Structural Bioinformatics

**fast_protein_cluster: parallel and optimized clustering of large scale protein modeling data**

Ling-Hong Hung* and Ram Samudrala*

*Department of Microbiology, University of Washington Seattle WA

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXXXX

**ABSTRACT**

**Motivation:** fast_protein_cluster is a fast, parallel, and memory efficient package used to cluster 60,000 sets of protein models (with up to 550,000 models per set) generated by the Nutritious Rice for the World project.

**Results:** fast_protein_cluster is an optimized and extensible toolkit that supports RMSD and TM-Score as metrics. RMSD calculations using a laptop CPU are 60x faster than qcprot and 3x faster than current GPU implementations. New GPU code further increases the speed of RMSD and TM-Score calculations. fast_protein_cluster provides novel k-means and hierarchical clustering methods that are up to 250x and 2000x faster respectively than Clusco, and identify significantly more accurate models than Spicker and Clusco.

**Implementation and Availability:** fast_protein_cluster is written in C++ using OpenMP for multi-threading support. Custom Streaming SIMD Extensions (SSE), and Advanced Vector eXtensions (AVX) intrinsics code accelerate CPU calculations and OpenCL kernels support AMD and NVIDIA Graphics Processing Units (GPUs). fast_protein_cluster is available under the M.I.T. license. (http://software.compbio.washington.edu/fast_protein_cluster)

1 INTRODUCTION

Many protein structure prediction and protein folding simulations generate a large ensemble of candidate structures using different starting conditions. By analyzing an ensemble of predicted structures, the overall consistency and accuracy of the final prediction can be increased. More accurate models are more structurally similar to the correct structure and will thus tend to be similar to each other and can be identified by clustering. Finding models at the center of clusters is thus a very effective method of identifying the best structures in an ensemble. The Nutritious Rice for the World project (Hung, et al.), an IBM World Community Grid project, generated de novo models of all modelable protein sequences in the rice proteome. More than 60,000 sets of protein models were generated with up to 550,000 models in a set. The size and number of sets exceeded the capability of existing clustering software.

fast_protein_cluster was written to analyze this large dataset using a Linux cluster consisting of 1200 CPU cores and 5 GPUs. The new software is able to cluster a set of 450,000 protein models in 1.5 hours on a single workstation node and clustered all 60,000 sets in 6 weeks. The fast implementation also makes it possible to employ new clustering strategies and we describe two methods that identify significantly higher quality models than the widely used Spicker (Zhang and Skolnick, 2004) and the recently published Clusco (Jamroz and Kolinski, 2013) packages.

2 IMPLEMENTATION

Clustering involves partitioning models into sets of similar structures. fast_protein_cluster implements k-means, and hierarchical clustering methods using RMSD or TM-Score as similarity metrics. Both the calculation of structural similarity and the partitioning methods have been accelerated. Previously, we had described a new faster algorithm for RMSD and TM-Score calculations (Hung and Samudrala, 2012). We now provide new multithreaded and SIMD assembler language implementations for RMSD and TM-Score calculations for CPUs and a new faster GPU implementation for RMSD.

Hierarchical clustering is implemented using a new parallelized O(N²) algorithm (Müllner, 2013; Murtagh, et al., 2011) instead of the O(N³) hclust algorithm used in Clusco. k-means partitioning is implemented using a novel and much faster variant of the standard methodology. The faster and parallel approach is exploited in a multi-k-means strategy where multiple k-means partitioning solutions are generated from different random starting clusters and the best solution chosen using the criterion of maximum homogeneity. Finally, the memory usage is decreased using an optional compact 1 byte representation of the similarity matrix which does not result in loss of clustering accuracy (see supplemental materials). The main routines are written in optimised C++ using assembler intrinsics for SIMD code and OpenCL for GPU kernels. Much of the speed of the software, especially those of the partitioning methods is due to algorithmic improvements that are independent of the hardware (see supplemental materials). The code is portable and provides acceleration on hardware ranging from 10 year old P4 CPUs to modern workstations and GPUs.

To assess the accuracy of the new multi-k-means and complete linkage hierarchical approaches, the models at the center of clusters generated by fast_protein_cluster were compared to the centroids from clusters generated by Spicker and Clusco. The test set consisting of 56 ensembles of size 11,500 to 32,000 models generated de novo by I-TASSER and used originally to test Spicker. Details of the algorithms, implementation, testing and additional benchmarks are provided in supplemental materials.

