Supporting Information

Supporting Information A: Quadratic interpolation for a lookup table

The function $f(x)$ is approximated by quadratic interpolation between lookup table knots. Knots are spaced by width w, and $FL(0,iw) = f(iw)$ at each knot where i is the lowest nearby knot number and z is distance from that knot. The equations are

$f(x) \approx FL(z,i)$

$FL(z,i) = az^2 + bz + c$

$z = x - iw$

With boundary conditions at (1) $z = 0$; (2) $z = w/2$; and (3) $z = w$

$$FL(z=0,i) = f(iw) = c \quad (1)$$

$$FL(z=w/2,i) = f(iw + 0.5w) = a(0.5w)^2 + b(0.5w) + c \quad (2)$$

$$FL(z=w,i) = f((i+1)w) = aw^2 + bw + c \quad (3)$$

Then it follows that

$FL(z=w,i) - c = aw^2 + bw = f((i+1)w) - f(iw) = g$

$FL(z=0.5w,i) = a0.25w^2 + b0.5w = f(iw + 0.5w) - f(iw) = h$

Where for convenience we introduced the notation: $g = f((i+1)w) - f(iw)$ and $h = f(iw + 0.5w) - f(iw)$ To solve for coefficients a and b of the 2nd order polynomial $aw^2 + bw$ for any knot width w, we solve the simple linear system $A\boldsymbol{x} = \boldsymbol{v}$ where A is the coefficient matrix $[\boldsymbol{a\ b}]$ and $\boldsymbol{v} = [g, h]$:

$$\left(\begin{array}{cc|c} w^2 & w & g \\ (0.5w)^2 & 0.5w & h \end{array}\right) \rightarrow \left(\begin{array}{cc|c} 1 & 0 & (2g-4h)/w^2 \\ 0 & 1 & (4h-g)/w \end{array}\right)$$

We get: $a = (2g - 4h)/w^2$ and $b = (4h - g)/w$.

Supporting Information B: Pseudo code for list generation

Note: actual code segments are printed *italic*.

The main simulation box is divided spatially into grid boxes. For example, the DHFR system has 7^3 boxes. Each CUDA thread block (a collection of threads that share memory and type of tasks, Figure 1) directly corresponds to one of these grid boxes and conducts the calculations associate with it. Each thread in this CUDA block represents one of the atoms in that grid box. Each thread therefore computes forces on one unique atom. The sum of all thread blocks comprises the entire simulation box. At the start of the program a list of 27 neighbor boxes (including self) is generated for each grid box applying periodic boundary translational symmetry on edge boxes. The symmetry operation for each box neighbor box is also stored in a list. One CUDA streaming multi-processor (SM) can execute up to 8 blocks but in practice is resource-limited to fewer. Blocks can be executed in any order. In principle, on a GPU with sufficient SMs, (or simulation with sufficiently few boxes), the entire list generation can be done in parallel. We start by

declaring SHARED float (arrays of size max atoms in box) jx[], jy[], jz[], ja[], jb[], jq[] that will contain neighboring box atom coordinates, Lennard-Jones parameters and charge. We also declare a SHARED integer array j[] (of size max atoms in box) that will contain the atom number of the neighboring box with atoms in order obtained from the box atom list.

The number of CUDA thread block equals a grid box number. It is one of the main simulation grid boxes for which we will loop over neighbor boxes with periodic symmetry.

Determine atom i from box atom list with index thread number (thread index of this block is its box atom list index) read from GLOBAL RAM box atom lists.

Read from GLOBAL RAM atom coordinates, charge and Lennard-Jones parameters of i and store these in LOCAL REGISTERS ix, iy, iz, iq, ia, ib respectively.

Read from GLOBAL RAM binary exclusion for atom i (integer: retain in LOCAL REGISTERS iExcl).

Loop over neighbor grid boxes

Get neighbor box number and neighbor box periodic symmetry operation from global RAM (2 integers),

Prepare symmetry vector valid for all atoms of the neighbor box and keep in LOCAL REGISTERS (floats tx, ty, tz where for example tx = either zero or (+)/(-) grid box-length-x).

Find atom index of one neighbor of current neighbor box corresponding to index of this thread (jLoad).

Read coordinates, parameters and charge of neighbor atom jLoad from GLOBAL RAM and store these into SHARED memory arrays listed above. For example in jx[thread number] we store coordinate x of atom jLoad which is an atom in the neighbor box atom list at index equaling this thread number and j[thread number] = jLoad. We now have all atoms of the current neighbor box in SHARED memory in one step since each of these operations is done in parallel.

Loop over neighbor atoms n of current neighbor box (note NO GLOBAL READS within this loop).

