JCTC Journal of Chemical Theory and Computation

Article

pubs.acs.org/JCTC

Efficient Maintenance and Update of Nonbonded Lists in Macromolecular Simulations

Rezaul Chowdhury,[†] Dmitri Beglov,[‡] Mohammad Moghadasi,[§] Ioannis Ch. Paschalidis,[§,∥] Pirooz Vakili,[‡] Sandor Vajda,[‡] Chandrajit Bajaj,[#] and Dima Kozakov*[,‡]

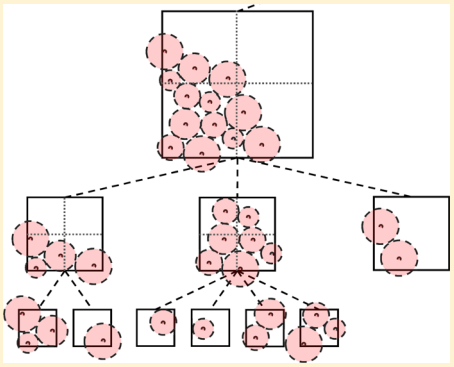
[†]Computer Science Department, Stony Brook University, Stony Brook, New York 11790, United States

[‡]Department of Mechanical Engineering, [§]Division of Systems Engineering, and [∥]Department of Electrical and Computer Engineering, Boston University, Boston, Massachusetts 02215, United States

[#]Department of Computer Science, University of Texas at Austin, Austin, Texas 78712, United States

Supporting Information

ABSTRACT: Molecular mechanics and dynamics simulations use distance based cutoff approximations for faster computation of pairwise van der Waals and electrostatic energy terms. These approximations traditionally use a precalculated and periodically updated list of interacting atom pairs, known as the "nonbonded neighborhood lists" or nblists, in order to reduce the overhead of finding atom pairs that are within distance cutoff. The size of nblists grows linearly with the number of atoms in the system and superlinearly with the distance cutoff, and as a result, they require significant amount of memory for large molecular systems. The high space usage leads to poor cache performance, which slows computation for large distance cutoffs. Also, the high cost of updates means that one cannot afford to keep the data structure always synchronized with the configuration of the molecules when efficiency is at stake. We propose a dynamic octree data structure for implicit maintenance of nblists using space linear in the number of atoms but independent of the distance cutoff. The list can be updated very efficiently as the coordinates of atoms change during the simulation. Unlike explicit nblists, a single octree works for all distance cutoffs. In addition, octree is a cache-friendly data structure, and hence, it is less prone to cache miss slowdowns on modern memory hierarchies than nblists. Octrees use almost 2 orders of magnitude less memory, which is crucial for simulation of large systems, and while they are comparable in performance to nblists when the distance cutoff is small, they outperform nblists for larger systems and large cutoffs. Our tests show that octree implementation is approximately 1.5 times faster in practical use case scenarios as compared to nblists.



The most memory and time-consuming step in molecular mechanics and molecular dynamics simulations is the calculation of the nonbonded terms in the energy function,[1] which requires summations of pairwise van der Waals and electrostatics interactions. The computation times for these summations are proportional to the number of interacting pairs of atoms, which grows quadratically with the total number of atoms in the molecular system. However, since the interactions decay with distance, distance-based truncation (cutoff) approximations are widely used in practice to trade-off accuracy for speed, reducing the overall computation time to a linear (or nearly linear) function of the number of atoms.[2] These cutoff approximations have traditionally been handled through initially determined and periodically updated list of interacting atomic pairs—the so-called *nonbonded neighborhood lists*[3] or nblists. The efficiency of molecular simulations is thus critically dependent on the space- and time-efficient maintenance of nblists. In the past, this efficiency had been achieved by dividing the system into smaller regions based on cutoff distance[4] or chemical connectivity[5] or the "By-Clusters-in-Cubes" (BYCC) method.[1] The BYCC method combines the connectivity and spatial separation criteria by clustering atoms of the molecular system. This clustering, however, is done only once at the beginning of the molecular simulation. The resulting clusters are then placed inside a cubic grid with the dimension of each grid cell being roughly equal to the nblist's cutoff distance plus the maximum cluster size. This cubic division facilitates search through the occupied neighboring cubes to determine the list of interacting pairs of atoms. The BYCC method through its reliance on cluster−cluster pairwise distances (rather than pairwise atomic distances) is much faster than its predecessors.[4,5] The efficiency of the clustering step of BYCC, based on connectivity is obviously higher for larger molecules. Since here we are interested in simulation of the macromolecular systems (e.g., several protein chains), we use BYCC as a comparison benchmark. Explicit nblists can also be constructed using dynamic packing grids,[6] as well as binary hierarchy approaches.[7] However, explicit nblists have high space requirement, poor cache performance, high update cost, and need to be

