CHAPTER 18

Electrostatic potential map

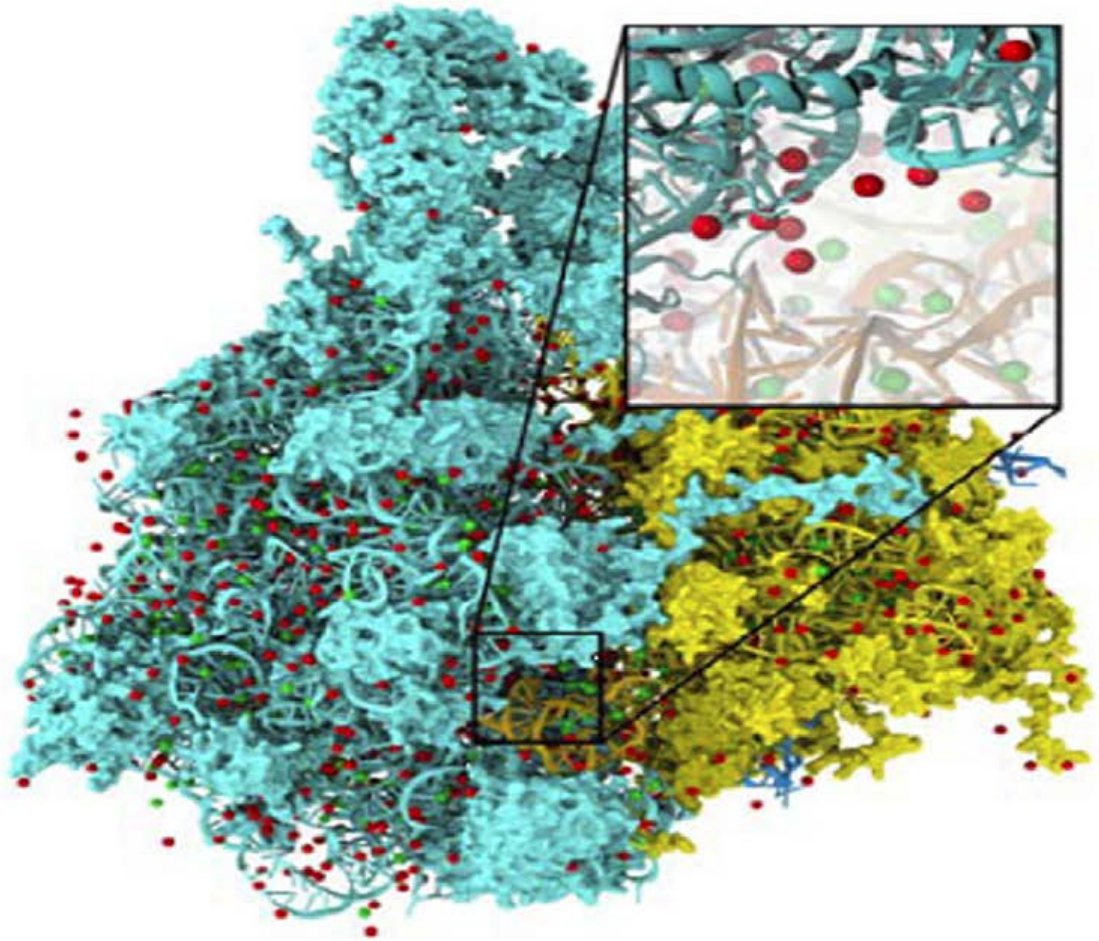


FIGURE 18.1

Electrostatic potential map used in building stable structures for molecular dynamics simulation.

