A Fast Parallel Clustering Algorithm for Molecular Simulation Trajectories

Yutong Zhao,[a,b] Fu Kit Sheong,[a] Jian Sun,[c] Pedro Sander,[b] and Xuhui Huang*[a,d,e]

We implemented a GPU-powered parallel *k*-centers algorithm to perform clustering on the conformations of molecular dynamics (MD) simulations. The algorithm is up to two orders of magnitude faster than the CPU implementation. We tested our algorithm on four protein MD simulation datasets ranging from the small Alanine Dipeptide to a 370-residue Maltose Binding Protein (MBP). It is capable of grouping 250,000 conformations of the MBP into 4000 clusters within 40 seconds. To achieve this, we effectively parallelized the code on the GPU and utilize the triangle inequality of metric spaces. Furthermore, the algorithm's running time is linear with respect to the number of cluster centers. In addition, we found the triangle inequality to be less effective in higher dimensions and provide a mathematical rationale. Finally, using Alanine Dipeptide as an example, we show a strong correlation between cluster populations resulting from the *k*-centers algorithm and the underlying density. © 2012 Wiley Periodicals, Inc.

DOI: 10.1002/jcc.23110

Introduction

Molecular dynamics (MD) and Monte Carlo are powerful techniques to explore conformational space and to probe dynamics of macromolecular systems. Recent advances in the molecular simulations have enabled us to generate massive datasets with millions of conformers.[1–7] A popular approach to reduce the complexity of such large datasets is through the use of clustering algorithms, whereby conformers of similar shapes are lumped together[8–11] to select representative conformations from MD trajectories.[9] However, clustering based on geometry alone tends to be insufficient to recover meaningful kinetic information.[8,12] Instead, geometric clustering is often used to preprocess conformational space prior to kinetic clustering, such as in the construction of Markov state models (MSMs).[7,13–17]

Almost all geometric clustering algorithms essentially have the same goal: given a collection of conformations and a distance function, partition conformations into disjoint subsets so as to minimize the overall distance of each conformation to an associated cluster center; some algorithms such as *k*-means and *k*-medoids[18,19] attempt to minimize the total distance of each point to its assigned center, while others such as *k*-centers[20] attempt to minimize maximum cluster radii. Ideally, the choice of the distance function, coordinates, and algorithm, should prescribe some notion of similarity for data conformations within each cluster.[9,21] For example, Euclidean distance on internal coordinates has been used to cluster RNA structures.[22] In almost all cases, given the very large datasets resulting from large-scale MD simulations, clustering algorithms need to perform in approximately linear O(N) time or better, where N is the number of conformations. However, many popular clustering algorithms tend to be quadratic $O(N^2)$ in running time, and thus are prohibitively slow for datasets with millions of conformers and an expensive distance function such as the optimally superimposed root mean squared deviation ($RMSD_{opt}$). In some applications,[23] geometric clustering is repeated numerous times—exacerbating the need for a fast clustering algorithm.

k-medoids and *k*-means are two popular clustering methods that attempt to minimize the sum of squared distances within each cluster.[18,19] They require the number of centers to be specified *a priori*, by means of silhouetting[24] or some other appropriate techniques. Furthermore, it has been reported that *k*-medoids has a propensity to lump densely sampled folded conformations and sparsely sampled unfolded conformations into the same cluster.[12,16,25] For *k*-medoids, the popular implementation[26] has $O(N^2)$ running time and is unsuitable for large datasets. The neighbor algorithm is another widely adopted clustering algorithm on MD conformations.[11,27] As opposed to requiring a predetermined number of centers, it instead predefines a cut-off distance and picks cluster centers *ad hoc*. In addition, the algorithm is unable to properly identify elongated clusters and is

[a] *Y. Zhao, F. K. Sheong, X. Huang*
Department of Chemistry, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong
[b] *Y. Zhao, P. Sander*
Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong
[c] *J. Sun*
Mathematical Sciences Center, Tsinghua University, Beijing 100084, China
[d,e] *X. Huang*
Center of Systems Biology and Human Health, School of Science and Institute for Advance Study, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, Division of Biomedical Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong
E-mail: xuhuihuang@ust.hk

Contract/grant sponsor: Hong Kong Research Grants Council GRF; Contract/grant numbers: 661011, F-HK29/11T, HKUST2/CRF/10, 619509; Contract/grant sponsor: University Grants Council; Contract/grant number: SBI12SC01; Contract/grant sponsor: NIH; Contract/grant number: R01-GM062868.

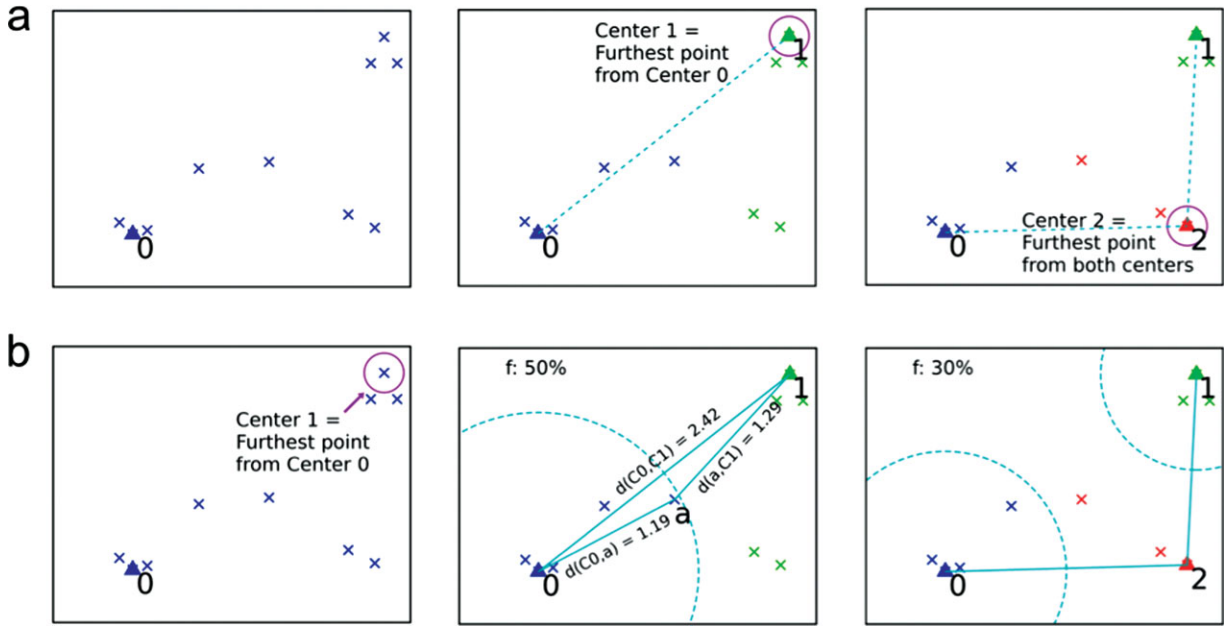

Figure 1. A step-by-step schematic diagram illustrating the (a) the naive *k*-centers algorithm and (b) the application of the triangle inequality. In the *k*-centers algorithm, the choice of subsequent centers is always the point furthest away from all previously found centers (denoted by triangles). However, it is possible to speed up this algorithm drastically by first pre-computing the center-to-center distances and applying the triangle inequality. As shown in the middle panel of b, the blue semidotted circle denotes the area of coverage provided by the triangle inequality, whereby all points within this dotted circle do not need a distance calculation. For example, the distance between centers 0 and 1 is 2.42, the distance between point a and center 0 is 1.19, which is less than 2.42/2 = 1.21. Indeed, the triangle inequality then tells us this point must be closer to center 0 than center 1, and thus there is no need to explicitly calculate the distance between point a and center 1. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

strongly biased by the cut-off distance.[8] Other clustering algorithms such as linkage and common-nearest-neighbor have also been applied to analyze MD trajectories,[8,9] but they all suffer from being approximately $O(N^2)$ in running time.