Get neighbor j from shared array (j[n]) and get coordinates and parameters of j from the other shared arrays (jx[n] etc.).

Translate neighbor coordinates (by symmetry tx,ty,tz) and calculate r^2 (square of length of the difference in coordinate vector).

Determine valid neighbor:

bool valid = (r^2 is below list cutoff^2 and i is not j);

Continue determine valid with binary exclusion list (see code segment below).

Binary exclusion (c++ code segment)

```
int diff = j[n] - i;
int dshift = 1<<(diff – 1); //-- obtain decimal value bit position.
bool val = (diff > 0) && (diff < 33);
bool valid = valid && !((dshift & iExcl) && val);

if (valid)
```

Calculate geometric averages A_{ij}, B_{ij} and q_{ij}. Example A_{ij}= ia * ja[n].

Along with atom number j, store A_{ij}, B_{ij} and q_{ij} in float4 structure "pars".

Write "pars" structure to GLOBAL RAM with (matrix) indexing for coalesced parallel read by force kernel later. **(Supporting Information C)**

Increment number of neighbors for i by 1.

end if

end loop

end loop

Store number of neighbors for i in GLOBAL RAM.

Supporting Information C: Pseudo code for real space calculations of non-bonded interactions

Atom number i is thread number, plus number of threads per block times block number. So again this thread is one unique atom. In this case atoms are simply in order of original input coordinate file.

Declare LOCAL REGISTER floats (OR doubles) in which to accumulate forces (iFx, iFy, iFz).

If option 4, declare SHARED float arrays FLa[], FLb[], FLc[] size 256 or 512 in which the quadratic lookup coefficients for electrostatic force term (**Supporting Information A**) are now stored by read from GLOBAL RAM (by all threads with thread number < 256). Each thread of a block reads one knot index of three coefficients. In one step, the whole lookup table is stored in shared memory.

Read atom coordinates and number of neighbors from GLOBAL RAM (3 floats and one integer) and store in LOCAL REGISTERS (floats ix, iy, iz and integer iNrNbrs).

Loop over neighbors

while (loop index <= number of neighbors)

Get the pair info float4 structure "pars" from neighbor list in GLOBAL RAM. This read is 100% coalesced and data kept for the duration of the force calculation below.

Read neighbor coordinates (into LOCAL REGISTER floats jx, jy, jz) from GLOBAL RAM directly (cached on Fermi) OR through TEXTURE (cached).

Get difference vector (LOCAL REGISTER floats rx, ry, rz) where rx = ix – jx

Prepare symmetry vector LOCAL REGISTER floats tx, ty, tz where symmetry is determined by rx is greater than list cutoff or less than (-)list cutoff.

Perform symmetry translation (e.g. rx = rx + tx).

Perform force calculation (see **Force calculation option code segments** below).

OPTION 1: no long range; calculate whole NB force without lookup – NO reads.

OPTION 2: linear lookup all – read one float4 from TEXTURE using built-in linear interpolation.

OPTION 3: Quadratic interpolation electrostatic – read one float4 from TEXTURE no interpolation.

OPTION 4: 256 or 512 knot Quadratic interpolation electrostatic read 3 floats from SHARED

Synchronize threads.

end while

Force calculation options (c++/CUDA code segments)

The "pars" float4 data structure was read from the neighbor list where pars.x = j, pars.y = A_{ij}, pars.z = B_{ij}, pars.w = q_{ij}. The neighbor atom (j) coordinates are then read and all options are preceded by symmetry translation of neighbor coordinates as shown above. Subsequently:

```
float r2 = rx*rx + ry*ry + rz*rz;
float r = sqrt(r2);
```

OPTION 1. Explicit calculation of all three force terms (no PME):

```
float invr2 = 1.0f/r2;  //-- note isqrt can be used here since r is not needed
float invr6 = invr2*invr2*invr2;
float FLJ = -12.0f * pars.y * invr6*invr6*invr2 + 6.0f * pars.z * invr6*invr2;
float Fel = pars.w * sqrt(invr2) *invr2;
```

OPTION 2. Texture memory linear interpolation of all three force terms (with PME)

```
float4 F; F.x = F.y = F.z = F.w = 0.0f;
if (valid)
        F = tex1D(tFLUa, r * 0.05f); //-- tex1D is a CUDA function
        //-- Texture force lookup with built-in linear filtering (interpolation)
        //--  tFLUa is the table of force values (all three terms)
float FLJ = pars.y * F.x + pars.z * F.y;
float Fel = pars.w * F.z * valid;
```

