Biophysical Journal, Volume *112*

Supplemental Information

OpenRBC: A Fast Simulator of Red Blood Cells at Protein Resolution

Yu-Hang Tang, Lu Lu, He Li, Constantinos Evangelinos, Leopold Grinberg, Vipin Sachdeva, and George Em Karniadakis

The CGMD RBC model

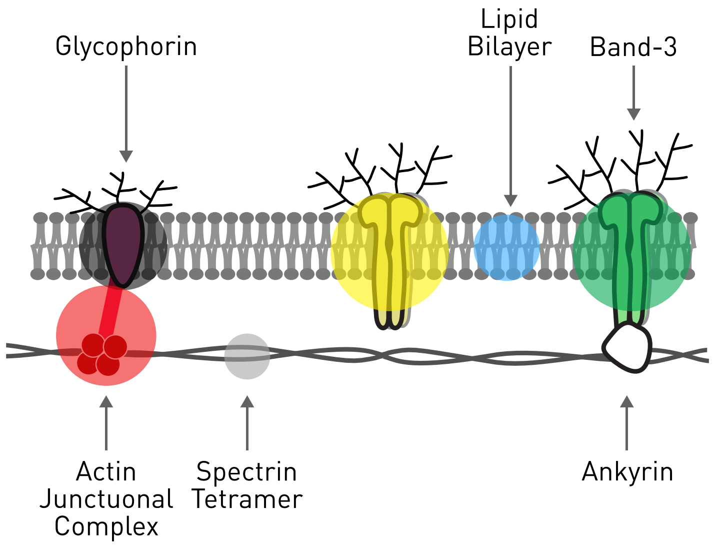

Figure 1: A schematic illustration of the CGMD model. Blue: lipid; red: actin junctional complex, silver: spectrin, black: glycophorin, yellow: mobile band-3, green: immobile band-3.

Here we briefly introduce the two-component CGMD RBC membrane model. For more detailed description see Refeference [1, 2]. As illustrated in Figure 1, the model describes the RBC as a two-component system, comprised of a cytoskeleton and a lipid bilayer. The cytoskeleton consists of spectrin filaments connected at actin junctional complexes forming a hexagonal network. The actin junctional complexes, as represented by the red particles, have a diameter of approximately 15 nm and are connected to the lipid bilayer via glycophorin. Spectrin is a protein tetramer formed by head-to-head association of two identical heterodimers. Each spectrin filament is represented by 19 spectrin particles connected by unbreakable springs. Spectrin chains are linked to band-3 particles via a spring potential. The two ends of the spectrin chains are also connected to the actin junctional complexes via the spring potential. Spectrin particles that are not connected by the spring potential interact with each other via a Lennard-Jones potential.

The CG particles, which form the lipid bilayer and transmembrane proteins, carry both translational and rotational degrees of freedom (x_i, n_i), where x_i and n_i are the position and the orientation (direction vector) of particle i, respectively. The rotational degrees of freedom obey the normality condition $|n_i| = 1$.

The lipid particles interact with each other through a pairwise addtive potential:

$$u_{ij}(n_i, n_j, x_{ij}) = u_R(r_{ij}) + A(\alpha, a(n_i, n_j, x_{ij}))u_A(r_{ij}) \tag{1}$$

$$u_R(r) = \begin{cases} \epsilon(\frac{r_c - r}{r_c - r_{eq}})^8, & r < r_c \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

$$u_A(r) = \begin{cases} -2\epsilon(\frac{r_c - r}{r_c - r_{eq}})^4, & r < r_c \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

$$A(\alpha, a(n_i, n_j, x_{ij})) = 1 + \alpha(a(n_i, n_j, \hat{x}_{ij}) - 1) \tag{4}$$

$$a(n_i, n_j, \hat{x}_{ij}) = (n_i \times \hat{x}_{ij}) \cdot (n_j \times \hat{x}_{ij}) = n_i \cdot n_j - (n_i \cdot \hat{x}_{ij})(n_j \cdot \hat{x}_{ij}) \tag{5}$$

where $x_{ij} = x_j - x_i$ is the distance vector between particles i and j, $r_{ij} = \|x_{ij}\|$ is the distance between i and j, $u_R(r_{ij})$ and $u_A(r_{ij})$ are the repulsive and attractive components of the pair potential, respectively. α is a tunable linear amplification factor. The function $A(\alpha, a(n_i, n_j, x_{ij})) = 1 + \alpha(a(n_i, n_j, x_{ij}) - 1)$ tunes the energy well of the potential, through which the fluid-like behavior of the membrane is regulated.

