Biophysical *Journal*
Computational Tool

Biophysical Society

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

Yu-Hang Tang,[1] Lu Lu,[1] He Li,[1] Constantinos Evangelinos,[2] Leopold Grinberg,[2] Vipin Sachdeva,[2] and George Em Karniadakis[1,*]
[1]Division of Applied Mathematics, Brown University, Providence, Rhode Island; and [2]IBM T.J. Watson Research Center, Cambridge, Massachusetts

ABSTRACT We present OpenRBC, a coarse-grained molecular dynamics code, which is capable of performing an unprecedented in silico experiment—simulating an entire mammal red blood cell lipid bilayer and cytoskeleton as modeled by multiple millions of mesoscopic particles—using a single shared memory commodity workstation. To achieve this, we invented an adaptive spatial-searching algorithm to accelerate the computation of short-range pairwise interactions in an extremely sparse three-dimensional space. The algorithm is based on a Voronoi partitioning of the point cloud of coarse-grained particles, and is continuously updated over the course of the simulation. The algorithm enables the construction of the key spatial searching data structure in our code, i.e., a lattice-free cell list, with a time and space cost linearly proportional to the number of particles in the system. The position and the shape of the cells also adapt automatically to the local density and curvature. The code implements OpenMP parallelization and scales to hundreds of hardware threads. It outperforms a legacy simulator by almost an order of magnitude in time-to-solution and >40 times in problem size, thus providing, to our knowledge, a new platform for probing the biomechanics of red blood cells.

The red blood cell (RBC) is one of the simplest, yet most important cells in the circulatory system due to its indispensable role in oxygen transport. An average RBC assumes a biconcave shape with a diameter of 8 μm and a thickness of 2 μm. Without any intracellular organelles, it is supported by a cytoskeleton of a triangular spectrin network anchored by junctions on the inner side of the membrane. Therefore, the mechanical properties of an RBC can be strongly influenced by molecular level structural details that alter the cytoskeleton and lipid bilayer properties.

Both continuum models (1–4) and particle-based models (5–8) have been developed with the aim to help uncover the correlation between RBC membrane structure and property. Continuum models are computationally efficient, but require a priori knowledge of cellular mechanical properties such as bending and shear modulus. Particle models are useful for extracting RBC properties from low-level descriptions of the membrane structure and defects. However, it is computationally demanding, if not prohibitive, to simulate the large number of particles required for modeling the membrane of an entire RBC. To the best of our knowledge, a bottom-up simulation of the RBC membrane at the cellular scale using particle methods remains absent.

Recently, a two-component coarse-grained molecular dynamics (CGMD) RBC membrane model that explicitly accounts for both the cytoskeleton and the lipid bilayer was proposed (9). The model could potentially be used for particle-based whole-cell RBC modeling because its coarse-grained nature can drastically reduce computational workload while still preserving necessary details from the molecular level. However, due to the orders-of-magnitude difference in the length scale between a cell and a single protein, a total of about four million particles is still needed to represent an entire RBC. In addition, the implicit treatment of the plasma in this model eliminates the overhead for tracking the solvent particles, but also exposes a notable spatial density heterogeneity because all CG particles are exclusively located on the surface of a biconcave shell. The space inside and outside of the RBC membrane remains empty. This density imbalance imposes a serious challenge on the efficient evaluation of the pairwise force using conventional algorithms and data structures, such as the cell list and the Verlet list, which typically assume a uniform spatial density and a bounded rectilinear simulation box.

In this article, we present OpenRBC—a new software tailored for simulations of an entire RBC using the

Submitted January 9, 2017, and accepted for publication April 10, 2017.
*Correspondence: george_karniadakis@brown.edu
Yu-Hang Tang and Lu Lu contributed equally to this work.
Editor: Nathan Baker.
http://dx.doi.org/10.1016/j.bpj.2017.04.020

two-component CGMD model on multicore CPUs. As illustrated in Fig. 1, the simulator can take as input a triangular mesh of the cytoskeleton of a RBC and reconstruct a CGMD model at protein resolution with explicit representations of both the cytoskeleton and the lipid bilayer. This type of whole cell simulation of RBCs can thus realize an array of in silico measurements and explorations of the following: 1) RBC shear and bending modulus, 2) membrane loss through vesiculation in spherocytosis and elliptocytosis (10), 3) anomalous diffusion of membrane proteins (11), 4) interaction between sickle hemoglobin fibers and RBC membrane in sickle cell disease (12,13), 5) uncoupling between the lipid bilayer and cytoskeleton (14), 6) ATP release due to deformation (15), 7) nitric oxide-modulated mechanical property change (16), and 8) cellular uptake of elastic nanoparticles (17).

Software overview

OpenRBC is written in C++ using features from the C++11 standard. To maximize portability and allow easy integration into other software systems (18), the project is organized as a header-only library with no external dependencies. The software implements SIMD vectorization (19) and OpenMP shared memory parallelization, and was specifically optimized toward making efficient use of large numbers of simultaneous hardware threads.