In this article, we choose to focus on the *k*-centers algorithm, a linear $O(kN)$ running time algorithm that attempts to minimize the maximum cluster radii.[20] The general problem of finding the set of centers that best minimizes the sum of maximum cluster radii, $\min\sum_k r_{max}$, out of all $C(N, k)$ possible combinations of the *k*-centers chosen from *N* points, is very complicated (at least NP-Hard).[20] However, Hochbaum and Shmoys[20] have shown that the score $(\sum_k r_{max})$of an approximate implementation of the *k*-centers algorithm is off from min $\sum_k r_{max}$ by at most a factor of two. In this implementation, we start with one center, and iteratively walk up to a total of *k*-cluster centers by finding the point furthest away from all previously found cluster centers. Under the Euclidean metric, *k*-centers algorithm sequentially partitions the dataset into convex Voronoi cells each of approximately equal size. Currently, the *k*-centers algorithm is a critical component of MSMBuilder,[13,16] a statistical framework to construct MSMs from MD trajectories. However, large datasets can take up to one day or more to cluster, and successfully constructing a MSM may require repeated *k*-centers clustering trials. For these reasons, a fast *k*-centers clustering algorithm will be very useful to accelerate clustering and analysis of MD datasets and will enable us to reach much larger number of cluster centers if need be.

In this work, we exploit the properties of metric space to rapidly accelerate the speed of *k*-centers by means of the triangle inequality and parallelize the algorithm on the GPU. Overall our code is up to two orders of magnitude faster when compared to the CPU code. The GPU has several distinct features that make it particularly suitable for parallelization of the *k*-centers algorithm. First, the high density of the GPU's Arithmetic Logic Units makes it naturally suitable for expensive operations such as $RMSD_{opt}$. Second, in the *k*-centers algorithm, all the threads must be repeatedly synced together to find the subsequent cluster center; the low latency in the GPU's inter-thread communication makes it particularly suitable for this task. Finally, in the *k*-centers algorithm, operations such as finding the point with the greatest distance to a given center reduces to the classic and well-studied GPU parallel reduction problem.[28] We implemented our algorithm using NVIDIA's CUDA platform[29] due to its versatile and easy to use API. In general, clustering algorithms exhibit a natural structure amenable for parallelization in that the calculation of the distance from one point to a center is independent of other points. As one might expect, GPU parallelization of clustering algorithms has been investigated in the past.[30,31] However, they were not optimized for MD conformations within general metric spaces endowed with expensive distance functions, and they certainly do not leverage the powerful triangle inequality.

Methodology

The k-centers Algorithm

The implementation of the *k*-centers algorithm[20] is herein described (Fig. 1a). For notational purposes, we use the terms point and conformation interchangeably, as a conformation is represented as a point in R^{3D}.

- Initially, we pick a random point as the starting center, denoted as center 0. Then, we calculate the distances of all N points to the current center and find the point Y furthest from center 0. Point Y then becomes the new center for the next iteration.
- We then calculate the distance of all N points to the new center and see if it is closer than the currently closest center. If so, this new center becomes assigned to the point.
- Afterwards, we find the point that is furthest away from all previously found centers.
- We repeat this process for k iterations or until some minimum cluster radii criteria is satisfied.

We choose to prescribe the number of centers *a priori*. On each iteration, we consider the set of minimum distances of each point to all possible centers. The choice of the subsequent center is the point that attains a maximum in the aforementioned set. As such, the k-centers algorithm thereby minimizes the maximum distance of each point to all possible centers. It is easy to see that the running time of the naive implementation of the k-centers algorithm is $O(kN)$, where k is the number of centers, and N is the number of points.

Distance Function

Given a system with N atoms, a particular conformation resides in R^{3D} Euclidean space, where D corresponds to the chosen number of heavy atoms or C-α atoms. We formally define the distance function, $\mathrm{RMSD}_{\mathrm{opt}}(x,y)$, between two conformations x and y as follows.

Let $Y \subset R^{3D}$ be the set of all possible rigid-body translations and rotations of a $y \in R^{3D}$

Then,

$$\mathrm{RMSD}_{\mathrm{opt}}(x,y) = \min_{y \in Y} \sqrt{\frac{1}{D}\sum_{1}^{3D}(x_i - y_i)^2} \tag{1}$$

In other words, the distance function is the standard RMSD after we optimally superimpose one conformer onto the other. We implemented Theobald's[32] quaternion characteristic polynomial (QCP) method to calculate $\mathrm{RMSD}_{\mathrm{opt}}(x,y)$, due to its speed and accuracy. Alternative methods include exact solvers[16] and the cyclic-Jacobi method.[33]

We emphasize that $\mathrm{RMSD}_{\mathrm{opt}}$ is very expensive to calculate, as our GPU implementation of the QCP method requires hundreds to thousands of mixed single and double precision floating point operations per calculation. Therefore, it is of critical importance that we reduce the total number of distance calculations.

The triangle inequality

Steipe[34] have previously provided a complete and rigorous proof showing that $\mathrm{RMSD}_{\mathrm{opt}}$ indeed satisfies the axioms of a metric. That is, (R^{3D}, $\mathrm{RMSD}_{\mathrm{opt}}$) forms a metric space, and $\mathrm{RMSD}_{\mathrm{opt}}$ is non-negative, discernible, symmetric, and satisfies the triangle inequality. In particular, we can take advantage of the triangle inequality to drastically speed-up the k-centers clustering algorithm. We show how this is done as follows.

By the triangle inequality:

$$d(c_i, c_j) \leq d(x, c_i) + d(x, c_j) \tag{2}$$

and suppose, we know $d(x, c_i) \leq \frac{d(c_i,c_j)}{2}$, then we know by substituting in (2):

$$d(x, c_i) \leq d(x, c_j) \tag{3}$$

where x is a particular point, c_j refers to the currently tested center, c_i refers the center that x is assigned to, and $d(x,y)$ is just short form for $\mathrm{RMSD}_{\mathrm{opt}}(x,y)$.

The incurred penalty for the triangle inequality is the cost of calculating the center-to-center distances at the start of each iteration—taking an additional $O(k^2)$ time and $O(k)$ memory compared to the naive implementation. An illustration of the triangle inequality is explicitly shown in Figure 1b, and the pseudo-code is available in Supporting Information Figure S1. Geometrically, the triangle inequality filters out the most of the points within a circle of some radius $\frac{d(c_i,c_j)}{2}$ and partitions the dataset into two disjoint sets: the set of points A that lie inside any circle, and the set of points B that lie outside every circle. All points in B fail the triangle inequality, and most of the points in A pass the triangle inequality.

