ENABLING SCALABLE DATA ANALYSIS FOR LARGE COMPUTATIONAL STRUCTURAL BIOLOGY DATASETS ON LARGE DISTRIBUTED MEMORY SYSTEMS SUPPORTED BY THE MAPREDUCE PARADIGM

by

Boyu Zhang

A dissertation submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science

Spring 2015

ENABLING SCALABLE DATA ANALYSIS FOR LARGE COMPUTATIONAL STRUCTURAL BIOLOGY DATASETS ON LARGE DISTRIBUTED MEMORY SYSTEMS SUPPORTED BY THE MAPREDUCE PARADIGM

by

Boyu Zhang

Approved: _____

Errol L. Lloyd, Ph.D.
Chair of the Department of Computer and Information Sciences

Approved: _____

Babatunde A. Ogunnaike, Ph.D.
Dean of the College of Engineering

Approved: _____

James G. Richards, Ph.D.
Vice Provost for Graduate and Professional Education

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Michela Taufer, Ph.D.
Professor in charge of dissertation

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Cathy H. Wu, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Li Liao, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Ming-Ying Leung, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Pietro Cicotti, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Trilce Estrada, Ph.D.
Member of dissertation committee

ACKNOWLEDGEMENTS

This work would have not been possible without the mentorship and support of several intelligent and passionate colleagues. First of all, I thank my research advisor Dr. Michela Taufer. She helped me focus on important research problems, guided me through the challenges and obstacles along this journey, and trained me to become more effective in communications and more independent in research. When I look back, I can see the changes that I went through, and I am thankful that I have her as my advisor and my mentor. I also thank my thesis committee members, Drs. Cathy H. Wu, Li Liao, Ming-Ying Leung, Pietro Cicotti, and Trilce Estrada, who provided valuable feedback, guidance, and directions for my thesis. I gained broader visions and ideas from the questions and discussions that we had.

Drs. Estrada, Leung, and Cicotti have played a special part in my professional growth. Dr. Estrada was a colleague when I first joined the group and now is a successful faculty member. She has been a tremendous role model for me and taught me various skills to enjoy (and survive) my Ph.D. time. Dr. Leung brought the indispensable statistical knowledge into my RNA work. I learned fundamental statistical concepts and methods in this process and was able to apply that knowledge in other problems. Dr. Cicotti provided me with invaluable support and guidance related with the use of the Hadoop framework and the Gordon supercomputer. I gained valuable experience and knowledge working with Gordon.

I thank Dr. Pavan Balaji and his group at the Argonne National Laboratory for providing guidance on using the Fusion supercomputer and performing research at a much larger scale than I was used to during my internship; all the Docking@Home volunteers for participating in the Docking@Home project, generating the ligand datasets, and helping advance science; Dr. Vijay Pande for providing me with the datasets of

Folding@Home; and D. E. Shaw research group for providing me the datasets of protein BPTI generated by the Anton supercomputer.

I feel lucky because I had the opportunity to work together with my lab colleagues at the Global Computing Lab. They always gave me the feedback I needed and created an open environment for discussions.

I acknowledge various funding agencies who provided grant support to the work in this thesis including the National Science Foundation, the Extreme Science and Engineering Discovery Environment (XSEDE) Program, and the U.S. Department of Energy, Office of Science.

I am deeply thankful to my family for always being supportive and believing in me. I especially thank my husband, Zhengyu Cao, who is patient, supportive, and helpful with his big heart and his knowledge in statistical and quantitative analysis. It takes a great deal of support and collaboration to make this thesis happen, and I feel deeply grateful.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Today, petascale distributed memory systems perform large-scale simulations and generate massive amounts of data in a distributed fashion at unprecedented rates. This massive amount of data presents new challenges for the scientists analyzing the data. In order to classify and cluster this data, traditional analysis methods require the comparison of single records with each other in an iterative process and therefore involve moving data across nodes of the system. When both the data and the number of nodes increase, classification and clustering methods can put increasing pressure on the system's storage and bandwidth. Thus, the methods become inefficient and do not scale. New methodologies are needed to analyze data when it is distributed across nodes of large distributed memory systems.

In general, when analyzing such scientific data, we focus on specific properties of the data records. For example, in structural biology datasets, properties include the molecular geometry or the location of a molecule in a docking pocket. Based on this observation, we propose a methodology that enables the scalable analysis for large datasets, composed of millions of individual data records, in a distributed manner on large distributed memory systems. The methodology comprises two general steps. The first step extracts concise properties or features of each data record in isolation and represents them as metadata in parallel. The second step performs the analysis (i.e., classification or clustering) on the extracted properties (i.e., metadata) using machine learning techniques.

We apply the methodology to three different computational structural biology datasets to (1) identify class memberships for large RNA sequences from their secondary structures, (2) identify geometrical features that can be used to predict class memberships for structural biology datasets containing ligand conformations from

protein-ligand docking simulations, and (3) find recurrent folding patterns within and across trajectories (i.e., intra- and inter-trajectory, respectively) in multiple trajectories sampled from folding simulations.

Since our method naturally fits in the MapReduce paradigm, we adapt it for different MapReduce frameworks (i.e., Hadoop and MapReduce-MPI) and use the frameworks on high-end clusters for the three scientific challenges listed above. Our results show that our approach enables scalable classification and clustering analyses for large-scale computational structural biology datasets on large distributed memory systems. In addition, compared with traditional analysis approaches, our method achieves similar or better accuracy.

Chapter 1

THESIS OVERVIEW

1.1 Problem, Motivation, and Proposed Solution

Nowadays, massive amounts of data are generated by petascale distributed memory systems performing large-scale simulations in a distributed fashion at unprecedented rates. When scientists seek to analyze the scientific meaning of the data, however, this massive amount of data presents new challenges. Specifically, traditional analysis methods of classification and clustering require the comparison of single records with each other in an iterative process. In the case of distributed large datasets, this comparison then involves moving data across nodes of the system. When both the datasets and the number of nodes increase, these methods can increase pressure on the system storage and bandwidth. Thus, these methods become inefficient and do not scale. New analysis methods are needed whereby the analysis moves to the distributed data and explicit all-to-all comparisons among data records are avoided.

This thesis focuses on large multi-dimensional structural biology datasets on large distributed memory systems. In general, when analyzing scientific data, scientists focus on specific properties of the data. In the structural biology datasets considered in this thesis, properties include the molecular geometry or the location of a molecule in a docking pocket. Based on this observation, we claim that the data properties can be captured across the dataset concurrently by analyzing each single data record independently and representing the data properties with metadata. The extracted properties (i.e., metadata) can be efficiently analyzed (i.e., classification or clustering) in a distributed fashion by using machine learning techniques.

Accordingly, in this thesis, we propose a transformative data analysis method that comprises two general steps. The first step extracts concise properties or features

of each data record in isolation and represents them as metadata in parallel. The second step performs the analysis (i.e., classification or clustering) on the extracted properties (i.e., metadata). We focus on three types of analysis: the identification of class memberships from a specific feature or property, the identification of features that can be used to predict class memberships, and the finding of recurrent patterns in datasets [31]. For each type of analysis, we consider types of datasets of real scientific records in the two-dimensional, three-dimensional, and four-dimensional spaces of computational structural biology, respectively. Figure 1.1 summarizes the three types.

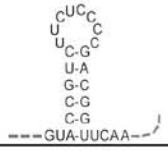
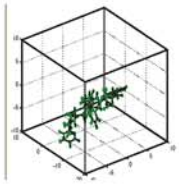
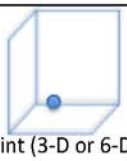
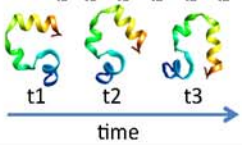
Analysis Type	Dataset	Metadata	Analysis	Pub.
Identify class membership from features • Case study: classification of RNA secondary structures	2-D: ACUGUCAAGUC... 	Strings representing chunk-based secondary structures: ...(((::::)))...	Classification: statistical-based	[83] [87] [88]
Identify features to predict class membership • Case study: clustering of ligand geometries	3-D: $x_1, y_1, z_1, x_2, y_2, z_2, \ldots$ 	 N-D point (3-D or 6-D) representing geometric shape features	Clustering: N-D tree-based	[28] [29] [84] [86]
Identify recurrent patterns • Case study: clustering of protein folding trajectories	4-D: $x_{t1}, y_{t1}, z_{t1}, x_{t2}, y_{t2}, z_{t2}, \ldots$ 	A set of 3-D points representing geometric shape features in time 	Clustering: hierarchical probabilistic-based	[85]

Figure 1.1: Three scientific case studies.

As an example of datasets for identifying class memberships from a specific feature or property, we consider RNA sequences and their secondary structures. In this case, we need to classify secondary structures of a given RNA sequence as more or less likely to happen in nature based on its chunk-based structural features and

the family model of the RNA sequence. Specifically, our method first extracts the relevant hairpins and pseudoknots contained in the sequence's chunks represented as strings metadata in parallel and then rebuilds several possible secondary structures of the whole RNA sequence using the chunks' structures. The method then classifies the secondary structures as more or less likely to occur in nature, using an accuracy retention obtained by comparing them with the predicted structures using the whole sequence and the known structure. In addition, our method offers the advantage of easy integration into our Hadoop-based framework of training a statistical model (i.e., the RNA family model) for the given sequence using known structures of other RNA sequences in the same family and classifying each of the several predicted secondary structures of the given sequence as more or less likely to happen in nature based on the family model.

As an example of identifying features that can be used to predict class memberships, we consider structural biology datasets containing ligand conformations from protein-ligand docking simulations. In this case, we want to cluster the sampled conformations of a given ligand when docked into a protein pocket based on the conformations' geometries. Our method first concurrently extracts the geometry of each ligand conformation as a single N-dimensional (i.e., 3-D or 6-D) point metadata. Then it selects multiple potential near-native conformations (i.e., ligand conformations that well-docked into the protein site) by using the N-dimensional clustering (i.e., octree-based clustering for 3-D metadata or 6-D tree-based clustering for 6-D metadata) technique.

As an example of finding recurrent patterns in datasets, we consider multiple trajectories sampled from folding simulations to identify any folding patterns within and across trajectories (i.e., intra- and inter-trajectory, respectively). Our method first extracts the geometric shape features of each protein conformation in a folding frame when the time evolves (i.e., a consecutive subset of the folding protein conformations) as a sequence of 3-D points metadata. Then the set of 3-D points in the time dimension are interpreted in terms of recurrent patterns by using a hierarchical probabilistic clustering technique.

Our method naturally fits in the MapReduce programming model. The MapReduce programming model was originally proposed and written by Google to index and annotate data on the Internet [19]. The model consists of two functions: map and reduce. The map function divides a large chunk of work into smaller chunks and performs computation on each smaller chunk. Its input comprises a paired argument: a key that is abbreviated as k_1, and a value that is abbreviated as v_1. The function's output consists of a list of intermediate key and value pairs, i.e., $list\,\langle k_2, v_2\rangle$. The output values (i.e., v_2) associated with the same key k_2 are aggregated by the runtime system without user intervention. The values become the input to the reduce function, which takes a paired of key and values as input (i.e., $\langle k_2, list(v_2)\rangle$), and outputs a list of values (i.e., $\langle list(v_3)\rangle$.

Many in-house software tools have been developed for specific problems and datasets, but they cannot be easily adapted to other problems. Thus, the benefits of these in-house software tools to the scientific community are limited. In contrast, we aim to provide a general method and a framework that allows existing algorithms to be easily integrated as software modules in a plug-and-play fashion. Instead of implementing specific ad hoc MPI-based applications, one for each dataset and problem, we adapt our method to the MapReduce programming model by structuring the extraction of metadata as a map function and the classification or clustering operation as a reduce function or as a combination of map and reduce functions. MapReduce is ideal for this purpose because different algorithms can be easily plugged in as map or reduce functions, and the communication between map and reduce steps is automatically handled by the runtime, removing a major burden on the developers. We integrate the map and reduce functions into different MapReduce frameworks (i.e., Hadoop and MapReduce-MPI), as well as the Parallel MATLAB framework; and we use the frameworks for three scientific datasets (i.e., RNA secondary structures, ligand conformations, and folding proteins trajectories).

For each of the three datasets, we evaluate the scalability of our method when the size of the dataset and that of the distributed memory system increase. We also study

the accuracy of our method by statistically comparing the scientific results obtained by our method with the results of traditional approaches for each dataset. The results show that our method can achieve both better scalability and higher accuracy comparing to the respective traditional approaches.

1.2 Thesis Statement

In this thesis, we claim that the relevant properties of data can be captured across the dataset as metadata concurrently by analyzing each single data record independently. We also claim that the class memberships or patterns of data records can be identified by using classification or clustering techniques on the metadata. Further, we claim that using our analysis method, we can enable scalable and accurate data analyses (i.e., classification and clustering) for large-scale computational structural biology datasets on petascale distributed memory systems.

To validate the thesis statement and prove both the scalability and accuracy of our method, we present three representative case studies, one for each type of analysis considered in this thesis. We show how our method can do the following:

- Classify the secondary structures as more or less likely to occur in nature based on their chunk-based structural features and the RNA sequence's family model.
- Cluster ligand geometries into sets with different probabilities of well-docking into the protein pocket based on the densities of geometries.
- Cluster folding trajectories into patterns with different recurrences within and across trajectories (i.e., intra- and inter-trajectory, respectively) based on geometrical variations in time of the folding protein conformations.

1.3 Contributions

The contributions of this thesis are as follows.

- We introduce a transformative, general data analysis method together with algorithms supported by a MapReduce-style parallel programming model. The method avoids moving data to a centralized server, enables classification and clustering on large-scale data in a distrusted fashion, and ensures both scalability and accuracy of the analysis.

- We apply the method to three representative and diverse computational structural biology datasets. The different datasets are (1) large datasets of RNA sequences, to identify sequence features of RNAs and classify the secondary structures as more or less likely to occur; (2) large datasets of docked ligand conformations, to identify geometrical features of the ligand conformations and cluster the geometries in groups with high probabilities of well-docking into a protein pocket; and (3) large datasets of folding trajectories, to identify recurrent patterns in protein folding trajectories and cluster the folding trajectories into meta-stable and transition stages.

1.4 Organization

The remainder of the thesis is organized as follows. Chapter 2 presents the application of our method to RNA sequence datasets. Chapter 3 presents the application of our method to ligand conformation datasets. Chapter 4 presents the application of our method to protein folding trajectory datasets. Chapter 5 summarizes the accomplished work in this thesis, presents future work beyond this thesis, and discusses the broader impact of the thesis.

Chapter 2

CLASSIFICATION OF RNA SECONDARY STRUCTURES

Given an RNA sequence, different computational methods can predict multiple secondary structures, where each structure has a certain probability of happening. The first analysis problem tackled in this thesis is a classification problem in which we identify class memberships for these multiple secondary structures. Specifically, we classify a secondary structure as more or less likely to occur based on its chunk-based secondary structures and the RNA sequence's family model. To extract the secondary structures of an RNA sequence into metadata, we cut the sequence into shorter chunks using statistical information, predict the secondary structure of each chunk independently using existing prediction programs, and reconstruct whole chunk-based secondary structures from the chunks' predictions. The classification of secondary structures requires our method to learn the RNA family model from other secondary structures in the same family and classifying the new structures using the model.

In this thesis, we design and implement a classification framework supporting our method outlined above, and we integrate existing chunk-based prediction algorithms. We evaluate the framework using three datasets. The first dataset consists of 50 non-pseudoknotted sequences from the RFAM database, and the lengths of sequences range from 127 to 568 bases [12]. The second dataset consists of 23 pseudoknotted sequences from the RFAM and Pseudobase++ databases, and the lengths of sequences in this dataset range from 77 to 451 bases [12], [77]. The third dataset consists of 14 sequences from the virus family *Nodaviridae*, and the lengths of sequences range from 1,305 to 3,204 bases [42], [79]. The results show that our method exhibits linear scalability and can predict longer sequences than commonly used prediction programs

can. Moreover, our chunk-based method generates more accurate secondary structures than the same prediction programs do using non-chunk-based methods.

The rest of this chapter is organized as follows. Section 2.1 gives background information on RNA molecules and their secondary structures, existing chunking methods, the MapReduce programming model, and the Hadoop runtime system. Section 2.2 reviews the related work. Section 2.3 presents our method. Sections 2.4 and 2.5 present the performance scalability and accuracy evaluation, respectively, of the RFAM, Pseudobase++ database, and *Nodaviridae* virus dataset. Section 2.6 summarizes the research results of the thesis and discusses future work.

2.1 Background

2.1.1 RNA molecules

Ribonucleic acid (RNA) is made up of four types of nucleotide bases: adenine (A), cytosine (C), guanine (G), and uracil (U). A sequence of these bases is strung together to form a single strand RNA molecule. RNA plays important roles in many biological processes including gene expression and regulation. RNA molecules vary greatly in size, ranging from short sequence of 19 nucleotide bases in microRNAs [32] to long polymers of over 30,000 bases in complete viral genomes [78]. Although an RNA molecule is a linear polymer, it tends to fold back on itself to form a three-dimensional functional structure, mostly by pairing complementary bases. Among the four nucleotide bases, C and G form complementary base pairs by hydrogen bonding, as do A and U; in RNA (but not DNA), G can also base pair with U residues. The overall stability of an RNA structure is determined by its "minimal free energy," defined as the amount of energy it takes to completely unpair all of the base pairs that hold it together (e.g., by denaturing it with heat).

The three-dimensional (3-D) structure of an RNA molecule is often the key to its function. Because of the instability of RNA molecules, experimental determination of their precise 3-D structures is a time-consuming and costly process. However,

useful information about the molecule can be gained from knowing its secondary structure, that is, the collection of hydrogen-bonded base pairs in the molecule [63]. RNA secondary structures can be classified into two basic categories: stem loops and pseudoknots (see Figure 2.1). Both kinds of secondary structures, which are implicated in important biological processes such as gene expression and gene regulation [10], must contain at least one inversion, that is, a string of nucleotides followed closely by its inverse complementary sequence. Figure 2.2 shows an example of an inversion, with the 6-nucleotide string "ACCGCA" followed by its inverse complementary sequence "UGCGGU" after a gap of three nucleotides.

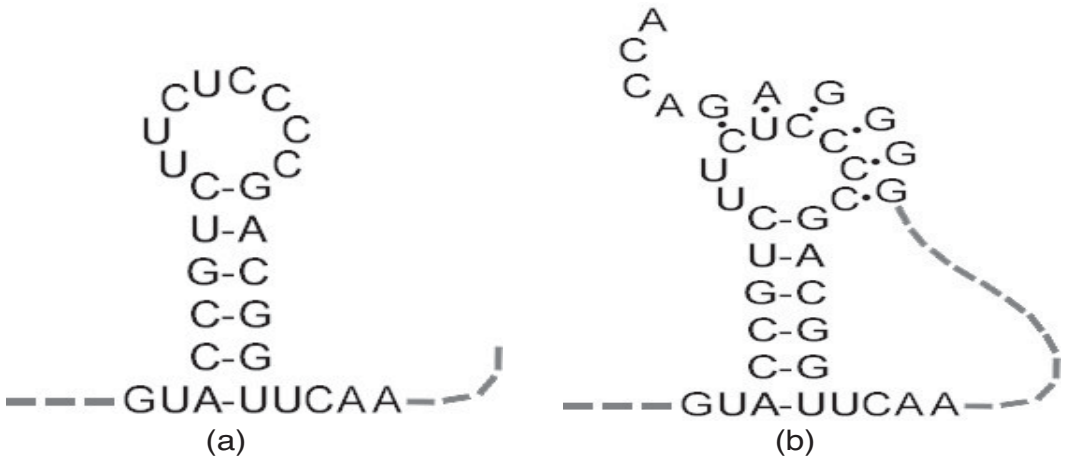


Figure 2.1: Stem loop (a) and pseudoknot (b).

2.1.2 Chunking methods based on inversions

Inversion excursion: The identification of regions in the sequence with high concentrations of inversions is essential to rebuilding the secondary structure of a long RNA sequence from the predictions of a shorter RNA sequence's chunks. These regions are likely to be generating the RNA's stem loops and pseudoknots, and thus it is recommended to avoid cutting them into separate chunks [88].

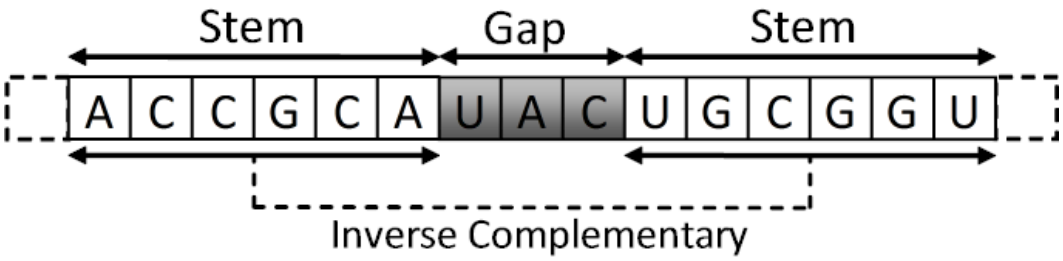


Figure 2.2: Inversion with stem length 6 and gap size 3.

The overall chunking method relies on a general excursion approach first formulated in [43] for a variety of DNA sequence analysis problems, but adapted in this thesis for RNA secondary structure predictions. In many bioinformatics applications, sequence analysis problems go beyond DNAs and call for identifying high-concentration regions of a certain property in the bases of biomolecular sequences. For example, replication origins in viral genomes have been predicted by looking for regions that are unusually rich in the nucleotides A and T in DNA sequences [17]. Leung et al. follow the same approach for RNA sequences by focusing on whether the nucleotide base is found inside an inversion [88]. Leung et al. refer to the excursions generated by this property as "inversion excursions." The excursion method requires assigning a positive score to each nucleotide if it is part of an inversion (including the two stems and the gap between them), and a negative score if it is not. The approach processes the entire nucleotide sequence in order to accumulate the scores to form inversion excursions.

To facilitate their analysis, Leung et al. used a parsing program to convert an RNA sequence into a binary sequence with the same length. If a nucleotide base is included in an inversion identified by the InversFinder program, it is given a value of "1"; if not, it is assigned a value of "0," as illustrated in Figure 2.3. Each "1" in the binary sequence is given a score of 1, and each "0" a negative score of s, which is determined as follows. The binary sequence is considered a realization of a sequence of independent and identically distributed (i.i.d.) random variables, $X_1, X_2, ..., X_n$, where

n is the length of the RNA sequence (i.e., number of bases). These random variables take values of either 1 or s. Let $p = Pr(X_i = 1)$ and $q = 1 - p = Pr(X_i = s)$. The parameter p is traditionally estimated by the percentage of bases contained in one or more inversions in the RNA sequence, that is, the percentage of "1"s in the binary sequence. The score per base $\mu = p + q * s$ is expected to be negative. This requirement prevents the tendency for long segments to have high scores. As done in [17] and other applications, the value of s can be conveniently selected by giving μ a value of -0.5 and then determining the value of s according to Equation 2.1.