As shown in Fig. 2 *A*, the main body of the simulator is a time-stepping loop, where the force and torque acting on each particle is solved for and used for the iterative updating of the position and orientation according to a Newtonian equation of motion. The time distribution of each task in a typical simulation is given in Fig. 2 *B*. The majority of time is spent in force evaluation, which is compute-bound. This makes the code capable of utilizing the high thread count of modern CPUs with the shared memory programming paradigm.

Initial structure generation

As shown in Fig. 1, a two-component CGMD RBC system can be generated from a triangular mesh, which resembles the biconcave shape of a RBC at equilibrium. Note that the geometry may be alternatively sourced from experimental data using techniques such as optical image reconstruction because the algorithm itself is general enough to adapt to an arbitrary triangular mesh. This feature can be useful for simulating RBCs with morphological anomalies. Actin and glycophorin protein particles are placed on the vertices of the mesh, whereas spectrin and immobile band-3 particles are generated along the edges. The band-3-spectrin connections and actin-spectrin connections can be modified to simulate RBCs with structural defects. Lipid and mobile band-3 particles are randomly placed on each triangular face by uniformly sampling each triangle defined by the three vertices (20). A minimum interparticle distance is enforced to prevent clutter between protein and lipid particles. The system is then optimized using a velocity quenching algorithm to remove collision between the particles.

Spatial searching algorithm

Pairwise force evaluation accounts for >70% of the computation time in OpenRBC as well as other molecular dynamics softwares (21,22). To efficiently simulate the

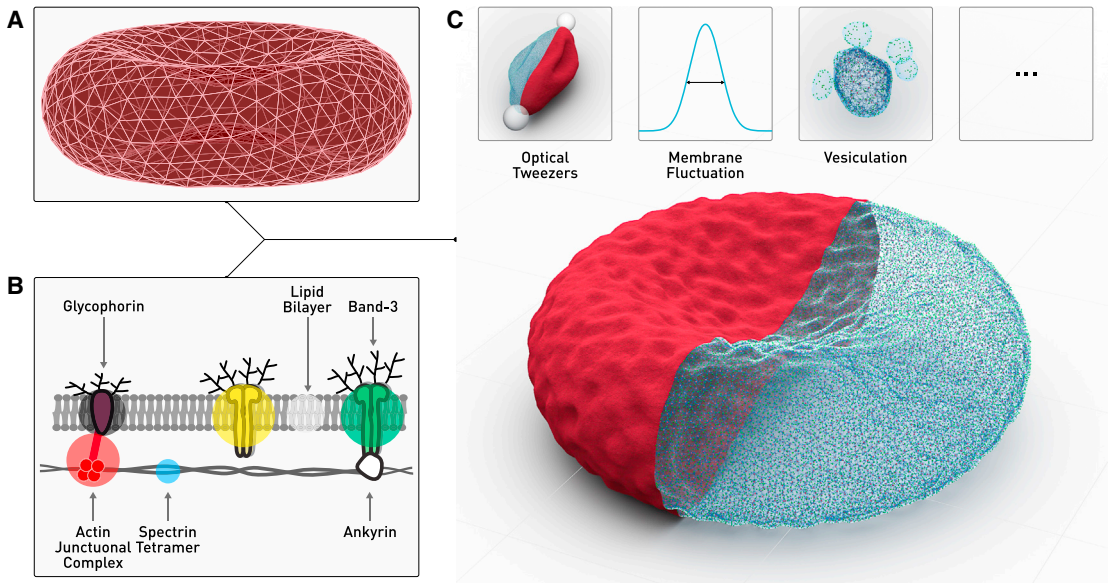

FIGURE 1 (*A*) A canonical hexagonal triangular mesh of a biconcave surface representing the cytoskeleton network is used together with (*B*) the two-component CGMD RBC membrane model to reconstruct (*C*) a full-scale virtual RBC, which allows for a wide range of computational experiments. To see this figure in color, go online.

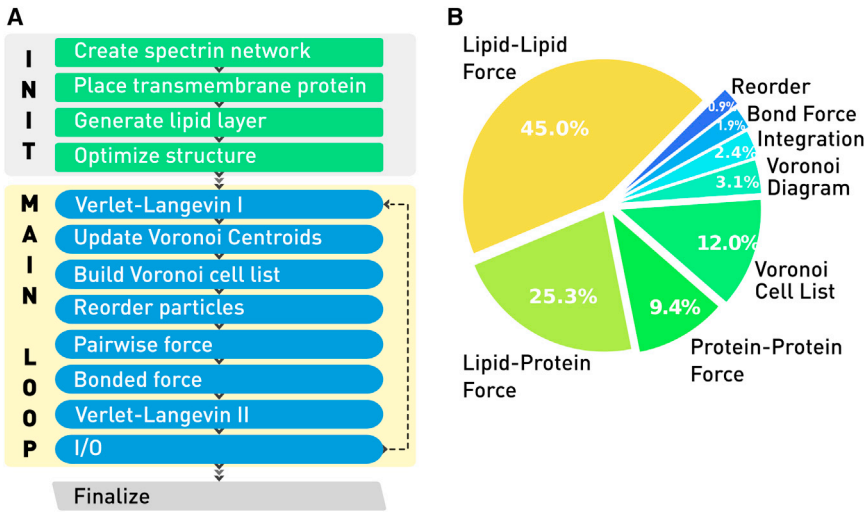

FIGURE 2 Shown here is the (*A*) flow chart and (*B*) typical wall time distribution of OpenRBC. To see this figure in color, go online.