*To whom correspondence should be addressed.
3 RESULTS AND DISCUSSION

Existing clustering approaches have approximately the same performance on the Spicker set (Jamroz and Kolinski, 2013). However, Table 1 shows that the models at the centers of the clusters identified by fast_protein_cluster are superior to those identified by Spicker and Clusco. The improvements are statistically significant (p<0.05) when comparing the best centroid model from the largest 5 clusters. To give some context to these differences, we note that the standard deviation of the average TM-Score of model 1 from the top 20 groups at the recent Critical Assessment of Protein Structure Prediction (CASP) 10 was in the same range (0.016) (Zhang, 2013).

The variability of de novo modeling can result in sub-populations of models which share different sets of locally correct structural features. These local similarities are detectable through their contribution to the global similarity metric. Complete-linkage hierarchical clustering uses the maximum distance between members in two clusters to determine which clusters to join. This is more conducive to the formation of more divergent final clusters of models that share common inconsistently predicted local features. Similarly, we attribute the improvement using multi-k-means method to its increased clustering accuracy resulting in better detection of the subtle effects of shared local similarity on the global metric.

Table 1. Mean TM-Score of centroids relative to native structure

<table>
<thead>
<tr>
<th>Clustering method</th>
<th>Centroid of largest cluster</th>
<th>Best centroid of 5 largest clusters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spicker</td>
<td>0.584</td>
<td>0.607</td>
</tr>
<tr>
<td>Clusco/k-means/RMSD</td>
<td>0.585</td>
<td>0.612</td>
</tr>
<tr>
<td>Multi-k-means/RMSD</td>
<td>0.590</td>
<td>0.624</td>
</tr>
<tr>
<td>Multi-k-means/TM-Score</td>
<td>0.592</td>
<td>0.624</td>
</tr>
<tr>
<td>Hierarchical/RMSD</td>
<td>0.588</td>
<td>0.626</td>
</tr>
<tr>
<td>Hierarchical/TM-Score</td>
<td>0.595</td>
<td>0.624</td>
</tr>
</tbody>
</table>

1 fast_protein_cluster k-means values are the average of 5 separate runs to control for different starting seeds. Distance matrices were calculated using CA atom coordinates.

2 TM-Score means that are significantly better (paired t-test with p<0.05) than Spicker are in bold, and those significantly better than Clusco are underlined. The quality of the best model among the centroids of the 5 largest clusters is significantly improved when fast_protein_cluster is used as the clustering method.

In figure 1, we demonstrate very significant improvements in performance on the Spicker test set. For TM-Score calculations, the GPU approach has been previously described and we have extended support to NVIDIA GPUs. The multithreaded SIMD CPU implementation is new and is the fastest CPU version of TM-Score providing an increase of 80% over scalar code. For RMSD calculations, the new GPU implementation is several times faster than Clusco. The SIMD acceleration is especially effective for RMSD calculations, resulting in a 3-4 fold speedup over scalar code. On a single core, the CPU code achieves a 15-fold increase over qcpot (Theobald, 2005) and can match GPU speeds when using multiple threads. The increases in partitioning speed are even greater — up to 250 fold for k-means and 2000 fold for hierarchical clustering.

The speed and modular nature of fast_protein_cluster allow for exploration of new metrics and partitioning approaches on very large sets of proteins. Its development has allowed us to cluster the data generated by the Nutritious Rice for the World project. Furthermore, clustering large sets is a common problem in bioinformatics. The algorithms and code are very portable — user defined similarity matrices are supported and new partitioning and input methods can be easily added to existing classes to extend the software’s functionality for applications beyond protein simulations. fast_protein_cluster is available under the permissive M.I.T. license. The source code, Makefile for Linux compilation, test set and documentation can be downloaded (http://software.compbio.washington.edu/fast_protein_cluster).

4 ACKNOWLEDGEMENTS

We thank Michael Guerquin, Michael Shannon and Hayehoi Taing and the IBM World Community Grid team of Viktors Berstis, Bill Bovermann, Juan Hindo, Tedi Hahn and Keith Uplinger. Most of all we thank the volunteers of the World Community Grid for donating their time and effort to this project.

Funding: This work was funded by an NIH director’s Pioneer Award DP1LM011509 to RS.

5 REFERENCES

S1 ALGORITHMIC DETAILS

S1.1 Structure comparison algorithms

Most clustering methods involve calculation of a matrix which enumerate the similarity or distance between every pair of models. The cost in memory and computation of the similarity matrix grows with the square of the number of proteins (O(N^2)). The calculation of the matrix has been the rate limiting step for protein clustering. For protein structure comparisons, the most widely used similarity measure is found by calculating a transformation that superposes corresponding atoms from the one structure onto the second structure and minimizes the root-mean-square-deviation (RMSD) between the coordinates of the superposed structures. This is done by Single Value Decomposition (SVD) of the covariance matrix formed from the two sets of coordinates of the structures being compared (Kabsch, 1976). However, RMSD is a globally optimal transformation that minimizes the distances between all superposed atom pairs and can be dominated by a small set of atoms in divergent loop regions. Several advanced metrics have been developed, such as GDT (Zemla, 2003), MaxSub (Siew, et al., 2000), and TM-Score (Zhang and Skolnick, 2004) that address these shortcomings at the cost of execution speed. For example, TM-Score is 3 orders of magnitude slower than RMSD. Recently Clusco has provided support for these and other measures in similarity matrices but there are no detailed published benchmarks about the performance of these advanced metrics for clustering.