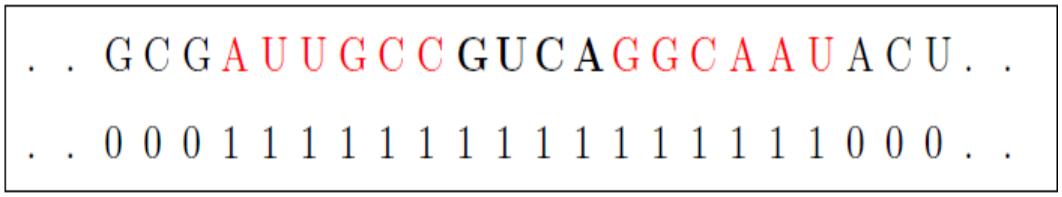


Figure 2.3: Binary sequence around an inversion. If a nucleotide base is included in an inversion identified by the InversFinder program, it is given a value of "1"; if not, it is assigned a value of "0."

$$s = \left\lfloor \frac{\mu - p}{q} \right\rfloor \tag{2.1}$$

The excursion score E_i at Position i of the sequence is defined recursively as in Equations 2.2 and 2.3.

$$E_0 = 0 \tag{2.2}$$

$$E_i = max(E_{i-1} + X_i, 0) \; for \; 1 \leq i \leq n \tag{2.3}$$

An excursion starts at a point i where E_i is zero, continues with a number of rising and falling stretches of positive values, and ends at $j > i$, where j is the next position with $E_j = 0$. The score then stays at zero until it becomes positive again when the next excursion begins. Plotting the excursion scores along the nucleotide positions of the RNA sequence offers an effective visualization of how inversion concentrations vary along the sequence. This plot can serve as a guide for choosing the cutting points

for the segmentation process. Figure 2.4 shows an example of an excursion plot. Note that rising stretches in the plot indicate the presence of inversions.

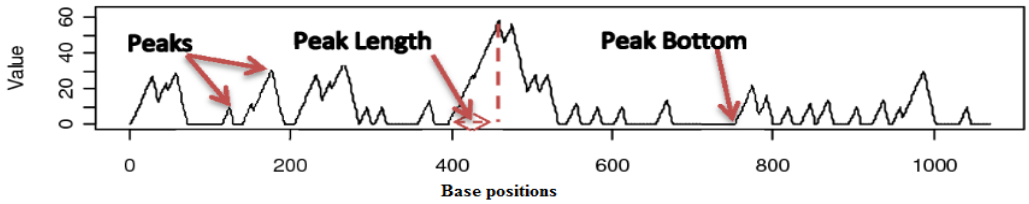


Figure 2.4: Excursion plot with peaks, peak bottoms, and peak lengths. Rising stretches in the plot indicate the presence of inversions.

After generating the excursion plot, the approach in [88] involves identifying the positions, called peaks, where the excursion scores are local maxima. Next the bottom of each peak (the last position with a zero excursion score right before the peak) is located. Then the length of the peak (the location difference between a peak and its peak bottom) is calculated. Note that since the chunk lengths are smaller than a prescribed maximum C_{max}, peak lengths greater than C_{max} have to be flagged and analyzed separately. Figure 2.4 also shows examples of peaks, peak bottoms, and peak lengths. Peaks are sorted in decreasing order based on their excursion scores. The sorted peaks are then used to cut sequences in chunks by the centered and optimized chunking methods.

Centered chunking method: The centered method cuts the sequence by identifying inversions and building the chunks around them. The objective is to segment the RNA sequence in such a way as to avoid losing structural information as much as possible by centering the longest spanning inversion clusters in the chunks. After peaks are identified, they are sorted in decreasing order of their excursion values. The peak with the highest excursion value is considered first, then the second highest peak, and so on. The algorithm stops either when all the peaks are exhausted or when all the inversion regions of the sequence (i.e., all "1"s in the binary sequence) are included in the chunks, whichever occurs first. Overlapping chunks are adjusted so that any

nucleotide base is captured by only one chunk, with priority given to the peak with a higher excursion score.

For each of the selected peaks, the positions of the inversions or peak length positions are centered within the maximum chunk length of C_{max} bases where C_{max} is defined by the user. The method starts at the bottom of this peak, follows the excursion until it returns to 0 the next time, and locates the position of the last peak before the excursion returns to 0. The method takes the sequence segment between the peak bottom and the position of the last peak and place the sequence segment in the center of the chunk, as illustrated in Figure 2.5. Suppose this centered segment contains x nucleotide bases. If $(c - x)$ is even, then the resulting chunk has $(c - x)/2$ bases on each side of the centered segment. If $(c - x)$ is odd, then the centered method adjusts the lengths on each side to the integers below and above $(c - x)/2$, allowing one side (chosen at random) to have one more nucleotide base than the other.

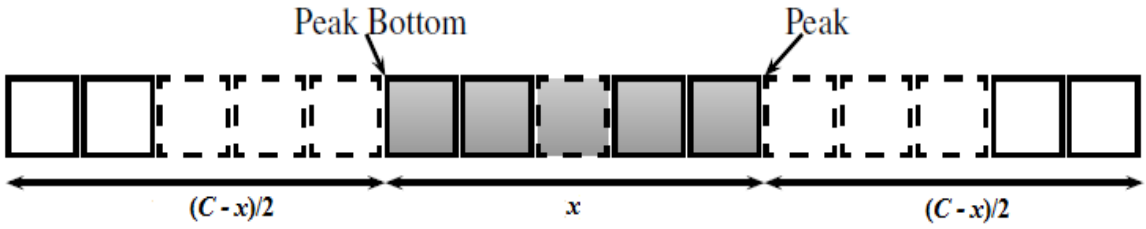


Figure 2.5: Centered chunking method where x = peak length. The centered method takes the sequence segment between the peak bottom and the position of the last peak in the excursion and places the sequence segment in the center of the chunk.

As an example, Leung et al. applied the aforementioned method to an RNA sequence in [83], that is, the 379-base RNA sequence RF00209_A in the RFAM database [12]. As shown in Figure 2.6, the sequence is segmented into six chunks using the centered chunking method. These six segments cover the entire sequence. Labels 1 through 6 in Figure 2.6 represent the six segments with decreasing order of peak excursion scores. After the peak scores are sorted, the peak with the highest excursion score is

considered first. In this example, the centered method uses the maximum chunk length $C_{max} = 100$. The highest peak is found at Position 297 with peak bottom at 257. Since there are other inversions after the highest scoring peak, the centered method follows the entire excursion to the end at Position 356. After locating the last peak in this excursion at 343, the centered method centers the sequence segment from 257 to 343 to produce the chunk covering the 100 positions from 250 to 349. Then the second highest scoring peak at Position 54 is considered, and the procedure is repeated. This time, the peak bottom is at Position 19, and the last peak before the end of this excursion is at Position 70. Centering the segment consisting of Positions 19–70 in a chunk of 100 requires 24 positions on each side, extending the chunk beyond the beginning of the sequence; therefore the centered method adjusts the chunk to start at Position 1 instead. Note that during the segmentation process, the centered method might get a chunk that overlaps with previously established chunks. In these cases, the method has to reconcile the situation by reducing one of the chunk lengths. For example, after establishing the first two chunks (labels 1 and 2 in Figure 2.6), the next highest peak to be processed is at Position 114, with peak bottom at Position 89. Centering this peak produces a chunk from Positions 52 to 151, overlapping with Chunk 2. The centered method resolves such conflicts by giving priority to the chunk with the higher number of bases within completely contained inversions. Using this rule, the centered method gives priority to Chunk 2 and reduces Chunk 3 to Positions 101–151. The process continues for the remaining Chunks 4–6.

Optimized chunking method: In the optimized method, cutting points are decided by choosing a segment containing the peak in an optimal position that yields the highest inversion score for the segment. The score is defined as the total number of nucleotide bases contained in the inversions that are entirely within the chunk. For example, consider a peak with peak length spanning the nucleotide bases between i and j and then all the chunks of size C_{max} covering this peak; that is, all segments with length C_{max} between Positions $j - (c - 1)$ and $i + (c - 1)$ are considered (see Figure 2.7). The chunk with the maximum inversion score is then selected. Beginning

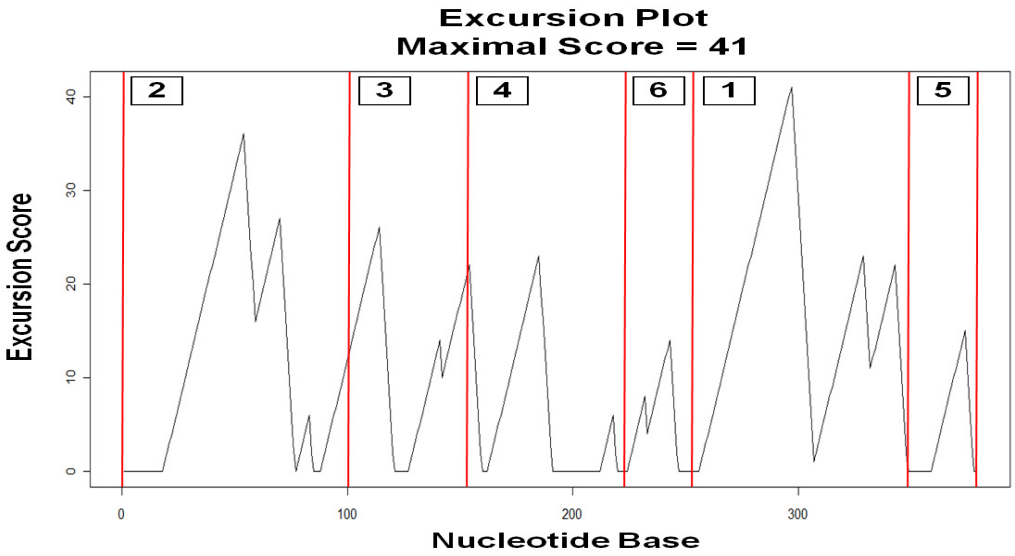


Figure 2.6: Six chunks obtained by using the centered method for the 379-base RNA sequence RF00209_A in the RFAM database.

with the highest peak, the process is repeated until either all the peaks are utilized or all the inversions of the sequence are contained in established chunks, whichever occurs first. When chunks overlap, the cutting points are adjusted in a similar way as described for the centered method. The optimized method ensures that peak length positions are included within a chunk but not necessarily in the center of the chunk.

As an example, Leung et al. applied the optimized method to the same RF00209_A RNA sequence file from the RFAM database in [88], as shown in Figure 2.8. The optimized method produced only five chunks covering all except the first 18 positions of the sequence. As shown in Figure 2.8, this method avoids cutting into sequence segments with rising excursion scores preceding the peaks. Also, the chunks produced by the optimized method cover only 96.3% of the sequence, leaving out those parts of the sequence where no inversions are found; therefore, wastage of computer resources is minimal with the optimized method.

Regular chunking method: The regular chunking method is the simplest method of segmentation and is used as a reference method in this thesis. This method cuts the

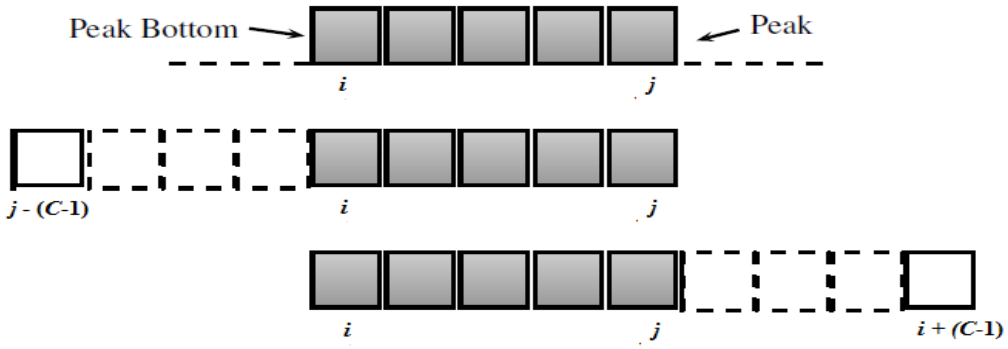


Figure 2.7: Chunks by the optimized method with peak spanning Positions i–j. All segments with length c between Positions $j-(c-1)$ and $i+(c-1)$ are considered. The chunk with the highest inversion score is selected.

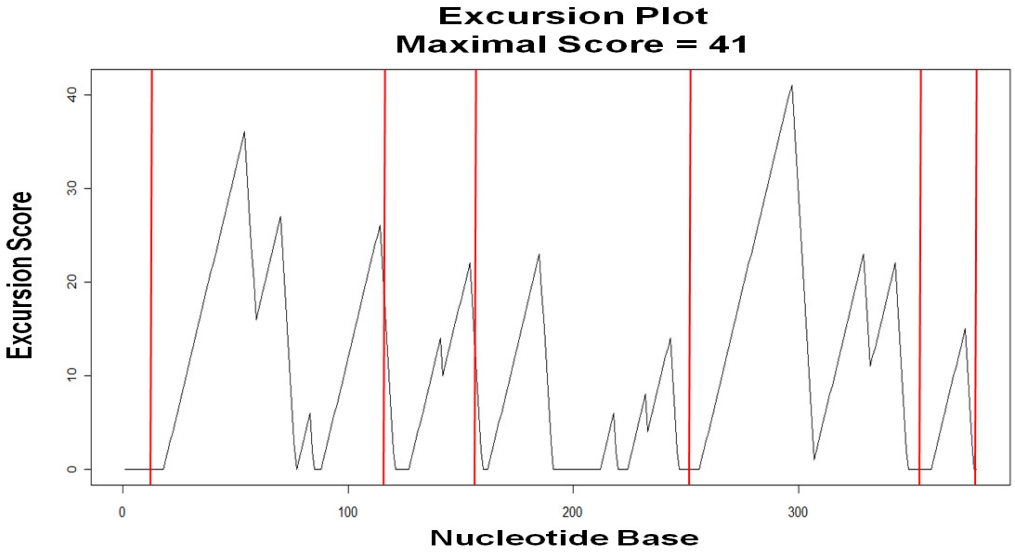


Figure 2.8: Five chunks obtained by the optimized method for the 379-base RNA sequence RF00209_A in the RFAM database. The chunks cover all but the first 18 positions of the sequence.

nucleotide sequence regularly into chunks of a specified maximum chunk length C_{max} until the sequence is exhausted.

For example, with $C_{max} = 100$, the sequence RF00209_A from the RFAM

database with 379 bases is cut into four chunks made up of nucleotide Positions 1–100, 101–200, 201–300, and 301–379 (Figure 2.9). Obviously, rising stretches in an excursion plot, which indicate the presence of inversions and are likely to be part of secondary structures, can often be cut by this method. As a result, important structural information can easily be lost. Intuitively, one expects that both the centered and optimized methods, which take the inversion locations into account when placing the chunks, perform better in retaining the secondary structure information in the sequences.

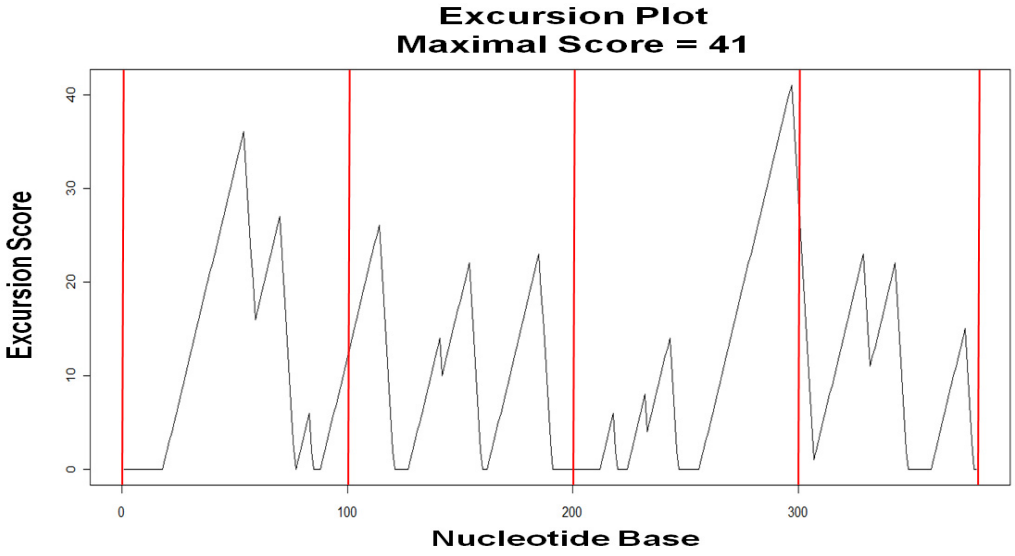


Figure 2.9: With $C_{max} = 100$, the sequence RF00209_A is cut into four chunks positioned at 1–100, 101–200, 201–300, and 301–379.

2.1.3 Hadoop

When implementing our method in a modularized framework we use Apache Hadoop [35]. Hadoop is an open source runtime library written in Java supporting the MapReduce programming model. Hadoop provides simple programming interfaces to developers. It handles load balancing and failure recovery automatically.

Hadoop includes two main components: Hadoop Distributed File System (HDFS) and Hadoop MapReduce. HDFS builds upon the local disks of the cluster nodes running Hadoop and provides a distributed file system that is accessible to all the Hadoop nodes. It is used to store input and output files of Hadoop programs. If a file is too large, HDFS divides the file into blocks, the size of which is determined by the programmer (64 MB by default). Hadoop MapReduce automatically runs a given Hadoop program in parallel using mappers and reducers processes. In execution, each mapper runs the map function on one block of the input and emits intermediate $\langle key, value \rangle$ pairs to the reducer (i.e., a map task); once finished, it runs another map task. The runtime library automatically groups all the intermediate values associated with the same key and sends them to a reducer. This process is the data-shuffling process. Each reducer runs the reduce function on all the values associated with the same key (i.e., a reduce task) and outputs another list of values as the final results. The programmer can control the behavior of the Hadoop program by specifying the map and reduce functions, as well as the number of mappers and reducers.

In Hadoop, load balancing is automatically handled by assigning map tasks to mappers in a first-in-first-out (FIFO) fashion. When the input files are large or the input dataset contains a large number of small files, a large number of map tasks are generated, each task processing one block of a file or one small file. Hadoop also automatically handles fault tolerance. When a failure happens to a map or a reduce task because of hardware or software issues on the node, Hadoop schedules the same task on another node to guarantee that every task is successfully executed exactly once.

2.2 Limits of Current Practice

2.2.1 RNA secondary structure predictions

Secondary structures are crucial for the RNA functionality, and therefore the prediction of the secondary structures has been widely studied. Development of mathematical models and computational prediction algorithms for stem-loop structures began in the early 1980s [56, 66, 90]. Pseudoknots, because of the extra base-pairings

involved, must be represented by more complex models and data structures that require large amounts of memory and computing time to obtain the optimal and suboptimal structures with minimal free energies. As a result, development of pseudoknot prediction algorithms began in the 1990s [64, 21].

Most existing secondary structure prediction algorithms are based on the minimization of a free energy (MFE) function and the search for the most thermodynamically stable structure for the whole RNA sequence [64, 62, 63, 39]. Searching for a structure with global minimal free energy may be memory and time intensive, especially for long sequences with pseudoknots. In order to overcome the tremendous demand on computing resources, various other algorithms have been proposed that restrict the types of pseudoknots for possible prediction in order to keep computation time and storage size under control [20]. Yet, most programs available to date for pseudoknot structures prediction can process sequences only of limited lengths—on the order of several hundred nucleotides). These programs therefore cannot be applied directly to larger RNA molecules such as the genomic RNA in viruses, which may be thousands of bases in length.

At the same time, minimal energy configurations may not be the most favorable structures for carrying out the biological functions of RNA, which often require the RNA to react and bind with other molecules (e.g., RNA binding proteins). Work of Taufer et al. suggests that local structures formed by pairing among nucleotides in close proximity and based on local minimal free energies rather than the global minimal free energy may better correlate with the real molecular structure of long RNA sequences [76]. This hypothesis has yet to be supported by more detailed experimental evidence. If proven correct, it can open the door to a new generation of programs based on segmenting long RNA sequences into shorter chunks, predicting the secondary structure of each chunk individually and then assembling the prediction results to give the structure of the original sequence. In previous work, Taufer et al. proposed predicting secondary structures for long RNA sequences using three steps: (1) cut the long sequence into shorter, fixed-size chunks; (2) predict the secondary

structures of the chunks individually by distributing them to different processors on a Condor grid; and (3) assemble the prediction results to give the structure of the original sequence. Taufer et al. used this approach on the genome sequences of the virus family *Nodaviridae*, leading to the discovery of secondary structures essential for RNA replication of the *Nodamura* virus [65]. However, the study also identified the need for a more effective segmentation strategy for cutting the sequence so that the predicted results of the chunks can be assembled to generate a reasonably accurate structure for the original sequence. Indeed, the selection of cutting points in the original RNA sequence is a crucial component of the segmenting step. In [83, 87], Leung et al. proposed to approach the problem by identifying inversion excursions in the RNA sequence and cutting around them. The authors considered two inversion-based segmentation strategies: the centered and optimized chunking methods. Both methods identify regions in the sequence with high concentrations of inversions and avoid cutting into these regions. In the centered method the longest spanning inversion clusters are centered in the chunks, while in the optimized method the number of bases covered by inversions is maximized in the chunks.

The centered and optimized chunking methods described above use predefined maximum chunk length (i.e., C_{max}) and maximum gap length (i.e., G_{max}) and thus result in chunks that contain no longer than C_{max} nucleotide bases. They are shown to be effective when the stem-loop structures in the given sequence are short and fit into single chunks. The methods have two main limitations, however. First, in long RNA sequences, nucleotide bases that are far away can still bind with each other to form a pseudoknot structure longer than C_{max}. With the two chunking methods, the small window for the lengths of the chunks (i.e., C_{max}) does not allow chunks longer than C_{max}. In this scenario, the longer secondary structure is cut, and the methods fail to predict the correct secondary structures.

Second, after assembling the prediction results, multiple possible secondary structures can be generated for one given sequence based on the different parameter values used (i.e., centered or optimized chunking method, C_{max}, L_{min}, and G_{max}). Each of

these secondary structures has a certain probability of occurring. In the work of Taufer et al. and Leung et al., the classification based on the likelihood that the secondary structure belongs to an RNA family does not take place. Instead, after obtaining the resulting structures from the merging processes, these structures are compared with known secondary structures. The overall approach is effective for validation purposes. When the known structures are not available a priori, however, defining the probability for which a sequence folds into a subset of possible secondary structures is not possible. In this thesis, we introduce a classification framework that explores the large parametric space of the chunk-based prediction method. Moreover, our framework allows easy integration of methods that do not rely on the known structure of the single sequence. We briefly discuss such classification methods in Section 2.6.

2.2.2 MapReduce in bioinformatics applications

The MapReduce programming model has been widely adapted for many bioinformatics applications. RNA sequence analysis studies include the work of Hong et al. [40] and Langmead et al. [45]. Hong et al. designed an RNA-Seq analysis tool for estimating gene expression levels and genomic variant calling and implemented it in Hadoop. Their work reduced the misalignments of short RNA-Seq reads originating from splicing junctions by using a transcriptome-based reference. Langmead et al. designed Myrna, a cloud-based efficient analysis software for transcriptome sequencing (RNA-Seq) data. Their work adapted the workflow of calculating differential gene expression in large RNA-Seq datasets into MapReduce on Amazon's Elastic Compute Cloud. Both works estimate the gene expression levels by performing sequence alignment. To the best of our knowledge, the work in this thesis is the first to adapt MapReduce into secondary structure predictions of long RNA sequences.