reconstructed RBC model, we invented a lattice-free spatial partitioning algorithm that is inspired by the concept of Voronoi diagram. The algorithm, at the high level, can be described as the following:

1. Group particles into a number of adaptive clusters,
2. Compute interactions between neighboring clusters, and
3. Update cluster composition after particle movement. Then
4. Repeat from Step 2.

As illustrated in Fig. 3, the algorithm adaptively partitions a particle system into a number of Voronoi cells that are approximately equally populated. In contrast, a lattice-based cell list leaves many cells vacant due to the density heterogeneity. Thus, the algorithm can provide very good performance in partitioning the system, maintaining data locality and searching for pairwise neighbors in a sparse three-dimensional space. It is implemented in our software using a *k*-means clustering algorithm, which is, in turn, enabled by a highly optimized implementation of the *k*-d tree searching algorithm, as explained below.

A Voronoi tessellation (23) is a partitioning of an *n*-dimensional space into regions based on distance to a set of points called the "centroids". Each point in the space is attributed to the closest centroid (usually in the L2 norm sense). An example of a Voronoi diagram generated by 12 centroids on a two-dimensional rectangle is given in Fig. 4 *A*.

The *k*-means clustering (24) is a method of data partitioning that aims to divide a given set of *n* vectors into *k* clusters in which each vector belongs to the cluster whose center is closest to it. The result is a partition of the vector space into a Voronoi tessellation generated by the cluster centers as shown in Fig. 4 *B*. Searching for the optimal clustering that minimizes the within-cluster sum of square distance is NP-hard, but efficient iterative heuristics based on, e.g., the expectation-maximization algorithm (25), can be used to quickly find a local minimum.

A *k*-d tree is a spatial partitioning data structure for organizing points in a *k*-dimensional space (26). It is essentially a binary tree that recursively bisects the points owned by each node into two disjoint sets as separated by an axial-parallel hyperplane. It can be used for the efficient searching of the nearest neighbors of a given query point in $O(\log N)$ time, where N is the total number of points, by pruning out a large portion of the search space using cheap overlap checking between bounding boxes.

The *k*-means/Voronoi partitioning of a point cloud adapts automatically to the local density and curvature of the points. As such, we exploit this property to create a generalization of the cell list algorithm using the Voronoi diagram. The algorithm can be described as a two-step

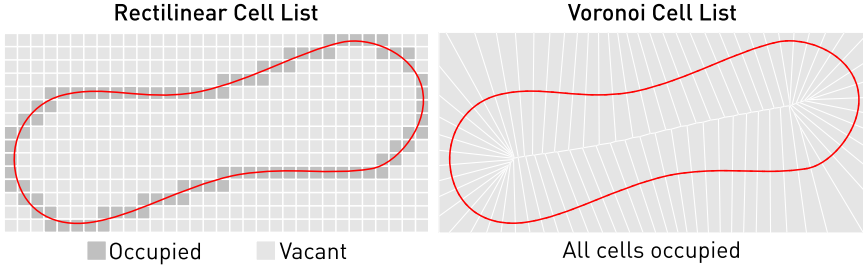

FIGURE 3 (*Left*) Only cells in dark gray are populated by CG particles in a cell list on a rectilinear lattice. This results in a waste of storage and memory bandwidth. (*Right*) All cells are evenly populated by CG particles in a cell list based on the Voronoi diagram generated from centroids located on the RBC membrane. To see this figure in color, go online.

