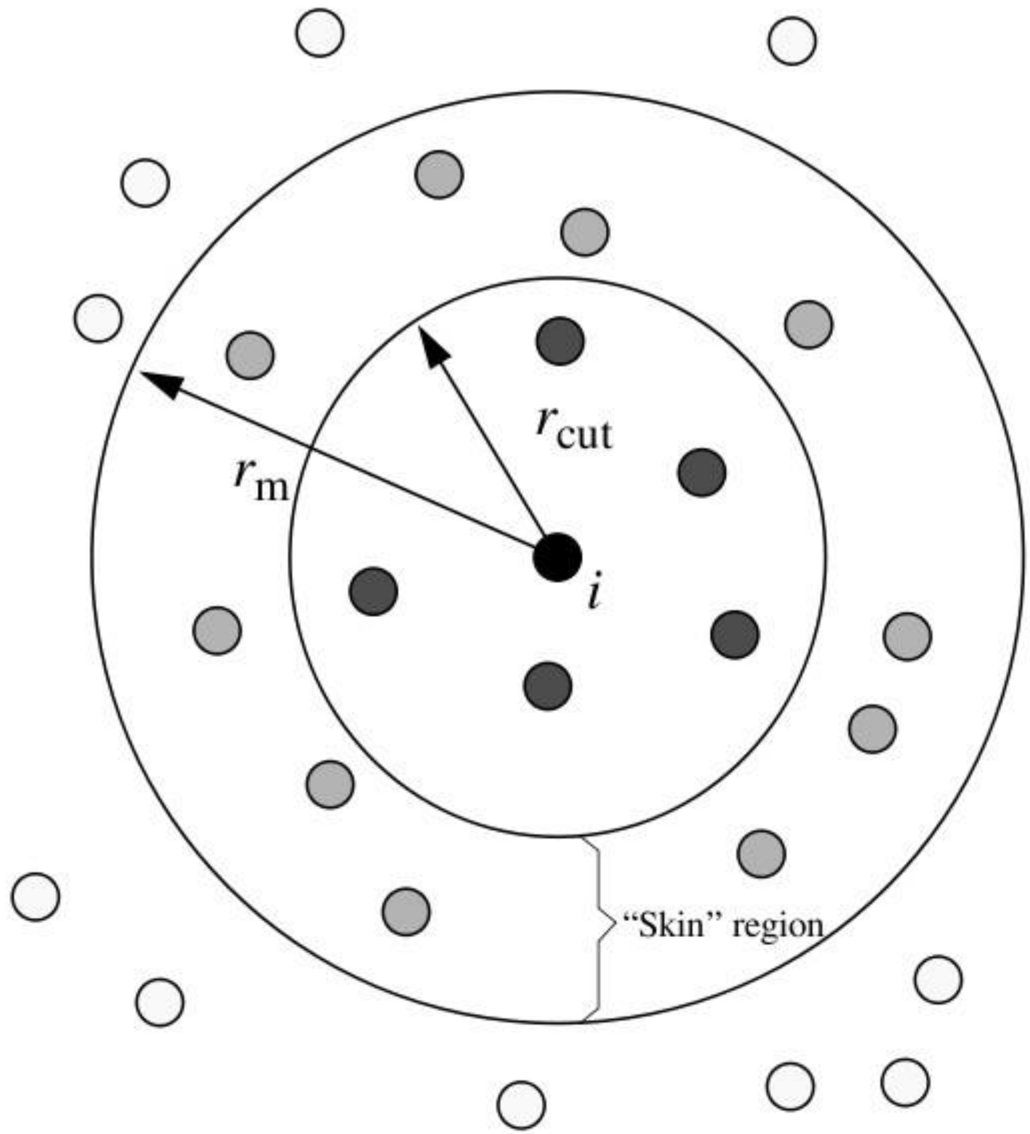
Computing forces between “nearby” particles

- Naïve algorithm is $O(n^2)$ for n particles
- Asymptotically faster algorithms are possible