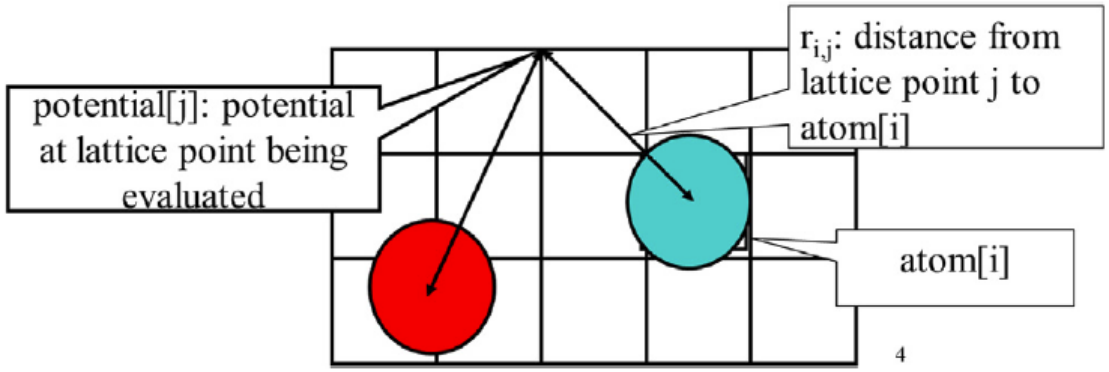


FIGURE 18.2

The contribution of atom[i] to the electrostatic potential at lattice point j (potential[j]) is atom[i]. charge/r_{ij}. In the direct coulomb summation method, the total potential at lattice point j is the sum of contributions from all atoms in the system.

```
01 void cenergy(float *energygrid, dim3 grid, float gridspacing, float z,
02                  const float *atoms, int numatoms) {
03   int atomarrdim = numatoms * 4; //x,y,z, and charge info for each atom
04   for (int j=0; j<grid.y; j++) {
        // calculate y coordinate of the grid point based on j
05      float y = gridspacing * (float) j;
06      for (int i=0; i<grid.x; i++) {
           // calculate x coordinate based on i
07         float x = gridspacing * (float) i;
08         float energy = 0.0f;
09         for (int n=0; n<atomarrdim; n+=4) {
10            float dx = x - atoms[n  ];
11            float dy = y - atoms[n+1];
12            float dz = z - atoms[n+2];
13            energy += atoms[n+3] / sqrtf(dx*dx + dy*dy+ dz*dz);
14         }
15         energygrid[grid.x*grid.y*z + grid.x*j + i] = energy;
16      }
17    }
18  }
```

FIGURE 18.3

An unoptimized direct Coulomb summation C code for a two-dimensional slice.

```
01 void cenergy(float *energygrid, dim3 grid, float gridspacing, float z,
02              const float *atoms, int numatoms) {
03   int atomarrdim = numatoms * 4; //x,y,z, and charge info for each atom
     // starting point of the slice in the energy grid
04   int grid_slice_offset = (grid.x*grid.y*z) / gridspacing;
     //  calculate potential contribution of each atom
05   for (int n=0; n<atomarrdim; n+=4) {
06      float dz = z - atoms[n+2];  // all grid points in a slice have the same
07      float dz2 = dz*dz;          // z  value, no recalculation in inner loops
08      float charge = atoms[n+3];
09      for (int j=0; j<grid.y; j++) {
10         float y = gridspacing * (float) j;
11         float dy = y - atoms[n+1];  // all grid points in a row have the same
12         float dy2 = dy*dy;          // y value
13         int grid_row_offset =  grid_slice_offset+ grid.x*j;
14         for (int i=0; i<grid.x; i++) {
15            float x = gridspacing * (float) i;
16            float dx = x - atoms[n    ];
17            energygrid[grid_row_offset+i] += charge / sqrtf(dx*dx + dy2+ dz2);
18         }
19      }
20    }
21  }
```

FIGURE 18.4

An optimized direct Coulomb summation C code for a two-dimensional slice.

```
01  __constant__ float atoms[CHUNK_SIZE*4];
02  void  __global__ cenergy(float *energygrid, dim3 grid, float gridspacing,
03                           float z) {
04    int n = (blockIdx.x * blockDim .x + threadIdx.x) * 4;
05    float dz = z-atoms[n+2];  // all grid points in a slice have the same
06    float dz2 = dz*dz;        // z value
      // starting position of the slice in the energy grid
07    int grid_slice_offset = (grid.x*grid.y*z) / gridspacing;
08    float charge = atoms[n+3];
09    for (int j=0; j<grid.y; j++) {
10       float y = gridspacing * (float) j;
11       float dy = y-atoms[n+1];   // all grid points in a row have the same
12       float dy2 = dy*dy;          // y value
         // starting position of the row in the energy grid
13       int grid_row_offset =  grid_slice_offset+ grid.x*j;
14       for (int i=0; i<grid.x; i++) {
15           float x = gridspacing * (float) i;
16           float dx = x - atoms[n    ];
17           atomicAdd(&energygrid[grid_row_offset+i],
18                     charge / sqrtf(dx*dx+dy2+dz2));
19       }
20    }
21  }
```