Work in other areas of bioinformatics includes that of Matsunaga et al. [51] and Schatz [68]. Matsunana et al. parallelized and deployed the commonly used bioinformatics tool NCBI BLAST [3], which finds regions of similarity between biological sequences including DNA and RNA, using MapReduce; this implementation is called

CloudBLAST. Schatz developed a new algorithm modeled after the short read-mapping program RMAP [72] to map next-generation sequence data to the human genome and other reference genomes; this implementation is called CloudBurst and is implemented in Hadoop. Both works focused on extending and implementing well-established software tools on cloud computing environments using MapReduce. In comparison, the work in this thesis not only identifies and adapts the workflow of chunk-based RNA secondary structure prediction but also allows easy integration of novel methods for chunking the long sequences and classifying resulting structures using the family model.

2.3 Methodology

In this section, we discuss the methodology of chunk-based prediction and classification of RNA secondary structures as shown in Figure 2.10. We present the research on exploring the large parametric space of the chunk-based prediction method by defining the workflow, adapting it for the MapReduce programming model, and investigating different levels of granularity for the search tree.

2.3.1 Workflow for parallel chunk-based predictions

Since the chunking process of an RNA sequence can be performed in different ways, the search for effective ways to cut sequences can require a large search space and generate a large number of independent prediction jobs that can potentially be performed in parallel. We define the workflow for a parallel chunk-based RNA secondary structure prediction as the combination of four steps:

1. Chunking: each RNA sequence is cut into multiple chunks (or segments) based on various chunking algorithms and parameters;
2. Prediction: the secondary structure for each chunk is predicted independently by using one or more well-known prediction programs;
3. Reconstruction: multiple whole chunk-based secondary structures of a sequence are reconstructed from predicted structures, one for each chunk; and
4. Analysis: reconstructed structures are compared with known structures to assess prediction accuracies.

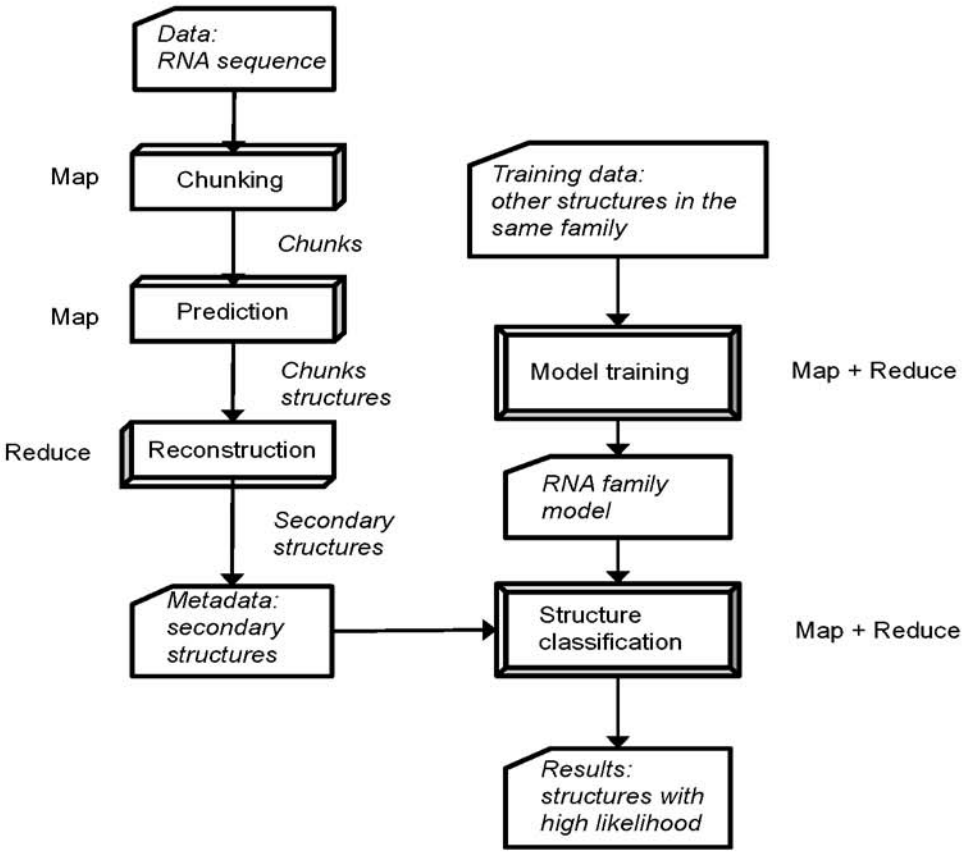


Figure 2.10: Overview of the classification of RNA secondary structures.

Figure 2.11(a) shows the prediction workflow. Note that the chunks do not necessarily have the same length: the lengths depend on the chunking method and parameters used. Also note that the chunk's prediction time and memory usage can vary based on the number of nucleotides in the chunk and the prediction program used. In most prediction programs the time and memory used do not grow linearly but exponentially or polynomially with the number of nucleotides, with the exponential factor and/or the polynomial components depending on the program complexity and its ability to capture complex RNA secondary structures such as pseudoknots.

Chunking process based on inversions: Given a long RNA sequence, we identify regions with high concentrations of inversions by using adapted versions of the centered

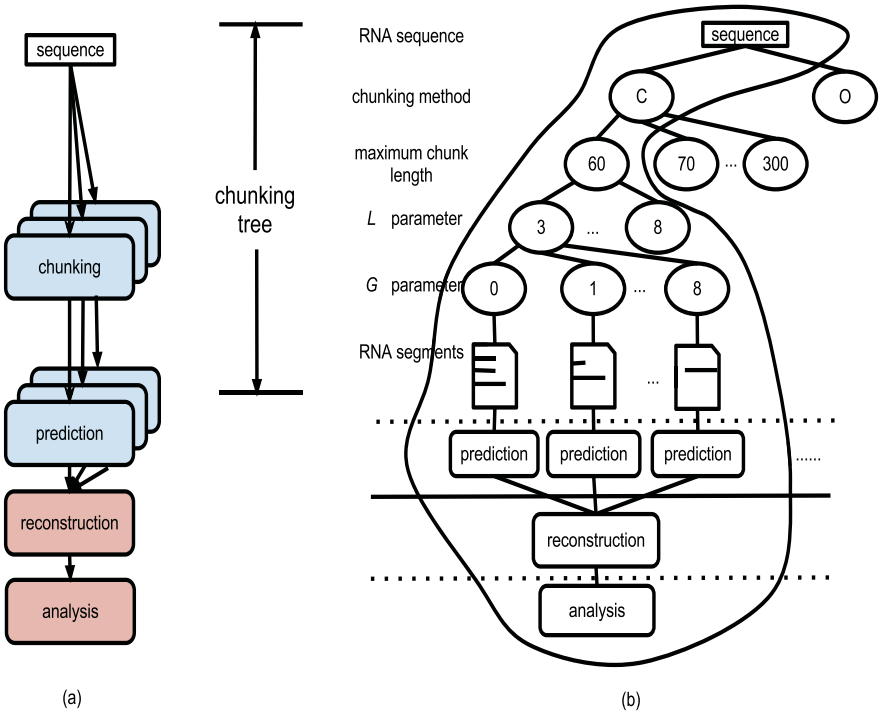


Figure 2.11: Workflow of the chunk-based RNA secondary structure prediction framework (a) and example of searching paths (b).

and optimized chunking algorithms of Leung et al. described in Section 2.1.2. We also adapt a simple version of the chunking method proposed by Taufer et al. that we call the regular chunking method and use it as a reference method in the thesis. This method cuts the nucleotide sequence regularly into chunks of a specified maximum chunk length C_{max} until the sequence is exhausted.

As described in Section 2.1.2, for the centered and optimized chunking methods, after generating the excursion plot, we identify the positions, called peaks, where the excursion scores are local maxima. Then, the bottom of each peak (the last position with a zero excursion score right before the peak) is located. After that, the length of the peak (the location difference between a peak and its peak bottom) is calculated. Note that since we require chunk lengths to be smaller than a prescribed maximum

length C_{max}, peak lengths greater than C_{max} have to be aggregated and analyzed separately. Figure 2.4 also shows examples of peaks, peak bottoms, and peak lengths. Peaks are sorted in decreasing order based on their excursion scores. The sorted peaks are then used to cut sequences in chunks by the centered and optimized chunking methods.

Prediction based on well-known algorithms: After the RNA sequence is cut into chunks, the structure of each chunk is predicted independently by using well-known algorithms and their programs. We also use the same prediction algorithms to predict the entire sequence without chunking for accuracy analysis. We employ seven commonly used prediction programs to test the chunking methods. These prediction programs typically involve some form of minimization of free energy, maximization of expected accuracy, or dynamic programming models in their algorithms. The programs that predict structures only for non-pseudoknotted sequences are UNAFOLD (2008) [50] and RNAfold (1994) [39]. The programs that predict both pseudoknotted and non-pseudoknotted sequences are IPknot (2011) [67], pknotsRG (2007) [62], HotKnots (2005) [63], NUPACK (2004) [20], and PKNOTS(1998) [64]. All the prediction programs used in this thesis are publicly available.

Reconstruction based on concatenation: The results of the chunk predictions are assembled to build a whole secondary structure. Currently, our framework simply concatenates all these predicted secondary structures to give the secondary structure for the whole sequence. This approach is possible because the cutting does not allow any overlap between two consecutive chunks. More sophisticated reconstruction methods that include partial chunk overlaps can be used with minor changes to our framework.

Accuracy analysis based on comparisons with known structures: Both the whole and the assembled predicted structures are compared with the known structure in order to obtain their respective prediction accuracies so that we can assess to what degree the chunking method can preserve the prediction accuracy of the program when applied without any segmentation. Figure 2.12 shows the RF00209_A nucleotide sequence along with the bracket view of its experimentally known secondary structure. In the bracket

view representation, bases that are hydrogen bonded with other bases are represented by a "(" or a ")"; a matching pair of "(" and ")" indicates that the bases at those positions are paired to be part of a secondary structure. Unpaired nucleotide bases are represented by a ":" (colon).

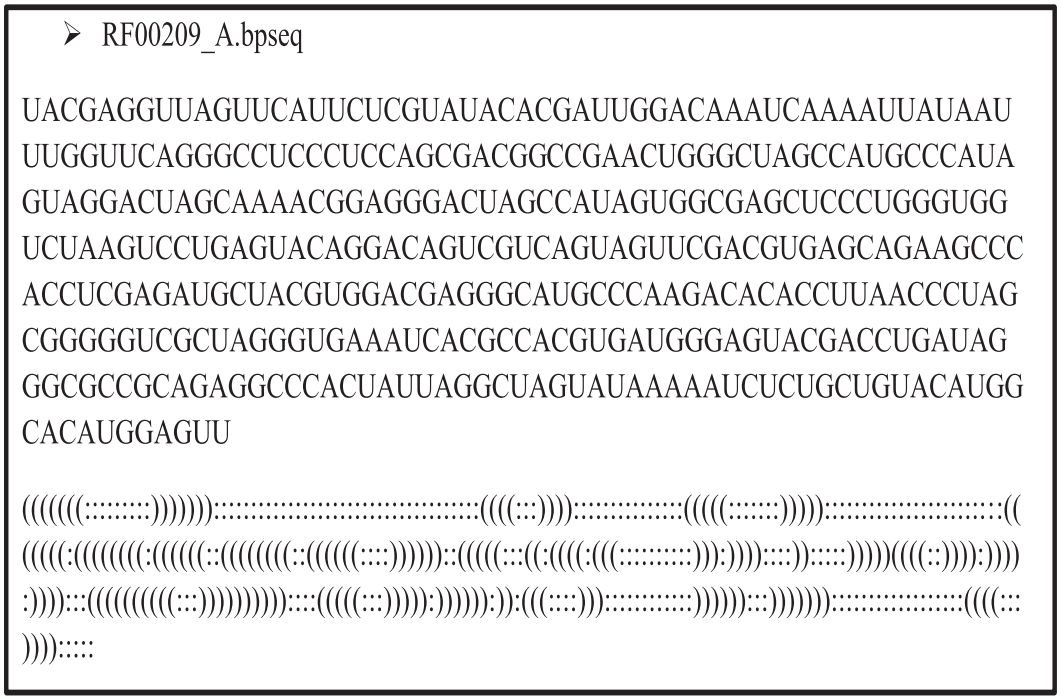


Figure 2.12: RF00209_A sequence and its experimental secondary structure from RFAM database. In the bracket view representation, bases that are hydrogen bonded with other bases are represented by a "(" or a ")"; a matching pair of "(" and ")" indicates that the bases at these positions are paired to be part of a secondary structure. Unpaired nucleotide bases are represented by a ":" (colon).

Various statistical tests are applied to the accuracy analysis for the different chunking methods, including t-tests, Pearson correlation analysis, and the non-parametric Friedman tests [73, 34]. We use the statistical functions provided by MATLAB [1]. Metrics of interests include: (1) accuracy chunking (AC), which is the accuracy of the predicted structure assembled from the chunks when compared with the known secondary structure; (2) accuracy whole (AW), which is the accuracy of the

predicted structure obtained from the whole sequence when compared with the known secondary structure; and (3) accuracy retention (AR), which is the ratio between AC and AW. While AC and AW reflect accuracies of the particular prediction in use with and without chunking, AR tells us how well a particular chunking method (i.e., centered, optimized, and regular) retains the accuracy of the original prediction program. AC and AW are given by the percentage agreement of the predicted structure with the known real structure calculated as

$$\frac{100 * [a + 2 * b]}{n}, \tag{2.4}$$

where a and b represent respectively the number of unpaired bases and the number of base pairs in common between the two structures and where n is the length of the RNA sequence. Large AC and AW values (close to 100%) for a predicted structure mean that it is highly similar to the real structure.

The accuracy retention (AR) is defined as

$$\frac{AC}{AW}. \tag{2.5}$$

AR provides a comparison of the prediction accuracies with chunking versus without chunking. Intuitively, we expect that a good chunking method causes only a minimal loss of prediction accuracy after cutting the sequence and has AR values somewhat less than but close to 1. In Section 2.5, however, we show that in most of the cases the AR values turn out to be greater than 1, meaning that secondary structure predicted using chunking is more similar to the real structure than it is to the secondary structure predicted by using the whole sequence. Several standard statistical tests, including t-tests, Pearson correlation analysis, and the non-parametric Friedman tests, are applied to analyze the AR values for the different chunking methods.

2.3.2 Integrating the searching paths into the MapReduce programming model

Given an RNA sequence, the search for the best set of chunking parameters (i.e., maximum chunk length C_{max}, centered or optimized chunking method, minimum stem length L_{min}, and maximum gap length G_{max}) requires traversing or searching a multilevel tree (i.e., the chunking tree in Figure 2.11(b)). In the chunking tree, each path from the root (RNA sequence) to the leaves (RNA chunks) represents a set of parameter values of the chunking method, namely, C_{max}, L_{min}, and G_{max}. The overall workflow (including the chunking, prediction, reconstruction, and analysis steps) naturally adapts to fit into the MapReduce programming model and can be easily implemented with Hadoop, for which the chunking and predictions can be solved by multiple mappers while the reconstruction and the analysis are done by a single reducer. In our framework, each MapReduce job is designed to partially traverse the multilevel tree. Multiple MapReduce jobs can be executed in parallel to explore the whole tree. The multiple searching paths combine attributes of both breadth-first search (performed by multiple MapReduce jobs in parallel) and depth-first search (performed by a single MapReduce job). While traversing the tree with multiple MapReduce jobs, we can explore the impact of different chunking methods as well as different C_{max}, L_{min}, and G_{max} values for a given sequence. An example of a MapReduce job is shown in the circled part of Figure 2.11(b), for which we assume the centered chunking method, with fixed $C_{max} = 60$ bases, and we vary L_{min} and G_{max} between 3 and 8 and between 3 and 8, respectively. As previously outlined in Section 2.3.1, for a sequence and a combination of parameters, the mappers perform the chunking and predictions. The input to each mapper is a $\langle k_1, v_1 \rangle$ value pair, in which k_1 is the ID of the sequence, and v_1 is the chunking parameters' values (including the chunking method). Each mapper cuts the sequence according to the chunking parameters values in the chunking step by identifying a variable number of chunks meeting the parameter requirements. Note that each combination of parameters (each branch of the tree) can result in a variable number of chunks. Each mapper performs the prediction on one or more chunks using

a certain prediction program. Here we use five secondary structure prediction programs capable of predicting pseudoknots (IPknot [67], pknotsRG [62], HotKnots [63], NUPACK [20], and PKNOTS [64]) and two programs that do not include this capability (i.e., UNAFOLD [50] and RNAfold [39]). Other programs can be easily used in our framework as a plug-and-play software module. After the prediction, each mapper outputs the list of $\langle k_2, v_2 \rangle$ pairs as the intermediate output to reduce. The k_2 is the ID of the whole secondary structure to which the predicted chunk belongs, and v_2 is the predicted secondary structure of the chunk. After the Hadoop runtime system groups all the values associated with the same key and passes the $\langle k_2, list(v_2) \rangle$ to the reducer, the reducer reconstructs the whole secondary structure of the sequence using all the v_2 (predicted chunk structures) associated with the same k_2. If required, the reducer analyzes the results in terms of their accuracy. After the accuracy has been computed, the reducer outputs the final results as $list(v_3)$, in which v_3 is the AR for reconstructed structures.

2.3.3 Tree granularity

In general, a mapper is the process that runs on a processor that applies the map function to a specific key and value pair. In our framework, each mapper runs the chunking process on an RNA sequence with a given set of parameter values and then predicts one or multiple chunks. The granularity of the mapping can vary based on the number of chunks each mapper is assigned to predict. Our MapReduce framework includes both a coarse-grained mapping and a fine-grained mapping as shown in Figure 2.13, in which each box represents a mapper. With the coarse-grained mapping, each mapper explores one branch of the chunking tree: it cuts the sequence into a set of segments based on a combination of L_{min} and G_{max} values and predicts all the segments it generates locally in order. With the fine-grained mapping, multiple mappers explore one branch of the chunking tree: each mapper cuts the same sequence into the same set of segments, but this time it predicts only one chunk that it generates. This means that if, for example, the sequence is cut into five segments, then five mappers are

exploring the same branch of the chunking tree, replicating the chunking process but predicting only one distinguished segment of the five chunks available. The mappers determine which segment to predict based on a hash function; thus the mappers do not need to synchronize their work or directly agree on what chunk to predict. The hash function uses the ASCII value of the chunk identifier as the key and the identifier of each mapper as the value. The function selects the segments to mappers in a round-robin fashion.

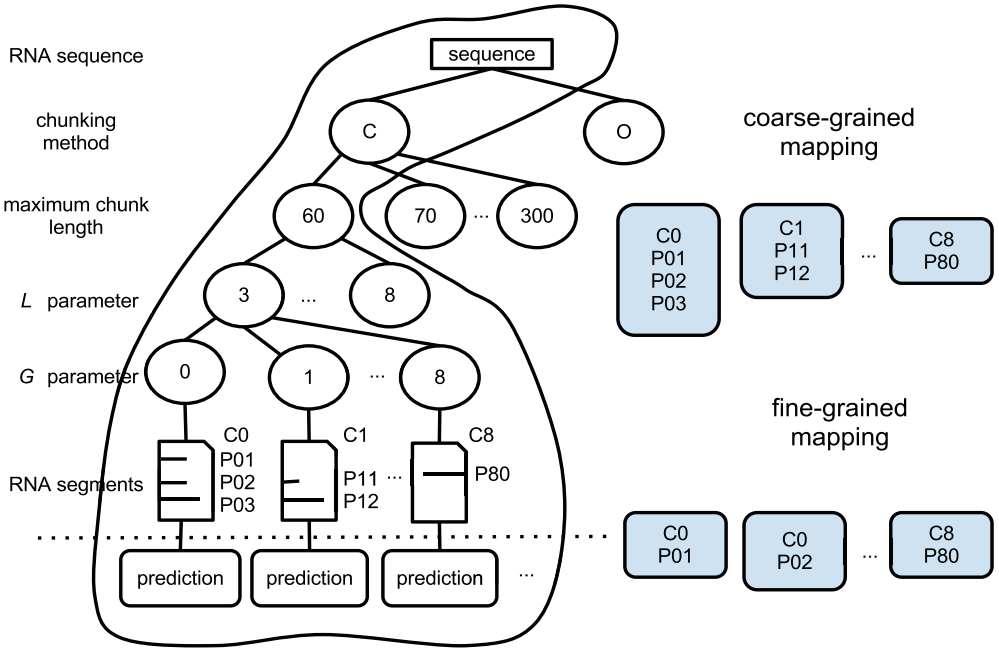


Figure 2.13: Mapping of tree granularity. Each mapper explores one branch of the tree and generates the set of segments as the output of the chunking program. The mapper predicts one or more segments generated based on the number of segments each mapper is assigned. Coarse-grained mappers explore one whole branch of the tree at a time. Fine-grained mappers predict one chunk at a time.

2.4 Performance

2.4.1 Platform

We ran the Hadoop framework on a UD cluster called Geronimo that is composed of 8 dual quad-core compute nodes (64 cores), each with two Intel Xeon 2.50 GHz quad-core processors. A front-end node is connected to the compute nodes and is used for compilation and job submissions. A high-speed DDR InfiniBand interconnect for application and I/O traffic and a Gigabit Ethernet interconnect for managing traffic connects the compute and front-end nodes. Our implementation is based on Hadoop 0.20.2.

2.4.2 Datasets

For the performance scalability analysis, we use two datasets of sequences. The first dataset contains 12 sequences from the RFAM database [12]. The lengths of the sequences range from 79 to 451 bases. Note that this is a subset of the 23 pseudoknotted sequences discussed before to show the performance scalability. We use the complete 23 sequences for the accuracy discussion later. The second dataset contains longer sequences from the virus family *Nodaviridae* [42, 79]. The lengths of the sequences range from 1,305 to 3,204 bases. Note also that because these RNA sequences are long and contain possible pseudoknots, none of the available well-known programs can predict the secondary structures for the entire sequences. The use of the Hadoop framework is vital for the exhaustive, efficient exploration of the tree branches. The virus family *Nodaviridae* is divided into two genera: *alphanodaviruses*, which primarily infect insects, and *betanodaviruses*, which infect only fish. These viruses share a common genome organization, namely, a bipartite positive strand RNA genome (i.e., mRNA sense). The longer genome segment RNA1 (ranging in size from 3,011 to 3,204 nucleotide bases) encodes the RNA-dependent RNA polymerase that catalyzes replication of both genome segments, while the shorter RNA 2 (ranging in size from 1,305 to 1,433 nucleotide bases) encodes the precursor of the viral capsid protein that encapsidates the RNA genome. The 14 sequences we analyze in this thesis are listed in

Table 2.4.2. These sequences are sorted based on their increasing lengths, and this order is preserved in all the figures and tables presented below.

Name	Geno. Segment	Length	Geno. Segment	Length
Boolarra virus (BoV)	RNA2	1,305	RNA1	3,096
Pariacoto virus (PaV)	RNA2	1,311	RNA1	3,011
Nodamura virus (NoV)	RNA2	1,336	RNA1	3,204
Black beetle virus (BBV)	RNA2	1,393	RNA1	3,099
Flock house virus (FHV)	RNA2	1,400	RNA1	3,107
Striped jack nervous necrosis virus (SJNNV)	RNA2	1,421	RNA1	3,107
Epinephelus tauvina nervous necrosis virus (ETNNV)	RNA2	1,433	RNA1	3,103

Table 2.1: Fourteen sequences from the virus family *Nodaviridae*.