Cell List



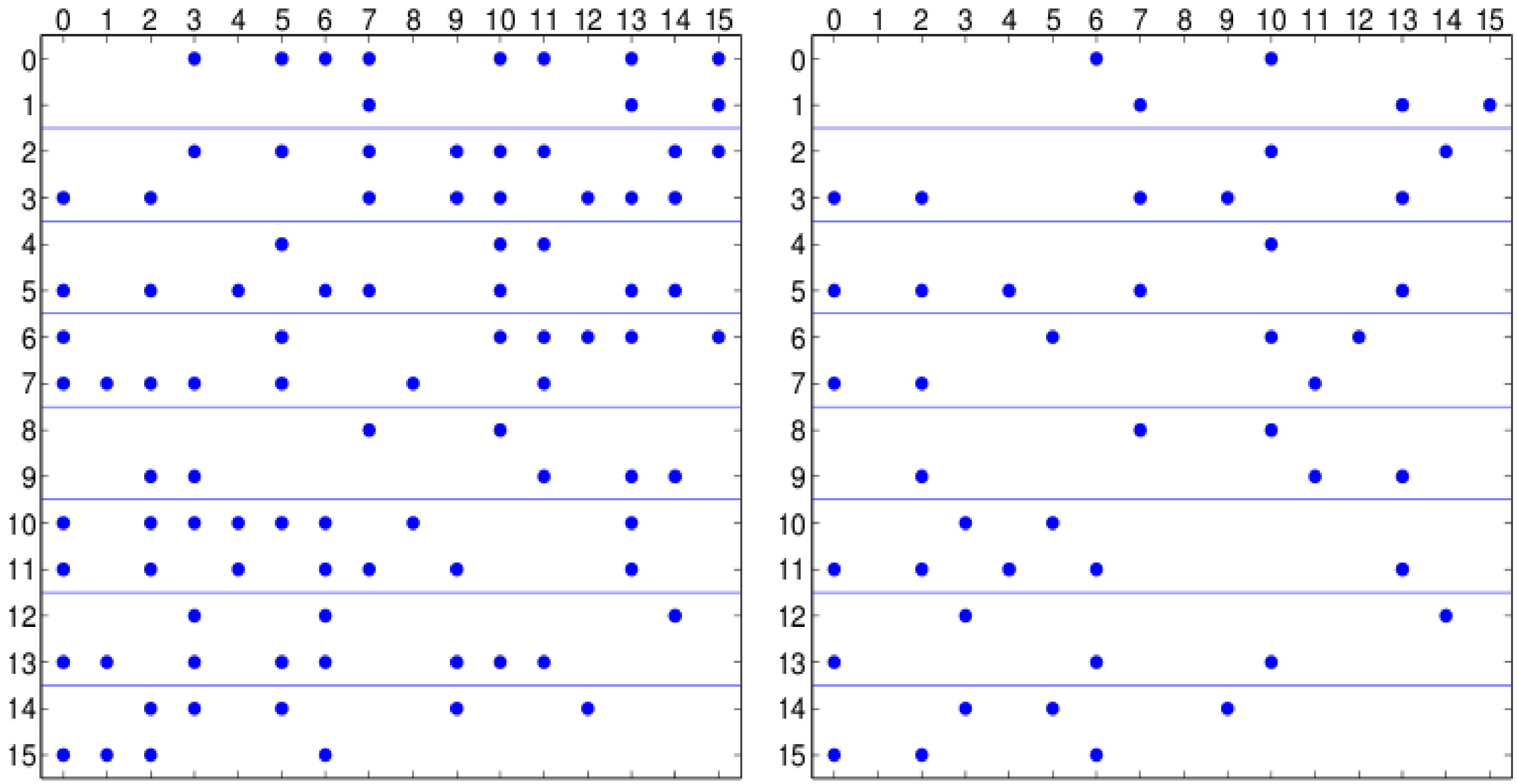
Verlet neighbor list



Parallelization of short-range forces

- Atom decompositions
  - Same process computes total force for given atom
- Force decompositions
  - Process computes force contribution depending on id of owner process
- Spatial decompositions
  - Ownership depends on spatial decomposition
- Neutral Territory methods
  - Force may be computed by process that does not own the participating atoms

Atom Decompositions



(a) Force matrix.

(b) Force matrix, redundancies removed.

Atom decomposition time step

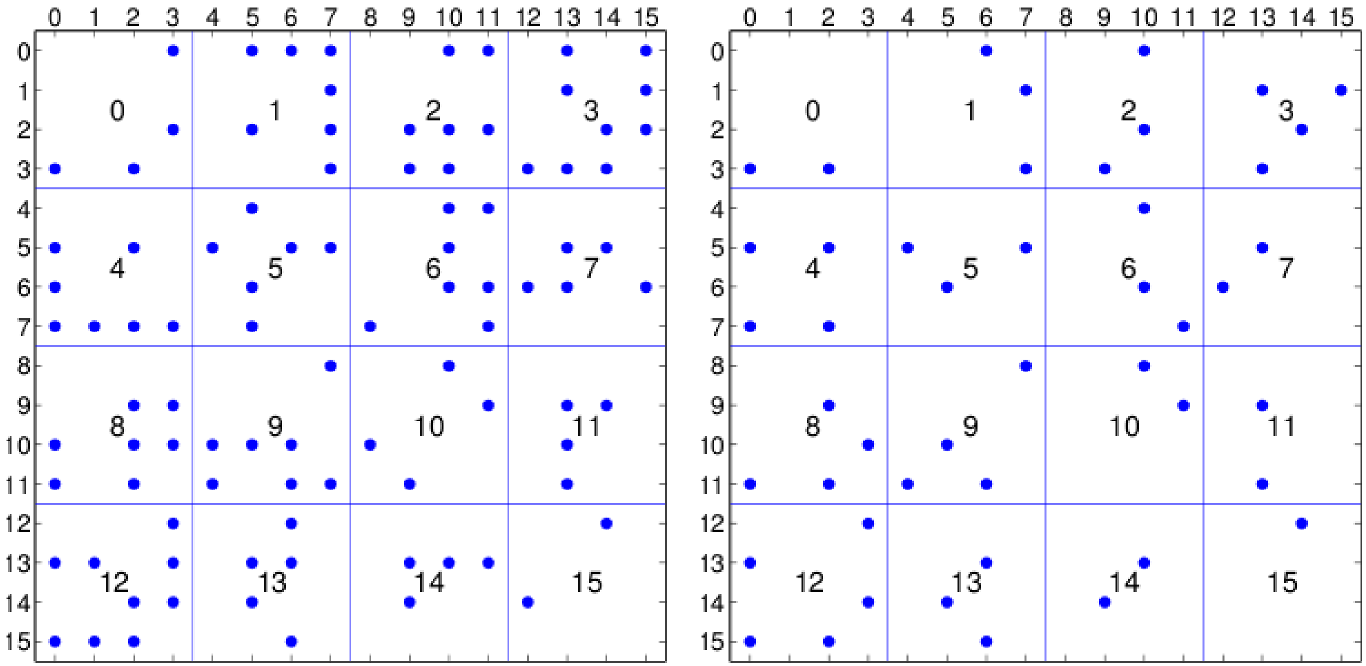
Algorithm 1 Atom decomposition time step