Strictly speaking, the above formulation is a weaker version of the triangle inequality, as the current implementation does not always guarantee that all points within all circles are able to bypass the distance calculation. For example, in the bottom right circle of Figure 4d (right panel), some points are red and fail the triangle inequality despite lying inside the said circle. However, these cases tend to take up a relatively small fraction of the total number of points. For the sake of completeness, we also derive a stronger triangle inequality that does provide more coverage (full details are available in Supporting Information). For technical reasons, we did not implement the stronger triangle inequality on the GPU because the implementation requires dynamically allocated arrays and linked lists. These data structures are not as amenable to the parallel single instruction multiple data processing and memory architecture of the GPU.

We denote f_{avg} as the average fraction of points that fail the triangle inequality. In general, we denote the f-value as the fraction of points that fail the triangle inequality on a given iteration.

The running time of k-centers using the triangle inequality is $O(k^2 + kf_{\mathrm{avg}}N)$, $0 < f_{\mathrm{avg}} < 1$. Note that when f_{avg} is approximately equal to 1, the triangle inequality k-centers in fact performs worse in running time than the naive implementation time due to the overhead associated with calculating center-to-center distances. However, we later on demonstrate that for purposes of clustering conformations from MD simulations, f_{avg} is almost always significantly smaller than 1 and that the implementation behaves exceptionally well for relevant datasets.

Parallelizing k-centers with triangle inequality on the GPU

We parallelized the triangle inequality k-centers algorithm on the GPU using CUDA. We parallelize across all the points, that

is, we launch N total threads to process N points independently. We arranged the coordinates of conformations in dimension-major convention to facilitate coalesced memory access (so that threads read from and write to memory contiguously). Furthermore, since the number of centers is usually much smaller than the number of points, we allocate an auxiliary storage array in global memory for the coordinates of previously found centers so that loading of their coordinates also proceeds in a coalesced manner. We also fine-tuned the double-precision QCP algorithm[32] into mixed single and double precision to maximize performance. Furthermore, we implemented a parallel reduction kernel to find the point furthest away from all previous centers.[35]

To minimize idle threads in the GPU, we also parallelize the triangle inequality component by compacting all the points that fail the triangle inequality into a single continuous array. Intra-warp conditionals (ifs, elseifs, switches) and nested conditionals are deleterious to performance due to the GPU's single instruction, multiple data architecture. In particular, unless all threads within a warp branch off in the same manner, multiple passes are needed for each conditional. Otherwise, idle threads on the streaming multiprocessor will adversely affect the performance. Thus, it is of critical importance to circumvent divergent conditionals. Because it is unlikely that all points within an execution warp of 32 threads fail the triangle inequality, a significant fraction of threads will idle on the streaming multiprocessor. To circumvent this problem, during each iteration, we first preprocess by launching a kernel with N threads in ceil(N/TPB) blocks, where TPB is the number of threads per block. We utilize Harris et al.'s[36] scan-based compaction kernel to sieve the points, whereby we identify all the points that necessarily fail the triangle inequality and compact them into a contiguous array that is then subsequently passed into a new RMSD kernel. Compaction is very fast and has an almost negligible overhead. This new kernel launches $f*N$ threads in ceil((fN)/TPB) blocks. Now every thread in this new kernel necessarily fails the triangle inequality and there is no more thread divergence, except possibly in the last block.

We tested our data on both the 3GB GTX 580, a consumer-grade gaming card, and the 3 GB Tesla C2070, a professional computing card. Our datasets were generated from MD simulations.

MD simulation datasets

Alanine Dipeptide (Dipep). We adopted the simulation dataset of the terminally blocked alanine peptide Ac-ala-NHMe generated by Chodera et al.[23] In particular, the conformations are extracted from the trajectories obtained from the 400 K replica of a replica exchange simulation. Altogether, there are 975 trajectories, each of which contains 20 ps simulation with conformations stored every 0.1 ps, thus totally 195,000 conformations. See Ref [23] for additional details of this simulation dataset.

β-hairpin Tryptophan Zipper 2 (Trpzip2). The MD simulation dataset for the Trpzip2 peptide was partially taken from Zhuang et al.[37] In particular, we have extended the 350 K MD simulations reported in Ref. [37] to contain 1000 simulations. As these simulations were generated in the Folding@Home distributed computing environment,[38] their length slightly varies and altogether we have collected 78,305 ns worth of simulation time. With a saving interval of 5 ps, our dataset reaches ~15.68 million conformations in total. The simulations were performed using the AMBER03 force field,[39] and the simulation box contains 2923 TIP3P[40] waters and two Cl^- ions. All the simulations were performed using a version of the Gromacs[41] modified for the Folding@Home infrastructure.[38] Please refer to Ref. [37] for additional details of the simulation setup.

Human Islet Amyloid Polypeptide (hIAPP). The MD dataset for hIAPP contains 20 simulations with slightly various lengths at an average of 198 ns each. We saved conformations every 2 ps, thus the total number of conformations is ~1.98 million. The initial conformations of our 20 simulations were randomly selected from a 300-ns NVT simulation at 600 K. The initial structure for this 600 K simulation was taken from the solution NMR structure of hIAPP (PDB ID: 2KB8[42]). The simulations were performed using the AMBER99SB force field.[43] The peptide was solvated in the implicit solvent using the OBC GB model[44] with solvent $\sigma = 78.3$ and surface tension $= 2.25936$ kJ/(mol nm^2). The Andersen thermostat[45] was adopted for the temperature coupling with a coupling constant of 1 ps^{-1}. The simulations were performed using the GPU module of the Gromacs 4[46] software supported by the OpenMM library.[47]

Maltose Binding Protein. The MBP dataset consists of 25 50-ns MD simulations with the saving interval of 5 ps with a total of 0.25 million conformations. The simulations were performed using the AMBER03 force field.[39] The initial protein structure was taken from the X-ray crystal structure of apo-open Maltose Binding Protein (MBP; PDB ID: 1OMP[48]). The protein molecule was solvated in a water box with 19,038 TIP3P[40] waters and eight Na^+ ions. The simulation system was minimized using a steepest descent algorithm, followed by a 200-ps MD simulation applying a position restraint potential to the protein heavy atoms. All the simulations were performed under NPT ensemble with $P = 1$ bar and $T = 310$ K. Temperature annealing was also applied to raise the temperature from 50 to 310 K at the first nanosecond of each simulation. The velocity-rescaling thermostat[49] with a coupling constant of 0.1 ps^{-1} and the Berendsen barostat[50] with a coupling constant of 1 ps^{-1} were used for the temperature and pressure coupling respectively. A cutoff of 12 Å was used for both van der Waals and short-range electrostatic interactions. Long-range electrostatic interactions were treated with the particle-mesh Ewald method.[51] Nonbonded pair-lists were updated every 10 steps with an integration step size of 2 fs in all simulations. All bonds were constrained using the LINCS algorithm.[52] The simulations were performed using the Gromacs 4 software.[46]