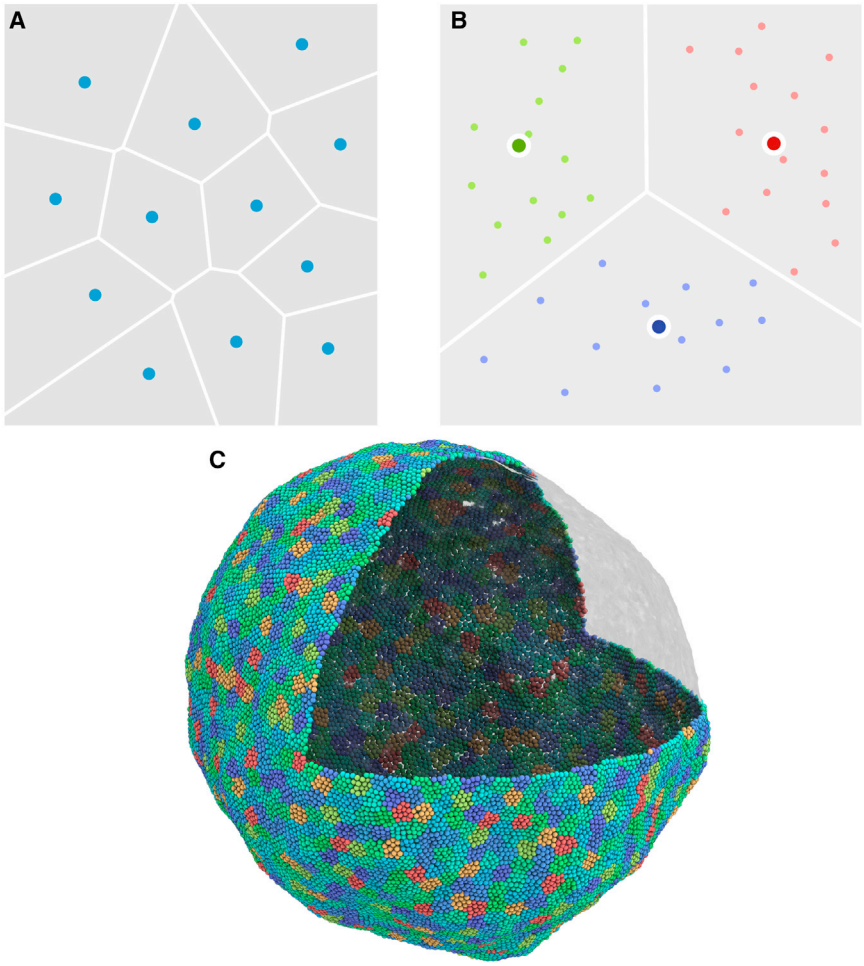

FIGURE 4 (*A*) Shown here is a Voronoi partitioning of a square as generated by centroids marked by the blue dots. (*B*) Shown here is a *k*-means ($k = 3$) clustering of a number of points on a two-dimensional plane. (*C*) Shown here is a vesicle of 32,673 CG particles partitioned into 2000 Voronoi cells. To see this figure in color, go online.

procedure: 1) clustering all the particles in the system using *k*-means, followed by an online expectation-maximization algorithm that continuously updates the system's Voronoi cells centroid location and particle ownership; and 2) sorting the centroids and particles with a two-level data reordering scheme, where we first order the Voronoi centroids along a space-filling curve (a Morton curve, specifically) and then reorder the particles according to the Voronoi cell that they belong to. The pseudocode for the algorithm can be found in the Supporting Material. The reordering step in updating the Voronoi cells ensures that neighboring particles in the physical space are also statistically close to each other in the program memory space. This locality can speed up the *k*-d tree nearest-neighbor search by allowing us to use the closest centroid of the last particle as the initial guess for the next particle. This heuristic helps to further prune out most of the *k*-d tree search space and essentially reduces the complexity of a nearest-neighbor query from $O(\log N)$ to $O(1)$. In practice, this brings $\sim$100 times acceleration when searching through 200,000 centroids. As shown by Fig. 4 *C*, the Voronoi cells generated from a *k*-means clustering of the CG particles are uniformly distributed on the surface of the lipid membrane.

Force evaluation

Lipid particles accounts for 80% of the population in the whole-cell CGMD system. 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 the Supporting Material. Because the cytoskeleton of a healthy RBC is always attached to the lipid bilayer, its protein particles are also distributed following the local curvature of the lipid particles. This means that we can reuse the Voronoi cells of the lipid particles, but with a wider searching cutoff, to compute both the lipid-protein and protein-protein pairwise interactions. For diseased RBCs with fully or partially detached cytoskeletons, a separate set of Voronoi cells can be set up for the cytoskeleton proteins to compute the force. A list of bonds between proteins is maintained and used for computing the forces between proteins that are physically linked to each other.

A commonly used technique in serial programs to speed up the force computation is to take advantage of the Newton's third law of action and reaction. Thus, the force between each pair of interacting particles is only computed once and added to both particles. However, this generates a race condition in a parallel context because two threads may end up simultaneously computing the force on a particle shared by two or more pairwise interactions.

Our solution takes advantage of the strong spatial locality of the particles as maintained by the two-level reordering algorithm, and decomposes the workload both spatially and linearly-in-memory into patches by splitting the linear range of cells indices among OpenMP threads. Each thread will be calculating the forces acting on the particles within its own patch. As shown in Fig. 5, force accumulation without triggering racing condition can be realized by only exploiting the Newton's third law on pairwise interactions where both Voronoi cells belong to a thread's own patch. Interactions involving a pair of particles from different patches are calculated twice (once for each particle) by each thread. The strong particle locality minimizes the shared contour length between two patches, and hence also minimizes the number of interpatch interactions.