1: send/receive particle positions to/from all other processors
2: (if nonbonded cutoffs are used) determine which nonbonded forces need to be computed
3: compute forces for particles assigned to this processor
4: update positions (integration) for particles assigned to this processor

Algorithm 2 Atom decomposition time step, without redundant calculations

1: send/receive particle positions to/from all other processors
2: (if nonbonded cutoffs are used) determine which nonbonded forces need to be computed
3: compute *partial* forces for particles assigned to this processor
4: send particle forces needed by other processors and receive particle forces needed by this processor
5: update positions (integration) for particles assigned to this processor

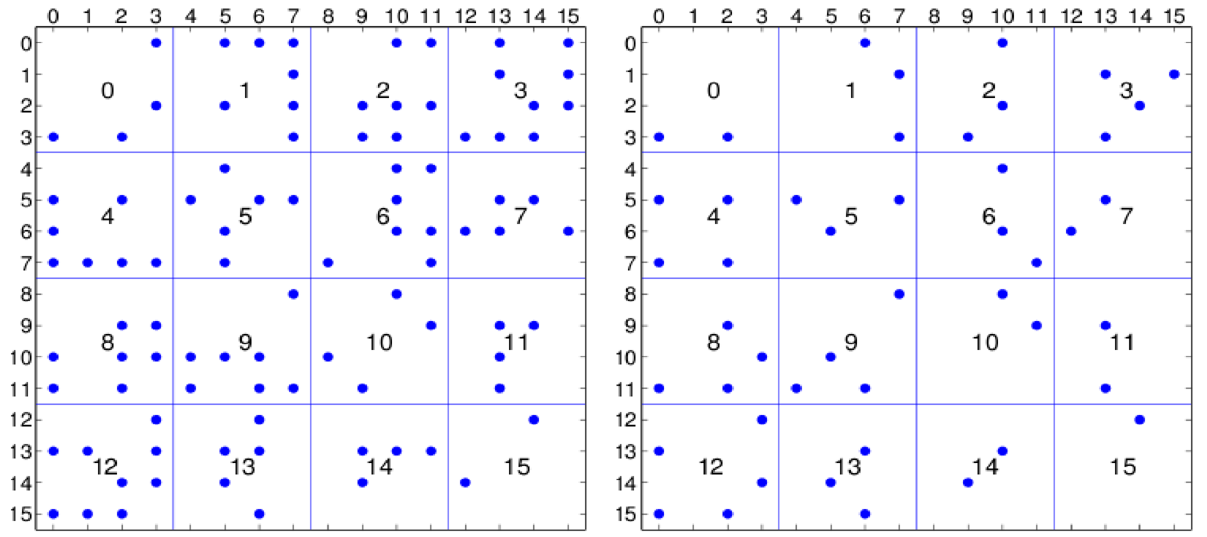
Force Decompositions



(a) Force matrix.

(b) Force matrix, redundancies removed.

Force decomposition time step



Algorithm 3 Force decomposition time step

1: send positions of my assigned particles which are needed by other processors; receive *row* particle positions needed by my processor (this communication is between processors in the same processor row, e.g., processor 3 communicates with processors 0, 1, 2, 3)

2: receive *column* particle positions needed by my processor (this communication is generally with processors in another processor row, e.g., processor 3 communicates with processors 12, 13, 14, 15)

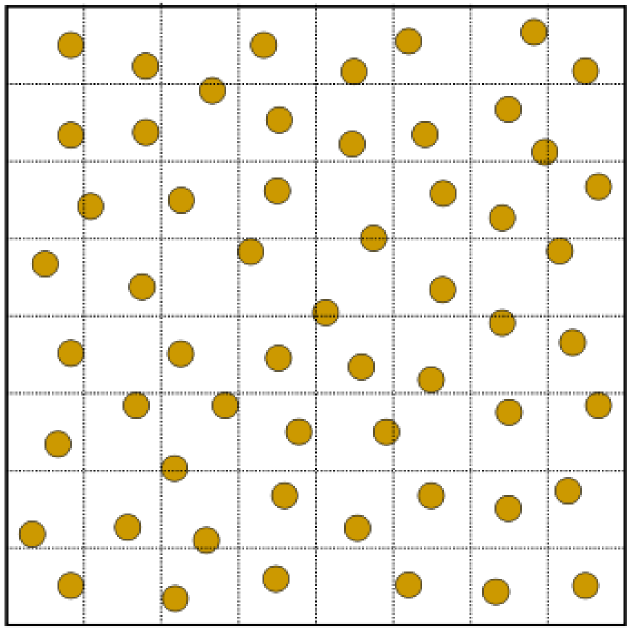
3: (if nonbonded cutoffs are used) determine which nonbonded forces need to be computed