One approach that has been used to address the problem of speed and memory is to avoid calculating a full distance matrix. I-TASSER, (Zhang, 2007) the best performing software package at recent Critical Assessment of Structure Prediction (CASP) competitions, uses Spicker for clustering. Spicker samples the full similarity matrix and discards outliers in its clustering method. Other packages (Li and Ng, 2010) use a full matrix and either accept the computational time required, or more recently, use parallel and multi-core implementations of RMSD and TM-Score (Farrar, 2007; Hung, et al., 2011). However, optimized Kabsch implementations can be significantly faster than quaternion approaches (Hun and Samudrala, 2012) (Jamroz and Kolinski, 2013). Many un-optimized Kabsch methods use library QC decomposition routines for SVD calculations, which is a very fast method for arbitrary sized matrices. However, for the 3x3 symmetric matrices in the Kabsch algorithm, eigenvalues can be more quickly obtained by directly solving a cubic characteristic equation. In contrast, quaternion methods require the solution of a quartic characteristic equation which cannot be reliably calculated without using iterative methods. This is an especially important difference when the models are divergent, and the eigenvalue calculations of the covariance derived matrices can converge more slowly for these iterative methods. Clusco has recently implemented a fast Kabsch based algorithm for NVIDIA GPUs. However, as noted previously (Hung and Samudrala, 2012), the major bottleneck in calculating RMSD is fetching the coordinates from memory. fast_protein_cluster pre-loads coordinates into the limited fast memory of the GPU before proceeding with the calculations and also orders the calculations to minimize the number of these slow data transfers. The resulting implementation is more than twice as fast as the Clusco software on the same GPU. fast_protein_cluster uses OpenCL and runs on both NVIDIA and AMD GPUs. It also uses multiple GPUs when available for further acceleration (see figure 1).

A further difficulty with the Kabsch approach arises when the optimal rotation matrix must be calculated as is the case for TM-Score calculations. To ensure that the transformation is a true rotation and not a reflection requires that the code branch to take care of the different cases. Single instruction multiple data (SIMD) and Single Instruction Multiple Thread (SIMT) systems used by GPUs do not actually decide which branch of code to follow but instead execute every possible branch and discard unwanted results. Thus, efficient parallelization of the standard Kabsch algorithm for TM-Score is difficult. We have previously described and implemented the TM-Score algorithm on GPUs using a faster hybrid Kabsch/quaternion algorithm with a more linear execution path. It also avoids the iteration found in the quaternion methods (Hung and Samudrala, 2012). fast_protein_cluster uses this algorithm for TM-Score and extends it to function on NVIDIA GPUs and also to use more than one GPU simultaneously.

S1.2 Acceleration of protein structure comparison using CPU SIMD units

Most modern CPUs also have an on-chip Single Instruction Multiple Data (SIMD) unit specialized for vector operations. A single SIMD vector operation replaces separate operations on each of the individual components. There are several instruction sets that are used to program the SIMD unit. fast_protein_cluster supports most common instruction sets Streaming SIMD Extensions (SSE), and Advanced Vector eXtensions (AVX). SSE instruction sets are supported in Intel and AMD chips of the past 10 years, while AVX is supported in newer Intel and AMD chips. Similar instructions sets exist for SIMD units on other chips. SSE operations involve vectors of 4 components, effectively performing 4 floating point operations in parallel. AVX operations can work with 8-component vectors and potentially offer even greater acceleration. The SIMD unit was originally put in place to help with graphics processing which involve many of these vector operations. Optimizing compilers such as GCC will attempt to automatically vectorize code and map operations to SIMD units when possible to increase the execution speed. However, with the exception of SIMD accelerated dynamic programming (Farrar, 2007), there has been little direct usage of SIMD units in bioinformatics applications. This can be partially attributed to the complexity of adapting the algorithms in an optimized manner. Although OpenCL and other vector math libraries allow the native SIMD instructions to be used indirectly in high level languages, more involved algorithms require finer control of scarce memory registers to achieve a significant increase in speed. Initial attempts to use these high level libraries for fast_protein_cluster actually resulted in a decrease in speed. Straight assembly language is another possibility but is not very portable and can interfere with automatic optimizations that the compiler performs. Modern compilers...
also support assembly language style snippets, or intrinsics. The compiler is aware that these snippets can be used and can allow for them in their optimizations schemes in present and future versions. Intrinsics have now become sufficiently robust to offer a portable alternative to assembly. Using intrinsics, significantly faster RMSD and TM-Score algorithms have been implemented in fast_protein_cluster. Moreover, fast_protein_cluster is multi-threaded and the SIMD acceleration scales with additional threads allowing commodity laptop 4-core CPU to perform RMSD calculations at GPU speeds (see figure 1). The SIMD code is very portable - it has been tested on 10 year old P4 CPUs where it still provides 3-fold acceleration over scalar code. These types of improvements will become increasingly important as the number of CPU cores expands and newer more powerful instruction sets such as AVX2 blur the distinction between on chip SIMD units and GPUs. GPUs, however, remain important for more demanding calculations such as TM-Score and for volunteer grid projects where they provide most of the computational power