The translational motion of the particles is governed by the Newtonian equation of motion (EOM) of force, velocity, and position

$$\ddot{x} = \dot{v} = -\nabla U/m \tag{6}$$

while the rotational motion for the CG particles forming the lipid bilayer and proteins in the lipid bilayer is governed by

$$\tilde{m}_i \ddot{\mathbf{n}}_i = -\frac{\partial(\sum_{j=1}^N u_{\text{mem},ij})}{\partial \mathbf{n}_i} + (\frac{\partial(\sum_{j=1}^N u_{\text{mem},ij})}{\partial \mathbf{n}_i} \cdot \mathbf{n}_i)\mathbf{n}_i - \tilde{m}_i(\dot{\mathbf{n}}_i \cdot \dot{\mathbf{n}}_i)\mathbf{n}_i \tag{7}$$

where m is the mass of each particle, $\tilde{m}$ is a pseudo-mass with dimension of energy $\cdot$ time^2, and the right-hand side of Eq. 7 obeys the normality constraint $|n_i| = 1$. The Verlet algorithm combined with a Langevin thermostat is used to update the particle's position and orientation according to the EOMs.

Voronoi Cell List

A pseudocode demonstrating the algorithm for building a Voronoi cell list, as detailed in the main text, is shown in Algorithm 1.

Algorithm 1 The Voronoi cell list construction algorithm.

```
Class Voronoi_Celllist
  Ncell = SIMULATION CASE-SPECIFIC VALUE
  centroids = EMPTY
  bin_start = EMPTY
  cell_list = EMPTY
  tree      = KDTree()

  # find k centroids to minimize the within-cluster sum of squares
  Method KMeans( k: integer,
                 N: integer,
                 points: real[N][3] )
    ...


  Method UpdateCentroids( coord: real[Np][3] ):
    for i = 0:Ncell
      com = (0, 0, 0)
      n = bin_start[i+1] - bin_start[i]
      for j = 0:n
        com = com + coord[ bin_start[i] + j ]
      centroids[i] = com / n

  Method BuildCelllist( Np: integer, coord: real[Np][3] ):
    if centroids = EMPTY
      centroids = KMeans( Ncell, Np, coord )
    else
      Update_Centroids( coord )

    local_seq = integer[N]
    cid       = integer[N]
    # Calculate the Voronoi cell that particles fall in
    # Compute cell size and local indices for particles
    tree.rebuild( centroids )
    previous = ( id = 0, dist = Infinity )
    for i = 0:N
      nearest = tree.find_nearest( coord[i], previous )
      cid[i] = nearest.id
      local_seq[i] = bin_size[ nearest.id ]
      ++bin_size[ nearest.id ]
      previous = nearest

    # O(N) prefix sum for the starting index of each cell
    bin_start = zeros[Ncell]
    for i = 1:Ncell
      bin_start[i] = bin_start[i-1] + bin_size[i-1]

    # Scatter particle indices into corresponding cell
    cell_list = integer[N]
    for i = 0:N
      cell_list[ bin_start[cid[i]] + local_seq[i] ] = i

    destroy cid, local_seq, bin_size
    return cell_list, bin_start
```

The Voronoi cells can be used directly for efficient pairwise force computation between lipids with a quad loop that ranges over all Voronoi cell v_i, all neighboring cells v_j of v_i, all particles in v_i, and all particles in v_j as shown in Algorithm 2. The cell-wise neighbors for each cell v is determined by the criterion:

$$d_{ij} \leq r_i + r_j + r_c$$

where d_{ij} is the centroid-to-centroid distance between the Voronoi cell v_i and v_j, r_i is $\max_{k \in v_i} \|x_k - c_i\|$, and r_j is $\max_{k \in v_j} \|x_k - c_j\|$.

Particle Storage

The layout of particle storage has an effect on the performance of our **OpenRBC** simulator. The data of the lipid and protein particles are stored separately in two containers for the following reasons. First, the protein-protein and protein-lipid potentials are much more

Algorithm 2 Algorithm for pairwise force evaluation using the Voronoi cell list.

```
Method ComputePairwise( x : real[N][3], # coordinate
                        f : real[N][3], # force
                        o : real[N][3], # orientation
                        t : real[N][3], # torque
                        tree : KDTree,
                        voronoi_cell )
  for i = 0 : voronoi_cell.n_cells
    for each j in tree.find_around( i )
      for p1 in voronoi_cell[i]
        for p2 in voronoi_cell[j]
            if dist( x[p1], x[p2] ) < cutoff
              f, tau = pairwise_force( x[p1], x[p2],
                                       o[p1], o[p2] )
              f[p1] += f
              f[p2] -= f
              t[p1] -= tau
              t[p2] -= tau
```

complicated than the lipid-lipid potential. Thus, frequently choosing between the corresponding force kernels by particle type will incur a lot of branching instructions which may hurt the processor's front end performance. This can also be solved by working separately on the two classes of particles. Second, protein particles carry more information, *e.g.* type, tag, and bonds, than lipid particles. Hence a separation between the two classes of particles can save 2 arrays of size $O(N)$, which can be significant in terms of cache performance when millions of particles are present in the system.