4: compute forces for my assigned particles

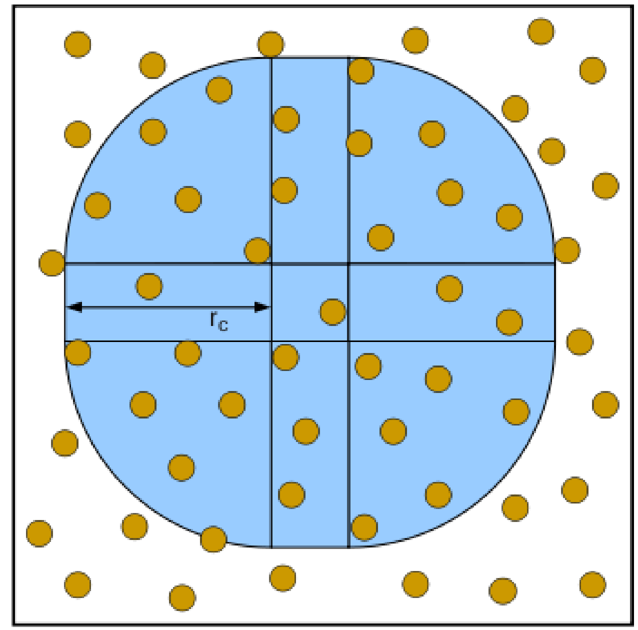
5: send forces needed by other processors; receive forces needed for my assigned particles (this communication is between processors in the same processor row, e.g., processor 3 communicates with processors 0, 1, 2, 3)

6: update positions (integration) for my assigned particles

Spatial decomposition



(a) Decomposition into 64 cells.

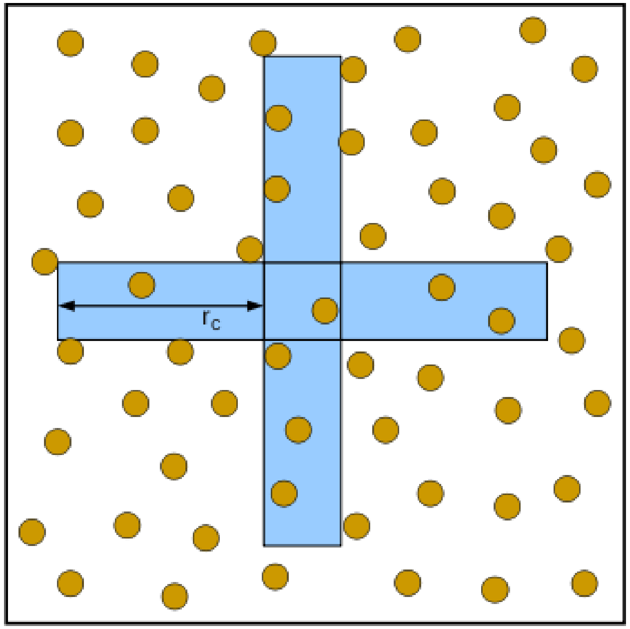


(b) Import region for one cell.

Algorithm 5 Spatial decomposition time step

1: send positions needed by other processors for particles in their import regions; receive positions for particles in my import region
2: compute forces for my assigned particles
3: update positions (integration) for my assigned particles

Neutral territory method



Algorithm 6 Neutral territory method time step

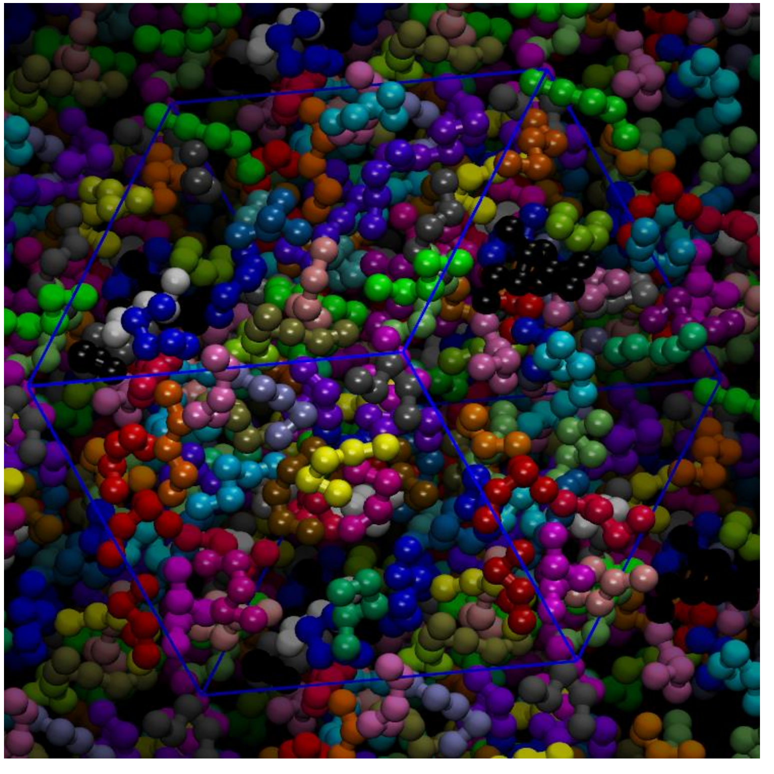
1: send and receive particle positions corresponding to import regions
2: compute forces assigned to this processor
3: send and receive forces required for integration
4: update positions (integration) for particles assigned to this processor