Overall, the Dipep, Trpzip2, hIAPP, and MBP simulation datasets contain around 0.20, 15.68, 1.98, and 0.25 million conformations respectively. In addition, we have also generated several sets of uniformly random datasets to serve as references for analyzing the performance of the triangle inequality. The random datasets have the same dimension as the

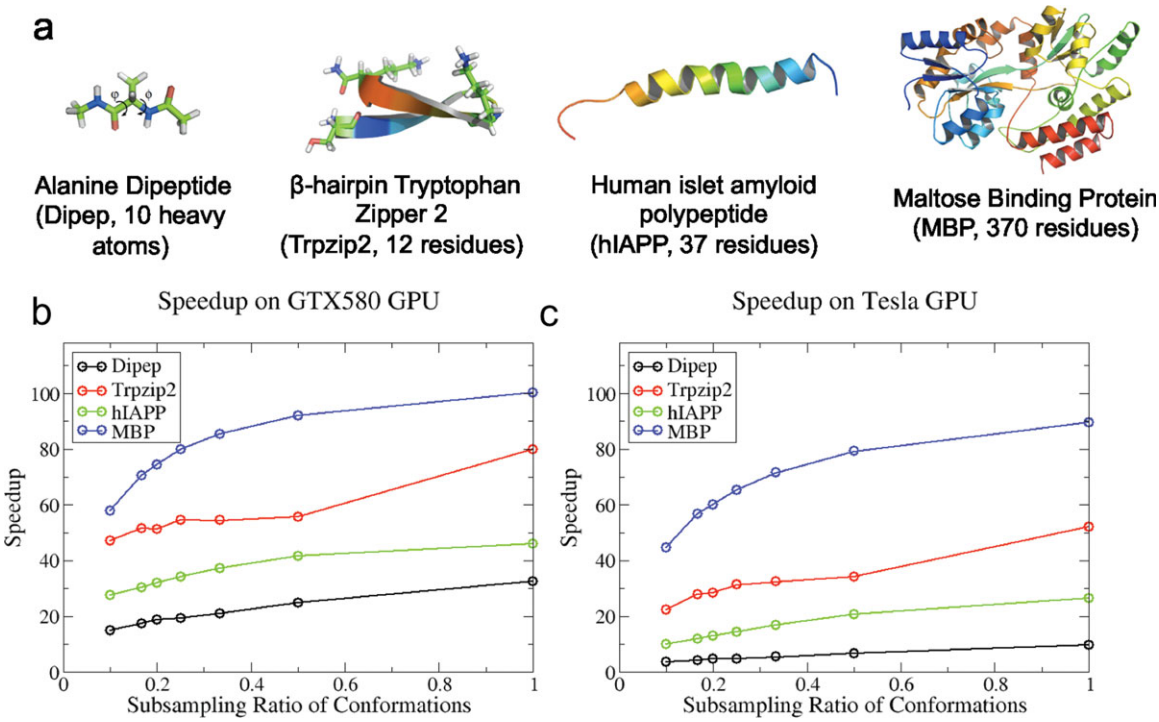

Figure 2. Speedup using GPU code compared against the CPU code on four different MD simulation datasets. (a) The four systems of interest are: Alanine Dipeptide (Dipep, 0.20 million conformations), β-hairpin Tryptophan Zipper 2 peptide with 12 residues (Trpzip2, PDB ID: 1LE1, 15.68 million conformations), Human Islet Amyloid Polypeptide with 37 residues (hIAPP, PDB ID: 2KB8, 1.98 million conformations), and MBP with 370 residues (MBP, PDB ID: 1OMP, 0.25 conformations). Speedup as a function of subsampling ratio, the fraction of the total number of available conformations, of the datasets using the GTX 580 GPU and Tesla GPU are shown in (b) and (c), respectively. The speedup is tested with the number of centers $k = 4000$. For all systems except Dipep, we use only the backbone C-α atoms. For Dipep, we use 10 heavy atoms to compute RMSD.

corresponding protein MD datasets, but do not have any bond length or angle constraints.

Results and Discussions

In this section, we test our algorithm on four different protein datasets and show that the performance of our algorithm is up to two orders of magnitude faster than the CPU implementation. In particular, the algorithm performs better on large datasets. The algorithm is also well-behaved under varying number of clusters (k), and is essentially linear in running time with respect to k. Furthermore, we show that the triangle inequality performs relatively well on protein datasets even in high dimensions when compared against corresponding randomly distributed datasets. Finally, in a simple dipeptide example, we show that the k-centers algorithm is able to faithfully recover the underlying density of the dataset in contrast to the popular k-medoids algorithm. Note that for purposes of evaluating the RMSD function, we use Cα atoms for MBP, hIAPP, and Trpzip2 datasets. For Dipep, we include the 10 heavy atoms for the RMSD calculations.

Speedup compared to the CPU implementations

We compare the speedup of the GPU *versus* the CPU on four different protein datasets, ranging from the small Alanine Dipeptide to the large 370-residue MBP (Fig. 2a). Although benchmarking, we chose to vary the number of conformations by varying the fraction of the total number of conformers available, for example, a subsampling ratio of 0.4 uses 40% of the total available conformers for that dataset. As shown in Figure 2b, when using the full-dataset, we observe a 100-fold speedup in MBP (68 min *vs*. 40 s, 0.25 million conformations), an 80-fold speed-up in hIAPP (4 min *vs*. 5 s, 1.98 million conformations), a 42-fold speedup in Trpzip2 (40 min *vs*. 30 s 15.68 million conformations), and a 32-fold speedup in Dipep (39 s *vs*. 1.2 s, 0.20 million conformations). In general, the larger the fraction of total available conformations and the larger the size of the protein, the better the performance is. We attribute the inferior speedup of the Dipep dataset to its small size (both in the number of conformers, and the number of dimensions), which in turn, is unable to fully utilize the arithmetic logic units on the GPU.

Overall, the observed speedup is likely a result of several components: the coalesced reads and writes from GPU memory, the fast all-against-one RMSD_{opt} kernel, and parallel reduction. In addition, we have been able to efficiently parallelize the triangle inequality by avoiding divergent threads. Surprisingly, the algorithm performs slightly worse on the professional grade Tesla C2070 (Fig. 2c) than the consumer grade GTX580 (Fig. 2b)—despite the C2070's superior double precision floating operation performance. Upon further investigation, we attribute this to the GTX580's superior single precision floating-point operation speed and the greater memory