2.4.3 Results and discussion

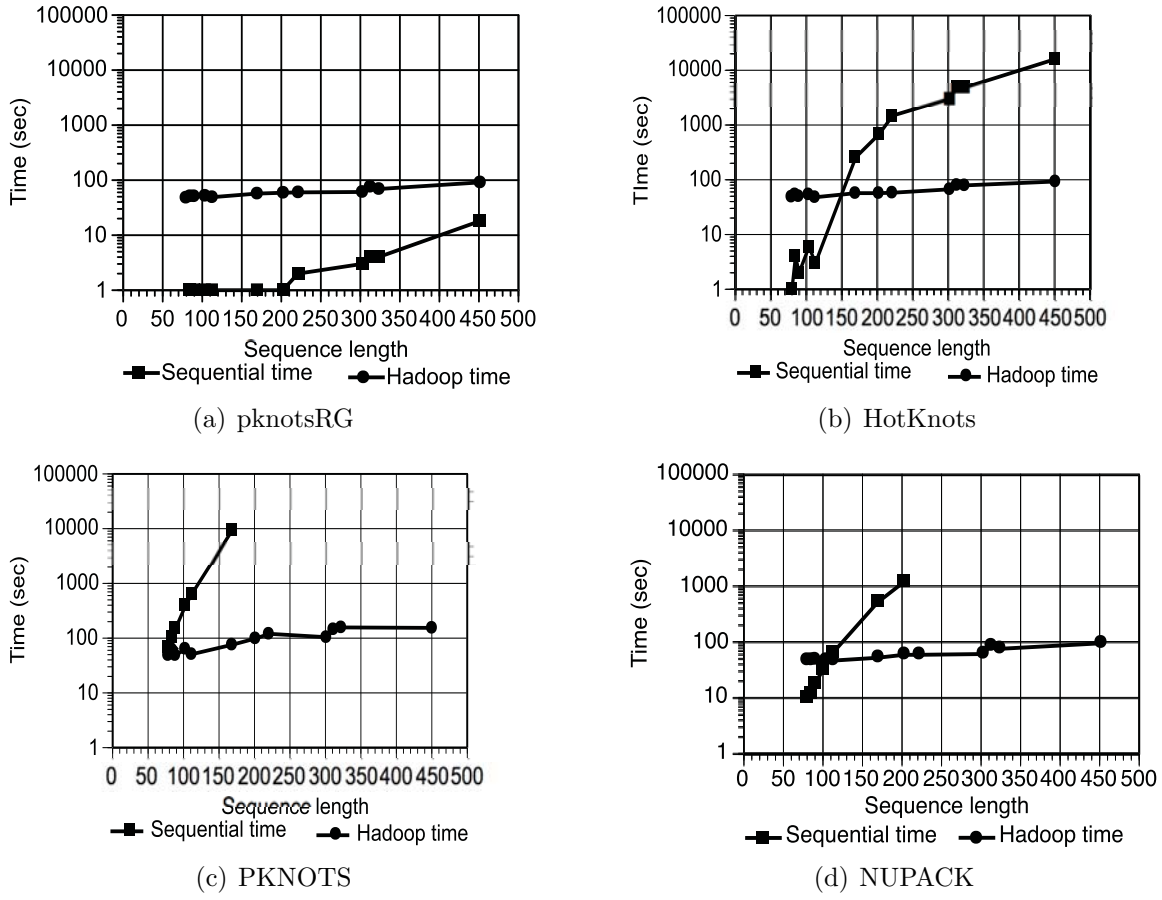
In this section, we show the scalability of our method by comparing the execution time of our chunk-based method with the well-known non-chunk-based prediction programs using the whole sequences of the RFAM dataset. Then we discuss the performance aspects of our method when running with long sequences from the virus family *Nodaviridae*, which the well-known prediction programs cannot handle. The performance aspects include the impact of using coarse- or fine-grained mapping, using the centered or optimized chunking method, and the efficiency of searching the chunking tree.

Chunk-based vs. non-chunk-based sequential prediction programs on RFAM sequence benchmarks: Our objective in measuring whole sequences is to understand how the execution time changes when the length of the sequence increases for our method and for the well-known prediction programs. When considering the chunk-based predictions, for each sequence and each prediction program, we run our framework using the regular, centered, and optimized chunking methods with maximum chunk length C_{max} equal to 60, L_{min} ranging from 3 to 8, and G_{max} ranging from 3 to 8. When

considering the prediction of each sequence as a whole (non-chunk-based) using each well-known prediction program, we use one of the compute nodes on our cluster.

In Figure 2.14, we present the execution time for both the Hadoop chunk-based predictions and the sequential predictions using the four prediction programs: (a) pknotsRG, (b) HotKnots, (c) PKNOTS, and (d) NUPACK. In each subfigure, the x-axis is the 12 RFAM sequences sorted based on their lengths in increasing order; the y-axis is the execution time in seconds in logarithmic scale for both chunk-based MR and non-chunk-based sequential predictions. Note that in PKNOTS and NUPACK, some long sequences are missing. The reason is that these prediction programs cannot predict the whole sequence sequentially, because of memory limitations. From Figure 2.14, we observe that for HotKnots and NUPACK, the execution time for chunk-based predictions is larger than the sequential prediction when the sequence length is short (less than 150 bases). When the sequence length grows (more than 150 bases), however, the chunk-based predictions run significantly faster than the sequential prediction. For PKNOTS, our chunk-based framework always runs faster than the sequential prediction. For pknotsRG, our framework runs slower than the sequential prediction, but the runtime of the sequential prediction grows rapidly with the length of the sequences while our method stays relatively constant. This result indicates that for longer sequences, our framework may run faster than the sequential prediction. Overall we observe that as the length of the sequence grows, the execution time of the sequential prediction grows exponentially or polynomially with the length of the sequences for all the 4 prediction programs. However, in the prediction of RNA secondary structure using the chunk-based method and using Hadoop implementation, we observe that the execution time for chunk-based predictions does not grow significantly with the sequence length. The reason is that our chunking methods cut the whole RNA sequence into segments smaller than or equal to 60 bases. In other words, as the length of the whole sequence grows, the lengths of chunked segments are still smaller than or equal to 60 bases. The predictions are performed in parallel across the Hadoop nodes of the distributed memory system. Our chunking methods cut the sequence into more

segments and by doing so add execution time for chunking (overhead) and prediction. Our measurements show, however, that this overhead is not significant. With very long RNA sequences (e.g., RNA viral genomes with thousands of bases), the chunk-based method is promising for two reasons. First, it allows us to predict secondary structures that cannot be predicted otherwise, namely, when considering the sequence as a whole. Second, it allows us to keep under control the total execution time by controlling the max length of the chunk.



(a) pknotsRG (b) HotKnots

(c) PKNOTS (d) NUPACK

Figure 2.14: Profile of execution time for chunk-based MapReduce (MR) method and non-chunk-based sequential method.

Execution time on the virus family Nodaviridae sequence benchmarks: When the sequences are long (i.e., thousands of nucleotide bases), the well-known programs cannot predict the secondary structures of such sequences. The use of the Hadoop

framework is vital for the exhaustive, efficient exploration of the tree branches. We measure the total time needed to explore the chunking tree of each sequence using either the centered or optimized methods and with either the coarse-grained or fine-grained mapping. The total time includes chunking and predictions (map time), reconstruction (reduce time), exchanging of predictions among nodes (shuffling time), and any overhead due to load imbalance and synchronizations. Note that this time does not include analysis since the secondary structures of the sequences considered here are not known experimentally and thus an analysis in terms of accuracy is not feasible. We use IPknot for our predictions since it is the most recently implemented code and its accuracy values are very high, as Section 2.5 will show.

Each of the four subfigures in Figure 2.15 shows the total times in seconds for exploring the prediction trees (left y-axes) and the number of map tasks (right y-axes) for the 14 sequences when a max chunk length C_{max} of 60, 150, and 300 bases, L_{min} ranging from 3 to 8, and G_{max} ranging from 3 to 8 are used. Each subfigure shows three groups of times, one for each maximum chunk length. Each group lists the 14 sequences sorted based on their lengths in nucleotide bases. More specifically, Figure 2.15(a) presents the times and number of map tasks when the coarse-grained MapReduce implementation and the centered chunking method are used; Figure 2.15(b) presents the times and number of map tasks when the coarse-grained MapReduce implementation and the optimized chunking method are used; Figure 2.15(c) presents the times and number of map tasks when the fine-grained MapReduce implementation and the centered chunking method are used; and Figure 2.15(d) presents the times and number of map tasks when the fine-grained MapReduce implementation and the optimized chunking method are used. As presented above, when using coarse-grained mapping, each mapper performs the chunking for the assigned sequence using a set of parameter values for the max length of stems (L) and gaps (G); the mapper then predicts the secondary structures of all its local chunks. This approach results in the exploration of a whole branch of the tree by the mapper. The total number of branches (and map tasks) is given by the combinations of L and G values (i.e., 54). When using fine-grained

mapping, chunking of a sequence based on a set of L and G values is performed across mappers and mappers are assigned resulting chunks in a round-robin fashion. Computationally this is performed by replicating the chunking processes across mappers and by using a hash function to assign different chunks to different mappers. The number of chunks equals the number of map tasks and depends on the number of inversions identified in the chunking process. We observe that using coarse-grained mapping, our method predicts the secondary structures for the group of sequences with shorter lengths (i.e., ranging in size from 1,305 to 1,433 nucleotide bases) in less than 200 seconds and for the groups of sequences with longer lengths (ranging in size from 3,011 to 3,204 nucleotide bases) in less than 1,200 seconds (i.e., under 20 minutes). We also observe that using fine-grained mapping, our method predicts the secondary structures for the group of sequences with shorter lengths in less than 500 seconds and for the group of the sequences with longer lengths in around 3,500 seconds. Note that these sequences are too long for the well-known prediction programs to handle using the whole sequence.

Centered vs. optimized chunking method: Here we discuss the performance difference by using the centered or the optimized chunking method on the two groups of sequences from the virus family *Nodaviridae*. When comparing centered with optimized chunking methods for the coarse-grained mapping, we observe that the two methods result in similar execution times (Figure 2.15(a) and Figure 2.15(b)). Table 2.2 quantifies the similarity for both subgroups (i.e., RNA2 and RNA1), which is within 3%. This observation differs from previous results in which the centered method resulted in shorter execution times because a different implementation of the chunking methods and a different program were used.

When comparing centered with optimized chunking methods for the fine-grained mapping, the optimized method results in a slightly lower execution time. As shown in Table 2.3, the execution times of fine-grained mapping when using the optimized chunking method for both subgroups (RNA2 and RNA1) is 11% to 18% slower than using the centered method. Table 2.4 shows the average number of chunks (i.e., map

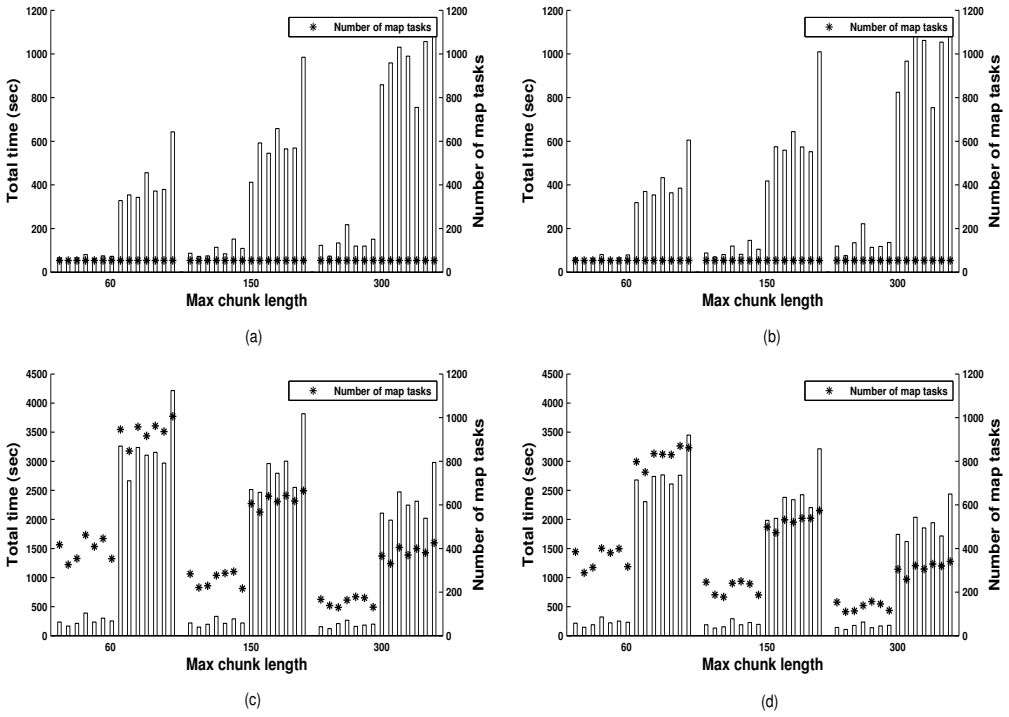


Figure 2.15: Total time in seconds for coarse- vs. fine-grained mapping and centered vs. optimized methods: (a) coarse-grained mapping using centered method, (b) coarse-grained mapping using optimized method, (c) fine-grained mapping using centered method, and (d) fine-grained mapping using optimized method.

Table 2.2: Average total times for the seven sequences in RNA2 and in RNA1 for coarse-grained mapping using centered and optimized methods.

Mean Total	RNA2(∼1300 bases)			RNA1(∼3100 bases)		
Time (sec)	60	150	300	60	150	300
Centered	68.7	99.0	134.3	410.7	618.0	971.7
Optimized	66.7	98.9	131.6	404.3	618.9	986.1
Opti./Cent.	0.97	1.00	0.98	0.98	1.00	1.01

tasks) for both subgroups using centered and optimized methods. We can see that optimized method results in 10% to 19% fewer chunks. The reason is that the optimized method tends to cut sequences into fewer chunks; this process leads to fewer map tasks

Table 2.3: Average total times for the seven sequences in RNA2 and in RNA1 for fine-grained mapping using centered and optimized methods.

Mean Total	RNA2(∼1300 bases)			RNA1(∼3100 bases)		
Time (sec)	60	150	300	60	150	300
Centered	257.7	232.3	185.7	3228.7	2871.7	2303.9
Optimized	226.6	197.4	164.6	2758.0	2366.0	1907.7
Opti./Cent.	0.88	0.85	0.89	0.85	0.82	0.83

Table 2.4: Average number of chunks (i.e., number of map tasks) for the seven sequences in RNA2 and in RNA1 for fine-grained mapping using centered and optimized methods.

Mean Total	RNA2(∼1300 bases)			RNA1(∼3100 bases)		
Time (sec)	60	150	300	60	150	300
Centered	395	258	155	939	622	383
Optimized	355	218	134	825	525	312
Opti./Cent.	0.90	0.84	0.86	0.88	0.84	0.81

and shorter MapReduce total times.

Coarse- vs. fine-grained mapping: We next discuss the performance of our method by using the coarse- or the fine-grained mapping on the two groups of sequences from the virus family *Nodaviridae*. We can see from Figures 2.15(a) and 2.15(b) that coarse-grained mapping results in shorter execution time compared with fine-grained mapping, as shown in Figure 2.15(c) and 2.15(d), independently of the chunking method used. Also we observe the trend that when the max chunk length grows from 60 to 300, the time gain of coarse-grained mapping over fine-grained mapping decreases. The speedup of coarse-grained mapping over fine-grained mapping using the centered chunking method for RNA2 subgroup of sequences decreases from 3.75 to 1.38, and for RNA1 it decreases from 7.86 to 2.37. A similar behavior is observed for the optimized chunking method: speedup of coarse-grained mapping over fine-grained mapping for RNA2 subgroup of sequences decreases from 3.4 to 1.25, and for RNA1 it decreases from 6.82 to 1.93.

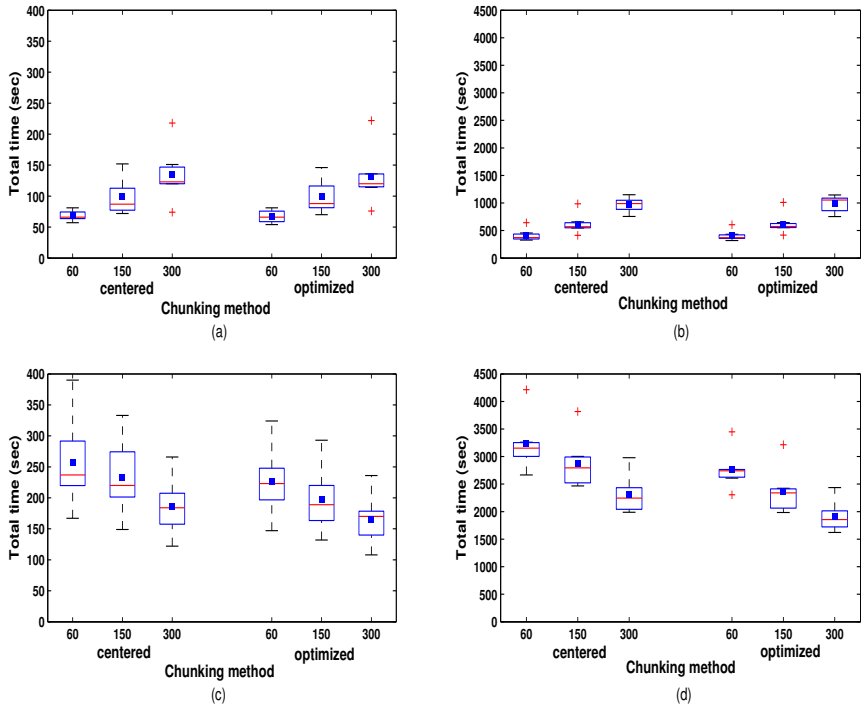


Figure 2.16: Box-and-whisker diagram of total MapReduce times for each subgroup of sequences (RNA2 and RNA1) using centered and optimized methods as well as max chunk length of 60, 150, and 300: (a) coarse-grained mapping on RNA2 (short sequences), (b) coarse-grained mapping on RNA1 (long sequences), (c) fine-grained mapping on RNA2 (short sequences), and (d) fine-grained mapping on RNA1 (long sequences).

The same trend of the total times is summarized in the box-and-whisker diagram of min, median, mean, and max execution time for each subgroup of sequences (RNA2 and RNA1) in Figure 2.16. More specifically, in Figure 2.16(a), we show the box-and-whisker diagram of the total times for the RNA2 subgroup of sequences (∼1300 nucleotides bases) using coarse-grained mapping, both centered and optimized methods, and max chunk lengths of 60, 150, and 300. In Figure 2.16(b), we show a similar box-and-whisker diagram but for the RNA1 subgroup of sequences (∼3100 bases). In Figure 2.16(c), we show the box-and-whisker diagram of the min, mean, max execution time for the RNA2 subgroup of sequences using fine-grained mapping, centered and

optimized methods and max chunk lengths of 60, 150, and 300. In Figure 2.16(d), we show a similar box- and-whisker diagram but for the RNA1 subgroup of sequences. We observe that for coarse-grained mapping, when using the centered and optimized methods, the average total times increase with the max chunk length at the rate of 2.0 for RNA2 and 2.4 for RNA1. On the contrary, for the fine-grained mapping, when using centered and optimized methods, the average total times decrease with the max chunk length at the rate of 0.7 for both subgroup of sequences. These results suggest that for larger max chunk length and sequence lengths, the fine-grained mapping potentially could outperform the coarse-grained mapping.

Map efficiency and load imbalance: When decoupling the total time in its components, we observe that the times for the reduce function and shuffling are marginal compared with the times used for the mapping functions (around 1% of the total time). We also observe that as a prediction tree is explored, some mappers are performing more work than others, resulting in idle time and low efficiency. The load imbalance among mappers depends on the granularity and chunking methods used. To better understand the causes of load imbalance, we reduce the mapping times into compute time (i.e., chunking and predictions) and idle time (i.e., waiting for all the mappers to complete their predictions). Figure 2.17(a) and Figure 2.17(b) show the percentages for compute and idle times for the coarse-grained framework with centered and optimized methods, respectively; and Figure 2.17(c) and Figure 2.17(d) show the same percentages but for the fine-grained framework and the two chunking methods.

Independently from the max chunk length, Figure 2.17 shows how fine-grained mapping reaches better efficiency compared with coarse-grained mapping. In other words, with fine-grained mapping, the mappers spend more time doing real chunking and predictions. We observe in Figure 2.15 (right y-axes) that fine-grained mapping has a larger number of map tasks and that each map task is shorter (it predicts only one chunk), making it easier for the Hadoop scheduler to allocate the several tasks efficiently by using a first-in-first-out policy. On the other hand, coarse-grained mapping has a smaller number of map tasks, and each map task is longer (all the sequence chunks

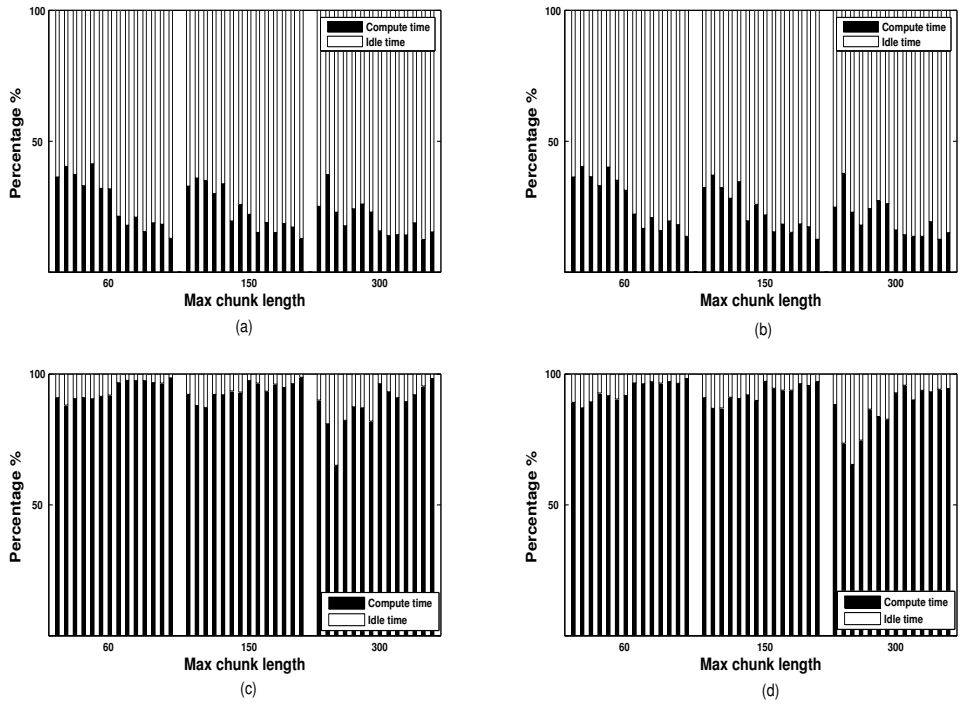


Figure 2.17: Percentages of compute and idle time in map function for coarse- vs. fine-grained mapping and centered vs. optimized methods: (a) coarse-grained mapping using centered method, (b) coarse-grained mapping using optimized method, (c) fine-grained mapping using centered method, and (d) fine-grained mapping using optimized method.

of a given L and G combination are predicted by a single mapper). In this case, once the scheduler assigns a longer task to a mapper, it has to wait for its completion, even if the other mappers have generated their chunk predictions, before proceeding to the reduce phase. We also observe that as the max chunk length increases from 60 to 300 bases, the map efficiency tends to drop. More specifically, the average map efficiency for coarse-grained mapping on RNA2 decreases from 36% to 25% and from 18% to 15% on RNA1 when using centered or optimized chunking methods. The average map efficiency for fine-grained mapping on RNA2 decreases from 91% to 79% and from 97% to 93% on RNA1. The reason is that when using a max chunk length of 60, the centered and optimized chunking methods tend to produce more chunks with shorter

chunk lengths. On the other hand, when using a max chunk 300, the same methods tend to produce fewer chunks each with longer lengths.