Validation and benchmark

In this section we present validation of our software by comparing simulation and experimental data. We also compare the program performance against that of the legacy CGMD RBC simulator used in Li and Lykotrafitis (9). The legacy simulator, which performs reasonably well for a small number of particles in a periodic rectangular box, was written in C and parallelized with the message passing interface using a rectilinear domain decomposition scheme and a distributed memory model. Three computer systems were used in the benchmark, each equipped with a different mainstream CPU microarchitecture, i.e., the Intel Haswell, the AMD Piledriver, and the IBM Power8 (27). The machine specifications are given in Table 1.

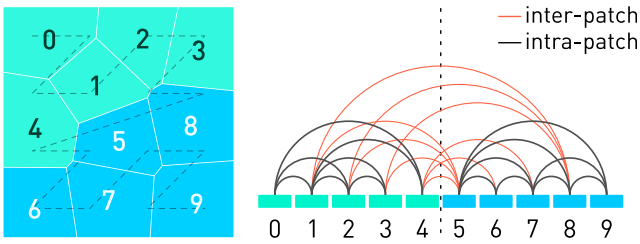

FIGURE 5 Thanks to the spatial locality ensured by reordering particles along the Morton curve (*dashed line*), we can simply divide the cells between two threads by their index into two patches each containing five consecutive Voronoi cells. The force between cells from the same patch is computed only once using Newton's third law, whereas the force between cells from different patches is computed twice on each side. To see this figure in color, go online.

TABLE 1 A Summary of Capability and Design Highlights of OpenRBC and the Specifications of the Computer Systems Used in the Benchmark

Capability and Design						Performance—Time Steps/Day					
OpenRBC Legacy Improvement						Cores	Particles	OpenRBC	Legacy	Speedup	
Maximum system size (number of particles)	$>8 \times 10^6$	2×10^5	>40 times		20	8.34×10^6	3.90×10^5	—	—		
Line of code	4677	7424	37% less		4	1.88×10^5	3.86×10^6	0.42×10^6	9.2		
CPU	Architecture	Instruction set	Frequency (GHz)	Physical cores	Hardware threads	Total threads	Last level cache (MB)	GFLOPS (SP)	Achieved FLOPS by OpenRBC (%)		Memory bandwidth (GB/s)
IBM POWER 8 "Minsky"	Power8	Power	3.5	10×2	8	160	80×2	560.0	8.7		230
Intel Xeon E5-2695 v3	Haswell	x86–64	2.3	14×2	2	56	35×2	1030.4	4.6		136
AMD Opteron 6378	Piledriver	x86–64	2.4	16×4	1	64	16×4	614.4	4.5		204

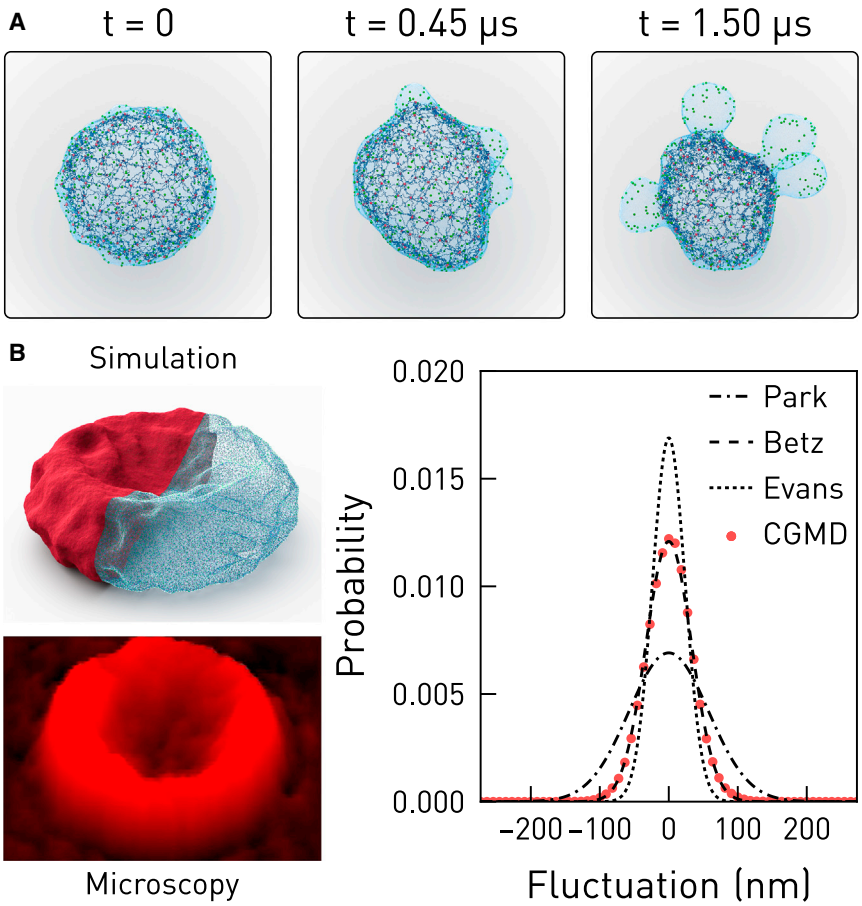

FIGURE 6 (*A*) Shown here is the vesiculation procedure of a miniature RBC. (*B*) The instantaneous fluctuation of a full-size RBC in OpenRBC compares to that from experiments (28–30). Microscopy image is reprinted with permission from Park et al. (28). To see this figure in color, go online.