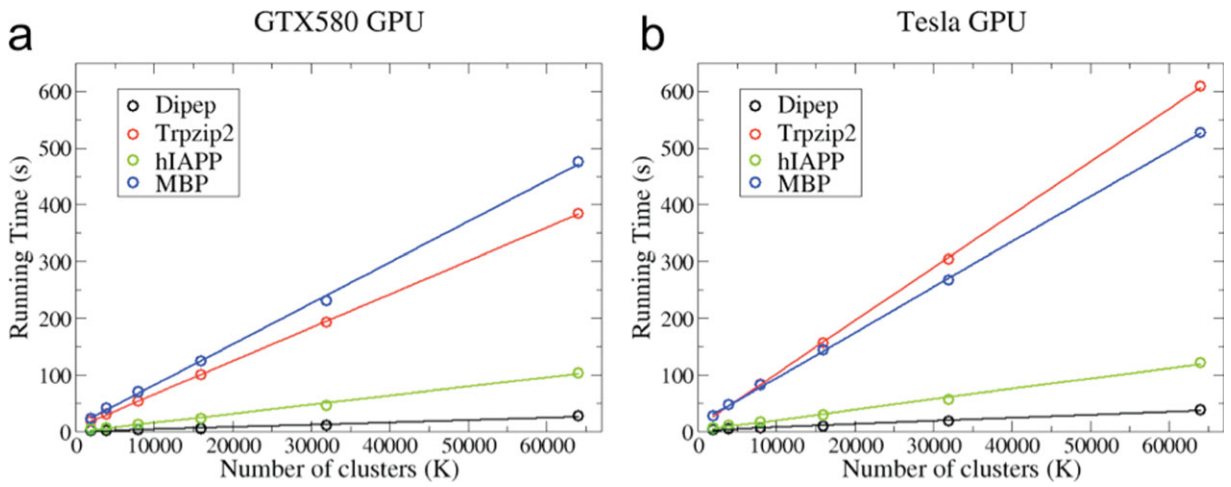

Figure 3. Linearity in the running time as we increase the number of clusters (*K*). [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

bandwidth. We also note that the CPU version of the RMSD calculation (from MSMBuilder) uses exclusively single precision operations[13]—as such, our GPU algorithm should in fact fare better against a mixed single-double precision implementation of Theobald's algorithm.[32] Note that these results disregard the *I/O* time from the hard-drive to CPU memory in both CPU and GPU implementations, as (1) *I/O* from hard drive is not always needed (the data may already exist in memory), and (2) transfer from hard drive to memory is very much implementation and hardware dependent. For all benchmarks, we note that both GPU and CPU implementations take advantage of the triangle inequality (see the next section for more details).

Our algorithm is also well behaved as we increase the number of clusters in all datasets. As shown in Figure 3, we observe that the algorithm is linear in running time with respect to *k*, with correlation coefficient > 0.99 for all datasets. If desired, we are able to increase the number of centers by more than an order of magnitude with only a linear increase in the amount of time needed. Thus, the algorithm allows one to create a much higher resolution of clusters if needed. For example, in the hIAPP dataset (1.98 million conformations, 37 residues), we generated 128,000 clusters in only 4 min and 19 s.

Triangle inequality

The triangle inequality is able to greatly speedup the algorithm by reducing the total of RMSD calculations needed. As shown in Table 1, the triangle inequality alone is able to speedup clustering of the hIAPP dataset by almost 17 times. For Trpzip2 the observed speedup is about 11 times. The difference in performance is due to various factors, such as the dimensionality of the data, the underlying density, and the overhead in parallelizing the algorithm. For example, in the Dipep dataset, the observed speedup is only threefold, most likely due to the inherent cost of the overhead associated with GPU memory transfer and other initialization routines. For MBP, a dataset with over 370 residues, the speedup is only approximately twofold due to the large associated dimensionality of the dataset. The effect of the dimensionality and underlying density is explained in detail in the next two paragraphs.

To better understand why a high dimensional space makes the triangle inequality less effective, we must consider the effective coverage of a high dimensional *n*-sphere within an *n*-hypercube [see Fig. 1b for the two-dimensional (2D) case]. To aid us in our understanding, we analyzed the property of the triangle inequality in high dimensional space endowed with the Euclidean metric. Suppose, we have an *n*-cube with length at most a constant factor *p* bigger than the radius of an *n*-sphere. Then, the ratio of the volume of a 2*n*-sphere to the volume of a 2*n*-hypercube is:

$$\frac{SV_{2n}}{CV_{2n}} = \frac{\pi^n}{n!p^{2n}} \tag{4}$$

Similarly, the ratio of the volume of a $2n + 1$ sphere to the volume of a $2n + 1$ hypercube is:

$$\frac{SV_{2n+1}}{CV_{2n+1}} = \frac{2^{2n+1}n!\pi^n}{(2n+1)!p^{2n+1}} \tag{5}$$

and in the limit as *n* tends to infinity, an *n*-sphere takes up a negligible amount of space in an *n*-cube as the factorial term in the denominator dominates, as $p > 1$. In our algorithm, we have at most a constant number of clusters, and therefore a constant number of *n*-spheres. Even if all the *n*-spheres are disjoint in coverage, we still multiply the above ratio by at most a constant factor, for example, 4000 clusters. For a derivation of eqs. (4) and (5) please refer to Supporting Information.

Table 1. Running time (s) of the datasets with and without the triangle inequality on the GTX 580 GPU.

Time (s)	Dipep	Trpzip2	hIAPP	MBP
With T-Ineq	1.20	29.94	5.49	40.70
Without T-Ineq	3.72	332.77	92.89	89.90

We used $k = 4000$ and the full number of conformations for each dataset (See Methods for details).

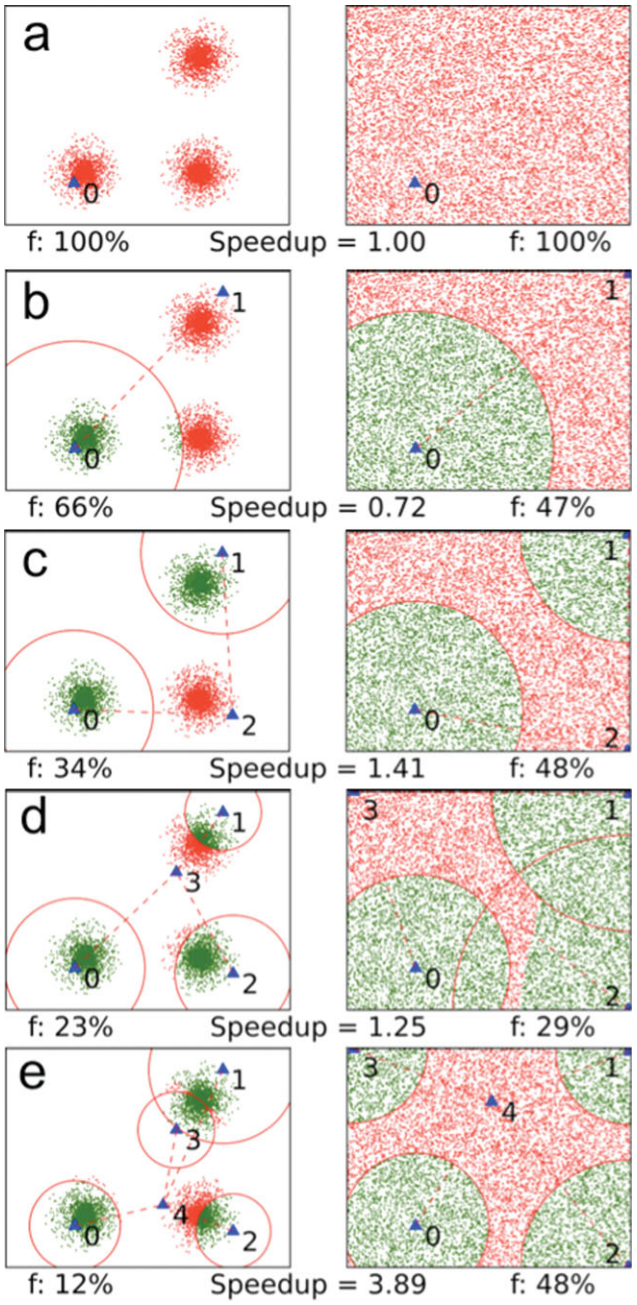

Figure 4. The performance of the triangle inequality on *k*-centers clustering for two 2D datasets with differing densities. The results for $k = 1$–5 are shown in (a)–(e), respectively. Left panels correspond to a dataset with distinct dense regions, which mimic the free energy landscape with multiple metastable regions. Right panels correspond to a dataset with a uniformly random distribution. The red circles within each diagram depict the total amount of coverage provided by the triangle inequality. Green points pass the triangle inequality, and red points fail the triangle inequality. *f* is the fraction of the points that fail the triangle inequality. The speedup is how much faster the left panel takes to compute when compared to the right panel.