Diverse chunk lengths within a prediction can also cause inefficiency. To study this phenomenon, we consider the *Nodamura* virus (NoV) RNA2 sequence, which in Figure 2.17 shows the largest drop in efficiency when moving from 60 to 300 max chunk length. Figure 2.18 and Figure 2.19 show the number of chunks and their lengths (i.e., max, min and median) for the different L and G parameter combinations with centered and optimized methods when the max chunk length is equal to 60 and when the length is equal to 300, respectively. When the maximum length grows from 60 to 300, the number of resulting chunks for each combination of L and G parameters decreases. At the same time the length of each set of chunks increases as well as the length variability within the set of chunks for a defined combination of L and G values. Note that for some combinations of L and G, the chunking process does not identify any set of chunks, and we do not report any result for these cases. This situation confirms our observation that as the number of chunks decreases, the chunk lengths increase but not homogeneously within a prediction, causing load imbalance and loss in efficiency. Selecting the shorter max length for the sake of efficiency is not always a wise decision: a max chunk length of 60 bases may be too short for the type of RNA sequences we are considering. In Figure 2.18, for example, the median is close to the max length of 60 for the centered methods, indicating that we are cutting out valuable parts of the inversion and ultimately of the secondary structures we are predicting.

The overall results suggest that the best set of parameter values to achieve higher performance depends on multiple aspects, including the targeted sequence and the different chunking method and mapping used.

2.5 Accuracy

2.5.1 Platform

The accuracy tests are run on the same platform as the performance tests (as described in Section 2.4.1).

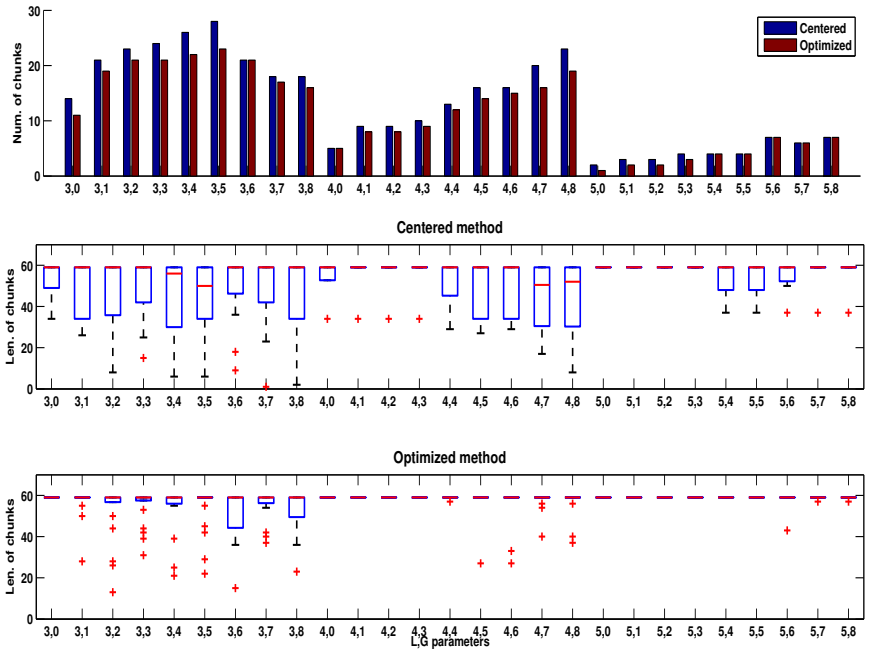


Figure 2.18: Number of chunks and chunk lengths for the *Nodamura* virus (NoV) RNA2 with centered and optimized methods and maximum chunk length of 60 bases.

2.5.2 Datasets

To study the framework accuracy, we use two datasets of sequences for which the secondary structures have been previously established. The first dataset consists of 50 non-pseudoknotted sequences from the RFAM database, and the lengths of sequences range from 127 to 568 bases. The second dataset consists of 23 pseudoknotted sequences from the RFAM and Pseudobase++ [21,29] databases, and the lengths of the sequences in this dataset range from 77 to 451 bases. Note that there are no large datasets of experimentally determined RNA secondary structures including pseudoknots, and to the best of our knowledge the one used in this thesis is one of the few available to the public for free.

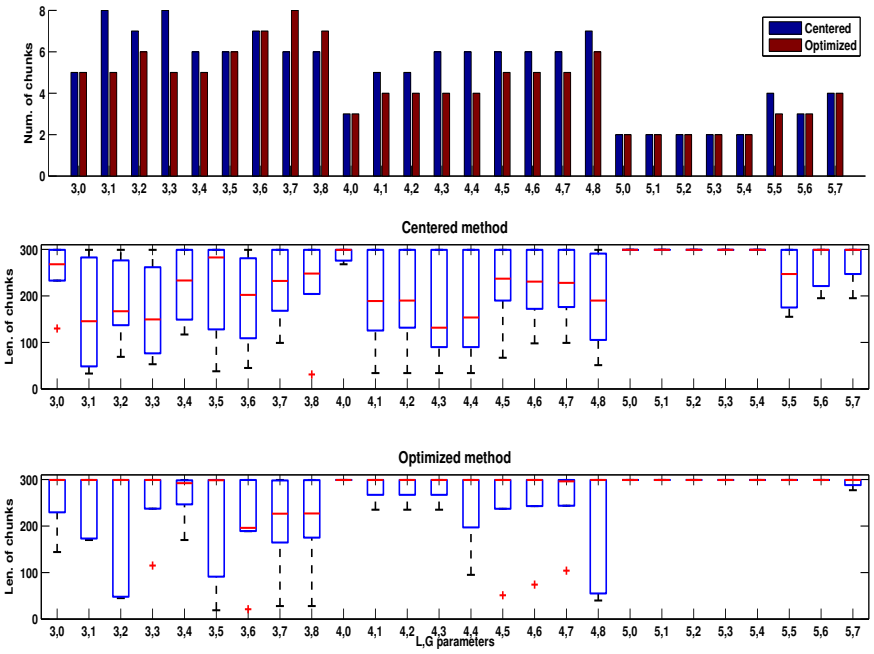


Figure 2.19: Number of chunks and chunk lengths for the *Nodamura* virus (NoV) RNA2 with centered and optimized methods and maximum chunk length of 300 bases.

2.5.3 Results and discussion

Here we compare the secondary structures obtained by our chunk-based method with the known secondary structures (AC) and with the secondary structures obtained by using the whole sequences (AR). Moreover, we examine the accuracy of our method when the length of the sequence increases. In addition, we discuss the accuracy of the centered and optimized chunking methods.

Chunk-based prediction vs. known structures (AC): To assess how well the predictions based on chunking agree with known RNA structures, we measure the maximum AC (MAC) values of the sequences in the two datasets when the maximum chunk length C_{max} ranges from 60 to 150 bases (incremental 10 bases each time), L_{min} ranges

from 3 to 8, and G_{max} ranges from 3 to 8. Figures 2.20(a), (b), and (c) show the box-and-whisker diagram for the regular, centered, and optimized methods, respectively, for the dataset of 50 non-pseudoknotted sequences. Figures 2.21 (a), (b), and (c) show the box-and-whisker diagram for the regular, centered, and optimized methods, respectively, for the dataset of 23 pseudoknotted sequences. In the figures, the lower and upper quartiles are at the top and bottom boundaries of the box for the kernels; the median is the band inside the box; the mean is the black square; the whiskers extend to the most extreme data points or outliers; and outliers are plotted individually as "+" symbols.

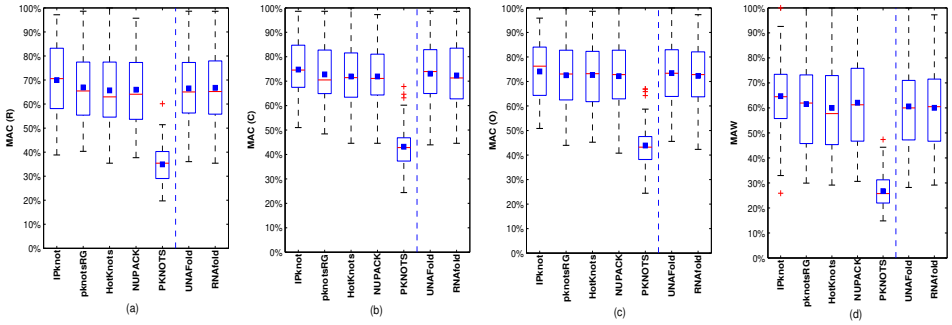


Figure 2.20: MAC and MAW values obtained using the prediction programs IPknot, pknotsRG, HotKnots, NUPACK, PKNOTS, RNAfold, and UNAFOLD for the dataset of 50 non-pseudoknotted sequences.

As described in Section 2.3, the AC value for a predicted RNA structure is the percentage of agreement between the known structure and the structure obtained by concatenating the predicted structures of the chunks. Likewise, the AW value is the percentage of agreement between the known structure and the predicted structure when the whole sequence is used. These values indicate how closely the predicted structure resembles the real structure. A larger AC value means that the chunk-based predicted structure is more similar to the real structure. For a given dataset, prediction program, and chunking method, our MapReduce framework collects multiple predicted structures associated with different C_{max}, L_{min}, and G_{max} parameters. The MAC value for a sequence gives the highest accuracy that can be attained for that sequence by the

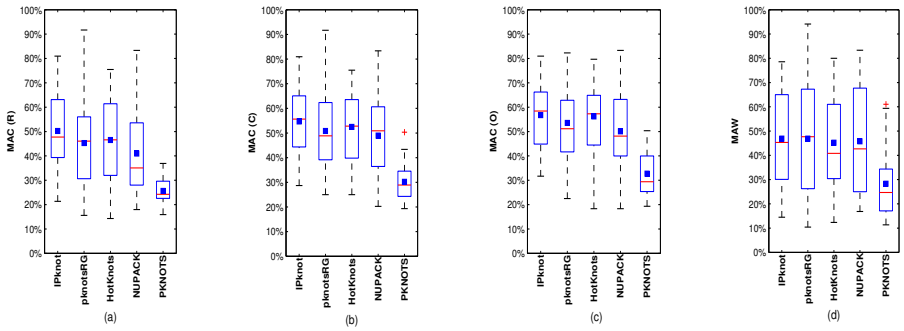


Figure 2.21: MAC and MAW values obtained using the prediction programs IPknot, pknotsRG, HotKnots, NUPACK, and PKNOTS for the dataset of 23 pseudoknotted sequences.

chunking method and the specific prediction program employed. Figure 2.20(d) and Figure 2.21(d) show the AW values of the sequences in the two datasets, respectively. From these figures, one can see that most of the prediction methods have similar accuracy ranges regardless of the chunking method used and whether the prediction was obtained with the whole sequence or with the chunks; however, the PKNOTS program produces somewhat lower accuracies. This lower accuracy is expected because PKNOTS is the earliest algorithm allowing for pseudoknot prediction. The other prediction programs with pseudoknot prediction capability that have developed afterwards have incorporated improvements over the original PKNOTS.

Chunk-based vs. non-chunk-based sequential programs predictions (AR): From Figures 2.20 and 2.21, the prediction accuracies with chunking (MAC values in (a) (c)) appear to be higher than those without (AW values in (d)), suggesting that the prediction accuracy, on average, can be enhanced by sequence segmentation. To get a clearer characterization of the effect of sequence segmentation, we carry out statistical tests on the maximum accuracy retention (MAR) obtained for each RNA sequence over the C_{max}, L_{min}, and G_{max} parameters. In the majority of the sequences in our dataset, the MAR turns out to be greater than 1. With a one-sample t-test, we test whether the mean MAR is significantly greater than 1 with p-value < 0.05. Tables 2.5 and 2.6 display the means, standard deviations, and p-values for the non-pseudoknotted and

pseudoknotted sequences respectively.

Table 2.5: Mean and standard deviations of MAR for regular, centered, and optimized chunking methods over 50 non-pseudoknotted sequences, and the corresponding p-values of the t-test for mean MAR > 1.

Cut	Regular			Centered			Optimized		
Prediction	Mean	Stdev	p	Mean	Stdev	p	Mean	Stdev	p
IPknot	1.13	0.32	0.002	1.23	0.36	0.000	1.21	0.36	0.000
pknotsRG	1.19	0.50	0.005	1.27	0.49	0.000	1.27	0.47	0.000
HotKnots	1.19	0.48	0.003	1.32	0.50	0.000	1.33	0.50	0.000
NUPACK	1.12	0.34	0.010	1.23	0.41	0.000	1.24	0.41	0.000
PKNOTS	1.33	0.19	0.000	1.65	0.35	0.000	1.70	0.35	0.000
UNAFold	1.19	0.49	0.003	1.31	0.47	0.000	1.31	0.46	0.000
RNAfold	1.19	0.46	0.002	1.31	0.48	0.000	1.30	0.45	0.000

Table 2.6: Mean and standard deviations of MAR for regular, centered, and optimized chunking methods over 23 pseudoknotted sequences, and the corresponding p-values of the t-test for mean MAR > 1.

Cut	Regular			Centered			Optimized		
Prediction	Mean	Stdev	p	Mean	Stdev	p	Mean	Stdev	p
IPknot	1.19	0.48	0.037	1.33	0.62	0.009	1.40	0.64	0.004
pknotsRG	1.21	0.84	0.116	1.39	0.99	0.036	1.48	1.00	0.016
HotKnots	1.11	0.41	0.098	1.32	0.71	0.021	1.43	0.80	0.009
NUPACK	0.93	0.18	0.955	1.14	0.39	0.071	1.17	0.35	0.032
PKNOTS	1.16	0.20	0.003	1.29	0.26	0.000	1.38	0.29	0.000

For non-pseudoknotted sequences, the mean MAR is significantly greater than 1 for all three chunking methods, whereas the mean MAR values for the pseudoknotted sequences are greater than 1 for the centered and optimized chunking methods. With the regular chunking method, one of the mean MAR values (with NUPACK) falls below 1 to 0.93. Looking at all the p-values, we can conclude that the average prediction accuracy attained with segmentation is not significantly less than that without. With the inversion-based centered and optimized chunking methods, we can conclude that the average prediction accuracies attained with segmentation are at least as good as, and often even better than, those without segmentation.

Shorter length vs. longer length sequences (ranging from 77 to 568): While the above results show that sequence segmentation does not reduce prediction accuracy on average, we still need to examine whether the MAR values decline as the whole sequence length grows, because a declining trend implies that the accuracy retention deteriorates when the segmentation approaches are applied to longer RNA sequences. To this end, for each dataset, chunking method, and prediction program, we perform the Pearson correlation analysis on the MAR values of the sequences [73]. For each dataset, we report both the correlation coefficient r and corresponding p-value between MAR and sequence length. If the r value is close to -1, it means that MAR and sequence lengths are negatively correlated, implying a decline in accuracy retention of the chunking method. If the associated p value is less than 0.05, we consider the correlation statistically significant; otherwise the correlation is not significant.

Figure 2.22 is a scatter plot of MAR values versus sequence lengths for one of the prediction programs, IPknot. Similar scatter plots for the other prediction programs have also been examined, and no statistically significant negative correlation has been detected in any of these plots. Table 2.7 presents the correlation coefficients r and their corresponding p-values when the null hypothesis of no correlation is tested against the alternative hypothesis of having a negative correlation. These p-values indicate that no significant negative correlation has been detected for any of the prediction programs and chunking methods. We therefore do not expect any substantial decline in accuracy retention of our chunking methods while sequence length increases.

Centered vs. optimized chunking method: Given the three chunking methods considered (i.e., regular, centered, and optimized) we also want to determine which among them is better at retaining the accuracies of the various prediction programs. For this purpose, we examine each sequence in our two datasets and keep track of which chunking method produces the highest MAR. Table 2.8 gives the total counts of sequences attaining the highest MAR for each of the chunking methods. If more than one chunking method get the same highest MAR for one sequence, we split the count of this sequence equally among the methods. We can see that the sequence counts in

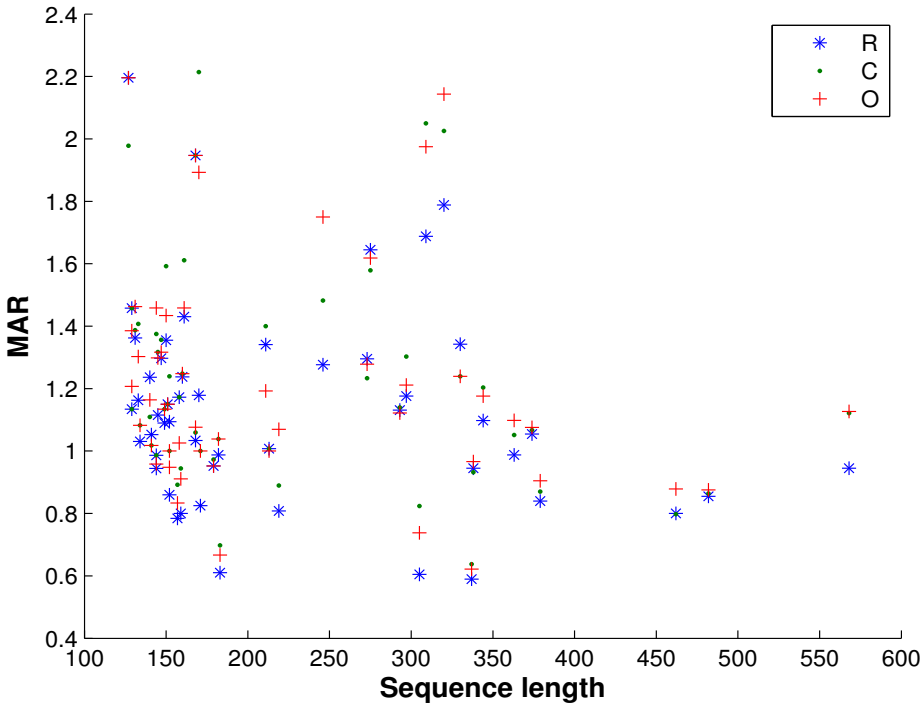


Figure 2.22: Scatter plot of MAR values versus sequence lengths for the IPknot program. Similar scatter plots for the other prediction programs have been examined, and no statistically significant negative correlation has been detected in any of these plots.

Table 2.8 for the centered and optimized (C and O) methods are higher than those for the regular method (R).

To see whether any differences exist among the accuracy retention capabilities among the three cutting methods, we perform the Friedman test for each dataset and each prediction program. The Friedman test is a non-parametric statistical test based on rank sums and requires ranking the MAR attained by each chunking method for each prediction program and each sequence in our datasets. The method producing the lowest MAR is given a rank of 1, and the method producing the highest MAR is given a rank of 3. Again, the ranks are averaged for ties. Table 2.9 shows the p-values of the Friedman tests in the "R-C-O" columns. From these very low p-values, we can conclude that significant differences do exist among the three methods.

Table 2.7: Correlation coefficients (r) between MAR and sequence lengths and corresponding p-values (p) when testing for a negative correlation.

		Non-Pseudoknotted Sequences			Pseudoknotted Sequences		
Prediction		r	C	O	r	C	O
IPknot	r	-0.2077	-0.2141	-0.1629	0.1379	0.1572	0.0841
	p	0.1478	0.1354	0.2582	0.5303	0.4738	0.7028
pknotsRG	r	-0.1550	-0.0670	-0.0756	0.2030	0.1971	0.1987
	p	0.2825	0.6437	0.6018	0.3528	0.3673	0.3634
HotKnots	r	-0.1434	-0.0476	-0.0732	-0.0360	-0.0669	-0.0855
	p	0.3204	0.7428	0.6136	0.8705	0.7618	0.6982
NUPACK	r	-0.1622	-0.0137	-0.0059	-0.0532	0.0666	-0.1331
	p	0.2604	0.9249	0.9676	0.8340	0.7928	0.5986
PKNOTS	r	0.4598	0.7053	0.7045	-0.0233	0.1001	0.0597
	p	0.0008	0.0000	0.0000	0.9294	0.7023	0.8199
UNAFold	r	-0.1449	-0.1055	-0.1311			
	p	0.3155	0.4658	0.3643			
RNAfold	r	-0.1056	-0.0646	-0.0538			
	p	0.4654	0.6559	0.7104			

Because the Friedman test does not reveal whether any one method is significantly better than another, we also perform the post hoc pairwise comparison test on each pair of the three chunking methods in order to confirm that the inversion based centered and optimized chunking methods are indeed superior to the naive regular method. The p-values, shown in the "R-C," "R-O," and "C-O" columns, indicate that both the centered and optimized methods are better than the regular method. Furthermore, the centered and optimized methods show no significant differences except when PKNOTS is applied to the pseudoknotted sequences.

These results demonstrate that for a variety of secondary structure prediction programs, our segmentation approach for handling the long RNA sequences can retain and even enhance the average prediction accuracy. Furthermore, using the inversion-based centered and optimized methods to cut the sequence produces better prediction accuracy than the naive r method does.

2.6 Summary and Future Work

Summary: This chapter presents the application of our general data analysis method to the classification problem in which we identify an RNA secondary structure as more or less likely to occur based on multiple chunk-based secondary structures. We discuss the workflow of mapping RNA sequences to chunk-based secondary structures (i.e., metadata) using inversion information, the classification of whole secondary structures into more or less likely to occur based on the accuracy retention, the integration of the workflow into the Hadoop framework, and the different granularity of the tree search in the Hadoop framework.

We evaluate the performance of our method on 8 nodes of the Geronimo cluster using the RFAM database and the virus family *Nodaviridae* with sequence lengths varying from 90 to 3,204 nucleotide bases. Results show that as the lengths of the sequences increase, our method achieves constant execution time whereas the execution times for traditional prediction methods grow exponentially or polynomially. In addition, the traditional prediction methods often fail on long RNA sequences because of memory limitations.

Moreover, we evaluate the accuracy of our chunk-based framework using two datasets from RFAM and Pseudobase++ database. Results show that the prediction accuracy, on average, is enhanced by our framework. In particular, the accuracy of our framework is significantly higher than that of the traditional method in 35 of 36 experiments. In the best case, our framework achieves 1.7 times higher accuracy compared with that of the traditional method using PKNOTS on the 50 non-pseudoknotted sequences.

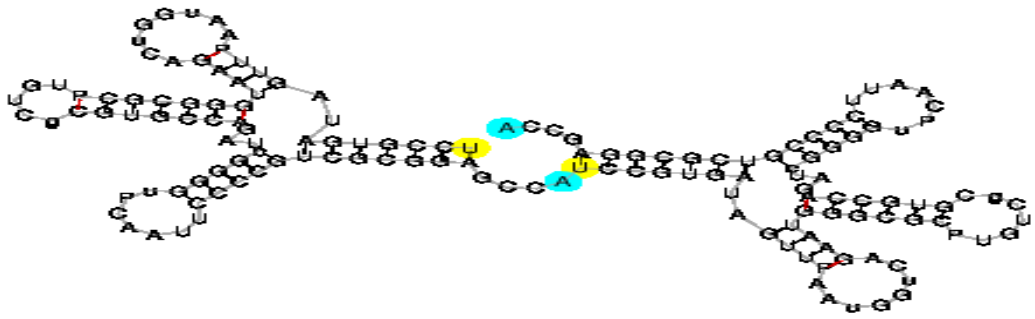
Future work: Future work lies in two directions: (1) further investigate how to cut the RNA sequence into chunks in a way that avoids cutting in long secondary structures, and (2) study how to classify the resulting secondary structures into those more or less likely to happen without referring to the known structure.