Summary

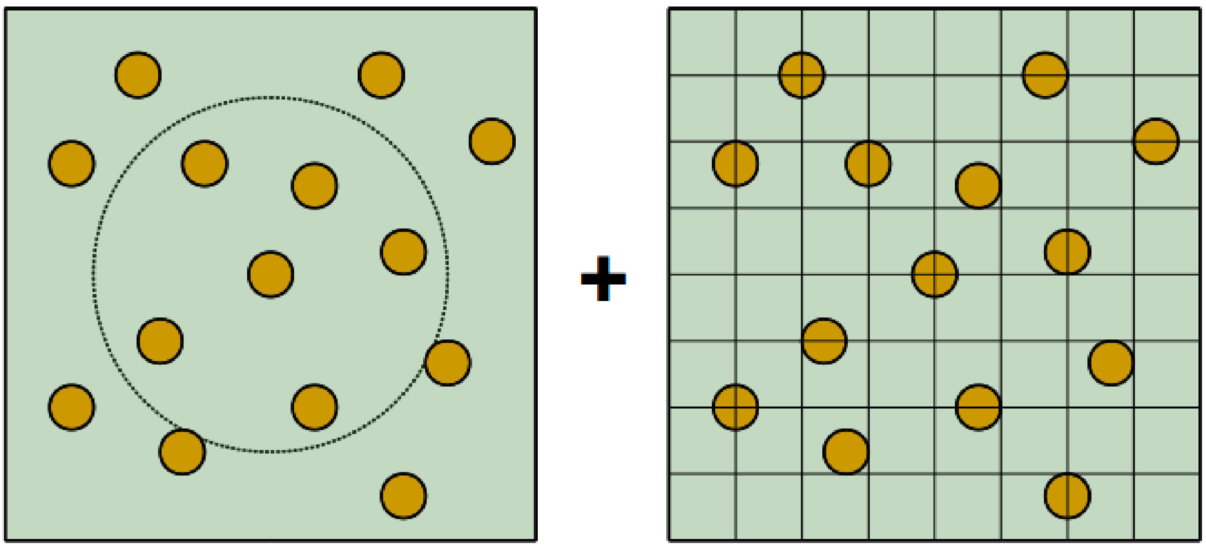
- Atom decompositions
  - Processes need to communicate with all other processes
- Force decompositions
  - Processes communicate with O(sqrt(p)) other processes
- Spatial decompositions
  - Processes only communicate with “neighbor” processes
- Neutral Territory methods
  - Communication volume is reduced compared to spatial decompositions

Methods for long-range forces

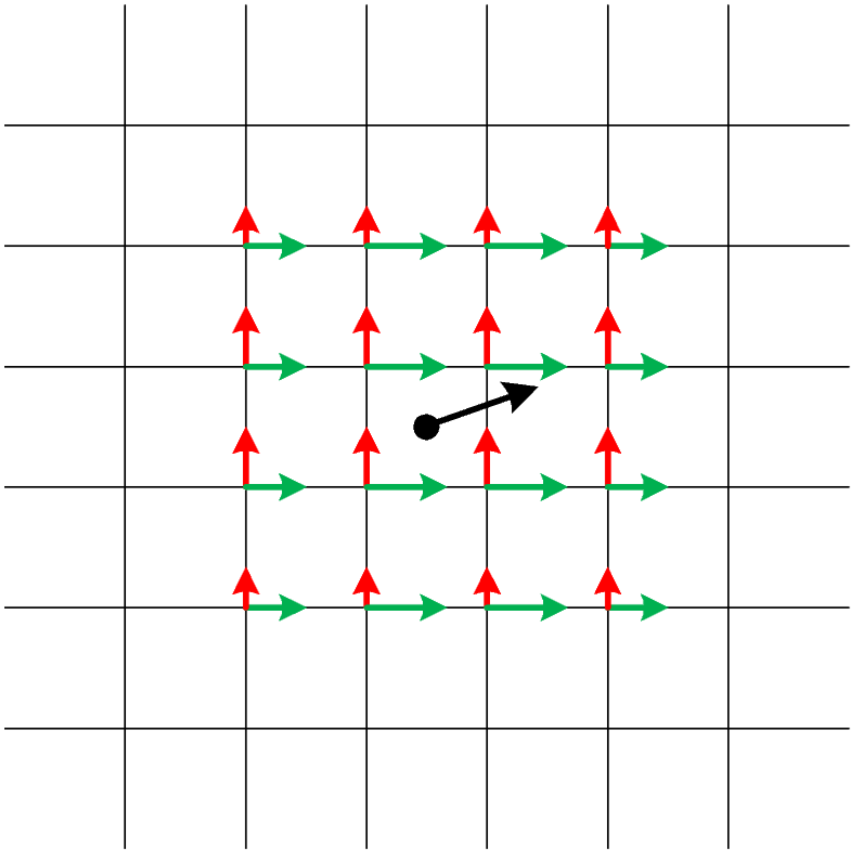
- Fast multipole method
- Particle-mesh Ewald (for periodic conditions)