FIGURE 18.5

Direct Coulomb summation kernel using the scatter approach.

```
01   __constant__ float atoms[CHUNK_SIZE*4];
02  void __global__ cenergy(float *energygrid, dim3 grid, float gridspacing,
03                          float z, int numatoms) {
04     int i = blockIdx.x * blockDim.x + threadIdx.x;
05     int j = blockIdx.y * blockDim.y + threadIdx.y;
06     int atomarrdim = numatoms * 4;
07     int k = z / gridspacing;
08     float y = gridspacing * (float) j;
09     float x = gridspacing * (float) i;
10     float energy = 0.0f;
       // calculate potential contribution from all atoms
11     for (int n=0; n<atomarrdim; n+=4) {
12        float dx = x - atoms[n  ];
13        float dy = y - atoms[n+1];
14        float dz = z - atoms[n+2];
15        energy += atoms[n+3] / sqrtf(dx*dx + dy*dy + dz*dz);
16     }
17     energygrid[grid.x*grid.y*k + grid.x*j + i] += energy;
18   }
```

FIGURE 18.6

Direct Coulomb summation kernel using the gather approach.

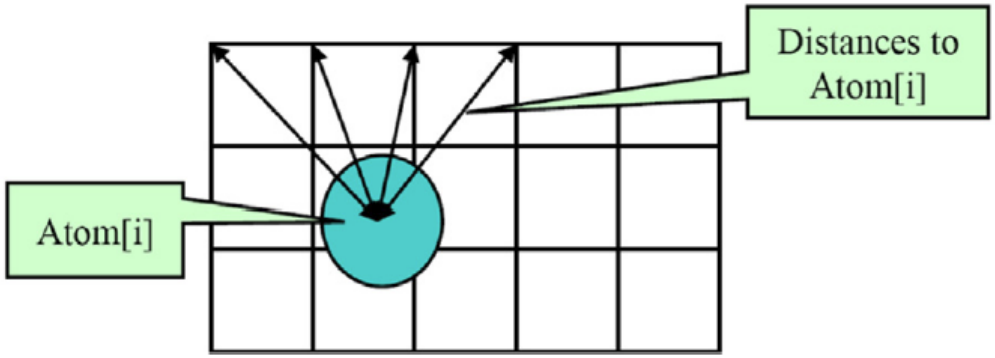


FIGURE 18.7

Reusing computation results among multiple grid points.

```
01  __constant__ float atoms[CHUNK_SIZE*4];
02  #define COARSEN_FACTOR 4
03  void __global__ cenergy(float *energygrid, dim3 grid, float gridspacing,
04                          float z, int numatoms) {
05     int i = blockIdx.x * blockDim.x*COARSEN_FACTOR + threadIdx.x;
06     int j = blockIdx.y * blockDim.y + threadIdx.y;
07     int atomarrdim = numatoms * 4;
08     int k = z / gridspacing;
09     float y = gridspacing * (float) j;
10     float x = gridspacing * (float) i;
11     float energy0 = 0.0f;
12     float energy1 = 0.0f;
13     float energy2 = 0.0f;
14     float energy3 = 0.0f;
15     // calculate potential contribution from all atoms
16     for (int n=0; n<atomarrdim; n+=4) {
17        float dx0 = x - atoms[n  ];
18        float dx1 = dx0 +   gridspacing;
19        float dx2 = dx0 + 2*gridspacing;
20        float dx3 = dx0 + 3*gridspacing;
21        float dy = y - atoms[n+1];
22        float dz = z - atoms[n+2];
23        float dysqdzsq = dy*dy + dz*dz;
24        float charge = atoms[n+3];
25        energy0 += charge / sqrtf(dx0*dx0 + dysqdzsq);
26        energy1 += charge / sqrtf(dx1*dx1 + dysqdzsq);
27        energy2 += charge / sqrtf(dx2*dx2 + dysqdzsq);
28        energy3 += charge / sqrtf(dx3*dx3 + dysqdzsq);
29     }
30     energygrid[grid.x*grid.y*k + grid.x*j + i  ] += energy0;
31     energygrid[grid.x*grid.y*k + grid.x*j + i+1] += energy1;
32     energygrid[grid.x*grid.y*k + grid.x*j + i+2] += energy2;
33     energygrid[grid.x*grid.y*k + grid.x*j + i+3] += energy3;
34  }
```