The triangle inequality tends to perform better on datasets with dense regions separated by sparse regions (dense dataset) than uniformly distributed datasets (uniform dataset). Figure 4 depicts two artificially generated 2D datasets of varying density. In the dense dataset, when compared against the uniform dataset, a single circle provided by the triangle inequality covers a greater proportion of all points because all the points are quite close to the center of the circle (left panels, Fig. 4). Furthermore, in the dense dataset, increasing the number of centers leads to a rapid decrease in *f*, the fraction of points that fail the triangle inequality (red points, Fig. 4). Overall this speeds up the algorithm due to the f_{avg} term in the $O(k^2 + kf_{avg}N)$ running time.

Biomolecular simulations datasets generally have features of dense regions (free energy minimums) separated by sparse regions (free energy barriers),[53] therefore, we expect that the triangle inequality perform well on these datasets. Indeed, as shown in Table 2, we observe that in all randomly generated datasets (of equal number of conformations and dimensions to the corresponding protein MD dataset), the *f*-values are greater than 90%, whereas in the corresponding protein MD dataset, the *f*-values are much smaller. As expected, protein datasets take considerably less time to cluster (5.6 s for hIAPP, compared to 144.7 s for the corresponding random dataset, see Table 2). In Figure 5, we look at the behavior of the *f*-value as we increase the number of clusters. For MBP, a protein system with a high dimensionality (370 residues), the *f*-value converges relatively slowly to about 0.32. However, for all the other MD datasets, the *f*-value decreases very quickly to less than 0.05. We also note that hIAPP has a lower *f*-value than Trpzip2 despite hIAPP's higher dimensionality (37 residues *vs*. 12 residues). We hypothesize that the behavior is due to the differences in underlying density distribution of these two datasets as Trpzip2 is a folded peptide with well-defined native structure with a large global minimum and hIAPP is an intrinsically disordered protein with a variegated energy landscape comprised of many distinct local minima.

Table 2. Performance comparison of the GPU *k*-centers clustering algorithm between the protein datasets and the uniformly random datasets (with the same number of atoms and conformations compared to the corresponding protein datasets) on running time (s) and the fraction of conformations that fail the triangle inequality.

Number of atoms included in RMSD calculation	Running time (s)		Fraction of conformations that fail the triangle inequality	
	Protein dataset	Random dataset	Protein dataset	Random dataset
10 (Dipep)	1.2	5.7	0.035	0.916
12 (Trpzip2)	40.8	444.1	0.026	0.991
37 (hIAPP)	5.6	144.7	0.019	0.999
370 (MBP)	30.0	118.7	0.390	0.993

We use $k = 4000$.

Cluster population as an indicator of density

In this section, we highlight the robustness of the *k*-centers algorithm by comparing it against the popular *k*-medoids algorithm in the ability to recover the ground-truth density of Alanine Dipeptide. As discussed earlier,[13,16,25,54] the *k*-centers clustering algorithm generates clusters of approximately

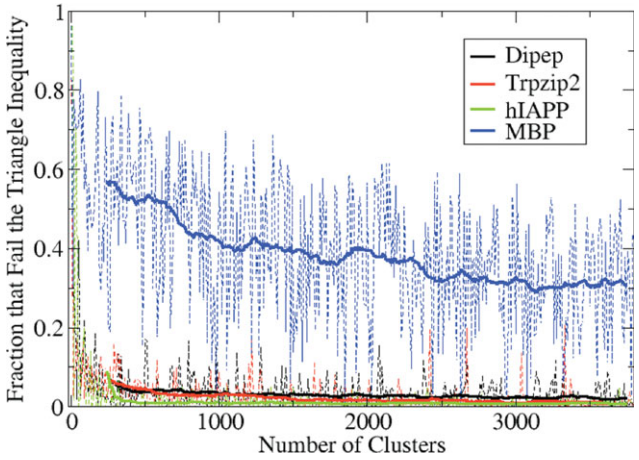

Figure 5. Fraction of conformations that fail the triangle inequality tests for different MD datasets. The dashed line denotes the fraction of points that fail at a certain number of clusters (K), with K spaced every 10 clusters up to 4000 clusters. The bold line denotes the running average (over 50 points) of the dashed line. For MD simulation dataset of protein conformations of relatively low dimensionality such as Dipep (10 heavy atoms, black), Trpzip2 (12 Cα atoms, red), and hIAPP (37 Cα atoms, green), the fraction of points that fail the triangle inequality rapidly decreases with increasing number of K. For high dimension simulation dataset MBP (370 Cα atoms, blue), the number of points that fail the triangle inequality decreases slower and oscillates rapidly. We also provide results for corresponding randomly distributed datasets in Supporting Information Table 1, whereby the average fraction of points that fail the triangle inequality is always greater than 90%. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

uniform distance to each cluster center, resulting in a strong correlation between the population of a cluster and its density. Alanine Dipeptide is an ideal system for verifying this correlation, because the logarithm of the density of conformations described by a pair of torsion angles (ϕ, ψ) is proportional to the potential of mean force (PMF) $W = -kT \ln(P_i/P_0)$,[55] where, P_i is the density estimator for a particular bin i. P_0 is a constant consistent across all three figures, here, we use $P_0 = 5 \times 10^{-6}$. We discretize the torsion plane by dividing it into square bins of 5° by 5°.

To generate the PMF plot for the reference density, defined as the underlying population density of the reference simulation datasets (Fig. 6a), P_i is the number of conformations that lie in bin i, divided by the total of number conformations. We can gauge the robustness of the two clustering algorithms by comparing their recovered PMF maps against the reference density. For the PMF maps resulting from the two clustering algorithms (Figs. 6b and 6c), the calculation of P_i is as follows: let N_i denote the set of clusters that belong to a particular bin i. A cluster belongs to a bin i if it owns at least one conformation positioned in bin i. Let n_j denote the number of conformations that belong to a given cluster j. Let B be the set of all bins. Then, $v_i = 1/|N_i| \sum_{j \in N_i} n_j$; and $P_i = v_i / \sum_{i \in B} v_i$. As shown in Figure 6b, the k-centers algorithm is able to sufficiently recover all six minima and their positions as found in the reference density. Moreover, the minima at around (−140°, 160°) and (−60°, 150°) are the deepest out of all six minima, a result consistent with the reference density. Furthermore, the k-medoids algorithm recovers a PMF map of roughly equal depth and essentially fails to identify any minima (Fig. 6c)—deviating significantly from the reference density (Fig. 6a).