Received: June 6, 2013
Published: September 5, 2014

   dx.doi.org/10.1021/ct400474w | *J. Chem. Theory Comput.* 2014, 10, 4449−4454

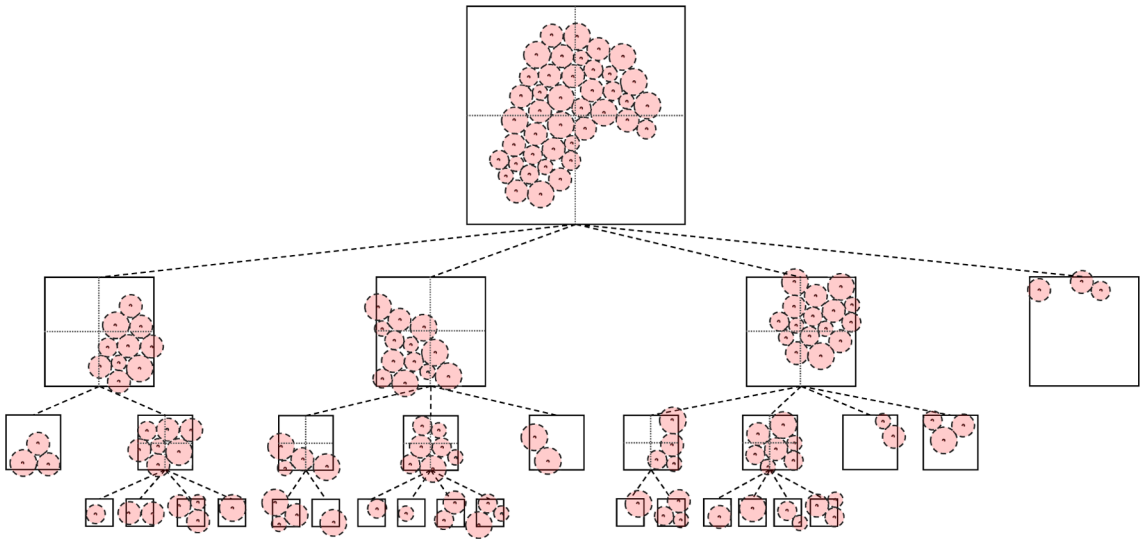


Figure 1. A (3,1)-admissible quadtree in which no leaf contains more than $1 \times 3 = 3$ points (i.e., centers) and each internal node contains more than $3/1 = 3$ points. The quadtree is constructed by recursive subdivision of the box bounding the initial set of points into quadrants.

regenerated explicitly for different distance cutoffs, problems that we now solve effectively in our new approach.

In this paper, we develop an octree[8]-based technique for space-efficient implicit maintenance of nblists and efficient evaluation of pairwise nonbonded energy terms under conformational changes of the molecules. In the following text, we will use the term nblists to indicate traditional nblists method, as opposed to octrees. While nblists use space superlinear in the distance cutoff d, space complexity of an octree is independent of d, and so, unlike nblists, the same octree can be used for all distance cutoffs. An octree is a cache-friendly data structure, and octree-based energy evaluations incur far fewer cache misses compared to nblists-based evaluations. Thus, octree-based computations are less prone to cache miss slowdowns on modern machines than computations involving nblists.

1. OCTREES FOR MAINTAINING MOLECULES UNDER CONFORMATIONAL CHANGES

An octree[8] is a tree data structure that recursively and adaptively subdivides the 3D space into octants, and is often used as a container for Cartesian space data. Here, we use octrees to store all atomic centers (or 3D atomic Cartesian coordinates) of a molecular system, organized recursively based on their spatial locality. Octrees have been used in the past for fast multipole-like approximations of force fields in particle simulations[9] and in molecular solvation energy and force calculations.[10]

In this work, we show that octrees can be used for space-efficient maintenance of nonbonded neighborhood lists (i.e., nblists) of a molecular system and can be updated very efficiently as the configuration of the system changes. A special type of octree, as defined below, will be used for the purpose. Octrees are more space-efficient, update-efficient, and cache-efficient in implicit maintenance of nblists compared to explicitly maintaining them. Also, unlike explicit nblists, a single octree can be used for any distance cutoff required for evaluating pairwise interactions among atoms.

An octree $\mathcal{T}$ is called $(\mathcal{K}, \alpha)$-admissible provided no leaf of $\mathcal{T}$ contains more than $\alpha\mathcal{K}$ points and each internal node has more than $\mathcal{K}/\alpha$ points, where $\mathcal{K} > 0$ is an integer and $\alpha \geq 1$.

Figure 1 shows an example of a (3,1)-admissible quadtree,[11] which is a 2D variant of octrees. The quadtree is constructed by recursive subdivision of the box bounding the initial set of points (i.e., circle centers) into nonempty quadrants. Observe that in the resulting tree each internal node contains more than $3/1 = 3$ points, but no leaf contains more than $3 \times 1 = 3$.