For the first research direction, a possible approach is to cut the RNA sequence into chunks such that long secondary structures are not cut in the middle. We refer to

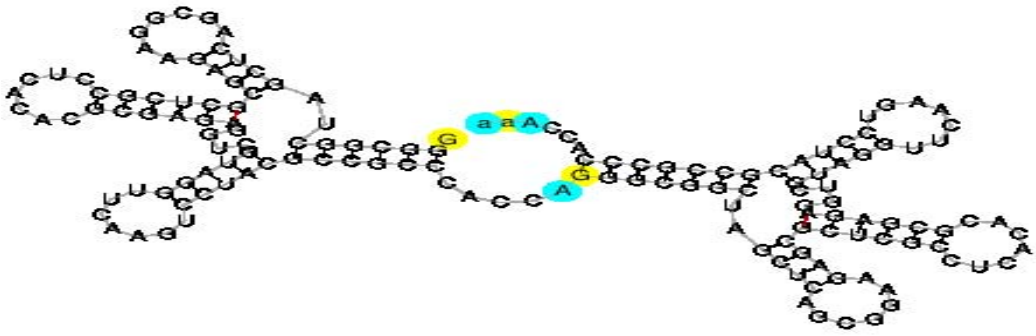
this proposed approach as the mega-chunk method. To this end, a possible strategy is to extend the chunking approach such that an RNA sequence is cut into multiple sets of chunks and the chunks in each set can have varying lengths and can contain inversions with larger gaps in between. The method will have to first identify all the inversions in the sequence, including the ones with large gaps, and then it will have to remove redundant inversions that are nested in longer inversions. All the possible cutting points that cut the sequence around inversions will have to be considered before cutting the sequence into sets of chunks (i.e., a brute-force exploration of all possible sets of chunks). In this way, we should be able to eliminate the threshold parameters in the current chunking methods (i.e., maximum chunk length C_{max}, minimum stem length L_{min}, and maximum gap length G_{max}). Because of allowing larger gaps, we should also be able to capture potential secondary structures, including pseudoknots that engage nucleotides in different regions of the sequence.

For the second research direction, we envision an approach capable of classifying resulting secondary structures into those more or less likely to occur in nature. To this end, the training of a family model using the known secondary structures of other sequences in the given family will be needed, and then the assignment of a score to each resulting structure of the sequence using the family model can naturally complete the classification process. Inspired by research in protein structure predictions, we hypothesize that the sequences in the same family share structural similarities. The proposed research direction can benefit from this feature to design, develop, and evaluate a method to classify new secondary structures. Figure 2.23 shows a case study for this scenario in which the known secondary structures for two RNA sequences PDB_00307 and PDB_00421 that belong to the same RNA family (transfer RNA) from the database RNA STRAND exhibit structural similarities [71]. We observe that although the two sequences have different lengths and nucleotide bases, their secondary structures share similarities. For example, they both have six hairpin loop structures and three multi-branched loops. Both descriptive statistical information and hidden Markov model (HMM) can be used to build the RNA family model and classify resulting secondary

structures.



(a)



(b)

Figure 2.23: RNA structures from the same family: (a) PDB_00307 and (b) PDB_00421.

Table 2.8: Count (co) and rank sum (rs) of sequences attaining the highest MAR with each chunking method for the various prediction programs.

	Non-Pseudoknotted Sequences						Pseudoknotted sequences					
Prediction	R-co	R-rs	C-co	C-rs	O-co	O-rs	R-co	R-rs	C-co	C-rs	O-co	O-rs
IPknot	7.7	74.0	24.7	115.5	17.7	110.5	2.0	32.0	7.0	47.0	14.0	59.0
pknotsRG	6.3	71.5	22.3	115.5	21.3	113.0	2.0	30.0	7.0	50.0	14.0	58.0
HotKnots	4.0	68.0	22.0	115.5	24.0	116.5	1.0	32.0	6.0	47.0	16.0	59.0
NUPACK	4.0	68.5	17.0	115.5	29.0	120.0	1.3	24.5	7.3	40.5	9.3	43.0
PKNOTS	1.0	55.5	17.5	114.5	31.5	130.0	1.5	21.0	4.0	35.5	11.5	45.5
UNAFold	4.0	67.0	25.0	118.0	21.0	115.0						
RNAfold	4.0	65.0	18.0	112.5	28.0	122.5						

Table 2.9: P-values from the Friedman test to compare the accuracy retention of the three chunking methods as well as the posthoc pairwise comparison tests.

	Non-Pseudoknotted Sequences				Pseudoknotted Sequences			
Prediction	R-C-O	C-O	R-C	R-O	R-C-O	C-O	R-C	R-O
IPknot	9.21E-06	5.27E-01	2.43E-05	4.02E-04	2.16E-05	4.55E-02	4.50E-03	3.74E-05
pknotsRG	1.82E-07	5.27E-01	1.86E-06	9.72E-06	3.65E-06	1.84E-02	1.08E-04	9.62E-05
HotKnots	8.80E-09	6.47E-01	4.20E-07	5.36E-07	2.47E-05	1.84E-02	1.30E-03	1.62E-04
NUPACK	5.16E-09	2.17E-01	5.79E-08	1.41E-06	3.70E-04	5.64E-01	9.11E-04	1.30E-03
PKNOTS	1.22E-15	2.69E-02	3.56E-10	1.18E-11	6.78E-06	6.70E-03	5.32E-04	1.83E-04
UNAFold	5.25E-09	2.74E-01	6.91E-07	2.96E-07				
RNAfold	1.18E-08	8.82E-01	1.29E-06	1.81E-07				

Chapter 3

CLUSTERING OF LIGAND GEOMETRIES

In studies of disease processes, a common problem is to search for small molecules (ligands) that can interact with a larger molecule such as a protein when the protein is involved in a disease state. More specifically, when ligands dock well in a protein, they can potentially be used as a drug to stop or prevent diseases associated with the protein's malfunction. The study of the docking process is computationally performed with docking simulations, which consist of sequences of independent docking trials. Since the process is highly parallelizable, it is efficiently performed on distributed memory systems (e.g., supercomputers and volunteer computing platforms), resulting in a distributed collection of hundreds of thousands of ligand conformations across the nodes of the system.

A second analysis problem, also tackled in this thesis, is a clustering problem in which we identify the geometries of ligand conformations and predict their probabilities of well-docking into a protein pocket based on their geometries in a distributed way, without moving the ligand conformations to a central server. We concurrently extract the geometry of a ligand conformation into metadata by performing the linear regression analysis on the ligand's atoms in which the coordinates are projected on three planes and interpolated in three different ways resulting in 3-D or 6-D metadata. The clustering is performed by an N-dimensional (N-D) clustering algorithm that searches for the densest subspace of metadata in two different variations. This subspace is expected to contain well-docked ligands. We integrate the metadata extraction and N-D clustering algorithm into the MapReduce programming model, and we study both the performance and the accuracy of the MapReduce framework for several structural biology datasets of up to several millions of ligand molecules.

The rest of the chapter is organized as follows. Section 3.1 gives the background information on protein-ligand docking simulations, the sampling process of the Docking@Home datasets used in the accuracy study, the distributed memory systems simulated in performance study, and the MapReduce-MPI framework. Section 3.2 reviews the related work on the centralized clustering of ligand conformations and the distributed clustering using MapReduce. Section 3.3 presents our methodology. Sections 3.4 and 3.5 present the performance and accuracy evaluations, respectively. Section 3.6 summarizes our research results and briefly discusses future work.

3.1 Background

3.1.1 Protein-ligand docking

Computationally, a protein-ligand docking search seeks to find near-native ligand conformations in a large dataset of conformations that are docked in a protein [41]. A conformation is considered near-native if the root-mean-square deviation (RMSD) of the heavy atom coordinates is smaller than or equal to two angstroms (Å) from the experimentally observed conformation. Algorithmically, a docking simulation consists of a sequence of independent docking trials. An independent docking trial starts by generating a series of random initial ligand conformations; each conformation is given multiple random orientations. The resulting conformations are docked into the protein-binding site. Hundreds of thousands of docking attempts are performed concurrently. Molecular dynamics simulated annealing is used to search for low energy conformations of the ligand on the protein pocket. Traditionally, docked conformations with minimum energy are assumed to be near native. Research has shown, however, that this is not always the case [28]. Docked conformations with minimum energy are not necessarily near native.

In previous work, we showed that compact clusters of docked conformations grouped by their geometries are more likely to be near native than are the individual conformations with lowest energy [28, 29]. Large numbers of ligand conformations were sampled through the Docking@Home (D@H) project in the past five years and are used

in this thesis as the dataset. Hundreds of millions of docked ligand conformations have to be compared with one another in terms of their geometries.

The docking process is only one of the key steps; once the results (ligand conformations) are collected, they need to be evaluated to predict the near-native ligand geometry. Selecting the near-native ligand geometry based on energy alone may result in incorrect conclusions. An alternative approach is to select the near-native geometry from clustering, but this approach can result in extensive computing and storage needs. Ideally the clustering methods have to be scalable, efficient, and accurate, allowing scientists to compare and select across a very large set of docking results.

3.1.2 Sampling conformational spaces with Docking@Home

Many computational molecular docking approaches for sampling large conformational spaces of ligands have been used for virtual screening [60, 2, 55]. Typically a docking method is evaluated with a selected number of experimentally determined protein-ligand complexes. In general, various docking methods differ from one another in the algorithm used in the conformational search [13, 57], the scoring function used to predict ligand geometries, and the scoring function used to rank compounds or to predict DGbinding [30].

Although the system software for sampling large conformation spaces is not the key contribution of this thesis, here we briefly describe how we collect the docking data for this thesis using volunteer computing (VC). VC is a well-established programming model for scientific projects. It is a form of distributed computing in which ordinary people volunteer processing and storage resources across the Internet to scientific simulations. The growing volunteer computing community includes 65 scientific projects, over 2 million volunteers, and 6 million computers distributed across the world. Berkeley Open Infrastructure for Network Computing (BOINC) [4], a well-known VC middleware, supports Docking@Home (D@H) version 2, the VC project producing the data used in this thesis. D@H is an NSF-funded project in molecular

docking that computationally searches for potential drug-like molecules against diseases, such as breast cancer and HIV. D@H generates a very large space of possible docking conformations. In order to extensively search this space, millions of independent docking attempts (jobs) are processed by the D@H server that distributes them for computation to clients across the Internet. Volunteers' computers perform the docking simulation and return results, consisting of the docked 3-D ligand conformation and its associated energy value. Currently, more than 89,000 volunteers and 188,000 computers worldwide support D@H.

To explore the conformational space of the ligands, D@H considers a representation of the solvent by using two docking methods: (1) a implicit representation of water using a distance-dependent dielectric coefficient (low if the atoms are close and progressively larger as the interatomic distance increases) and (2) a more physically accurate implicit representation of water using a generalized Born model [46]. The method based on the generalized Born model is a more compute- and memory-intensive method. At the same time it provides a more physically accurate description of the potential energy of a ligand where part of the ligand conformation is exposed to solvent. In many situations where a large portion of the ligand is solvent exposed, the generalized Born model should help significantly in providing better ligand conformations (e.g., when one orientation of a given ligand leaves a large bulky hydrophobic group exposed to solvent, this is penalized, where exposing a hydrophilic group such as a hydroxyl group to solvent is much more favorable).

The molecular docking is performed by using the CHARMM (Chemistry at HARvard Molecular Mechanics) molecular simulation package [11] and an intermediate-accuracy all-atom force field. The CHARMM script describing the docking process considers a protein-ligand complex as a composition of a flexible ligand and a rigid protein structure (i.e., on a three-dimensional lattice of regularly spaced points surrounding and centered on the active site of the protein, where each point on the grid stores the potential energy of a "probe" atom's interaction with the molecule). A D@H simulation consists of a sequence of independent trials. For each trial, either a

randomly generated conformation or a user-defined conformation for a ligand is used as the initial conformation. Random conformations are generated starting from the ligand crystal structure with random initial velocities on each ligand atom. Then the initial conformation is randomly rotated to produce a set of different orientations that are placed into the active site of the protein or docking pocket (docking attempts).

Once the ligand is docked into the protein site, a molecular dynamics simulation is performed consisting of a gradual heating phase of 4,000 1-femtosecond (1 fs) steps from 300K to 700K, followed by a cooling phase of 10,000 1 fs steps back to 300K. In order to facilitate the penetration of ligands into protein sites and to allow larger conformational changes, van der Waals (vdW) and electrostatic potentials with soft-core repulsions are utilized. A soft-core repulsion reduces the potential barrier at vanishing interatomic distances to a finite limit, allowing ligands to pass between conformational minima with a relatively small potential barrier that normally is very large and impossible to overcome with an unmodified standard potential. The detailed description of the docking method and its comparison with other docking codes is not in the scope of this thesis; this information can be found in past work [75, 74]. Once the results (ligand conformations) are collected, they need to be scored. Initially D@H used an energy-based scoring method; our previous work showed how this scoring approach can result in incorrect conclusions, however, because energy values are approximated by the simplified methods used in the computational algorithms. The alternative approach that we pursue in this thesis is to score ligands based on the geometry of their resulting conformations.

3.1.3 Semi-decentralized and fully-decentralized systems

Data considered in this thesis comprises a large number of individual data records and is distributed across the nodes of a large distributed memory system. More specifically, we consider semi-decentralized and fully-decentralized distributed memory systems. In a semi-decentralized system, processes report to more than one node, usually the closest one, and take advantage of locality by reducing expensive data transfer

and potential storage pressure. In contrast, in a fully-decentralized system, each node stores its own data, thus reducing the need for data transfers and increasing the amount of locally stored data. Logically, the entire dataset can converge toward one or multiple scientific properties, or it may not convey any scientific information. Physically, data with similar scientific properties may agglomerate in topologically close nodes or may be dispersed across nodes. Logical and physical tendencies are not known a priori when data is generated in semi- or fully-decentralized systems. In general, scientists must move data across nodes in order to analyze and understand it. This process can be extremely costly in terms of execution time. For the sake of completeness, we define three scenarios that resemble the challenging conditions faced by scientists when dealing with distributed systems with large amounts of data. The scenarios consist of data distributed as follows:

- A semi-decentralized manner in which data with similar properties is generated by and stored in specific nodes.
- A fully-decentralized, synchronous manner in which data is gathered at regular intervals producing a uniform distribution of data properties across the nodes in a round-robin fashion.
- A fully-decentralized, asynchronous manner in which every node acts by itself and properties are stored randomly across nodes.

3.1.4 MapReduce-MPI

MapReduce-MPI is a runtime library supporting the MapReduce programming model [59]. It is written in C++ and MPI. It runs a MapReduce program using MPI processes, the number of which is defined by the programmer. Each process runs both the map and the reduce functions. It first runs the map function on its partial data and outputs the intermediate $\langle key, value \rangle$ pairs in parallel. Then, the MapReduce-MPI framework communicates all the values with the same key across the distributed memory system to the same process. After this data shuffle stage, each process runs the reduce function and generates the final output.

Compared with Hadoop, MapReduce-MPI is more flexible when structuring computation: the programmer can specify any combination of map and reduce functions, including multiple map functions followed by one reduce, one map followed by multiple reduce functions, and multiple map followed by multiple reduce functions. In Hadoop, on the other hand, iterations must be expressed as a chain of MapReduce jobs. In addition, MapReduce-MPI eliminates the use of HDFS for data input and output by using the Lustre file system or the local disk of each node in the distributed memory system directly. In other words, it eliminates the time-consuming data-staging phases required in Hadoop. Moreover, it runs on any platform that supports MPI and C++, as most distributed memory systems including supercomputers do. In comparison, Hadoop requires installing HDFS and Hadoop MapReduce components on the system. MapReduce-MPI also utilizes the high-speed InfiniBand for data communication by default.

3.2 Limits of Current Practice

3.2.1 Centralized clustering of docking conformations

Traditionally, a naïve approach used to group similar structural biology conformations is through geometry-based clustering. Important work in this direction includes that of Lorenzen et al. [49], Bouvier et al. [9], Chang et al. [16], and Estrada et al. [27]. Lorenzen et al. [49] select near-native docking conformations by assuming that a bigger cluster is more likely to have better candidate conformations. Bouvier et al. [9] use a Kohonen self-organizing map that is trained in a preliminary phase by using drug-protein contact descriptors. Chang et al. [16] perform a simple cluster analysis for docking simulations and use the size of the clusters to estimate the vibrational entropy of the resulting conformations. Estrada et al. [27] identify near-native ligand conformations using a probabilistic hierarchical clustering and fuzzy c-means.

Such a technique requires that data be stored in a centralized location in order to compute the RMSD of each ligand with all the other ligands in the dataset. The analysis requires that the molecular dataset be moved in its entirety into a central

server. Figure 3.1 shows an abstract representation of a centralized data analysis system in which all the ligands have to be moved to a local storage where they can be compared and clustered. This fully-centralized approach is not scalable and can result in serious storage and bandwidth pressure on the server side. Thus, the challenge involves ways to efficiently find these dense ligand clusters of geometric representations, especially when the data is acquired in a distributed manner. While each conformation is small (on the order of ten kilobytes), the number of conformations that must be moved across the distributed memory system and compared with one another is extremely large; depending on the type and number of proteins, the conformation dataset can comprise tens or hundreds of millions of ligands. This scenario is expected when thousands of processes perform individual docking simulations and store their results locally. As an example, consider the D@H project that is supported by more than 188,000 hosts. If all the hosts communicate their local results to a centralized server simultaneously, even when each host communicates data in terms of 100 ligand conformations (i.e., around 7 MB in size), the data sending across the distributed memory system is more than 1 TB. This situation can lead to serious storage and bandwidth pressure on the server side.

3.2.2 Distributed clustering in MapReduce

The MapReduce programming model has been used in the past to analyze large data in science and engineering fields by using clustering techniques. Some efforts have investigated well-known clustering methods, such as k-means clustering and probabilistic hierarchical clustering, which were adapted to fit into the MapReduce framework [47, 25]. However, the resulting implementations suffer from the limitations of the clustering algorithms, which do not scale, despite being formulated in the MapReduce programming model.

A similar clustering approach based on the density of single points in an N-dimensional space was presented by Cordeiro et al. [18]. The three algorithms presented in that work rely on local clustering of subregions and the merging of local results into

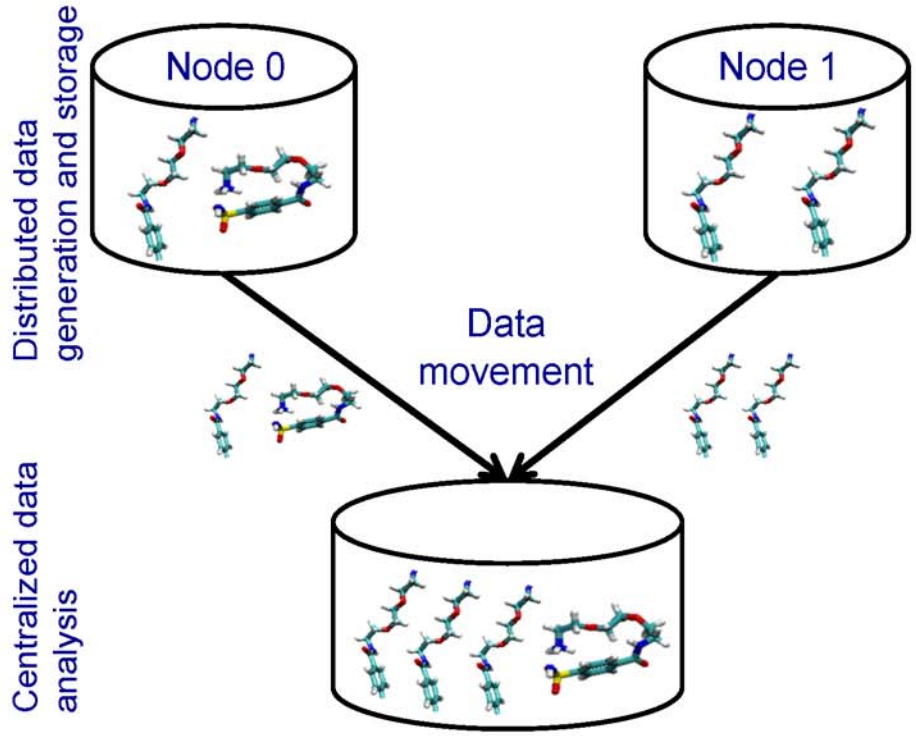


Figure 3.1: Traditional centralized comparison and clustering of large datasets generated on a distributed memory system.

a global solution (which can potentially suffer from accuracy issues). In contrast, our proposed approach considers the whole dataset and performs a single-pass analysis on it. Moreover, contrary to [18], when using the LG variation we no longer observe a correlation between space density and clustering efficiency.

Another group of research efforts focuses on a hashing step that partitions the input data into groups and then clusters each group in parallel. Important work in this direction includes that of Hefeeda et al. [37] and Rasheed et al. [61]. Hefeeda et al. present an approximation algorithm that reduces the computation and memory overhead in kernel-based machine learning algorithms. The approximation algorithm uses locality-sensitive hashing on the signature of each data record in the dataset to hash close records in the same bucket. This method potentially suffers from accuracy

issues when the level of approximation is low. In our work, we deliver scalable performance without sacrificing the accuracy. Rasheed et al. cluster metagenome sequence reads using a minwise hashing approach and agglomerative hierarchical clustering or a greedy clustering. Their work requires computing the similarity of a given metagenome sequence read with groups of sequence reads. In our work, on the other hand, no comparison is needed between data records (i.e., ligand conformations).

To the best of our knowledge, our approach is the first to emphasize a local, single pass of data to extract global properties or densities and, by doing so, to avoid major data movements [84]. In our work, scalability plays a key role and is particularly well supported by the LG variant. Our approach expands our previous work [28]. In this thesis we use MapReduce-MPI rather than Hadoop to move away from the overhead of the Hadoop Distributed File System (HDFS). HDFS acts as the central storage space for input and output data, adding additional space and operation time to move in and out data. We also extend the previous algorithm into two variations for fully-distributed environments. In the variations, data no longer needs to be moved a priori with HDFS, making our approach completely scalable. These variations allow us to study the performance impact of exchanging extracted properties in contrast to exchanging property densities; this evaluation was not achievable in Hadoop because of the coarse-grained control on data placement of HDFS.

3.3 Methodology

The overall method for clustering ligand geometry first extracts the relevant geometrical properties of each ligand conformation and represents the properties as N-dimensional points and then performs an N-D clustering by searching dense subspaces through counting properties or property aggregates. Figure 3.2 shows an overview of our approach.