To compare performance between OpenRBC and the legacy simulator, the membrane vesiculation process of a miniaturized RBC-like sphere with a surface area of 2.8 μm^2 was simulated. The evolution of the dynamic process is visualized from the simulation trajectory and shown in Fig. 6 *A*. OpenRBC achieves almost an order-of-magnitude speedup over the legacy solver in this case on all three computer systems, as shown in Table 1.

Furthermore, OpenRBC can efficiently simulate an entire RBC modeled by 3,200,000 particles and correctly reproduce the fluctuation and stiffness of the membrane as shown in Fig. 6 *B*. The legacy solver was not able to launch the simulation due to memory constraint. The simulation was carried out by implementing the experimental protocol of Park et al. (28), which measures the instantaneous vertical fluctuation $\Delta h(x,y)$ along the upper rim of a fixed RBC. In addition, a harmonic volume constraint is applied to maintain the correct surface-to-volume ratio of the RBC. We measured a membrane root-mean-square displacement of 33.5 nm, whereas previous experimental observations and simulation results range between 23.6 and 58.8 nm (28–31).

A scaling benchmark for the whole cell simulation on the three computer systems is given in Fig. 7. It can be seen that compute-bound tasks such as pairwise force evaluation can scale linearly across physical cores. Memory-bound tasks benefit less from hardware threading as expected, but thanks to thread pinning and a consistent workload decomposition between threads, there is no performance degradation from side effects such as cache and bandwidth contention.

It is also worth noting that Fu et al. (32) recently published an implementation of a related RBC model in LAMMPS, which can simulate 1.15×10^6 particles for 10^5 time steps on 864 CPU cores in 2761 s. However, the use of explicit solvent particles in their model generates difficulty in establishing a direct performance comparison between their implementation and OpenRBC. Nevertheless, as a rough estimate and assuming perfect scaling, their timing result can be translated into simulating 8.34×10^6 particles for 0.41×10^5 time steps per day on 864 cores. OpenRBC can complete roughly the same amount of time steps on 20 CPU cores. We do recognize that the explicit solvent model carries more computational workload, and that implementing nonrectilinear partitioning schemes may not be straightforward within the current software framework of LAMMPS. Nonetheless, this comparison does serve to demonstrate the potential of shared-memory programming

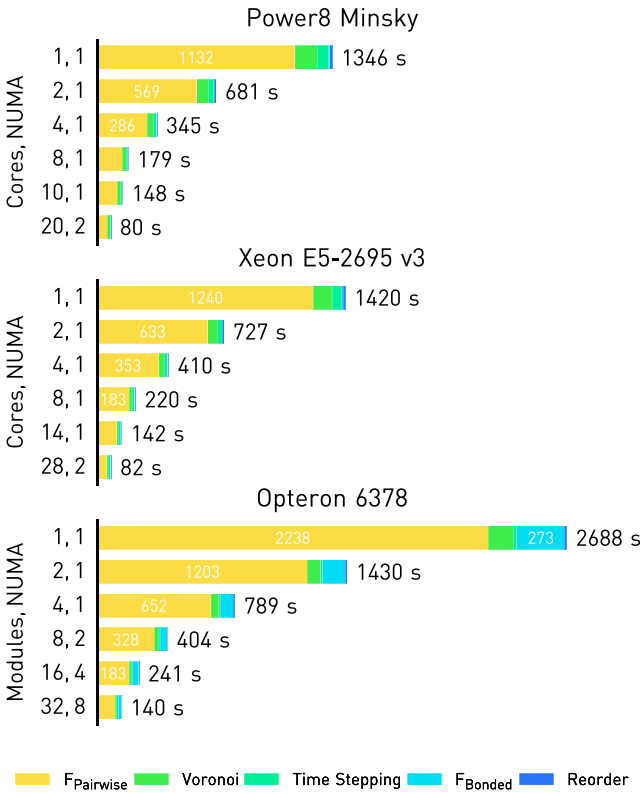

FIGURE 7 Shown here is the scaling of OpenRBC across physical cores and NUMA domains when simulating an RBC of 3,200,000 particles. To see this figure in color, go online.

paradigm on *fat* compute nodes with large numbers of strong cores and amounts of memory.