Particle simulations: short range forces and long range forces



Spreading force onto a mesh
(4x4 spreading)



GPU parallelization of spreading operation

- Input is a list of positions and forces
- Output is a 2D array of forces

GPU parallelization of spreading operation

- Input is a list of positions and forces
- Output is a 2D array of forces

- Each thread (of a thread block) gets one position/force and sums into the 2D array of forces (using atomic operations)
- Problem is write contention in the 2D array

- Propose a better solution!

Be grid-centric, not particle-centric

- Instead of particles ***scattering*** data to grid points, grid points should be ***gathering*** data from particles

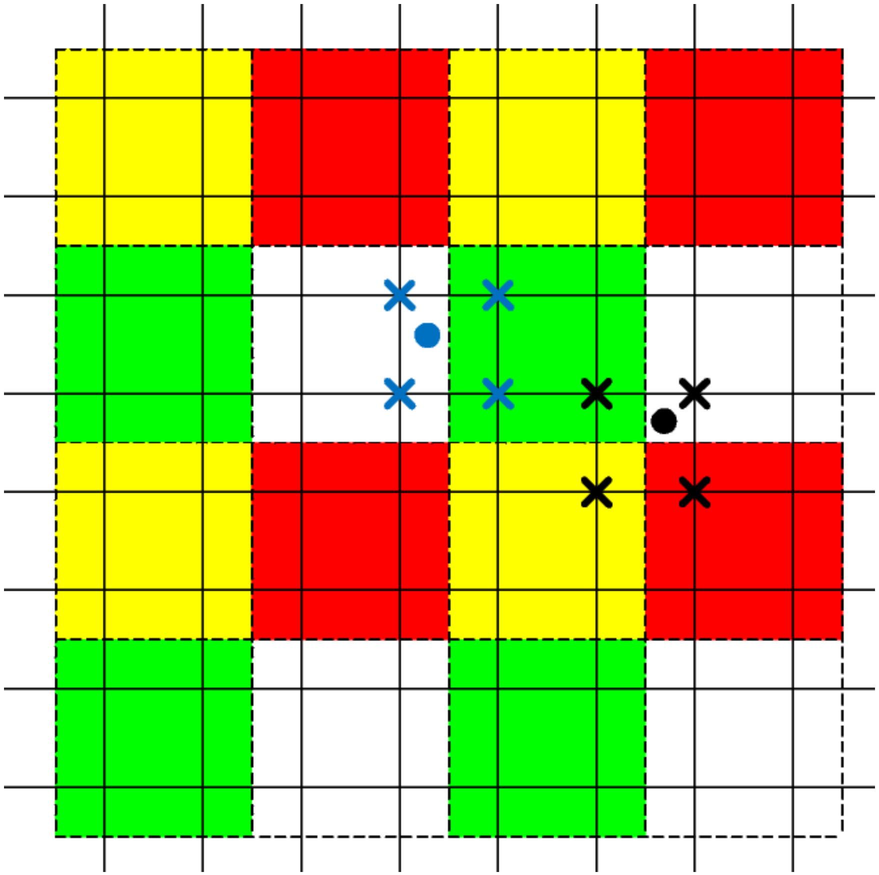
- One thread per grid point
- How do grid points gather data efficiently? (Each grid point does not examine all particles)

Placement, accumulation, overflow

- Map particles to 3D array of grid points in global memory (must still use atomics, but need 4x4x4 times fewer atomics)
- If more than one particle is assigned to a grid point, then use overflow list for each grid point
- GPU kernel, one thread per grid point. Gather 4x4x4 region from 3D array of grid points. Memory access is somewhat coalesced.
- Overflow lists are then processed separately.