S1.3 Partitioning methods applied to protein clustering

The calculation of similarity matrices is only half of the clustering process. Structures must also be partitioned into groups of similar structures based on the similarity measures. For the very large sets that can be clustered by fast_protein_cluster, partitioning speed now becomes the major rate limiting factor. fast_protein_cluster implements density, k-means, and hierarchical clustering methods which are described in the next subsections.

S1.3.1 Density

Density is the simplest method which treats the entire ensemble as a single cluster. Models in "dense" structural space will have many nearby models. We define density as the average distance to all other models. Density is defined in this manner for historical reasons so that density behaves like an energy potential with lower values being associated with better models. Density is easily calculated and does not require storage of the similarity matrix. This can be a very effective method for finding a single best model. It can also be a method for rapidly reducing the size of the set or removing outliers before a more complicated clustering method is applied. However, protein structure predictions, especially de novo predictions, are inconsistent and more than one final model is often desired to increase the probability of identifying one with a correct fold. CASP recognizes this and allows for the submission of 5 different models for each blind prediction sequence. More complicated partition methods divide the ensemble into multiple clusters of structures sharing similar features and then use the lowest density criterion to identify a representative model from each cluster. In this manner, inconsistently predicted but correct features that are only found in a sub-population of models can still be detected in one of the smaller outlier clusters.

S1.3.2 k-means

One of the simplest and most popular clustering methods is k-means. k models are randomly chosen as starting centers to seed the calculation. The distance from each model to the each of the k centers is determined. Models which are closest to the same center are assigned to the same cluster. New center models (centroids) are then calculated from the new clusters and the process repeated until the cluster membership does not change. The major rate limiting operation is the calculation of the new centroids after the new clusters are assigned. If the number of models is n then the average cluster size will be n/k and require (n/k)^2 comparisons to find the centroid for a single cluster. Thus approximately k(n/k)^2 = n^2/k operations are required to calculate all k centroids in each update. This gives rise to at least O(N^2) complexity. fast_protein_cluster uses a slightly different and much faster algorithm for k-means. The same procedure is used to obtain an initial assignment into k clusters from k random centers. Rather than re-assigning a model to the closest center, we assign it to the cluster where the average distance to all the other members of the cluster is the lowest, i.e. where it is most similar to the other members of the cluster. Let us define the cluster density D(i,m) as the average distance from model m to all the members of cluster i. All cluster densities can be maintained in a k x n matrix and each step model m is re-assigned to cluster i where D(i,m) is the lowest. For each model that changes its cluster membership, one needs to update the cluster densities for the original cluster and for the new cluster or a total of 2(n-1) operations per change or 2c(n-1) for c changes. Typically cluster membership converges quickly and after the first few iterative cycles, the number of changes c is small relative to the total number of models n and the variant is much faster than the usual k-means methodology. However, in the first couple of iterations, c can be rather large and it is more efficient to calculate the cluster densities de novo rather than updating them. The software does a benchmark simulation at the beginning to estimate the value of c where it becomes optimal to calculate cluster densities de novo rather than performing using a update procedure. The simulation follows a single trajectory and records the time taken to perform the update. Linear regression then is used to estimate the time required as a function of the number of changes.