FIGURE 18.8

Direct Coulomb summation kernel with thread coarsening.

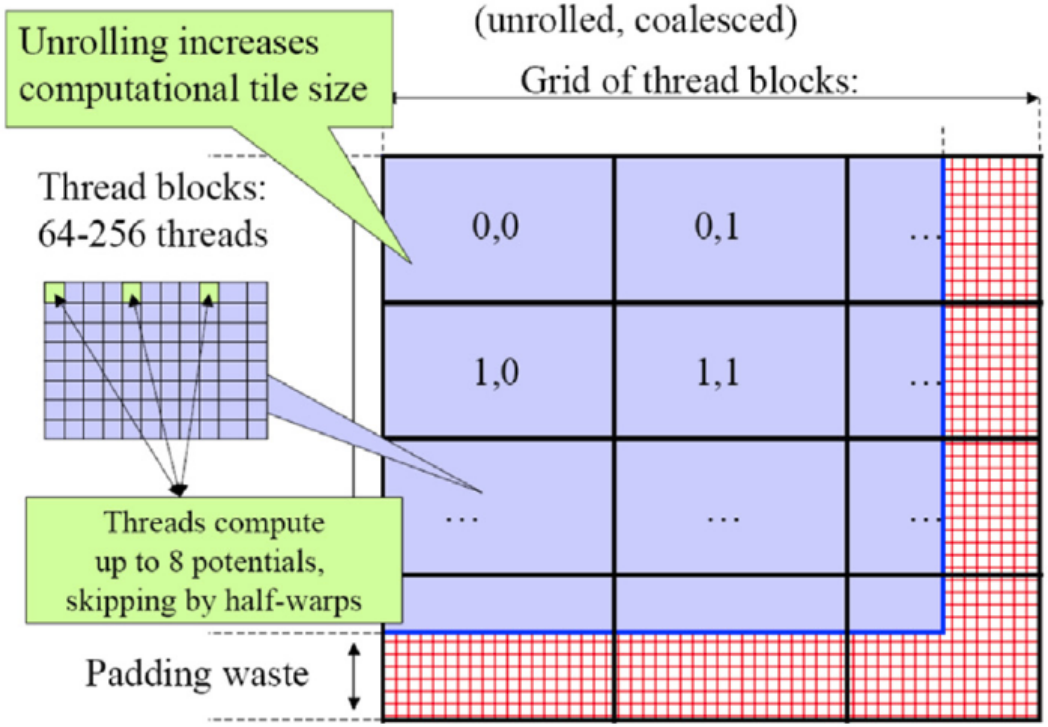


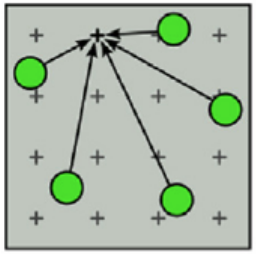
FIGURE 18.9

Organizing threads and memory layout for coalesced writes.