Ref: Harvey and de Fabritiis, 2009

Coloring of the mesh (2x2 spreading)



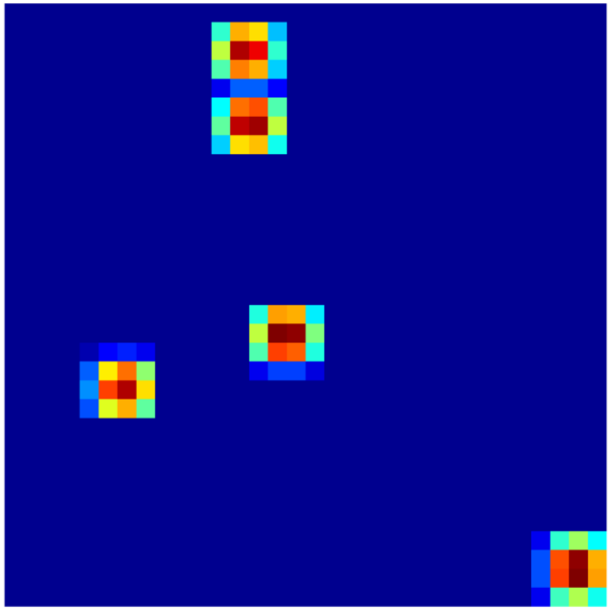
- The four independent sets are shown in different colors.
- Two particles (dots) from different blocks in the same independent set cannot spread to the same mesh points (crosses).
- One thread processes particles in one block. Threads with the same color can compute in parallel.

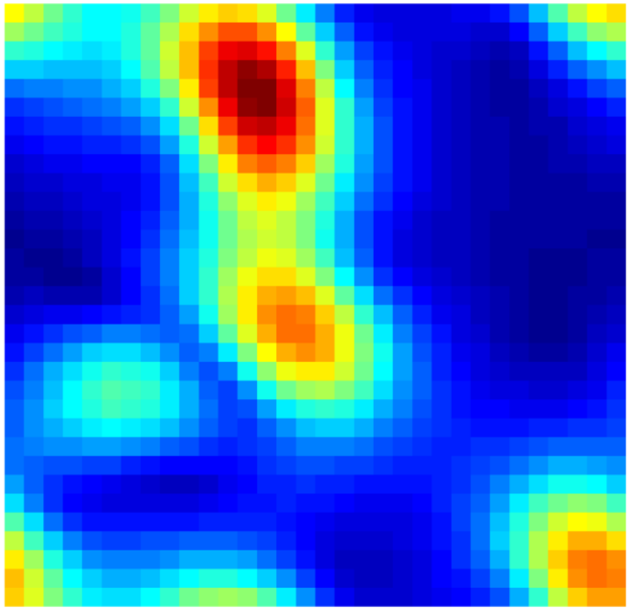
Similar problem: y = Ax

- Sparse matrix-vector multiply, when the matrix is partitioned by columns

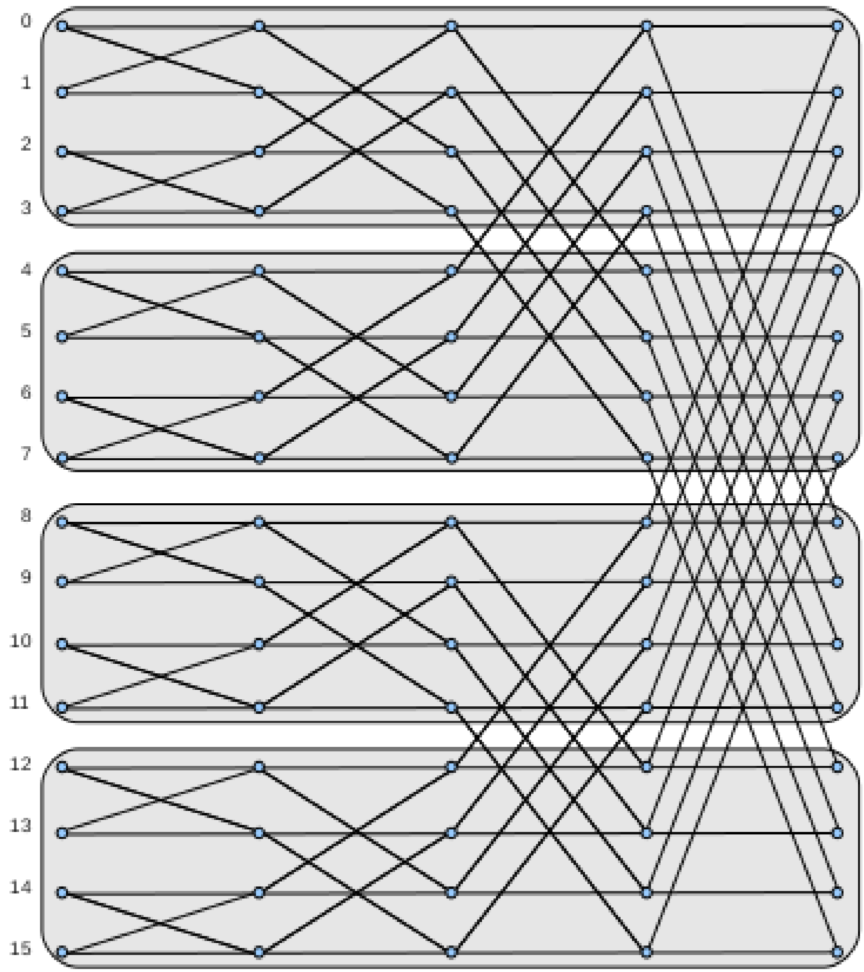
- One thread has one column of the matrix and one element of x
- Vector y is the sum of sparse vectors