On the Spicker dataset, fast_protein_cluster is up to 250x faster than Clusco which uses the standard k-means algorithm. We have exploited the increased speed of the calculations to improve the accuracy of the clustering. The final clusters obtained from a k-means iteration will depend somewhat on the starting clusters. For protein models, these differences can be significant, especially when clustering more variable models generated by de novo methods. fast_protein_cluster exploits the fast k-means implementation to calculate multiple clustering solutions using different starting seeds. The k-means solution that minimizes the total distance from the models to the center of their cluster is then chosen (i.e. a maximum homogeneity criterion is used for selection). This is easily and efficiently parallelized by having each thread calculate a separate k-means solution starting from a different set of centers. As shown in Table 1, this approach gives rise to final centroids that are closer to the experimental structure. One further advantage of the multi-k-means method is that it allows us to strictly enforce the requirement that there be k clusters and that the clusters be of a minimum size. Because we calculate multiple solutions, we can simply discard those that do not meet these criteria.
Hierarchical or agglomerative clustering successively merges the most similar clusters. Merging stops when the desired number of clusters is reached. Hierarchical clustering is much less dependent on starting clusters than k-means and is a deterministic method when all pairwise distances are distinct. The similarity or distance between the clusters can be calculated using a variety of different methods – the most popular being complete linkage (most distant members of two clusters determines distance), single linkage (closest members of two clusters determines distance) and average linkage (mean distance between members of two clusters determines distance). Single-linkage hierarchical clustering, like density based clustering does not require that the entire similarity matrix be kept in memory. This makes it an important technique for clustering large sets (Loewenstein, et al., 2008). However, single-linkage joins clusters based on the distance between the most similar members which means that an outlier cluster of divergent models will be joined to a cluster as long as there is one similar pair of models between the two clusters. For clustering of protein models, we are interested in finding alternative sub-populations of models that share different subsets of correct structural features. Thus we would want outlier clusters to remain separate, which would be better accomplished by complete-linkage clustering. In our initial benchmarking, single-linkage clustering was not significantly different from other methods whereas complete-linkage clustering is superior.

Hierarchical methods have had the unfortunate reputation of being slow, largely due to some popular implementations that are O(N^3) such as R’s default implementation, hclust. Similar to our k-means strategy, we had originally implemented a faster algorithm that maintained a list of the nearest neighbors which was updated after each agglomeration step. These types of strategies (Müllner, 2013; Murtagh, et al., 2011) remains worse case O(N^3) but are in practice O(N^2). Müllner’s approach improves upon the basic method by halving the number of comparisons necessary to update the nearest neighbor list. fast_protein_cluster takes Müllner’s basic algorithm and makes changes to make it more amenable for very large datasets. Our similarity matrices are implemented using our triangular matrix class. Two dimensional arrays are stored and accessed using a vector of pointers to vectors whereas the original implementation uses a single vector and a macro to map the two indices to an offset from the start of the vector. The original approach is better for smaller sets and may be optimal for memory accesses patterns typically found in R, for which it was designed. However, we have found that very large offsets result in much slower data access and our vector of pointers approach was faster for very large sets. It also allows us to use compact representations of the similarity matrix when memory is limiting. The original method also uses a heap to store nearest neighbor distances which has significant overhead and results in memory access patterns that are difficult to predict. To increase cache-friendliness and speed for large sets and allow for compact representations, we use our triangular matrix class instead of a heap. Finally, we were able to partially parallelize the algorithm, with the most performance gain coming from parallelizing the nearest neighbor list updates. Using this implementation, fast_protein_cluster takes 3 seconds to partition 10,000 models and 1 hour to partition 450,000 models on a workstation. In comparison, Clusco using a O(N^2) method takes 27 minutes for the smaller set and extrapolates to several years for the larger set. With fast_protein_cluster, hierarchical clustering becomes a viable and accurate protocol for medium and large sets of proteins models as shown in Table 1.

**S2 IMPLEMENTATION DETAILS**

**S2.1 Distance matrix calculation methodology**

fast_protein_cluster is written in C++. A triangular matrix class handles similarity matrices and masks the details of index manipulation, matrix calculation, matrix input/output and data type translation. fast_protein_cluster currently supports RMSD and TM-Score as metrics of protein similarity. Other metrics are supported by the ability to read text and binary matrices generated by any user algorithm. In addition, new methods can be easily added to the triangular matrix class. The TM-Score algorithm is a hybrid Kabsch-quaternion algorithm which has been previously described (Hung and Samudrala, 2012). The algorithm for RMSD calculations is a Kabsch implementation that solves a cubic characteristic equation as described in the introduction. Single precision is used for the calculation of covariances and determinants to increase speed with double precision only being reserved for the final steps of the eigenvalue calculation to preserve accuracy. The algorithm is further optimized by pre-calculating the center of mass and sum of squares of the coordinates of each model.