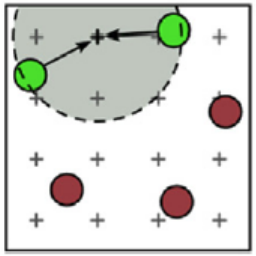
```
01  __constant__ float atoms[CHUNK_SIZE*4];
02  #define COARSEN_FACTOR 4
03  void __global__ cenergy(float *energygrid, dim3 grid, float gridspacing,
04                          float z, int numatoms) {
05     int i = blockIdx.x * blockDim.x*COARSEN_FACTOR + threadIdx.x;
06     int j = blockIdx.y * blockDim.y + threadIdx.y;
07     int atomarrdim = numatoms * 4;
08     int k = z / gridspacing;
09     float y = gridspacing * (float) j;
10     float x = gridspacing * (float) i;
11     float energy0 = 0.0f;
12     float energy1 = 0.0f;
13     float energy2 = 0.0f;
14     float energy3 = 0.0f;
15     // calculate potential contribution from all atoms
16     for (int n=0; n<atomarrdim; n+=4) {
17        float dx0 = x - atoms[n  ];
18        float dx1 = dx0 +   blockDim.x * gridspacing;
19        float dx2 = dx0 + 2*blockDim.x * gridspacing;
20        float dx3 = dx0 + 3*blockDim.x * gridspacing;
21        float dy = y - atoms[n+1];
22        float dz = z - atoms[n+2];
23        float dysqdzsq = dy*dy + dz*dz;
24        float charge = atoms[n+3];
25        energy0 += charge / sqrtf(dx0*dx0 + dysqdzsq);
26        energy1 += charge / sqrtf(dx1*dx1 + dysqdzsq);
27        energy2 += charge / sqrtf(dx2*dx2 + dysqdzsq);
28        energy3 += charge / sqrtf(dx3*dx3 + dysqdzsq);
29     }
30     energygrid[grid.x*grid.y*k + grid.x*j + i               ] += energy0;
31     energygrid[grid.x*grid.y*k + grid.x*j + i +   blockDim.x] += energy1;
32     energygrid[grid.x*grid.y*k + grid.x*j + i + 2*blockDim.x] += energy2;
33     energygrid[grid.x*grid.y*k + grid.x*j + i + 3*blockDim.x] += energy3;
34  }
```

FIGURE 18.10

Direct Coulomb summation kernel with thread coarsening and memory coalescing.



(A) Direct summation
At each grid point, sum the electrostatic potential from all charges



(B) Cutoff summation
Electrostatic potential from nearby charges summed; spatially sort charges first

FIGURE 18.11

(A) Cutoff summation versus (B) direct summation.

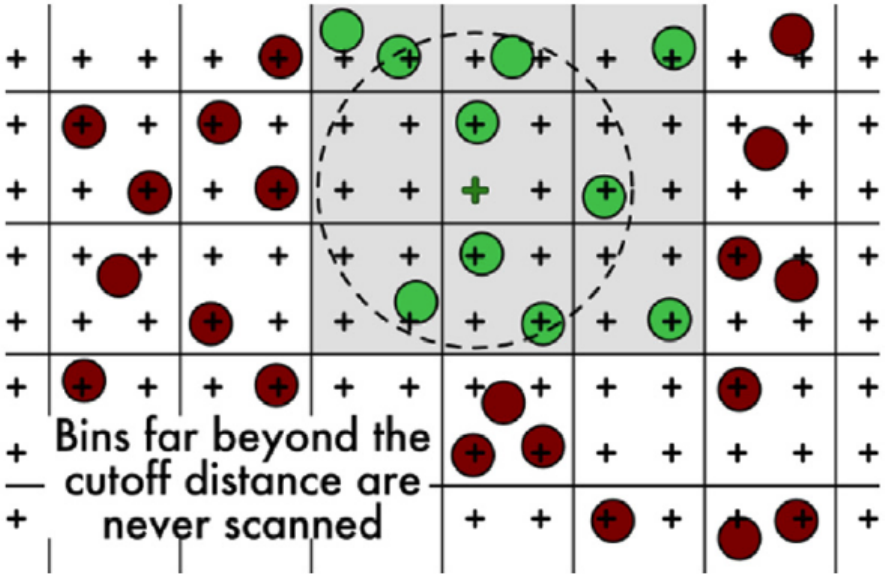


FIGURE 18.12

Neighborhood bins for a grid point.