1.1. Construction of Octrees. Given a set P of points (or atom centers) in three dimensions, representing a simulated molecular system, consisting of arbitrary number of chemically connected components, a $(\mathcal{K}, \alpha)$-admissible octree can be constructed by first finding a box (or cube) bounding P and then recursively subdividing the initial cube into smaller nonempty subcubes until each subcube encloses at most $\alpha\mathcal{K}$ points. The straightforward recursive algorithm is shown in Supporting Information Figure 1. It can be shown that such a tree can be constructed in $O(n \log n)$ time, where $n = |P|$ (see Supporting Information).

1.2. Space Usage. In this section, we show that a $(\mathcal{K}, \alpha)$-admissible octree can be modified easily to use space linear in the number of atoms in the molecular system it stores. The trick is to use a *contracted octree*, which is obtained from a standard octree $\mathcal{T}$ by directly connecting each node of $\mathcal{T}$ to its nearest proper ancestor with more than one child.

Thus, each internal node of a contracted octree has at least two children. All algorithms we have described so far can be easily modified to work on contracted octrees without any asymptotic increase in running times. It can be shown that a contracted $(\mathcal{K}, \alpha)$-admissible octree $\mathcal{T}$ storing a molecular system containing n atoms uses $\Theta(n)$ space, where $\mathcal{K}$ is a positive integer and $\alpha \geq 1$ (as shown in the Supporting Information).

1.3. Computing Interactions. Given two octrees $\mathcal{T}_A$ and $\mathcal{T}_B$ storing molecules A and B (possibly $A = B$ and thus $\mathcal{T}_A = \mathcal{T}_B$), respectively, the pairwise interactions between atoms of A and B can be computed as follows. Suppose we are only interested in computing interactions between a pair of atoms provided their centers lie within a given distance cutoff d. We perform a simultaneous recursive traversal of $\mathcal{T}_A$ and $\mathcal{T}_B$ starting from their root nodes. Suppose at some point we are at node u of $\mathcal{T}_A$ and node v of $\mathcal{T}_B$. If the two cubes corresponding to u and v are separated by a distance (surface-

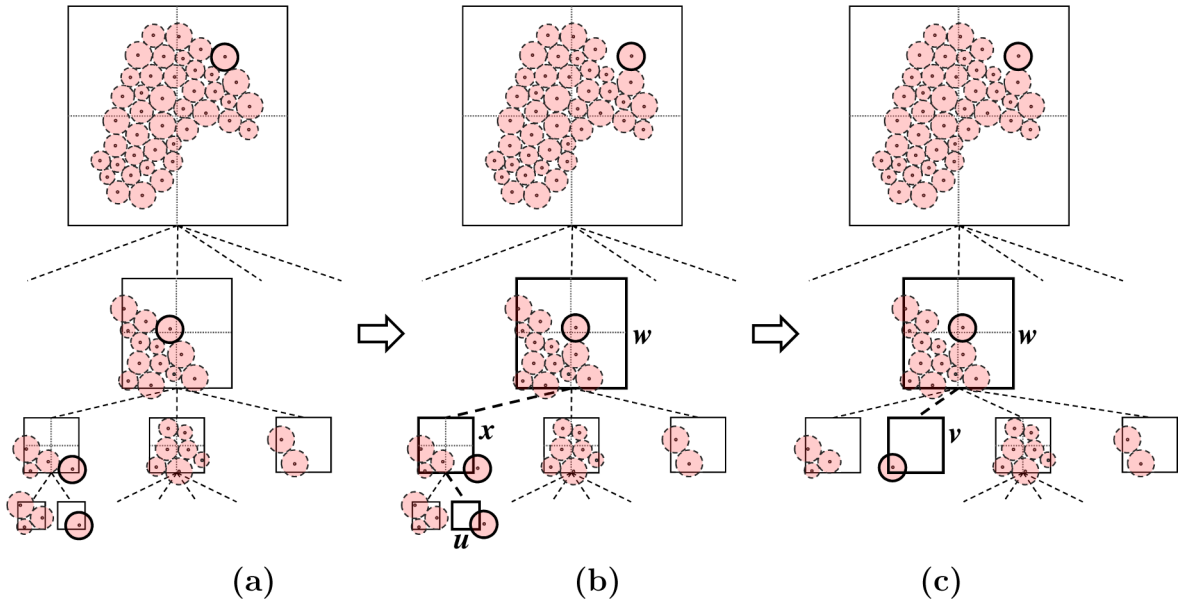


Figure 2. Updating the (3,1)-admissible quadtree from 1: (a) The point to be updated is marked with solid black outline. (b) The point is moved to a new location, and so it moves out of leaf u and internal node x but remains inside node w and all its ancestors. (c) Node u becomes empty, and so, it is removed. Node x now contains only 3 points, and so, it becomes a leaf. The updated point moves to the top-right quadrant of node w, which was empty before, and so, a new leaf v containing only the updated point is created.