**S2.1.1 SIMD acceleration**

fast_protein_cluster has the option to use SIMD accelerated RMSD and TM-Score algorithms. SSE2, SSE3 and AVX instruction sets are supported. GCC intrinsics were used to implement the SIMD routines. The coordinates are rearranged so that x coordinates, y coordinates and z coordinates are grouped as separate x vectors, y vectors and z vectors. SIMD vector operations then replace normal floating point scalar operations as described in section S1. SIMD operations are used to rearrange the coordinates into vectors, calculate the centers of mass, sums of squares and determinants, and apply rotations to coordinates. However, the major increase in speed comes from vectorizing the calculation of the covariances. The final part of the Kabsch algorithm, involving the actual solution of the cubic characteristic equation to determine the eigenvalues requires double precision and trigonometric calculations which are not amenable to SIMD acceleration and are performed with the usual scalar operations. For TM-Score, which involves iterative RMSD calculations and superposition transformations, SIMD operations require more extensive use of single precision variables. This results in an occasional slight loss of accuracy similar to that found for Clusco’s implementation. This accuracy loss is only observed for SIMD TM-Score calculations. There is no loss of accuracy for our GPU or scalar implementations and the results in Table 1 are not affected. The TM-Score methodology involves extensive
rotations to superpose the sets of coordinates. While these operations have been efficiently vectorized and converted to SIMD operations, the speedup is less than observed for covariances. This may be due to the optimizing compiler already partially vectorizing rotation operations. In any case the speedup for TM-Score is still 80%, and for RMSD calculations is 3-4 fold over optimized scalar code.

**S2.1.2 GPU distance matrix calculations**

fast_protein_cluster can use GPUs to calculate RMSD and TM-Score. Support is provided for both NVIDIA and AMD GPUs using OpenCL. Multiple GPU support is provided using separate OpenMP threads to manage data transfers to each GPU and requires a multi-core CPU and OpenMP. Since we have already described the algorithm for TM-Score previously, we will describe the GPU RMSD algorithm. The coordinates of models are read into an array on the CPU side and rearranged as described for SIMD. They are then transferred to the GPU where they reside in the slow but plentiful GPU global memory. On the GPU there are multiple compute units, each with multiple threads that share a common block of fast local memory (typically 16-64 KB per compute unit). There is even scarcer and faster private register memory that is available (up to 1 KB per 4 threads on AMD GPUs). The performance of most GPU processes are “memory-bound”, i.e. limited by the memory bandwidth rather than the speed of the actual computations and a common optimization is to ensure that computation is done as much as possible using data in fast memory.

The strategy is to have each compute unit calculate the RMSD between different pairs of models. Coordinates for the two models are copied from GPU global memory into the shared local memory and the RMSD calculated in parallel by the threads in the compute unit. All intermediate values from calculations are kept in very fast register memory. The number of slow memory transfers from global to local memory is minimized by assigning the calculations so that only one set of coordinates needs to be swapped into fast memory before the calculation of the next pairwise RMSD. GPU based calculations determine which coordinates to fetch from global memory. Pre-centering and rearrangement of coordinates is performed by the CPU as for CPU/SIMD case which reduces the number of arithmetic operations required. The overall result of these optimizations is up to a 2.5x speedup (on the same hardware), over Clusco which uses a very similar algorithm directly ported from C into CUDA.

**S2.1.3 Variables that can affect the speed of similarity matrix calculations**

Much of the main speed gain for the SIMD code comes in calculation of the covariance matrix. When larger numbers of atoms are compared, a larger proportion of the computational time is spent calculating covariances. Hence, for the calculations of RMSD of larger proteins and the calculations RMSD using all atom coordinates instead of just CA atom coordinates, the speed differences are greater (data not shown). Most of the optimizations for the GPU code also manifest themselves in the calculation of the covariance matrix – hence the additional increase in relative performance over Clusco’s GPU implementation on the same hardware when all atom RMSD is evaluated. As discussed in S2.1.1, SIMD acceleration for TM-Score is less effective overall than for RMSD as comparatively less time is spent on the calculation of the covariance matrix because of the extra time spent transforming coordinates.

Both SIMD and GPU performance can also be affected by data bottlenecks. This accounts for the slight non-linear scaling of SIMD performance with the number of threads. With multiple threads, there are likely more pauses as the computation of the SIMD units outpace the data pipelines. However, even with more threads than physical cores (hyperthreading), these wait states can be masked by switching from an idle thread to one that is ready. This accounts for the improvement in SIMD performance at 8 threads in figure 1 which is not observed for the slower scalar code. Data bottlenecks are even a greater factor for our compute nodes that have AMD Bulldozer chips where SIMD unit and data port is shared by 2 CPU cores. There is no hyperthreading to mask wait states and the SIMD code actually becomes slower than scalar for large numbers of threads (data not shown). On the GPU, data bottlenecks manifest themselves in the relatively poor performance of the NVIDIA 580 GPU for TM-Score. This is likely due to the smaller amount of fast register memory available on the test NVIDIA GPU. This is not a problem for RMSD calculations. The RMSD algorithm uses much less register memory than the TM-Score algorithm and the NVIDIA GPU is comparable in speed to the AMD GPU.