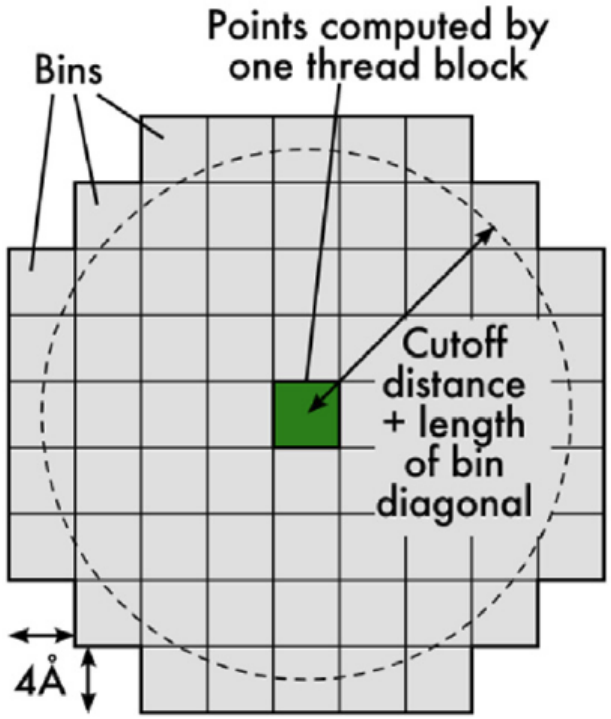


FIGURE 18.13

Identifying the neighborhood bins for all grid points processed by a block.

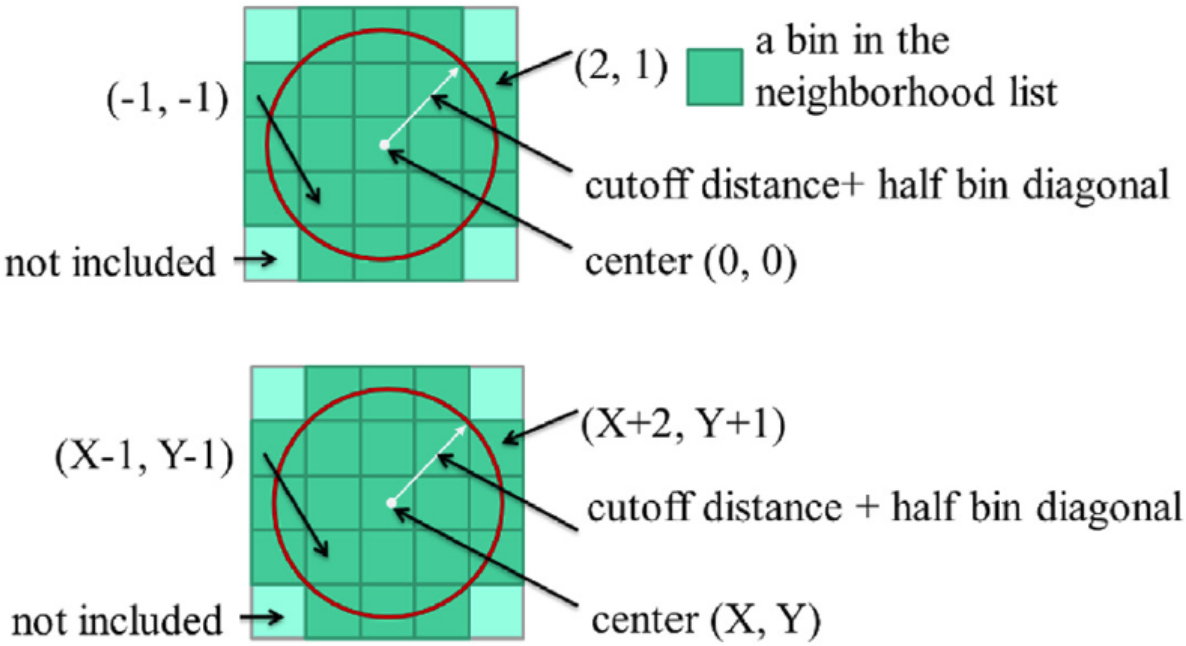


FIGURE 18.14

Neighborhood list using relative offsets.