Use FFTs to find solution on a mesh

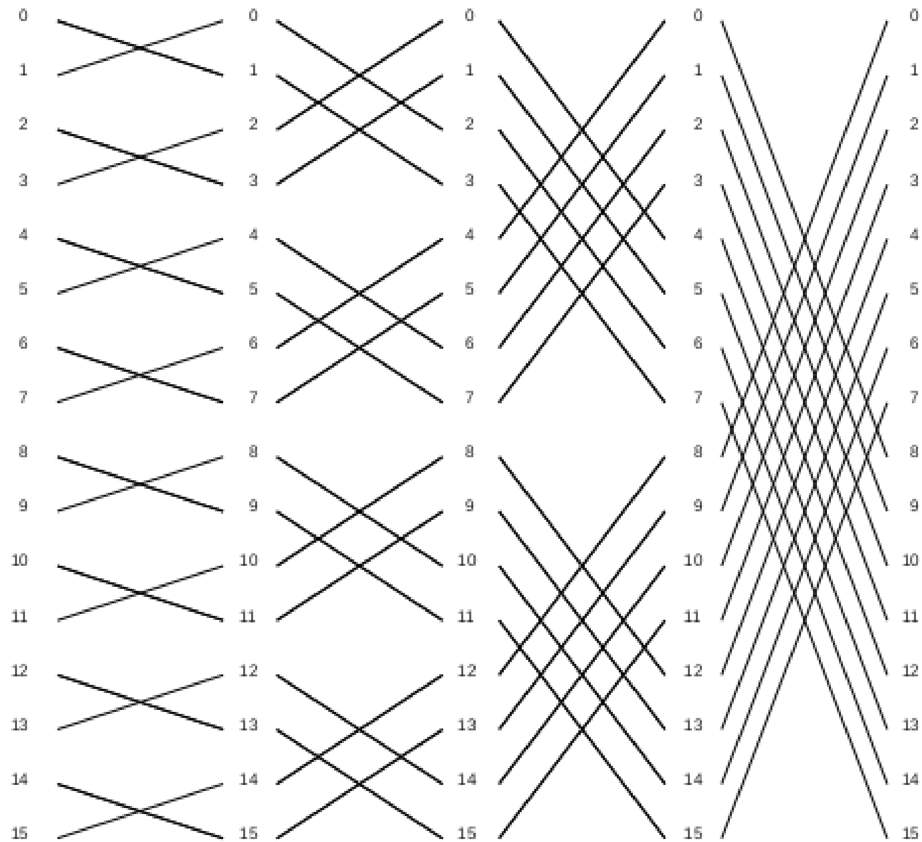




1-D FFT data flow diagram



1-D FFT with Transpose



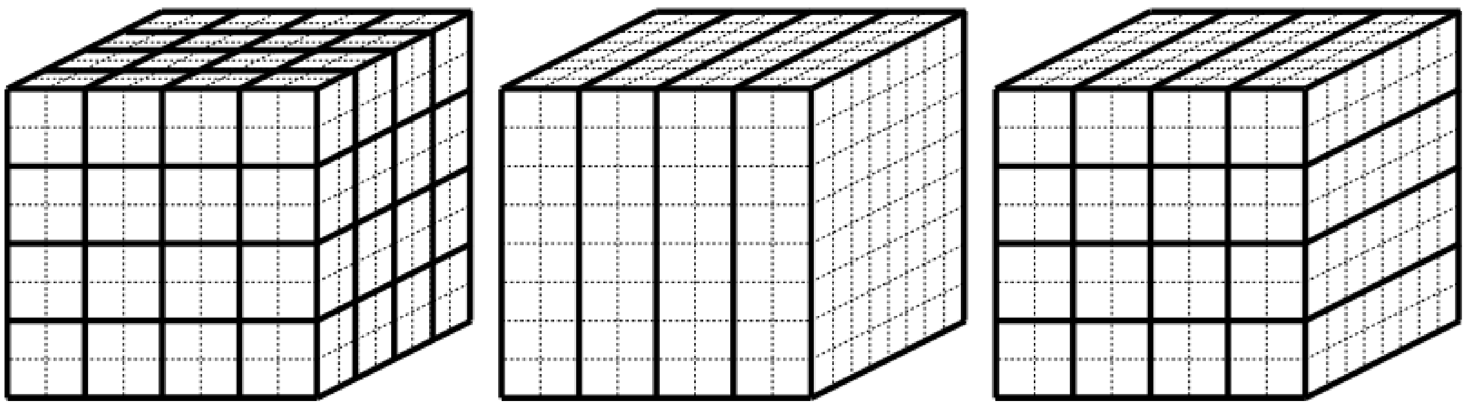
(a) Data flow diagram (shown without horizontal lines for clarity) for 1-D FFT for 16 points.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

(b) Partitioning of the indices before (left) and after (right) the transpose.

3-D FFT partitionings



(a) Block Decomposition (b) Slab Decomposition (c) Pencil Decomposition