to-surface) larger than d, then there can be no interaction. Otherwise, if both nodes are leaves, we consider each pair $\langle p \in u, q \in v \rangle$, and compute the interaction between p and q provided the two atom centers lie within distance d. If neither of the two conditions above hold, we move to the children of u and/or v and compute the interactions recursively. Supporting Information Figure 2 shows the pseudocode of the algorithm. It can be shown that all pairwise interactions between atoms within distance cutoff d can be computed in $O(nd^2(\delta d + \mathcal{K}^{1/3}))$ time, where δ is the time to compute one pairwise interaction (see Supporting Information). For any fixed value of d the running time of interaction calculation increases with the increase of $\mathcal{K}$. This happens because the larger the value of $\mathcal{K}$, the larger the region of unnecessary exploration. However, as computing interactions is a recursive function, the smaller the value of $\mathcal{K}$, the deeper the recursion and the larger the recursion overhead. Hence, one must chose a value of $\mathcal{K}$ that balances between these two types of overheads.

1.4. Updates. The key novelty of our work is that we allow fast updates of the octree under conformational changes of the molecular system, during the course of simulation. Figure 2 illustrates the basic idea behind such updates using the (3,1)-admissible quadtree from 1. Figure 2a shows the original state of the quadtree with the point (i.e., circle center) to be updated marked with solid black outline. The updated position of that point is shown in Figure 2b but without updating the structure of the quadtree. Observe that the point has moved out of leaf u and internal node x (i.e., parent of u) but remains inside node w (i.e., parent of x) and all its ancestors. Hence, in order to keep the structure of the quadtree correct, the point must be removed from nodes u and x. Figure 2c shows the quadtree after the removals. Observe that since leaf u is now empty it no longer appears in the quadtree and since the number of points inside x is less than 4 it has now become a leaf. Also observe that the updated point has now moved to the top-right quadrant of node w, which was originally empty. So, the new children (leaf v) of w are created corresponding to that quadrant containing the updated point. Figure 2c shows the structure of the quadtree after the update. Detailed description of the method together with pseudocode is given in the Supporting Information.

We prove in the Supporting Information that a $(\mathcal{K}, 2)$-admissible octree storing a molecular system of n atoms supports updates of atomic positions in $O(\log n)$ amortized time each, where $\mathcal{K}$ is a positive integer. However, observe that in molecular dynamics simulations atoms often move very short distances at every step, and the number of nodes on the shortest path between the leaves corresponding to the old and the new centers of the atom is very small. The update time will often be proportional to the length of this path, and thus much better than $O(\log n)$.

2. RESULTS AND DISCUSSIONS

We outline the experimental setup in section 2.1. The BYCC method for explicit nblists was originally implemented in the CHARMM program.[12] We reimplemented the BYCC method, and incorporated it into our molecular simulation library after a rigorous testing and validation phase that verified compliance with the CHARMM implementation. In order to avoid the costly update of nblists after each change of atomic position, we update them only when at least one atom has moved by more than 0.5 Å since the last update. The space requirements of explicit nblists and octrees are compared in section 2.1. Section 2.3 reports the running times and cache performance of a limited-memory BFGS minimizer[13] using the two data structures for maintaining molecular systems under conformational changes.

2.1. Experimental Setup. All algorithms were implemented in C, and tested on Intel Xeon 5680, using the icc compiler with optimization flag "-O3".

We ran our experiments on the following six proteins of varying sizes downloaded from the Protein Data Bank,[14,15] and prepared using CHARMM c34b2:[12,16] 1CLV (4680 atoms),