Summary

We presented a from-scratch development of a coarse-grained molecular dynamics software, OpenRBC, which exhibits exceptional efficiency when simulating systems of large density discrepancy. This capability is supported by an innovative algorithm that computes an adaptive partitioning of the particles using a Voronoi diagram. The program is parallelized with OpenMP and SIMD vector instructions, and implements threading affinity control, consistency loop partitioning, kernel fusion, and atomics-free pairwise force evaluation to increase the utilization of simultaneous hardware threads and to maximize memory performance across multiple NUMA domains. The software achieves an order-of-magnitude speedup in terms of time-to-solution over a legacy simulator, and can handle systems that are almost two orders-of-magnitude larger in particle count. The software enables, for the first time ever to our knowledge, simulations of an entire RBC with a resolution down to single proteins, and opens up the possibility for conducting many in silico experiments concerning the RBC cytomechanics and related blood disorders (33).

SUPPORTING MATERIAL

Supporting Materials and Methods, one figure, and one table are available at http://www.biophysj.org/biophysj/supplemental/S0006-3495(17)30436-8.

AUTHOR CONTRIBUTIONS

Y.-H.T. designed the algorithm. Y.-H.T. and L.L. implemented the software. Y.-H.T. and C.E. carried out the performance benchmark. H.L. developed the CGMD model. H.L. and Y.-H.T. performed validation and verification. Y.-H.T., L.L., and H.L. wrote the manuscript. L.G., C.E., and V.S. provided algorithm consultation and technical support. G.E.K. supervised the work.

ACKNOWLEDGMENTS

This work was supported by NIH grants No. U01HL114476 and U01HL116323 and partially by the Department of Energy (DOE) Collaboratory on Mathematics for Mesoscopic Modeling of Materials (CM4). Y.-H.T. acknowledges partial financial support from an IBM Ph.D. Scholarship Award. Simulations were partly 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 No. BIP118.

REFERENCES

1. Evans, E. A. 1974. Bending resistance and chemically induced moments in membrane bilayers. *Biophys. J.* 14:923–931.
2. Feng, F., and W. S. Klug. 2006. Finite element modeling of lipid bilayer membranes. *J. Comput. Phys.* 220:394–408.
3. Powers, T. R., G. Huber, and R. E. Goldstein. 2002. Fluid-membrane tethers: minimal surfaces and elastic boundary layers. *Phys. Rev. E Stat. Nonlin. Soft Matter Phys.* 65:041901.
4. Helfrich, W. 1973. Elastic properties of lipid bilayers: theory and possible experiments. *Z. Naturforsch. C.* 28:693–703.
5. Feller, S. E. 2000. Molecular dynamics simulations of lipid bilayers. *Curr. Opin. Colloid Interface Sci.* 5:217–223.
6. Saiz, L., S. Bandyopadhyay, and M. L. Klein. 2002. Towards an understanding of complex biological membranes from atomistic molecular dynamics simulations. *Biosci. Rep.* 22:151–173.
7. Tieleman, D. P., S.-J. Marrink, and H. J. Berendsen. 1997. A computer perspective of membranes: molecular dynamics studies of lipid bilayer systems. *Biochim. Biophys. Acta.* 1331:235–270.
8. Tu, K., M. L. Klein, and D. J. Tobias. 1998. Constant-pressure molecular dynamics investigation of cholesterol effects in a dipalmitoylphosphatidylcholine bilayer. *Biophys. J.* 75:2147–2156.
9. Li, H., and G. Lykotrafitis. 2014. Erythrocyte membrane model with explicit description of the lipid bilayer and the spectrin network. *Biophys. J.* 107:642–653.
10. Li, H., and G. Lykotrafitis. 2015. Vesiculation of healthy and defective red blood cells. *Phys. Rev. E Stat. Nonlin. Soft Matter Phys.* 92:012715.
11. Li, H., Y. Zhang, …, G. Lykotrafitis. 2016. Modeling of band-3 protein diffusion in the normal and defective red blood cell membrane. *Soft Matter.* 12:3643–3653.
12. Lei, H., and G. E. Karniadakis. 2012. Predicting the morphology of sickle red blood cells using coarse-grained models of intracellular aligned hemoglobin polymers. *Soft Matter.* 8:4507–4516.
13. Li, X., E. Du, …, G. E. Karniadakis. 2016. Patient-specific blood rheology in sickle-cell anaemia. *Interface Focus.* 6:20150065.
14. Peng, Z., X. Li, …, S. Suresh. 2013. Lipid bilayer and cytoskeletal interactions in a red blood cell. *Proc. Natl. Acad. Sci. USA.* 110:13356–13361.