**S2.2 Partitioning calculations**

The code that partitions the models into clusters is handled by a separate partition class. The class manages the data structures and methods specific to assigning and maintaining information about the cluster membership of the models. This allows for straightforward addition of new partitioning methodologies. Currently, density, k-means, and hierarchical clustering algorithms are supported by fast_protein_cluster. The basic algorithms and most of the optimizations have already been described in section S1.3. One additional optimization deserves mention. Both the k-means and hierarchical clustering methodologies depend on updating intermediate results based on cluster membership changes. All the changes are recorded and grouped by the clusters affected. The updates are then performed on one cluster at a time. Although this procedure involves more loops, it is far more cache friendly. It also allows for separate threads to update in a thread-safe manner without using expensive (and sometimes unreliable) OpenMP atomic or critical constructs.

**S2.3 Multi-threading**

fast_protein_cluster uses OpenMP to provide multi-threading support. For both TM-Score and RMSD CPU calculations, the parallelization is at the task level, i.e. each thread handles the complete task of calculating one RMSD or TM-Score between a pair of models. Lower level parallelism for matrix calculation is achieved through SIMD operations and GPU operations as previously described. For k-means, task
level parallelization is implemented by assigning each thread a separate k-means trajectory starting from a different set of random centers. fast_protein_cluster supports the option of terminating the calculation based on the total number k-means solutions calculated or based on the number of consecutive solutions that have been calculated without further improvement on the best solution. For hierarchical clustering, there are several separately parallelizable segments, including the matrix initialization and determination of closest clusters. The major part of the code, the cluster distance and nearest neighbor update after a cluster merge, can also be assigned to multiple threads. Unfortunately, there is no pattern to the cluster distances that will be updated and a handful of threads will do most of the work. Thus, even with dynamic allocation of threads, the increase in speed does not scale as well as task level parallelism. However, as shown in figure 1, both parallelization methods give rise to significant increases in speed with multiple threads.

S2.4 Compact matrices
To save memory, fast_protein_cluster has the option of using compact similarity matrices that store the distances as single byte characters. The values of RMSD or TM-Score are linearly mapped into 256 bins. The range of the bins can be selected by the user or be determined by the software. Specialization of the triangular matrix class is used to make this process completely transparent to the other parts of the code and makes it straightforward to extend support to short, integer and double precision types. Both k-means and hierarchical clustering using average-linkage protocols require the calculation of average distances. To reduce the impact of the small loss of accuracy from the compact representation, we maintain these average distances as floating point. For complete linkage and single linkage hierarchical clustering, average distances do not need to be calculated during the agglomeration process. Thus, cluster distances are also maintained as character values to save space when compact distance matrices are selected as an option. The compact representation does not affect the quality of the final centroid models obtained which are are by inspection similar to the full-matrix results and not significantly different in a paired z-test. (see Table S2).

S3 BENCHMARKING DETAILS
S3.1 Test set and hardware configurations
The test set was originally used to test Spicker and consists 56 sets of 11500 to 32000 de novo models generated by I-TASSER. These sets are considerably smaller than the Nutritious Rice for the World datasets which have an average size of 150,000. Therefore, a 450,000 model set from the Nutritious Rice for the World project was also used as a test case. The Spicker set was downloaded from the author's site and filtered to eliminate identical structures and empty coordinate files before analyses. The benchmarks were performed on a laptop computer with 8 GB Memory and a 2.2 Ghz Ivy-bridge I7 CPU with 4 physical cores and 8 logical cores. Tests on the 450,000 model set were run on a 4x16 core Bulldozer workstation with 256 GB memory. GPU benchmarks were performed on an AMD 6990 double GPU and a NVIDIA 580 GPU connected to similar workstations. All software was downloaded and re-compiled from source with optimization flags. C++ wrappers for qcprot and TM-Score were constructed to link to our own read/write routines which allowed use to exclude input/output from the timings. For qcprot, we also used the package's provided coordinate centering routine and removed the default calculation of rotation matrices for a fair comparison.