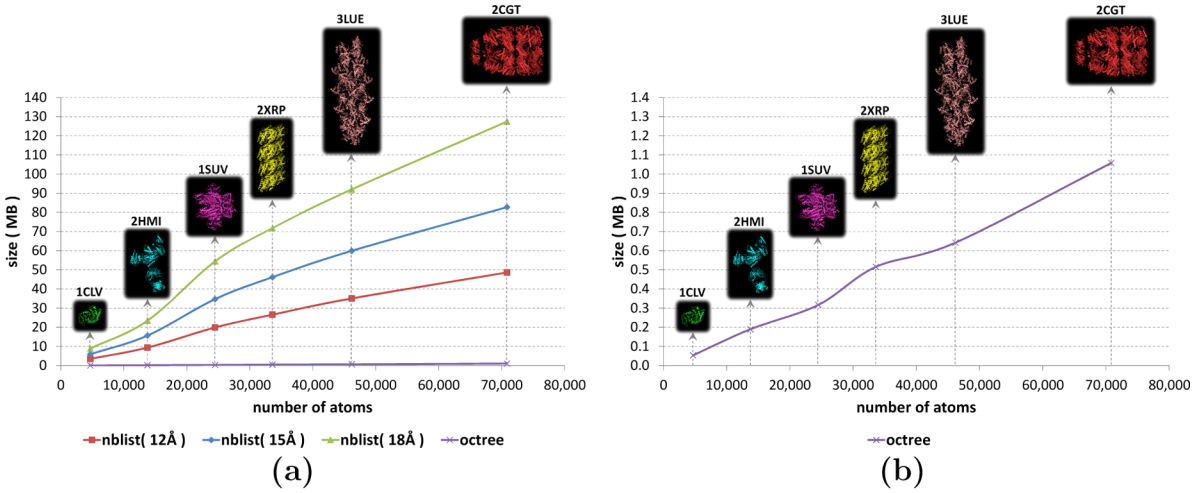


Figure 3. Plot a compares the sizes of of nblists and octrees constructed for six protonated proteins containing 4600−71 000 atoms. nblists are built for 12, 15, and 18 Å distance cutoff. Plot b zooms in on the octree curve in plot a.

2HMI (13 758 atoms), 1SUV (25 118 atoms), 2XRP (33 602 atoms), 3LUE (46 200 atoms), and 2CGT (70 847 atoms).

We have found that (60, 2) octrees work best for molecular simulation application.

In our experiments, we compare the cost of maintaining molecules under conformatonal changes using nblists and octrees. For this purpose, we use these data structures to store molecules that undergo changes in atomic positions during energy optimization using the limited-memory BFGS (LBFGS) algorithm.[13] LBFGS is a quasi-Newton optimization method that uses a limited memory variation of the Broyden−Fletcher−Goldfarb−Shanno (BFGS) algorithm[17−20] for solving nonlinear optimization problems. We have used our own C port of Jorge Nocedal's original Fortran implementation of the LBFGS algorithm.[13]

2.2. Space Usage. For each protein included in our experiments, we constructed nblists for each integral distance cutoff ranging from 9 to 20 Å.

Figure 3 compares the sizes of these data structures. In Figure 3a, we plot the space used by the octrees as well as by the nblists for 12, 15, and 18 Å distance cutoff. Figure 3b zooms in on the octree curve in Figure 3a. As evident from the plots both octrees and nblists for any given distance cutoff use space linear in the number of atoms in the molecular system. However, size of an nblist grows as the distance cutoff increases. Also, the size of an octree is much smaller than that of an nblist for the same system. For the six proteins in the plots, octree size varied from 55 KB (for 1CLV) to 1 MB (for 2CGT), while nblist size varied from 3.5 MB (for 1CLV) to around 50 MB (for 2CGT) for 12 Å distance cutoff, and from 9 MB to around 130 MB for 18 Å cutoff. Overall, nblists used from 45 to 65 times more space than octrees for 12 Å cutoff, and the range increased to 120−165 for 18 Å.

It turns out that for the proteins and the d values included in our experiments $3.7nd^{2.1}$ is a lower bound on the sizes of nblists while $1.7nd^{2.5}$ is an upper bound. In comparison, octrees use only $\Theta(n)$ space. As can be seen, octrees are orders of magnitude more memory efficient than nblists.

2.3. Running Times in Application to Local Minimization. Figure 4 compares the time required to complete 100 steps of the LBFGS minimizer using octrees and nblists as the distance cutoff for computing pairwise interactions grows. For smaller distance cutoffs the overhead of recursion in calculation of interactions slows down octree-based computation. However, that overhead gradually diminishes compared to the cost of actual computation as the cutoff value increases. Also, nblists constructed for large cutoff values are often too large to fit in the cache, and as a result, computations slow down due to costly cache misses. Octrees, on the other hand, incur very few cache misses as they are often small enough to fit into the cache. Detailed analysis of cache performance is provided in the Supporting Information.

We have compared performance of octrees vs nblists in different synthetic scenarios. We have used several nonbonded forces with different cutoffs.

Figure 4a plots the ratio of the running time of the nblists-based LBFGS minimizer to that of the octree-based minimizer when minimizing van der Waals (vdW) energy only. Octrees were up to 20% slower than nblists for distance cutoffs 9 Å−11 Å, comparable to nblists for cutoff values 12 and 13 Å and ran faster than nblists for all systems when the cutoff value reached 14 Å. Supporting Information Figure 6 reveals one of the major reasons behind the improving performance of octrees. As the distance cutoff increased, the sizes of nblists also increased, leading to increasing number of misses at various levels of the caches. On the other hand, much smaller sizes of octrees and their cache-friendly recursive traversal for energy evaluation meant that the number of cache misses incurred by the octree-based minimizer grew at a much slower rate.