Thus, we conclude that the k-centers algorithm is able to gauge estimates of density much better than the k-medoids algorithm. However, as discussed earlier in the article and in other works,[25] we note that the accuracy of the density estimation depends on the intrinsic dimensionality of the system. For systems with a low degree of freedom, it is empirically helpful to use k-centers as preliminary estimator of density.

Conclusion

We develop a GPU powered k-centers clustering algorithm that utilizes the triangle inequality with an improved running time of $O(k^2 + kf_{avg}N)$. We achieve this by optimizing memory access patterns, parallelizing the $RMSD_{opt}$ operations, and implementing the triangle inequality via compaction. With this new algorithm, we are able to speed up clustering of MD conformations by up to two orders of magnitude. For protein datasets with up to millions of conformations, the conformations can be partitioned into thousands of clusters in under a minute. Furthermore, we show that the algorithm behaves well even when we increase the number of clusters. In addition, using artificial 2D datasets, we show that the triangle inequality performs especially well for datasets comprised of dense and sparse regions. For biological datasets of low dimensionality, we show that the triangle inequality is highly effective as f_{avg} tends to be very small. We also provide a

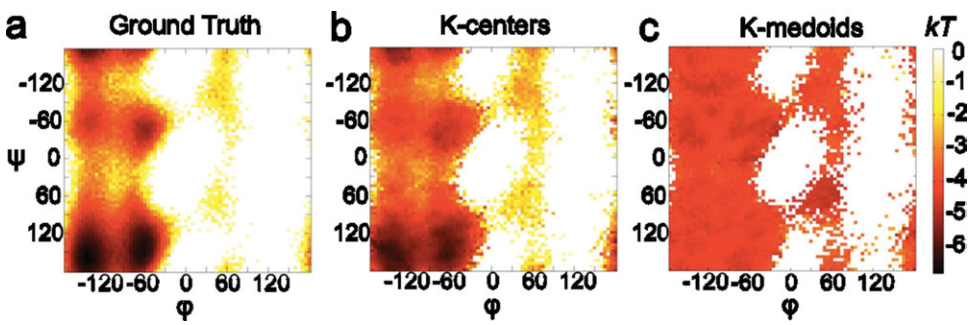

Figure 6. The projection of the PMF (kT) of Alanine Dipeptide (Dipep) onto the torsion angle plane spanned by ϕ (°) and φ (°) (see Fig. 2a for their definitions), (a) directly from the equilibrium MD simulations, (b) from the populations of the clusters obtained by the k-centers algorithm with $K = 4000$, (c) from the populations of the clusters obtained by the k-medoids algorithm with $K = 4000$. The ϕ–φ plane is divided into square bins of 5° by 5°. The free energy is calculated by $G = -\ln P_i/P_0$ for a particular bin i, where P_i is a measure of the density of the ith bin, and $P_0 = 5 \times 10^{-6}$. We refer the reader to the main text for the exact form of P_i. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

mathematical explanation showing why the triangle inequality tends to be not as effective for very high dimensions. Overall, the fast speed of the clustering algorithm enables us to address one of the key challenges in the construction of MSMs, which requires fast geometric clustering.[12,13,15,16]

We develop a triangle inequality which can be applied to other metric space clustering algorithms, such as *k*-medoids and *k*-means. By being able to parallelize the triangle inequality on the GPU, the methods we develop holds promise for applications in various clustering algorithms of general metric spaces. However, we do note that the *k*-centers algorithm has a propensity to be very sensitive to outliers, that is, the algorithm prefers choosing outliers in phase space as cluster centers over more dense points.[12,25,54] This problem can be addressed by several different approaches: a hybrid *k*-centers and *k*-medoids algorithm has been developed in MSMBuilder 2[16]; subsampling the datasets to disregard many of the outliers[12,56]; density-based hierarchical clustering.[25] We speculate that the *k*-centers clustering algorithm may also be used to generate landmark points for dimensionality reduction algorithms such as Isomap.[57] The code is publicly available at http://www.proteneer.com/.

Acknowledgments

The authors thank Douglas Theobald from Brandeis University for discussions regarding the implementation details of the RMSD algorithm; Sean Baxter from the Modern GPU for discussions on GPU reduction and scans. They also thank Qin Qiao for providing the hIAPP MD simulation dataset, and the Cloud-computing Center for Multidisciplinary Research at HKUST and the Key Laboratory of Advanced Optical Communication Systems and Networks in Shanghai for providing the GPU computing resources.

Keywords: molecular dynamics · clustering · triangle inequality · general-purpose computation on GPU · Markov state models

How to cite this article: Y. Zhao, F. K. Sheong, J. Sun, P. Sander, X. Huang, *J. Comput. Chem.* **2013,** *34,* 95–104. DOI: 10.1002/jcc.23110

Additional Supporting Information may be found in the online version of this article.