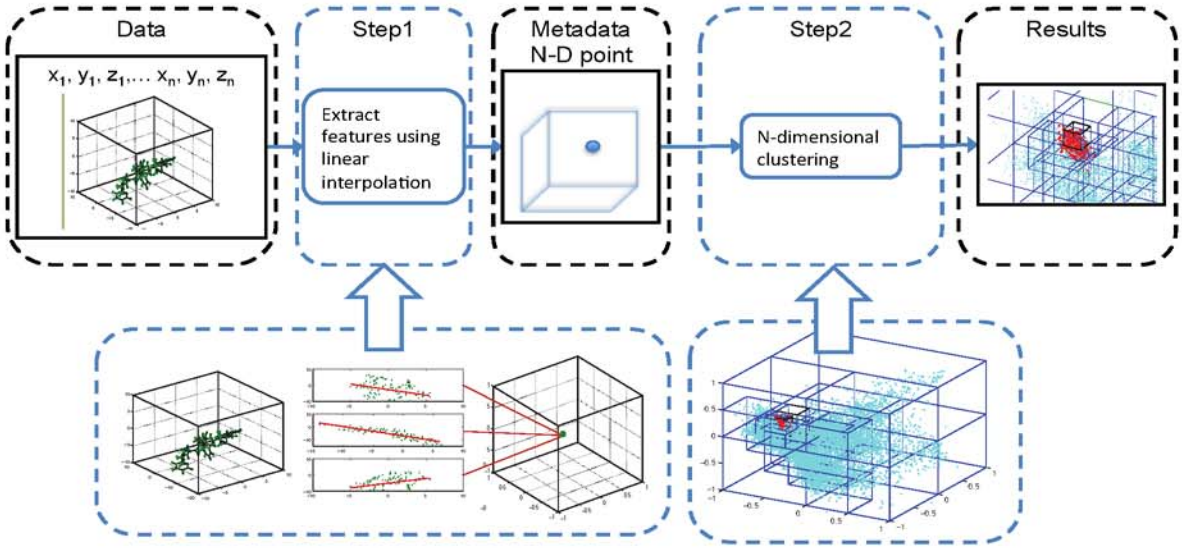


Figure 3.2: Overview of ligand clustering.

3.3.1 Capturing relevant geometrical properties

Our processing of docking simulation results requires extracting the geometrical shape (property) of each docked ligand in the docking pocket of a protein. To this end, we perform a space reduction from the atom coordinates of the ligand conformation to a single point of three or six coordinates in a 3-D or 6-D space, respectively, depending on which reduction technique we are using. The three or six coordinates are the metadata that represents the ligand's geometry in the protein's docking pocket. Our approach relies on the key assumption that when geometrical properties are properly captured, ligand conformations with similar geometries are mapped into similar metadata, as shown in Figure 3.3 where close 3-D points in a 3-D mapping represent the geometry of similar ligand conformations. In other words, our space reduction should have the desired property of projecting ligands with a similar geometry closer into the newly defined space, and the converged cluster with the highest density of points should include ligand conformations of interest for the pharmaceutical search. The empirical validation of this hypothesis is presented in Section 3.5.

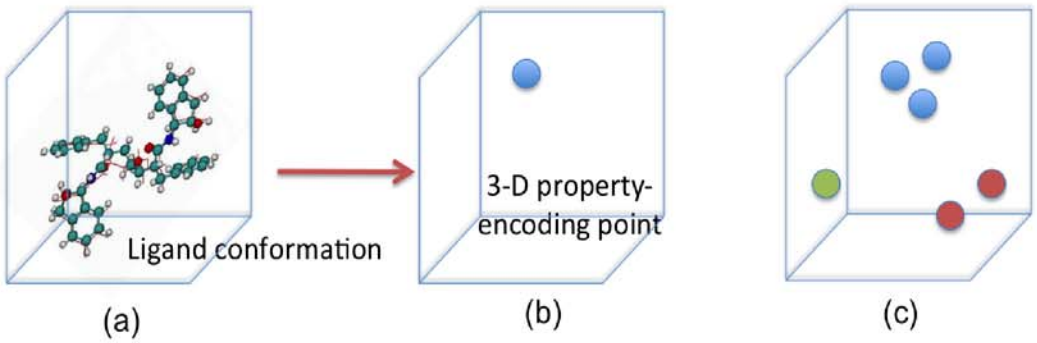


Figure 3.3: From the scientific data to extracted property: (a) a ligand conformation in the docking pocket of a protein, (b) the 3-D point that encodes the geometrical property using either the 3-D or 3-Dlog mapping, and (c) six points obtained in parallel that represent six ligand conformations clustered in three groups with three different geometries.

We consider three variations of the mapping algorithm; all are based on projections and linear interpolations. The variations share the backbone reduction technique but differ in terms of the final metadata representation. In all three variations, given a ligand with p atomic coordinates (x_i, y_i, z_i, with i from 1 to p), we perform a projection of the coordinates in the three planes (x, y), (y, z), and (z, x). Each projection results in a set of 2-D points on the associated 2-D plane. For each projection, we compute the best-fit linear regression line over the projected points and compute the three slopes of the three lines. In the first variation of our reduction, we use the three slopes as the coordinates of the 3-D point to encode the conformational geometry of its corresponding ligand. Figure 3.4 shows an example of metadata generated from multiple conformations of the ligand *1hbv* when docked in the HIV protease as part of the Docking@Home project [23]. We call this variation "3-D mapping."

In the second variation, we use the logarithmic values of the three slopes to encode the 3-D point representing the conformation geometry. If the slope is negative, we use the negative logarithm of the absolute value. Contrary to the "3-D mapping" variation, this variation better captures the geometrical properties of conformations that are in an almost-vertical position inside the protein pocket. In the 3-D mapping

variation, when ligand conformations are in an almost-vertical position in the protein pocket, the three resulting slopes are large. When the shape rotates or changes slightly, the resulting slopes change significantly. This situation may result in conformations with similar shape and in an almost-vertical position to be unmapped into a dense metadata subspace. When using the logarithmic of the slopes as metadata, we decrease the changes in the metadata coordinates and thus increase the chance for ligand conformations with similar shape to form a dense-enough subspace. We call this variation "3-Dlog mapping."

In the third variation, in addition to the slopes, we compute the intersection of the three linear regression lines with the x-axis for the line on the (x, y) plain, the y-axes for the line on the (y, z), and the z-axis for the line on the (z, x). We map each conformation into a 6-D point that coordinates the three slopes and the three line intersections. Unlike the other two variations, this variation not only captures the conformation shape and rotation but also stores the correct location of the ligand in the docking pocket. We call this variation "6-D mapping."

The advantage of our space reduction is that it does not rely on calculations of atomic distances between two or more ligand conformations as do most traditional analysis algorithms, such as k-means and fuzzy c-means clustering. Those calculations may require moving conformations across nodes, thus causing many frequent communications and multiple storages of the same data across nodes. On the contrary, our space reduction can be applied individually and concurrently to each ligand conformation by transforming each molecule containing p atomic coordinates in the three-dimensional space $(p * 3)$ into a single point of $(1 * 3)$ for the 3-D and 3-Dlog mappings and $(1 * 6)$ for the 6-D mapping, all in the Euclidean space. This transformation is performed locally on the compute node that generates and stores the ligand conformation, and thus no communication is required during this phase. The projections and interpolations are low-cost processes in terms of computing and memory requirements.

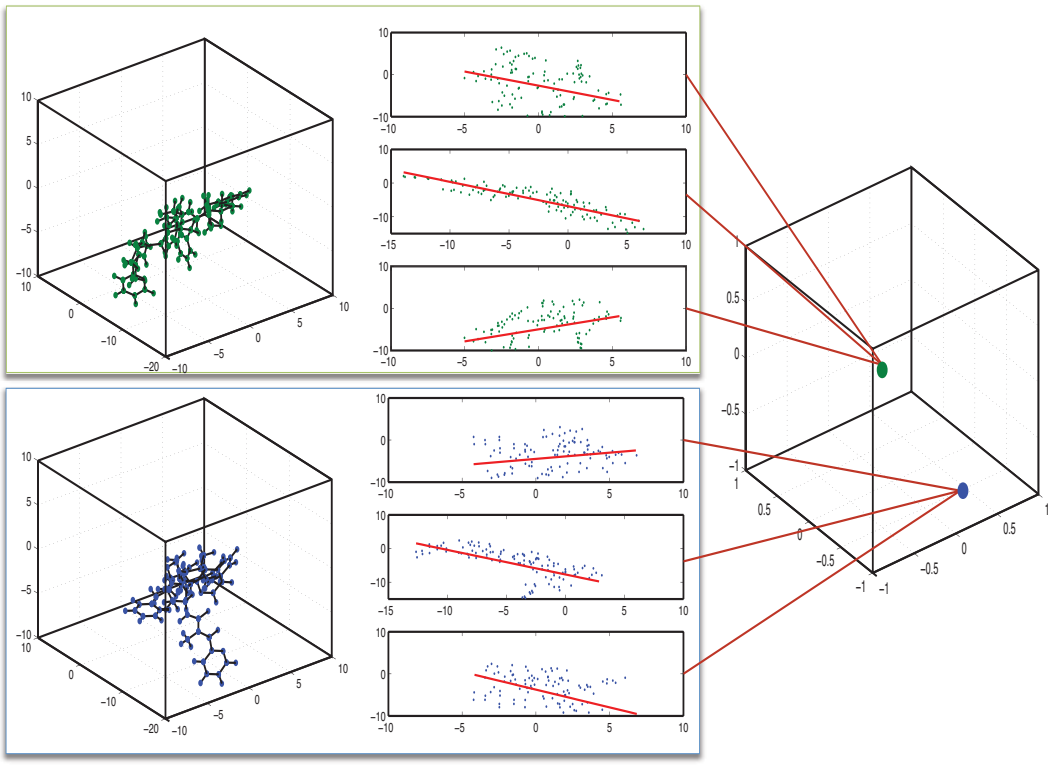
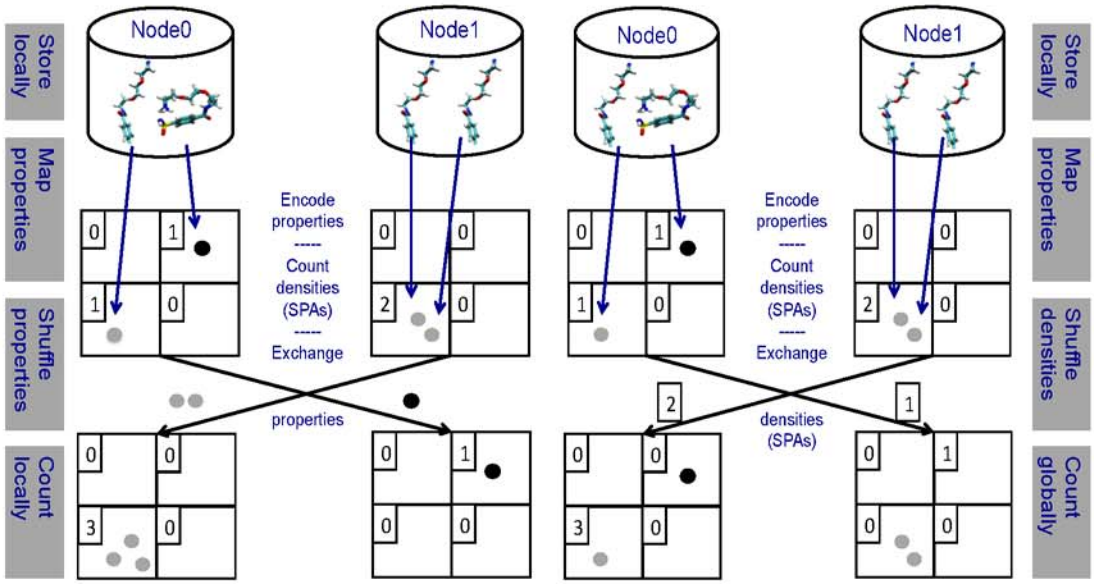


Figure 3.4: Capturing relevant geometrical properties by using projection and linear interpolation for the 3-D mapping variation.

3.3.2 Searching and counting property aggregate

By dealing with property-encoding metadata (i.e., three-dimensional points for the 3-D and 3-Dlog mappings or six-dimensional points for the 6-D mapping) rather than raw atom coordinates, we implicitly transform the analysis problem from a clustering or classification problem into a search of the smaller subspaces in the newly defined metadata space (i.e., an octant for the 3-D and 3-Dlog mappings or a 6-D space for the 6-D mapping) with high property aggregates. To perform the search, we build an N-D tree by recursively partitioning the N-D space into fixed-sized subspaces, each of which forms the tree nodes. Extracted properties can be unpredictably distributed across the N-D space of property-encoding points and across the compute nodes of the distributed memory system. Thus, the next challenge we face is how to efficiently count the aggregates of close property-encoding points in a distributed way. We address the

challenge by defining an effective search algorithm based on an N-D tree search. The search algorithm counts scalar property aggregates (SPAs) representing locally stored metadata densities and identifies the densest tree nodes in the N-D tree (by "densest tree nodes" we mean the deepest nodes with a minimum number of metadata items or aggregates). We design two general variations of the search algorithm that we call GlobalToLocal and LocalToGlobal. The two variations are depicted in Figure 3.5 for a simplified case study that maps the conformations into a 2-D space.



(a) GlobalToLocal (b) LocalToGlobal

Figure 3.5: Examples of exchange of properties in the GlobalToLocal variant and exchange of scalar property aggregations in the LocalToGlobal variant.

In GlobalToLocal, the 3-D and 6-D points representing the extracted metadata are exchanged among compute nodes across the distributed memory system to rebuild a global view of the information content in the scientific data so that similar points representing similar geometries eventually reside on the same compute node or nodes that are topologically close to each other; see Figure 3.5(a). The atom coordinates of the ligands generating the property-encoding points are not moved from their original compute nodes. Property densities are then iteratively counted locally while searching

for convergence.

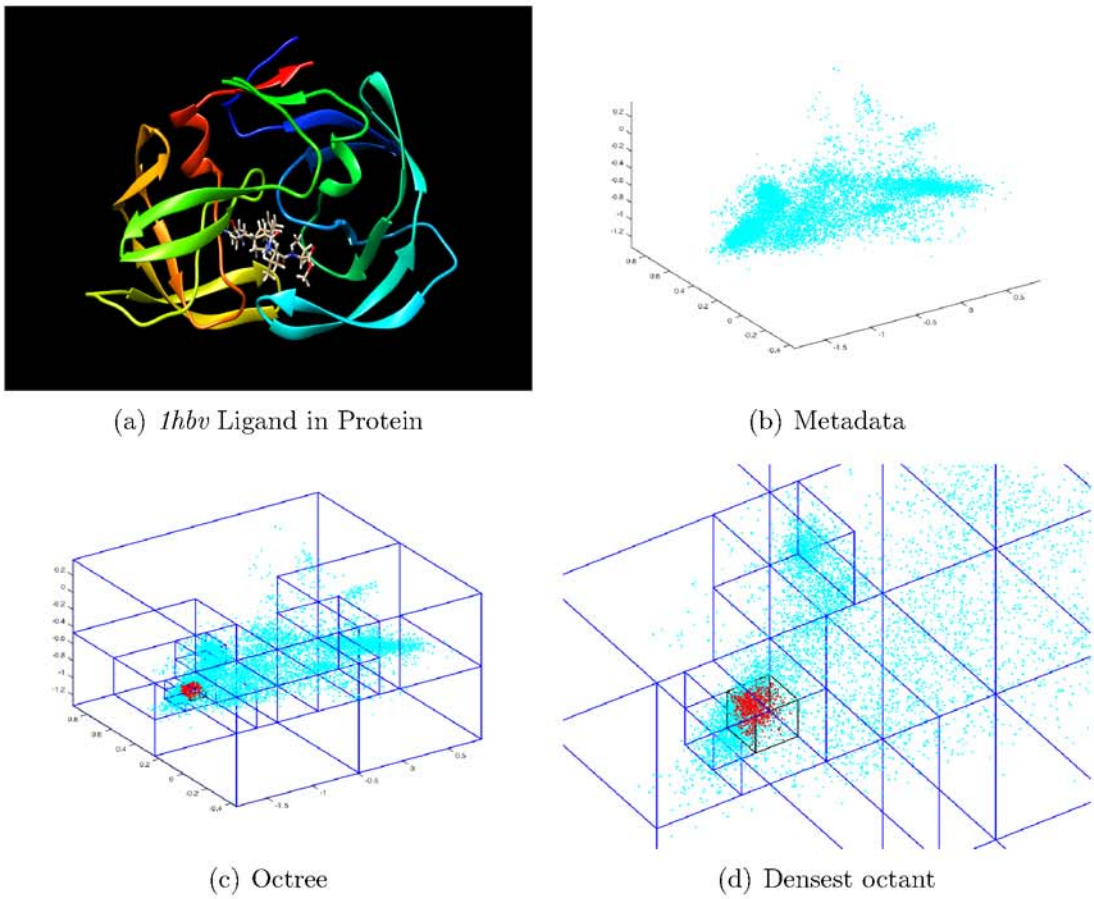
In LocalToGlobal, instead of exchanging extracted metadata, compute nodes exchange partial densities represented as scalar property aggregates; see Figure 3.5(b). Each compute node preserves a global vision of the metadata associated with its local data and counts its densities before shuffling the densities (or aggregates) rather than the metadata with other compute nodes. After shuffling the aggregates, each compute node sums the aggregates to obtain the global cluster densities while searching for convergence.

As already outlined above, the local property-encoding points can populate the space in an unpredictable way, the points can reside across multiple subspaces, and each compute node has a disjointed vision of the whole space. GlobalToLocal and LocalToGlobal address the uncertainty associated to the above described data distributions differently. In GlobalToLocal, each compute node has an assigned subspace of the N-D space, and the extracted properties (i.e., the 3-D or 6-D points) are communicated among compute nodes so that each node has all the information needed to analyze similar properties in the assigned subspace. Consequently the density analysis is applied to each compute node's local properties iteratively. On the other hand, in LocalToGlobal, after capturing relevant properties, each compute node applies the data analysis operation to its local extracted properties and computes scalar property aggregates based on densities for the entire assigned N-D space. In this scenario, each compute node has a disjointed view of the entire N-D space containing the metadata for the ligand conformations stored on its disk. The union of the disjointed N-D spaces is important for knowledge acquisition of the densest subspaces. This is pursued in LocalToGlobal through the communication of the local aggregates among compute nodes so that each compute node has all the partial results necessary to analyze the density globally. An aggregate operation (i.e., sum) is applied to the partial results. Communication of properties and aggregates is done iteratively. To sum up, communication in GlobalToLocal happens only once, and the communicated package is larger while communication in LocalToGlobal happens multiple times, and each time the

communicated package is smaller.

From the implementation point of view, in both variations (i.e., GlobalToLocal and LocalToGlobal) we reshape the space of property-encoding points into an N-D tree and search for the deepest, densest tree nodes. These nodes contain the solution to our analysis problem. Our search for dense tree nodes begins with each compute node generating its N-D tree on its own local data by recursively subdividing the space into 2^k subspaces: 8 (2^3) subspaces for the 3-D and 3-Dlog mapping and 64 (2^6) subspaces for the 6-D mapping. Figure 3.6 shows an example for a dataset of *1hbv* ligand conformations when docked in the HIV protease. Figure 3.6(a) is the result of the 3-D mapping, after the mapping of *1hbv* ligand confirmations into metadata has been performed. The compute nodes build an octree by assigning octkeys to the points, an N-D tree by assigning N-D keys to the points in the case of 6-D mapping. A point belongs to a specific tree node based on its key. The point's key is generated as follows. We initially determine the edge size (i.e., N-D resolution) of the N-D space containing all the projected conformations. Since we are dealing with the 3-D mapping in this example, we divide the initial space into eight subspaces of the same size, half the original edge size. A similar process is followed for the 3-Dlog mapping; however, when using the 6-D mapping, the metadata space is subdivided into 64 subspaces (not shown here). Every subspace is given an unique identifier ranging from 0 to 7 for the 3-D space or from 0 to 63 for the 6-D space, based on its position in the N-D space. The key of each point is extended by attaching the subspace identifier to the point's key by padding the left side with the identifier. This process is recursively repeated an arbitrary number of times on each subspace to produce a complete key for each point (key[1...Nkey]), where Nkey is the number of digits selected to represent each point. As previously observed in [29], Nkey can be empirically defined, and a value key of 15 digits is sufficient to capture diverse geometries in the dataset of ligand conformations considered in this work. Figure 3.6(c) shows an example of the generated octree for the 3-D mapping points in Figure 3.6(b).

The compute node explores the N-D space (the octree in Figure 3.6(c)), moving



(a) *1hbv* Ligand in Protein

(b) Metadata

(c) Octree

(d) Densest octant

Figure 3.6: Example of a metadata space of 3-D points generated from a dataset of ligand conformations and its octree built to identify the densest octant.

up or down along the tree branches depending on whether a "dense enough" tree node is found. The exploration is performed as a binary search along the tree levels. For example, if the tree is built with each point in the metadata space containing a key of 15 digits, then the tree has up to 15 levels. Our search starts at Level 8, and we reach a solution (i.e., the deepest tree node with a defined minimum number of points) by exploring up to 4 levels of the tree. Specifically, we start from Level 8 and branch to either Level 12 or Level 4 depending on whether any node is found at Level 8 with at least a given number of points or not. Once at the new level, the same criterion is applied to decide whether to move up or down within either the lower half the tree (moved down in the previous iteration) or the upper half (moved up). Figure 3.6(d)

shows the deepest and densest octant (points in red) that is identified by our tree search when looking for the deepest tree node with at least 500 points. For a 6-D tree, the same approach is applied but on a larger number of branches at each level. During the search, compute nodes may exchange either extracted properties (metadata) or scalar property aggregates based on which variation of the algorithm is used. As shown in Figures 3.5(a) and 3.5(b), in both variations each compute node transforms its locally stored ligand conformations into a local N-D space, as described in the steps involving capture of the relevant properties, and exchanges only partial knowledge on its metadata with the other compute nodes. Since compute nodes work on disjointed sets of ligand conformations and metadata, they can map ligand conformations into metadata concurrently and count aggregates locally in advance to perform a global summation.

3.3.3 Integration of the clustering algorithm into MapReduce

The MapReduce programming model naturally accommodates the capturing of properties from local data and the iterative search for either properties or densities in its map and reduce functions, respectively. Thus, we integrated our two variations of the search described above into the MapReduce-MPI framework rather than implementing a new MPI-based framework from scratch.

In the GlobalToLocal variation, the operation of capturing relevant properties is implemented as the map function. It takes the identifier and coordinates of each ligand conformation as the input key and value, respectively; applies the geometry reduction operation; and outputs the id and the property-encoding point of each conformation as the intermediate key and value pair. The MapReduce-MPI library shuffles the property-encoding points across the distributed memory system to rebuild the global knowledge of the N-D space on each node. It achieves this goal by communicating all the property-encoding points in one N-D subspace to one process such that this process has all the information needed to explore the N-D tree locally. Then, the operation of counting property densities is implemented as the reduce function. This function takes

as input the id of the tree node and all the property-encoding points in the node and iteratively explores its local N-D tree by counting the density of the nodes in one level of the N-D tree until the deepest and densest tree node is found.

The LocalToGlobal variation has two map functions. The first function captures the relevant geometrical properties in the same way as in the GlobalToLocal variation. After the relevant properties are extracted by the first map function, the second map function counts locally the property aggregates for a certain level of the N-D tree nodes. It takes the id and the property-encoding point of each conformation as the input key and value pair, respectively; counts the aggregates for tree nodes at a certain level; and outputs the id and aggregates of each tree node as the intermediate key and value pairs. The exploration of the N-D tree starts at the middle level of the tree and branches up or down depending on whether a dense enough node is found. The MapReduce-MPI framework shuffles the id and all the aggregates of each node across the distributed memory system. Then, the reduce function applies a sum operation to all the aggregates to compute the density of the nodes. The process of counting aggregates (i.e., the second map function) and summing aggregates (i.e., the reduce function) is iterated until the deepest and densest node is found.

As already outlined in the previous section, it is important to notice how the differences between the two variations lie in the communication stage (i.e., data shuffle stage). In the GlobalToLocal variation communication happens only once, and the size of the communicated data (i.e., the property-encoding points) is larger. In comparison, in the LocalToGlobal variation the communication happens multiple times, and each time the size of the communicated data (i.e., local aggregates) is smaller.