Figure 4b is similar to Figure 4a but uses hydrogen bonding (hbond) energy in addition to vdW energy. The nblists-based minimizer evaluated the hbond energy using the same nblist constructed for evaluating vdW energy. However, since the same octree works for all distance cutoffs, the octree-based minimizer always evaluated the hbond energy using a 3 Å distance cutoff irrespective of what cutoff value was used for vdW energy. Because of the lower overhead in evaluating hbond energy, octrees achieved even better speedup with respect to nblists than in Figure 4a. For example, for 16 Å cutoff, octrees were 10−30% faster than nblists in Figure 4a, but in Figure 4b, the speedup values improved to 20−40%.

Figure 4c is for minimizing the sum of vdW, hbond, and Coulomb electrostatic (Coul) energy. While the nblists-based minimizer used the same distance cutoff for evaluating all three

dx.doi.org/10.1021/ct400474w | J. Chem. Theory Comput. 2014, 10, 4449−4454

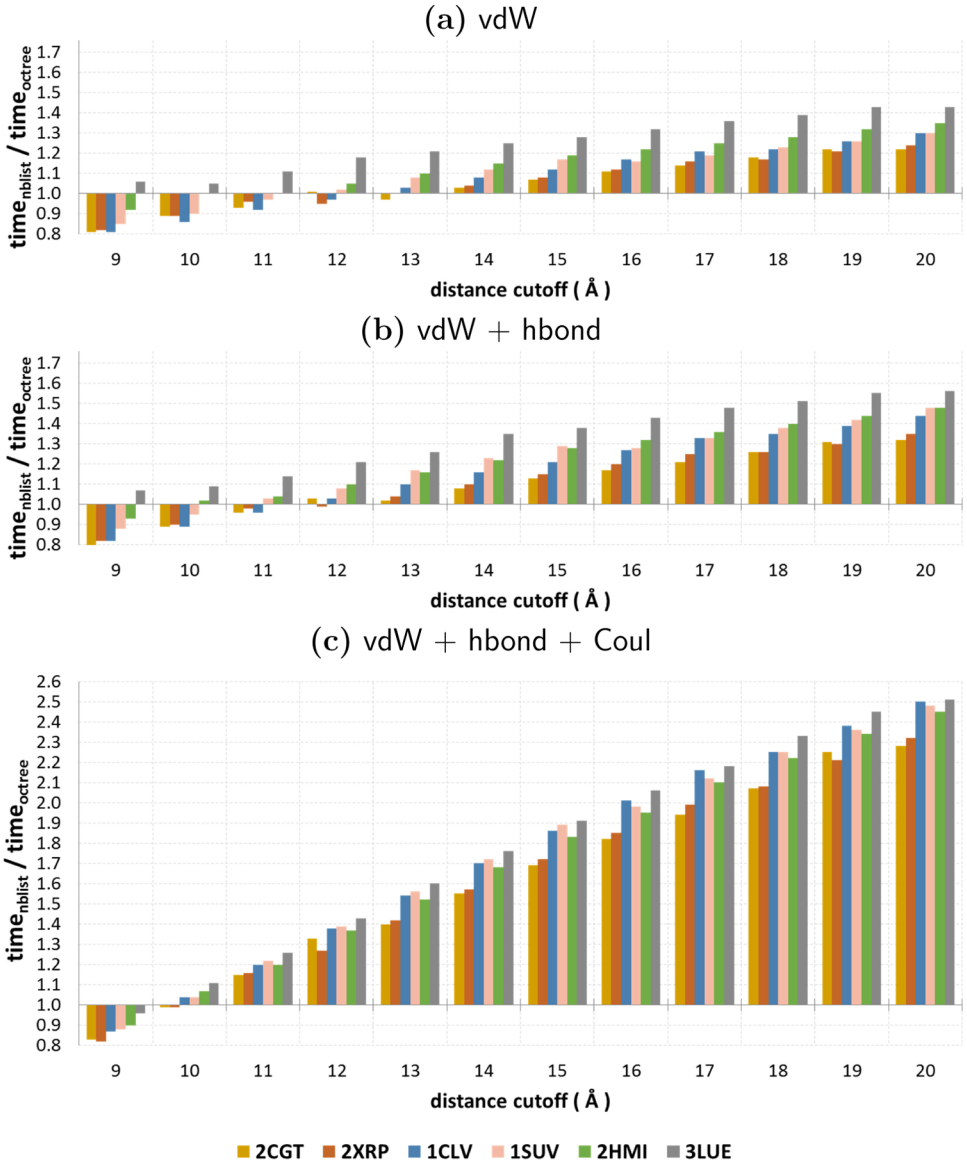


Figure 4. Time required for 100 steps of LBFGS minimization involving nblists and octrees: (a) minimize in (and hence compute) vdW energy for various distance cutoffs, where both data structures use the same cutoff value; (b) minimize in vdW plus hbond energy, where nblists use the same cutoff value (given in the horizontal axis) for both types of energy while octrees use a fixed 3 Å cutoff for hbond and the same cutoff as nblists for vdW (given in the horizontal axis); (c) minimize in vdW plus hbond plus Coul, where nblists use the same cutoff value (given in the horizontal axis) for all types of energy while octrees use a fixed 3 Å cutoff for hbond, a fixed 9 Å cutoff for vdW, and the same cutoff as nblists for Coul (given in the horizontal axis).