To evaluate the clustering accuracy, we used the standard Spicker settings and clustered into 10 clusters. This may not be the optimal number of clusters but since it is the default setting, it is not far from optimal for Spicker. The centroid from the largest cluster was deemed to be model 1 and the centroids from the top 5 clusters were also evaluated. This simulates a CASP scenario where 5 models are submitted for evaluation and the quality of model 1 and the best of the 5 models used to assess the submission. The quality of the models were determined by calculating the TM-Score of the structure to the experimentally determined structure. Because our k-means clustering solutions depend on random starting centers, we averaged the statistics from 5 runs when k-means was the partitioning method. The standard errors for the different runs ranged from 0.0008 to 0.0014. This is not necessary for Clusco or Spicker which do not use random seeds. To benchmark RMSD speed, the time was summed for the calculation of distance matrices for all 56 sets of models. For TM-Score, only the first 300 models were evaluated per set for the CPU benchmark due to the slowness of the method. For the GPU implementation, which is much faster, this is too small a sample for consistent timing and 1000 models per set were evaluated and the times normalized to be comparable to the CPU times.

S4 SUPPLEMENTAL RESULTS
S4.1 Real world timings on Nutritious Rice For the World dataset
Table S1 lists the average time required for the different methods to cluster the Spicker sets and the time for methods to cluster a 450,000 model set from the Nutritious Rice for the World project. These are the total times including file I/O times. The number of threads used is indicated in the brackets. For the larger set only, compact distance matrices were used that reduced the memory used by 4-fold without affecting the final accuracy (see Table S1 in supplemental materials). The table shows that hierarchical clustering is significantly faster than other methods even when only a single-thread is used. k-means calculation times are comparable to Spicker even when 500 k-means trajectories are calculated. Neither Spicker nor Clusco were able to give results for the 450,000 model set. For sets of this size, distance matrix
calculation speed becomes less important than the memory footprint and the speed of the partitioning methods. All three of these limiting factors have been addressed by optimized code and fast_protein_cluster is thus able to cluster the larger set within a reasonable amount of time.

Table S1  Comparison of total clustering time.

<table>
<thead>
<tr>
<th>Clustering method</th>
<th>Spicker set $^1$</th>
<th>450,000 model set $^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spicker</td>
<td>138s (1)</td>
<td></td>
</tr>
<tr>
<td>Clusco (k-means/RMSD)</td>
<td>191s (1)</td>
<td></td>
</tr>
<tr>
<td>Clusco (Hierarchical/RMSD)</td>
<td>3426s (8)</td>
<td></td>
</tr>
<tr>
<td>Hierarchical/RMSD</td>
<td>58s (1)</td>
<td>1.6h (64)</td>
</tr>
<tr>
<td>Hierarchical/RMSD</td>
<td>13s (8)</td>
<td>1.5h (64)+2GPU</td>
</tr>
<tr>
<td>k-means/RMSD</td>
<td>156s (8)</td>
<td>18.9h (64)+2GPU</td>
</tr>
</tbody>
</table>

$^1$Average time per set of models on a 4-core laptop (number of threads used are in brackets). k-means timings for fast_protein_cluster are based on the total time to calculate 500 k-means trajectories. AVX acceleration was used for timings.

$^2$Time using compact distance matrix option on a 64 core workstation. Even when including time to read and write data, fast_protein_cluster is up to an order of magnitude faster than Spicker and is able to cluster the very large test set.

Table S2. Comparison of average TM-Score to native for compact distance matrices on the Spicker sets

<table>
<thead>
<tr>
<th>Clustering method</th>
<th>Centroid of largest cluster</th>
<th>Best centroid of 5 largest clusters</th>
</tr>
</thead>
<tbody>
<tr>
<td>k-means/RMSD</td>
<td>0.590 +/- 0.023</td>
<td>0.624 +/- 0.022</td>
</tr>
<tr>
<td>k-means/RMSD/compact</td>
<td>0.590 +/- 0.022</td>
<td>0.626 +/- 0.022</td>
</tr>
<tr>
<td>k-means/TM-Score</td>
<td>0.592 +/- 0.024</td>
<td>0.624 +/- 0.022</td>
</tr>
<tr>
<td>k-means/TM-Score/compact</td>
<td>0.593 +/- 0.024</td>
<td>0.624 +/- 0.022</td>
</tr>
<tr>
<td>Hierarchical/RMSD</td>
<td>0.588 +/- 0.024</td>
<td>0.626 +/- 0.021</td>
</tr>
<tr>
<td>Hierarchical/RMSD/compact</td>
<td>0.586 +/- 0.024</td>
<td>0.626 +/- 0.021</td>
</tr>
<tr>
<td>Hierarchical/ TM-Score</td>
<td>0.595 +/- 0.024</td>
<td>0.624 +/- 0.021</td>
</tr>
<tr>
<td>Hierarchical/ TM-Score/compact</td>
<td>0.597 +/- 0.023</td>
<td>0.618 +/- 0.022</td>
</tr>
</tbody>
</table>

S5 REFERENCES


Murtagh, F., Langfelder, P. and Yau, C.M. (2011) flashClust: fast implementation of hierarchical clustering.