[1] G. R. Bowman, V. A. Voelz, V. S. Pande, *Curr. Opin. Struct. Biol.* **2011**, *21,* 4.
[2] X. Huang, G. R. Bowman, S. Bacallado, V. S. Pande, *Proc. Natl. Acad. Sci.* **2009**, *106,* 19765.
[3] R. Zhou, M. Eleftheriou, A. K. Royyuru and B. J. Berne, *Proc. Natl. Acad. Sci. USA* **2007**, *104,* 5824.
[4] D. E. Shaw, P. Maragakis, K. Lindorff-Larsen, S. Piana, R. O. Dror, M. P. Eastwood, J. A. Bank, J. M. Jumper, J. K. Salmon, Y. Shan, W. Wriggers, *Science* **2010**, *330,* 341.
[5] V. A. Voelz, G. R. Bowman, K. Beauchamp, V. S. Pande, *J. Am. Chem. Soc.* **2010**, *132,* 1526.
[6] P. L. Freddolino, F. Liu, M. Gruebele, K. Schulten, *Biophys. J.* **2008**, *94,* L75.
[7] F. Morcos, S. Chatterjee, C. L. McClendon, P. R. Brenner, R. Lopez-Rendon, J. Zintsmaster, M. Ercsey-Ravasz, C. R. Sweet, M. P. Jacobson, J. W. Peng and J. A. Izaguirre, *PLoS Comput. Biol.* **2010**, *6,* e1001015.
[8] B. Keller, X. Daura, W. F. Van Gunsteren, *J. Chem. Phys.* **2010**, *132,* 074110.
[9] J. Shao, S. W. Tanner, N. Thompson, T. E. Cheatham, *J. Chem. Theory Comput.* **2007**, *3,* 2312.
[10] M. E. Karpen, D. J. Tobias, C. L. Brooks, 3rd, *Biochemistry* **1993**, *32,* 412.
[11] X. Daura, W. F. van Gunsteren, A. E. Mark, *Proteins* **1999**, *34,* 269.
[12] G. R. Bowman, K. A. Beauchamp, G. Boxer, V. S. Pande, *J. Chem. Phys.* **2009**, *131,* 124101.
[13] G. R. Bowman, X. Huang, V. S. Pande, *Methods* **2009**, *49,* 197.
[14] D. A. Silva, G. R. Bowman, A. Sosa-Peinado, X. Huang, *PLoS Comput. Biol.* **2011**, *7,* e1002054.
[15] J.-H. Prinz, H. Wu, M. Sarich, B. Keller, M. Senne, M. Held, J. D. Chodera, C. Schütte, F. Noé, *J. Chem. Phys.* **2011**, *134,* 174105.
[16] K. A. Beauchamp, G. R. Bowman, T. J. Lane, L. Maibaum, I. S. Haque, V. S. Pande, *J Chem. Theory Comput.* **2011**, *7,* 3412.
[17] I. Buch, T. Giorgino, G. De Fabritiis, *Proc. Natl. Acad. Sci. USA* **2011**, *108,* 10184.
[18] J. A. Hartigan, Clustering Algorithms; Wiley, New York, **1975**.
[19] J. A. Hartigan, M. A. Wong, *J. R. Stat. Soc.* **1979**, *28,* 100.
[20] D. Hochbaum, D. Shmoys, *Math. Oper. Res.* **1985**, 10, 180.
[21] M. E. Karpen, D. J. Tobias, C. L. Brooks, *Biochemistry* **1993**, *32,* 412.
[22] A. Y. Sim, M. Levitt, *Proc. Natl. Acad. Sci. USA* **2011**, *108,* 3590.
[23] J. D. Chodera, N. Singhal, V. S. Pande, K. A. Dill, W. C. Swope, *J. Chem. Phys.* **2007**, *126,* 155101.
[24] P. Rousseeuw, *J. Comput. Appl. Math.* **1987**, *20,* 53.
[25] X. Huang, Y. Yao, G. R. Bowman, J. Sun, L. J. Guibas, G. Carlsson, V. S. Pande, *Pac. Symp. Biocomput* **2010**, *15*, 228.
[26] L. Kaufman, P. J. Rousseeuw, In Finding Groups in Data: An Introduction to Cluster Analysis, Wiley, New York, **2008**.
[27] W. F. van Gunsteren, D. Bakowies, R. Baron, I. Chandrasekhar, M. Christen, X. Daura, P. Gee, D. P. Geerke, A. Glättli, P. H. Hünenberger, *Angew. Chem. Int. Ed. Engl.* **2006**, *45,* 4064.
[28] S. Sengupta, M. Harris, Y. Zhang, J. D. Owens, In Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware, Eurographics Association, San Diego, California, **2007**; pp. 97–106.
[29] E. Lindholm, J. Nickolls, S. Oberman, J. Montrym, *Micro IEEE* **2008**, *28,* 39.
[30] K. J. Kohlhoff, M. H. Sosnick, W. T. Hsu, V. S. Pande, R. B. Altman, *Bioinformatics* **2011**, *27,* 2322.
[31] R. Wu, B. Zhang, M. Hsu, In Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop, ACM, Ischia, Italy, **2009**; pp. 1–6.
[32] D. L. Theobald, *Acta Crystallogr. A* **2005**, *61,* 478.
[33] L. H. Hung, M. Guerquin, R. Samudrala, *BMC Res. Notes* **2011**, *4,* 97.
[34] B. Steipe, *Acta Crystallogr. A* **2002**, *58,* 506.
[35] M. Harris, S. Sengupta, J. D. Owens, Parallel prefix sum (scan) with CUDA, NVIDIA Developer Technology, *GPU Gems.* **2007**, *3*, 851–876.
[36] S. Sengupta, M. Harris, Z. Yao, J.D. Owens, *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware, Eurographics Association Aire-la-Ville, Switzerland, Switzerland,* **2007**, 97–106. http://dl.acm.org/citation.cfm?id=1280110.
[37] W. Zhuang, R. Z. Cui, D. A. Silva, X. Huang, *J. Phys. Chem. B* **2011**, *115,* 5415.
[38] M. Shirts, V. S. Pande, *Science* **2000**, *290,* 1903.
[39] Y. Duan, C. Wu, S. Chowdhury, M. C. Lee, G. Xiong, W. Zhang, R. Yang, P. Cieplak, R. Luo, T. Lee, J. Caldwell, J. Wang, P. Kollman, *J. Comput. Chem.* **2003**, *24,* 1999.
[40] W. L. Jorgensen, J. Chandrasekhar, J. D. Madura, R. W. Impey, M. L. Klein, *J. Chem. Phys.* **1983**, *79,* 926.
[41] D. Van Der Spoel, E. Lindahl, B. Hess, G. Groenhof, A. E. Mark, H. J. Berendsen, *J. Comput. Chem.* **2005**, *26,* 1701.
[42] S. M. Patil, S. Xu, S. R. Sheftic, A. T. Alexandrescu, *J. Biol. Chem.* **2009**, *284,* 11982.
[43] V. Hornak, R. Abel, A. Okur, B. Strockbine, A. Roitberg, C. Simmerling, *Proteins* **2006**, *65,* 712.
[44] A. Onufriev, D. Bashford, D. A. Case, *Proteins* **2004**, *55,* 383.
[45] H. C. Andersen, *J. Chem. Phys.* **1980**, *72,* 2384.

[46] B. Hess, C. Kutzner, D. van der Spoel, E. Lindahl, *J. Chem. Theory Comput.* **2008**, *4,* 435.
[47] M. S. Friedrichs, P. Eastman, V. Vaidyanathan, M. Houston, S. Legrand, A. L. Beberg, D. L. Ensign, C. M. Bruns, V. S. Pande, *J. Comput. Chem.* **2009**, *30,* 864.
[48] A. J. Sharff, L. E. Rodseth, J. C. Spurlino, F. A. Quiocho, *Biochemistry* **1992**, *31,* 10657.
[49] G. Bussi, D. Donadio, M. Parrinello, *J. Chem. Phys.* **2007**, *126*, 014101.
[50] H. Berendsen, J. Postma, W. Van Gunsteren, A. DiNola, J. Haak, *J. Chem. Phys.* **1984**, *81,* 3684.
[51] U. Essmann, L. Perera, M. Berkowitz, T. Darden, H. Lee, L. Pedersen, *J. Chem. Phys.* **1995**, *103,* 8577.
[52] B. Hess, H. Bekker, H. Berendsen, J. Fraaije, *J. Comput. Chem.* **1997**, *18,* 1463.
[53] F. Noe, S. Fischer, *Curr. Opin. Struct. Biol.* **2008**, *18,* 154.
[54] X. Huang, G. R. Bowman, S. Bacallado, V. S. Pande, *Proc. Natl. Acad. Sci. USA* **2009**, *106,* 19765.
[55] J. D. Chodera, W. C. Swope, J. W. Pitera, K. A. Dill, *Multiscale Model. Simul.* **2006**, *5,* 1214.
[56] F. Noe, C. Schutte, E. Vanden-Eijnden, L. Reich, T. R. Weikl, *Proc. Natl. Acad. Sci. USA* **2009**, *106,* 19011.
[57] P. Das, M. Moll, H. Stamati, L. E. Kavraki, C. Clementi, *Proc. Natl. Acad. Sci. USA* **2006**, *103,* 9885.

Received: 1 July 2012
Revised: 14 August 2012
Accepted: 19 August 2012
Published online on 20 September 2012