types of energy, the octree-based minimizer always used a 3 Å cutoff for hbond, and 9 Å for vdW. Octrees always outperformed nblists and ran 80−100% faster (i.e., 1.8−2.0 times faster) for 16 Å cutoff (see Figure 4c).

Figure 5 shows a breakdown of the total time spent in minimizing 2CGT into the time required for updating the data structure and the time needed for computing the energy. Implementations based on nblists and octrees have been compared. Observe that octree update times are almost insensitive to distance cutoffs while time required for updating nblists increases with the cutoff value. Even for a 9 Å cutoff, octrees can be updated more than 20 times faster than nblists, and for 20 Å, updating octrees can be more than 100 times faster. Though octree-based energy computation is slower than nblists for smaller distance cutoffs, this relative speed improves and octrees start to beat nblists for large cutoffs.

CONCLUSION

We have developed memory and cache efficient octree implementation of implicit nblists, which has almost 2 orders of magnitude lower memory usage as compared to existing implementations. In addition to tremendous reduction in memory requirements the code runs approximately 1.5 faster than nblists in practical use case scenarios. The algorithm described in the paper is suitable for effective parallelization and vectorization, as described and demonstrated in the Supporting Information section 2.2, and thus can be applied to Molecular Dynamics (MD). Current implementation is directly useful for simulations requiring multiple simultaneous minimizations, such as multistart Monte Carlo Minimization (MCM). The code is available under open source license.

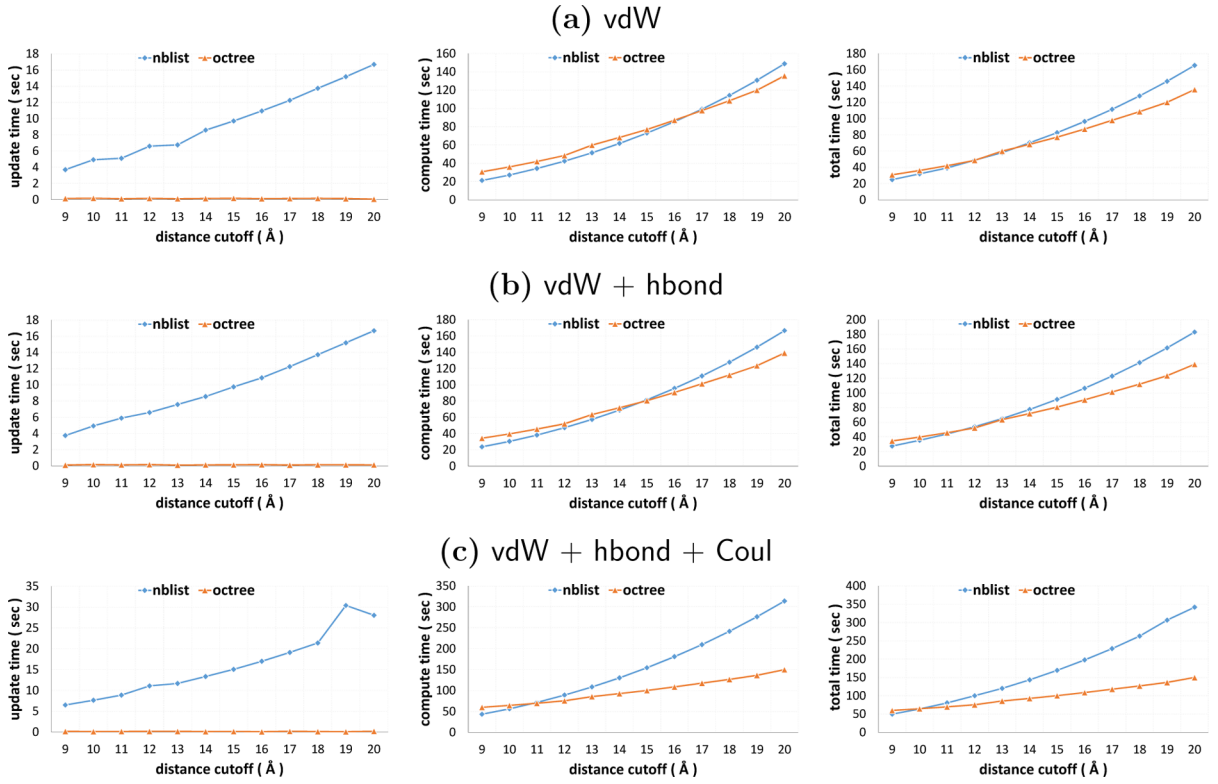


Figure 5. Plots show a breakdown of the total time required for running 100 steps of LBFGS minimization on 2CGT into update time and energy computation time. The nblists and octree implementations of the algorithm are compared. Please see 4 for the cutoff values used in parts a, b, and c.