15. Sprague, R. S., M. L. Ellsworth, …, A. J. Lonigro. 1998. Deformation-induced ATP release from red blood cells requires CFTR activity. *Am. J. Physiol.* 275:H1726–H1732.
16. Wood, K. C., M. M. Cortese-Krott, …, M. T. Gladwin. 2013. Circulating blood endothelial nitric oxide synthase contributes to the regulation of systemic blood pressure and nitrite homeostasis. *Arterioscler. Thromb. Vasc. Biol.* 33:1861–1871.
17. Zhao, Y., X. Sun, …, V. S.-Y. Lin. 2011. Interaction of mesoporous silica nanoparticles with human red blood cell membranes: size and surface effects. *ACS Nano.* 5:1366–1375.
18. Tang, Y.-H., S. Kudo, …, G. E. Karniadakis. 2015. Multiscale universal interface: a concurrent framework for coupling heterogeneous solvers. *J. Comput. Phys.* 297:13–31.
19. Abraham, M. J., T. Murtola, …, E. Lindahl. 2015. GROMACS: high performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX.* 1:19–25.
20. Osada, R., T. Funkhouser, …, D. Dobkin. 2002. Shape distributions. *ACM Trans. Graph.* 21:807–832.
21. Tang, Y.-H., and G. E. Karniadakis. 2014. Accelerating dissipative particle dynamics simulations on GPUs: algorithms, numerics and applications. *Comput. Phys. Commun.* 185:2809–2822.
22. Rossinelli, D., Y.-H. Tang, …, P. Koumoutsakos. 2015. The in-silico lab-on-a-chip: petascale and high-throughput simulations of microfluidics at cell resolution. *In* Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2015. ACM, New York, NY, pp. 2:1–2:12.
23. Edelsbrunner, H. 2014. Voronoi and Delaunay diagrams. *In* A Short Course in Computational Geometry and Topology. Springer, Berlin, Germany, pp. 9–15.
24. Hartigan, J. A., and M. A. Wong. 1979. Algorithm as 136: a k-means clustering algorithm. *J. R. Stat. Soc. Ser. C Appl. Stat.* 28:100–108.
25. Dempster, A. P., N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the Em algorithm. *J. R. Stat. Soc. B.* 39:1–38.
26. Bentley, J. L. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM.* 18:509–517.
27. Starke, W., J. Stuecheli, …, B. Blaner. 2015. The cache and memory subsystems of the IBM POWER8 processor. *IBM J. Res. Develop.* 59:3:1–3:13.
28. Park, Y., M. Diez-Silva, …, S. Suresh. 2008. Refractive index maps and membrane dynamics of human red blood cells parasitized by *Plasmodium falciparum*. *Proc. Natl. Acad. Sci. USA.* 105:13730–13735.
29. Evans, J., W. Gratzer, …, J. Sleep. 2008. Fluctuations of the red blood cell membrane: relation to mechanical properties and lack of ATP dependence. *Biophys. J.* 94:4134–4144.
30. Betz, T., M. Lenz, …, C. Sykes. 2009. ATP-dependent mechanics of red blood cells. *Proc. Natl. Acad. Sci. USA.* 106:15320–15325.
31. Fedosov, D. A., H. Lei, …, G. E. Karniadakis. 2011. Multiscale modeling of red blood cell mechanics and blood flow in malaria. *PLoS Comput. Biol.* 7:e1002270.
32. Fu, S.-P., Z. Peng, …, Y.-N. Young. 2017. Lennard-Jones type pair-potential method for coarse-grained lipid bilayer membrane simulations in LAMMPS. *Comput. Phys. Commun.* 210:193–203.
33. Li, X., H. Li, …, G. E. Karniadakis. 2016. Computational biomechanics of human red blood cells in hematological disorders. *J. Biomech. Eng.* 139:021008.

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

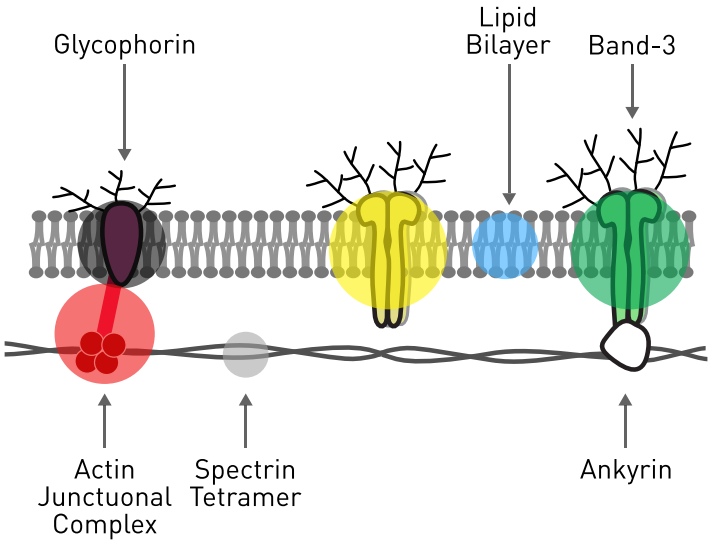

Figure 1: A schematic illustration of the CGMD model. Blue: lipid; red: actin junctional complex, silver: spectrin, black: glycophorin, yellow: mobile band-3, green: immobile band-3.

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.