Time Stepping

The Verlet integration algorithm coupled with the Langevin thermostat consists of embarrassingly parallel loops that iterates over particles to update their position, velocity, orientation and angular momentum. The implementation of the algorithm is divided into 2 stages, one before force evaluation and one after. Each of the stages consists of 3 passes that perform different tasks such as position updating, bounce back, orientation renormalization, force and torque reset, temperature calculation and temperature adjustment. Naively, each of the passes can be trivially parallelized with a single line of OpenMP `parallel for` directive. However, this will invoke a total of 12 parallel regions per time step to process both the lipid and protein container. Due to the low computation/transfer ratio of the arithmetics within each pass, the entire workload is largely memory bound. As a consequence, the software initially displayed a performance degradation when going from 4 to 8 hardware threads on Power8 CPUs.

We implemented a fused version of the time stepping algorithm by extracting the core algorithm inside each pass as functors that resemble GPU kernels. A fusion can then be performed to the greatly simplified kernels. A C++11 variadic driver function is then used to start a single parallel region, within which an arbitrary number of containers can be processes by the kernels. This effectively reduced the total number of parallel regions encountered per time step to 2, and maximizes cache line reuse without compromising program readability. The new time stepping scheme can benefit from using all the hardware threads available on the physical cores.

Memory Access

Non-uniform memory access (NUMA) / Non-uniform cache access (NUCA) are the prevalent memory system design in current processor architectures, where the latency of memory access depends on the link topology between the memory location relative to the processing elements. To maximize local memory access, the code will pin OpenMP threads to hardware threads in a depth-first manner such that consecutive threads reside on the same physical core. The scheduling of most OpenMP loops that operate on array-like objects are done with a central work scheduler which controls the range processed by each thread. This ensures consistency in the memory access footprint to each array object across different functions. The two most frequently accessed and performance-critical data structures in the program are the particle container and the Voronoi cell list. Thanks to the data locality as provided by the particle reordering algorithm, the partitions of the two structures can be aligned naturally with a simple linear split. Overall, in most of the parallel regions each thread will only need to work on its own partition without the need to touch data owned by other hardware cores. The non-local, pairwise nature of the particle interaction makes it inevitable for threads in OpenRBC to access part of the particle array which may be far away from its physical location. However, most of the non-local access is read-only and hence does not incur as much penalty as that in a read-write scenario.

Source Code Overview

To maximize customizability and reusability, we strive to decompose the source code into functionally independent modules. The structure of **OpenRBC** in terms of the source file organization is given in Table 1.

Table 1: List of source files relevant to customization

File	Functionality
openrbc.cpp	Main program loop
orbc-util.cpp	Utility program for file format conversion *etc.*
config_static.h	Compile-time options mostly relevant to performance optimization
forcefield_*.h	CGMD particles definition and interaction parameters
container.h	CGMD particle container
init_*.h	Initial structure generation algorithm
compute_*.h	Force evaluation driver (outer loop among adaptive cells)
pairwise_kernel_*.h	Pairwise force evaluation (inner loop between particles)
integrate_*.h	Time integrators, substeps, etc.
kdtree.h	k-d tree algorithm
math_vector_*.h	Vector operation support, SIMD wrapper
reorder_*.h	Sorting algorithm to improve data locality
runtime_parameter.h	Options controlling program behavior at run time
voronoi.h	Voronoi cell list
trajectory.h, topology.h	File I/O
util_*.h, service.h, timer.h	Miscellaneous utilities

Acknowledgment

This work was supported by National Institutes of Health (NIH) grants U01HL114476 and U01HL116323 and partially by the Department of Energy (DOE) Collaboratory on Mathematics for Mesoscopic Modeling of Materials (CM4). YHT acknowledges partial financial support from an IBM Ph.D. Scholarship Award. Part of the simulations were carried out at the Oak Ridge Leadership Computing Facility through the Innovative and Novel Computational Impact on Theory and Experiment program at Oak Ridge National Laboratory under project BIP118.

References

[1] H. Li and G. Lykotrafitis. Two-component coarse-grained molecular-dynamics model for the human erythrocyte membrane. *Biophysical journal*, 102(1):75–84, 2012.

[2] H. Li and G. Lykotrafitis. Erythrocyte membrane model with explicit description of the lipid bilayer and the spectrin network. *Biophysical journal*, 107(3):642–653, 2014.