■ ASSOCIATED CONTENT

Supporting Information

Proofs of the lemmas and corollaries. Detailed description of the octree update algorithm. Discussion of the cache complexity of the approach. Discussion of parallelization, load balancing, and vectorization, with practical implementation results. Pseudocode of the described algorithms on octree construction, traversing, and updates. This material is available free of charge via the Internet at http://pubs.acs.org/

■ AUTHOR INFORMATION

Corresponding Author

*Email: midas@bu.edu.

Notes

The authors declare no competing financial interest.

■ ACKNOWLEDGMENTS

This research was funded by National Institutes of Health (NIH) grants 1-R01-GM093147-01, R01-GM064700, 2-R01-GM061867, R01-GM074258, and R01-EB004873, National Science Foundation (NSF) grants DBI1047082 and CCF-1162196, grant 14.A18.21.1973 from Russian Ministry of Education and Science, and a grant from the UT-Portugal colab project.

■ REFERENCES

(1) Petrella, R.; Andricioaei, I.; Brooks, B.; Karplus, M. *J. Comput. Chem.* **2003**, *24*, 222−231.
(2) Steinbach, P.; Brooks, B. *J. Comput. Chem.* **1994**, *15*, 667−683.
(3) Verlet, L. *Phys. Rev.* **1967**, *159*, 98−103.
(4) Yip, V.; Elber, R. *J. Comput. Chem.* **1989**, *10*, 921−927.
(5) Brooks, B.; Bruccoleri, R.; Olafson, B.; States, D.; Swaminathan, S.; Karplus, M. *J. Comput. Chem.* **1983**, *4*, 187−217.
(6) Bajaj, C.; Chowdhury, R.; Rasheed, M. *Bioinformatics* **2011**, *27*, 55−62.
(7) Artemova, S.; Grudinin, S.; Redon, S. *J. Comput. Chem.* **2011**, *32*, 2865−2877.
(8) Jackins, C.; Tanimoto, S. *Computer Graphics and Image Processing* **1980**, *14*, 249−270.
(9) Greengard, L.; Rokhlin, V. *J. Comput. Phys.* **1987**, *73*, 325−348.
(10) Chowdhury, R.; Bajaj, C. Technical Report TR-10-17; Department of Computer Sciences, The University of Texas at Austin: Austin, TX, 2010, pp 548−556.
(11) Finkel, R.; Bentley, J. *Acta Inf.* **1974**, *4*, 1−9.
(12) Brooks, B.; Brooks, C.; Mackerell, A.; Nilsson, L.; Petrella, R.; Roux, B.; Won, Y.; Archontis, G.; Bartels, C.; Boresch, S.; Caflisch, A.; Caves, L.; Cui, Q.; Dinner, A.; Feig, M.; Fischer, S.; Gao, J.; Hodoscek, M.; Im, W.; Kuczera, K.; Lazaridis, T.; Ma, J.; Ovchinnikov, V.; Paci, E.; Pastor, R.; Post, C.; Pu, J.; Schaefer, M.; Tidor, B.; Venable, R.; Woodcock, H.; Wu, X.; Yang, W.; York, D.; Karplus, M. *J. Comput. Chem.* **2009**, *30*, 1545−1614.
(13) Liu, D.; Nocedal, J. *Mathematical Programming* **1989**, *45*, 503−528.
(14) Berman, H.; Westbrook, J.; Feng, Z.; Gilliland, G.; Bhat, T.; Weissig, H.; Shindyalov, I.; Bourne, P. *Nucleic Acids Res.* **2000**, *28*, 235−242.
(15) www-PDB, The Protein Data Bank. http://www.rcsb.org/pdb/ (accessed Aug. 28, 2014).
(16) www-CHARMM, CHARMM (Chemistry at HARvard Macromolecular Mechanics). http://www.charmm.org/ (accessed Aug. 28, 2014).
(17) Broyden, C. *IMA J. Appl. Math.* **1970**, *6*, 222.
(18) Fletcher, R. *Comput. J.* **1970**, *13*, 317.
(19) Goldfarb, D. *Math. Comput.* **1970**, *24*, 23−26.
(20) Shanno, D. *Math. Comput.* **1970**, *24*, 647−656.