OPTION 3. Quadratic interpolation of electrostatics with PME. Coefficients from texture memory:

```
float invr2 = 1.0f/r2;
float invr6 = invr2*invr2*invr2;
float FLJ = -12.0f * pars.y * invr6*invr6*invr2 + 6.0f * pars.z * invr6*invr2;
```

```
int ind = r * KNOTSbyR;  //--  KNOTSbyR = number of table knots / r-max
float findx = float(ind);
float dx = r - findx * InvKNOTSbyR; //--  InvKNOTSbyR = 1/ KNOTSbyR
float4 F; F.x = F.y = F.z = F.w = 0.0f;
if (valid)
        float4 c = tex1D(tELFLa, findx);
        //-- Texture lookup of interpolation coefficients (no filtering)
        //--  tELFLa is the table of coefficients for  quadratic interpolation
float Fel = (c.x + (c.y*dx + c.z)*dx) * pars.w;
//-- quadratic interpolation where c.x,y,z are analogous to FLa,b,c
```

OPTION 4. Quadratic interpolation of electrostatics with PME. Coefficient from shared memory:

```
float invr2 = 1.0f/r2;
float invr6 = invr2*invr2*invr2;
float FLJ = -12.0f * pars.y * invr6*invr6*invr2 + 6.0f * pars.z * invr6*invr2;
int ind = r * SH256_KNOTSbyR * valid; //-- SH256_KNOTSbyR = 256/r-max
float findx = float(ind);
float dx = r - findx * SH256_InvKNOTSbyR;
float4 F; F.x = F.y = F.z = F.w = 0.0f;
float Fel = (FLc[ind] + (FLa[ind]*dx + FLb[ind])*dx) * pars.w;
```

All options are completed with force accumulation at end of loop:

```
iFx += (FLJ - Fel)*rx; iFy += (FLJ - Fel)*ry; iFz += (FLJ – Fel)*rz;
```

Supporting Information D: Serial profile of MOIL (single-precision)
Flat profile:

```
Output from gnu profiler (gprof) for solvated DHFR (section III.1). The
output was re-organized by function. For example, the non-bonded
interactions (when PME is used) are calculated by the following
functions: watwat_ewald, cdie_ewald, symwat_ewald and symcdie_ewald.
The calculation time therefore is the sum of these four.

Flat profile:
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name

Non bonded force – real space
 43.03    105.16   105.16     1350     0.08     0.08  watwat_ewald_
 14.78    141.28    36.12     1350     0.03     0.03  cdie_ewald_
 12.08    170.80    29.52     1350     0.02     0.02  symwat_ewald_
  0.04    243.55     0.09     1350     0.00     0.00  symcdie_ewald_
```

```
List generation
  6.92    187.72    16.92      150     0.11     0.16  nbondm_
  3.43    196.10     8.38     1950     0.00     0.00  nbmsym_
  2.60    202.46     6.36      150     0.04     0.04  nbond_
  0.09    241.69     0.21     1950     0.00     0.00  nbsym_

PME including FFT
  2.22    207.89     5.43      450     0.01     0.01  scalar_sum_
  2.08    212.97     5.08      450     0.01     0.01  grad_sum_
  1.56    225.79     3.81      450     0.01     0.01  fill_charge_grid_
  0.90    228.00     2.21      900     0.00     0.01  pubz3d_
  0.79    231.96     1.94 16588800     0.00     0.00  passf4_
  0.67    235.27     1.63 16588800     0.00     0.00  passb4_
  0.58    236.70     1.43 63547202     0.00     0.00  one_pass_
  0.07    242.08     0.18      450     0.00     0.00
get_bspline_coeffs_
  0.07    242.76     0.17 31773601     0.00     0.00  fill_bspline_

(M)SHAKE
  2.06    218.01     5.04     1200     0.00     0.00  shakept_
  1.62    221.98     3.97     1200     0.00     0.00  mshakpt_
  0.69    233.64     1.68     1200     0.00     0.00  shakevl_
  0.47    237.85     1.15     1200     0.00     0.00  mshakvl_

Bonded forces
  0.26    239.14     0.63     1350     0.00     0.00  etors_
  0.14    240.83     0.35     1350     0.00     0.00  etheta_
  0.09    241.90     0.21     1350     0.00     0.00  ener14_
  0.04    243.36     0.10     1350     0.00     0.00  ebond_

Subtract excluded forces
  0.27    238.51     0.66     1350     0.00     0.00  cdie_ewald_excl_
  0.16    240.11     0.39 16167657     0.00     0.00  ewforceexcl_

Verlet/RESPA
  0.16    240.49     0.38     3000     0.00     0.00  vel_step_
  0.05    243.15     0.12     1200     0.00     0.00  coord_step_
```