3.4 Performance

3.4.1 Platforms

We consider three platforms with different scales in terms of number of nodes. At the small scale, we use a dedicated cluster called Geronimo at the University of Delaware (UD) and a shared cluster called Gordon at the San Diego Supercomputer

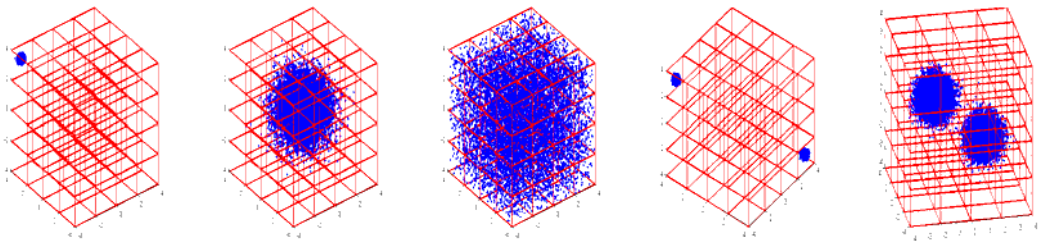
Center. Geronimo is composed of 8 dual quad-core compute nodes (64 cores), each with two Intel Xeon 2.5 GHz quad-core processors and 48 GB RAM. The nodes are connected by high-speed DDR InfiniBand. Gordon, a Track 2 supercomputer at the San Diego Supercomputer Center, features powerful flash memory storage private to each compute node that can be used to simulate a fully distributed memory system [15]. For the tests, we are limited to 64 nodes because of the allocation constraints defined by the provider. Each node contains two 8-core 2.6 GHz Intel EM64T Xeon E5 (Sandy Bridge) processors and 64 GB of DDR3-1333 memory and mounts a single 300 GB SSD. The nodes are connected by a 4 x 4 x 4 3-D torus with adjacent switches connected by three 4 x QDR InfiniBand links.

At the large scale, we use Fusion, a 320-node computing cluster at the Laboratory Computing Resource Center at Argonne National Laboratory. For these tests, we use 256 compute nodes. Each of Fusion's compute nodes contains two Nehalem 2.6 GHz dual-socket, quad-core Pentium Xeon processors, 36 GB of RAM, and 250 GB local disk. The nodes are connected by InfiniBand QDR at 4 GB/s per link.

3.4.2 Datasets

We define different type of datasets and different sizes for each type. The datasets are built from real Docking@Home data. Smaller datasets are in the order of 0.5 GB (200K ligand conformations) up to 4 GB (1.6 million ligand conformations). Larger datasets are in the order of 250 GB (100 million ligands) up to 2 TB (800 million ligands).

The datasets are built to fit five specific property distributions in the N-dimensional space (logical distributions) that reflect five scientific conclusions in terms of convergence toward specific ligand geometries. Figures 3.7 shows the five logical distributions for the 3-D and 3-Dlog mapping. The 6-D mapping is not presented here because of the complexity of its representation on a 2-D page, but similar conclusions hold for it.



(a) (b) (c) (d) (e)

Figure 3.7: Five scenarios of logical distributions of data: 1D (a), 1S (b), UN (c), 2D (d), and 2S (e).

As shown in Figure 3.7(a), in the first scenario with one dense cluster (1D), the information content in the scientific data strongly converges toward one ligand conformation; the 3-D points densely populate one small region of the metadata space while most of the remaining space is empty. We select ligands from the Docking@Home dataset whose geometries generate 3-D points with normal distribution around one point in the 3-D space with a standard deviation of 0.1 for this scenario. As shown in Figure 3.7(b), in the second scenario with one sparse cluster (1S), the information content in the data more loosely converges toward one ligand conformation; the 3-D points sparsely populate one larger region of the space. The ligand geometries generate points with normal distribution around one point in the 3-D space with a standard deviation of 1.

As shown in Figures 3.7(c) and 3.7(d), respectively, in the third scenario with two dense clusters (2D) and the fourth scenario with two sparse clusters (2S), we extend the first and second scenarios by presenting scientific cases in which information in the data either strongly or loosely converges toward two major conformations rather than one. More specifically, in the third scenario, ligand geometries are mapped onto points with normal distribution around two separate points with a standard deviation of 0.1. In the fourth scenario, we generate points with normal distribution around two separate points with a standard deviation of 0.5. As shown in Figure 3.7(a), in the fifth scenario with a uniform distribution of the information (UN), the 3-D

space does not convey any main scientific conclusion; no single ligand conformation was repeatedly generated. This scenario can happen, for example, with insufficient sampling or inaccurate mathematical modeling.

We also study the impact of the data distribution across the nodes' storage (physical distribution). Specifically, we study these three different ways in which distributed data can be generated and stored across the nodes' disks: Uniform, Round-robin, and Random. Figures 3.8 shows the three physical distributions considered in our study, respectively. In the Uniform physical distributions, property-encoding points that belong to the same subspace in the logical distribution are located in the same physical storage. This scenario happens most likely in a semi-decentralized distribution in which points mapping close properties are collected by the same node or topologically close nodes; hence, there is uniformity of the property-encoding points inside the nodes' storages. In the Round-robin physical distributions, points that belong to the same subspace in the logical distribution are stored in separate physical storage in a round-robin manner. This scenario happens most likely in a fully-decentralized, synchronous distribution in which points are collected at each predefined time interval; hence, the data points for each time interval are stored in separate storage across the distributed system. In the Random physical distributions, points are randomly stored in the physical storage of all the system nodes. This scenario simulates the fully-decentralized asynchronous manner. In each of the scenarios, the whole dataset is roughly evenly distributed among the physical storage sites.

3.4.3 Results and discussion

We study two important aspects of our method: first, whether our method is sensitive to data distributions at both small and large scales, and second, what is the data scalability of our method as the data size grows when comparing to a probabilistic hierarchical clustering [27] that more traditionally relies on data movement and root-mean-square deviation (RMSD) calculations to identify well-docked ligand conformations.

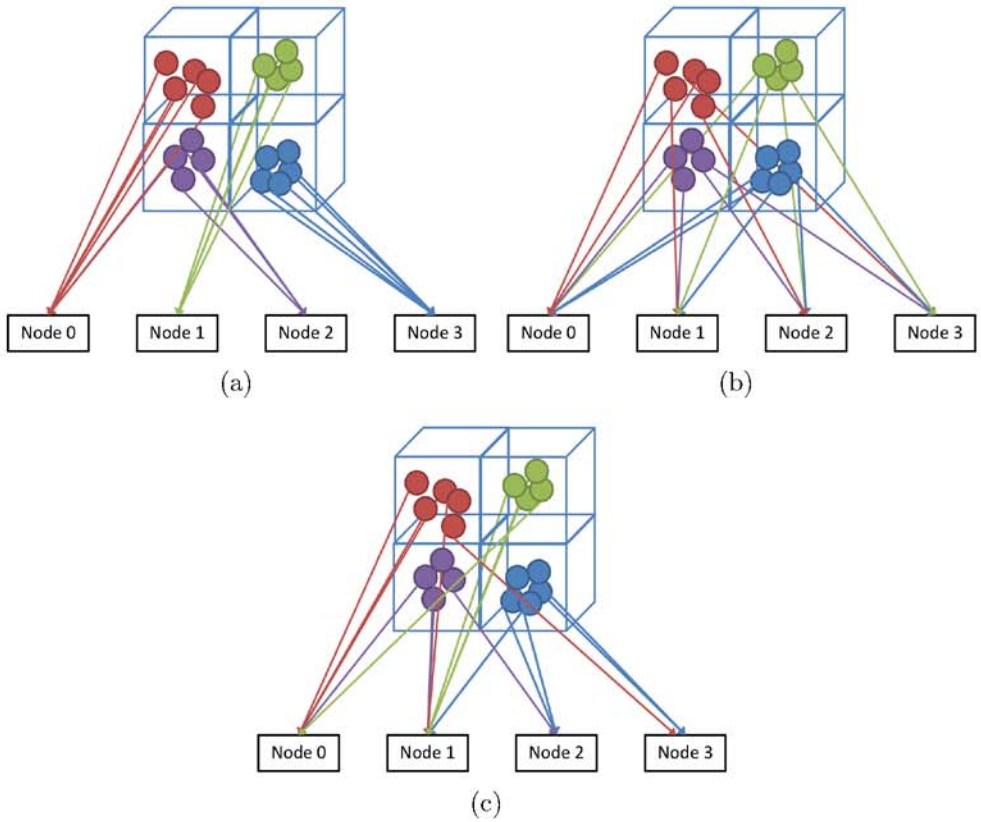
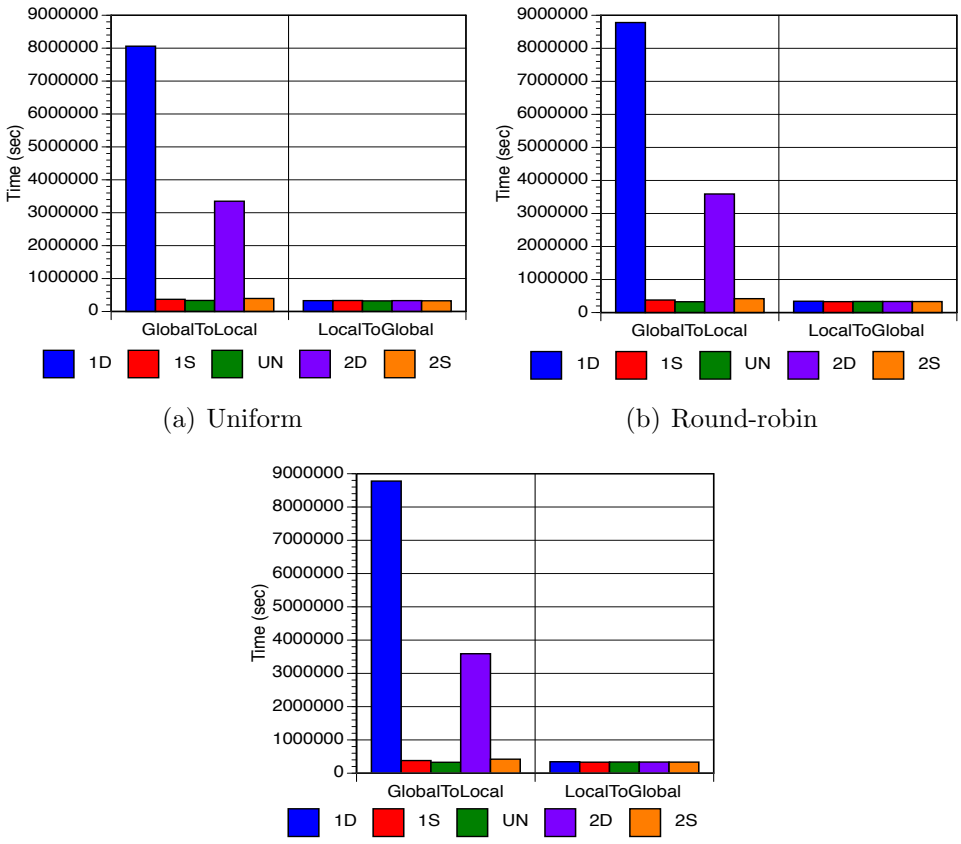


Figure 3.8: Three scenarios of physical distributions of data: Uniform (a), Round-robin (b), and Random (c).

Small-scale study: To assess our method at the small scale, we measure the performance of GlobalToLocal (GL) and LocalToGlobal (LG) for the five logical distributions (i.e., one dense cluster 1D, one sparse cluster 1S, uniform UN, two dense clusters 2D, and two sparse clusters 2S) and the three physical distributions (i.e., Uniform, Round-robin, and Random physical distribution) on Gordon. The data size for each distribution is 250 GB (100 million ligands) while the number of nodes for each distribution is 64 (1024 cores). Figure 3.9 shows the total time in seconds (aggregated across all processes). All the results are included in the discussion below, and the complete set of MapReduce times, including the Map, Shuffle, Overhead and Reduce times, is also reported in Table 3.1.

When looking at the execution times across the five logical distributions for



(a) Uniform

(b) Round-robin

(c) Random

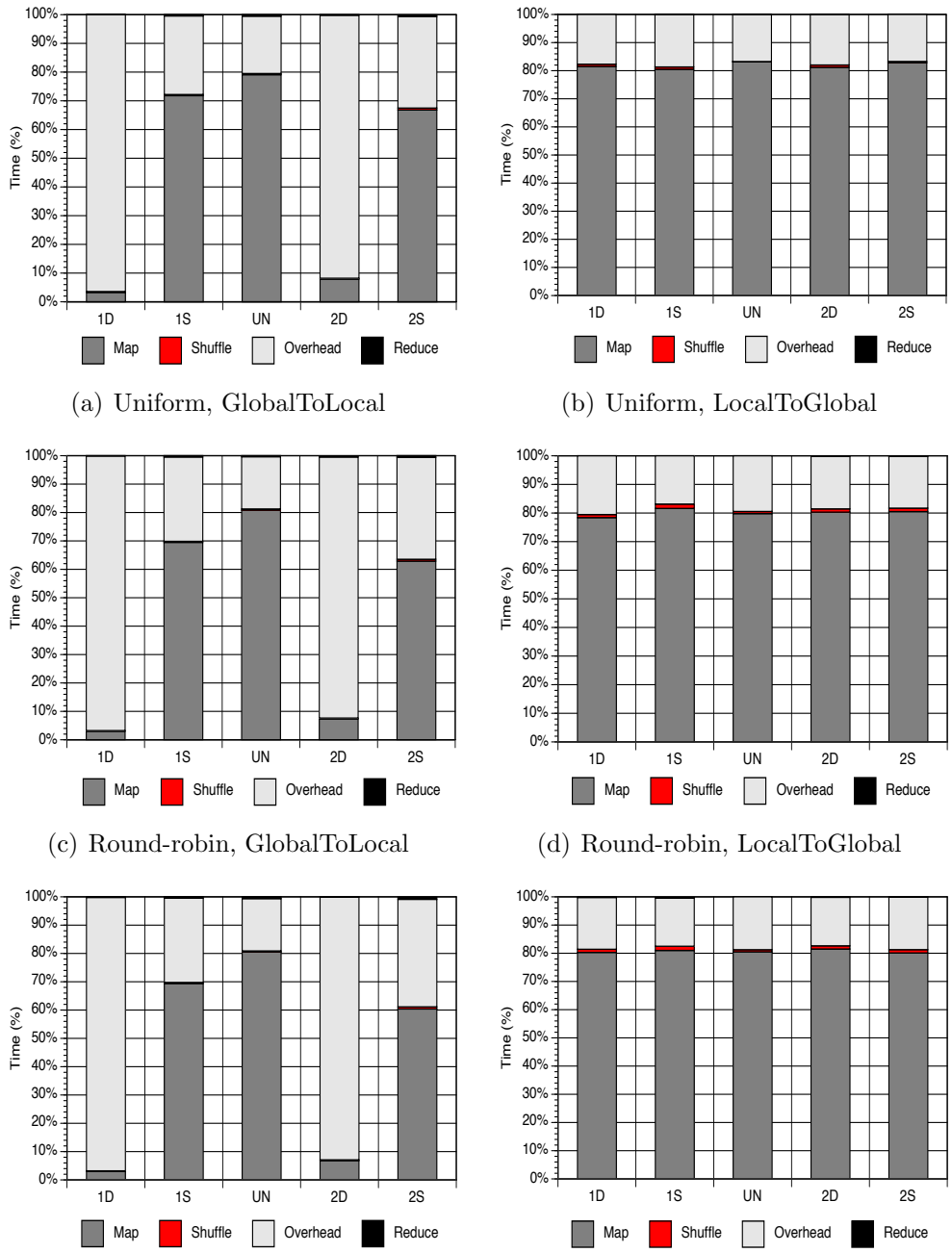
Figure 3.9: Total execution time for GlobalToLocal and LocalToGlobal on the five logical distributions and (a) (b) the Uniform physical distribution, (c) (d) the Round-robin physical distribution, and (e) (f) the Random physical distribution.

both GlobalToLocal and LocalToGlobal variations, we observe that the performance of GlobalToLocal is highly sensitive to the logical distributions, while LocalToGlobal delivers scalable performance across the five logical scenarios. For example, in the case of Uniform physical distribution, when the information content in the dataset strongly converges to one cluster (i.e., the 1D logical distribution), the total execution time for GlobalToLocal is more than one order of magnitude larger than when the information does not converge at all in the UN logical distribution (i.e., 8.06E06 seconds in 1D vs. 3.35E05 seconds in UN). When the information content in the dataset strongly converges to two clusters (i.e., the 2D logical distribution), the total execution time for

GlobalToLocal is one order of magnitude larger than with the UN logical distribution (i.e., 3.35E06 seconds in 2D vs. 3.35E05 seconds in UN), while for LocalToGlobal, the execution time varies only 3.4% across the five logical distributions. We observe similar results in the case of Round-robin and Random physical distributions. Note that scenarios such as one dense cluster 1D and two dense clusters 2D are scientifically meaningful because the information content of the science strongly converges toward a few conclusions. GlobalToLocal has significantly longer execution time for such scenarios, while LocalToGlobal has scalable performance regardless of the information content latent in the datasets.

To study the reason for the big variations in GlobalToLocal and the small variation in LocalToGlobal, we show in Figure 3.10 the four MapReduce time components (i.e., Map, Shuffle, Overhead and Reduce times) normalized with respect to the total execution time. Specifically, the figure presents the percentages of the four time components across Gordon's 1,024 cores. Map includes the time a process spends extracting properties during the preprocessing and searching across subspaces in the tree. Shuffle is the time spent exchanging properties in GlobalToLocal or densities in LocalToGlobal. Overhead is the time introduced by the MapReduce-MPI library for either synchronizing processes at the end of each MapReduce step (i.e., the implicit MPI_ALL_Reduce operations to communicate small bookkeeping information such as the total number of $\langle key, value \rangle$ pairs processed by the map or reduce function) or for awaiting certain processes to complete their Map or Reduce in case of load imbalance. Reduce is the time to aggregate properties in GlobalToLocal or the time to aggregate densities in LocalToGlobal. In the figure, we report the average execution time in seconds over three runs; the time traces are obtained by using TAU [70]. We instrumented the source code so that only a limited number of events (i.e., the time to perform the key Map, Shuffle, and Reduce functions in the two variations) are measured by TAU; thus, the overhead introduced by TAU is negligible.

When looking at the percentages of time for GlobalToLocal and LocalToGlobal across the five logical distributions in Figure 3.10, we observe that for all the three



(a) Uniform, GlobalToLocal

(b) Uniform, LocalToGlobal

(c) Round-robin, GlobalToLocal

(d) Round-robin, LocalToGlobal

(e) Random, GlobalToLocal

(f) Random, LocalToGlobal

Figure 3.10: Percentage of time for GlobalToLocal and LocalToGlobal for five logical and (a) the Uniform physical distribution, (b) the Round-robin physical distribution, and (c) the Random physical distribution.

physical distributions, the overhead percentage for GlobalToLocal varies significantly across the five logical distributions, while the overhead percentage for LocalToGlobal is relatively constant. More specifically, in the case of Uniform physical distribution, when the information content in the dataset strongly converges to one cluster or two clusters (i.e., the 1D and 2D logical distribution), the overhead for GlobalToLocal is 96.4% and 91.6%, respectively. When the information content loosely converges to one cluster or two clusters (i.e., the 1S and 2S logical distribution), the overhead is 27.4% and 31.9%, respectively. In the UN logical distribution, when the information content does not converge at all, the overhead is at minimum 20.0%. In comparison, for the LocalToGlobal variation, the overhead is consistently around 19.0% across all five logical distributions. Similar results are observed for the Round-robin and the Random physical distributions. In general, we see that when the datasets contain strong scientific conclusions (i.e., the 1D and 2D logical distributions), GlobalToLocal has a significant overhead, while LocalToGlobal has consistent overhead across the five logical distributions. The reason is that in the dense cluster 1D and two dense clusters 2D scenarios, the properties (3-D points) are densely populated in a small octant space; in order to explore the octree, GlobalToLocal requires relocating all the properties (3-D points) to one or two processes, respectively, on which the iterative search is performed locally. When points are sparsely distributed (one sparse cluster 1S and two sparse cluster 2S), GlobalToLocal overhead is still tangible but smaller than for one dense cluster 1D and two dense clusters 2D (i.e., up to one order of magnitude larger than in LocalToGlobal). Note that for GlobalToLocal, scenarios like one dense cluster 1D and two dense clusters 2D are associated with high total times because of the load imbalance and ultimately poor performance. On the other hand, LocalToGlobal performs similarly across scenarios (one dense cluster 1D, one sparse cluster 1S, two dense clusters 2D, and two sparse clusters 2S), and the overheads have lower magnitudes regardless of the embedded scientific results and the physical distributions.

When looking at the other time components, we observe that both GlobalToLocal and LocalToGlobal total Map times are similar; both variations perform similar

preprocessing. The GlobalToLocal total Reduce times are two orders of magnitude larger than the equivalent LocalToGlobal times; LocalToGlobal has been simplified to sum single density values (SPAs) rather than counting densities from properties and then summing them up. Dense and sparse clusters have different Shuffle times with dense scenarios taking longer than sparse scenarios. The reason is that shuffling all the properties to one or two processes for the global analysis, as in the case of dense scenarios, takes more time than does shuffling the same properties to a larger subset of processes. The scenarios with a logical uniform distribution always outperform the other scenarios in terms of performance independently from the physical distribution of the data, but these scenarios are also less desirable to work with from the science point of view since they do not convey any conclusion. When comparing the time components across physical distributions, we observe that semi-decentralized scenarios (i.e., Uniform) outperform fully-decentralized scenarios (i.e., Round-robin and Random) in GlobalToLocal and have similar performance for LocalToGlobal. The reason is that the 3-D points with similar properties are stored in the same node in the Uniform physical distribution; hence the data shuffled across the distributed memory system is smaller than with the Round-robin and Random physical distributions.

To better understand the reasons behind the observed overheads, in Figure 3.11 we present the typical time pattern per process for the Uniform, Round-robin, and Random physical distributions and the five logical distributions; each bar is a collection of 1,024 thin bars, one per each process. For each subfigure, each process time in the two bars is normalized with respect to the longest process time for both GlobalToLocal and LocalToGlobal. This approach allows us to better compare the two variations. Figure 3.11 shows that overheads have two components: (1) a much larger component that is present in GlobalToLocal only and is due to load imbalance and (2) a smaller component that is present in both variations and is due to a collective communication embedded into MapReduce-MPI when synchronizing all the processes. This is visible in the figure where thin black lines (corresponding to the Reduce phase) at the top of the bars are observed in one dense cluster 1D and two dense clusters 2D for one and

two processes, respectively, that perform most of the reduce task (i.e., the aggregate operation on the extracted properties). During this time, the remaining processes sit idle until the Reduce step is completed. For the one sparse cluster 1S and two sparse clusters 2S scenarios and GlobalToLocal, we still observe load imbalance (in the form of drifting lines at the top of the bars) but in a smaller proportion compared with one dense cluster 1D and two dense clusters 2D. This time, multiple processes across multiple nodes perform the aggregation operation on more dispersed properties. The synchronization time for the 1,024 processes is similar for the two variations, is negligible compared with the other overhead component in GlobalToLocal, and can take up to 20% of the total time in LocalToGlobal. Both Figures 3.10 and 3.11 convey the important findings that LocalToGlobal is able to perform implicit load balancing regardless of the logical and physical data distributions.

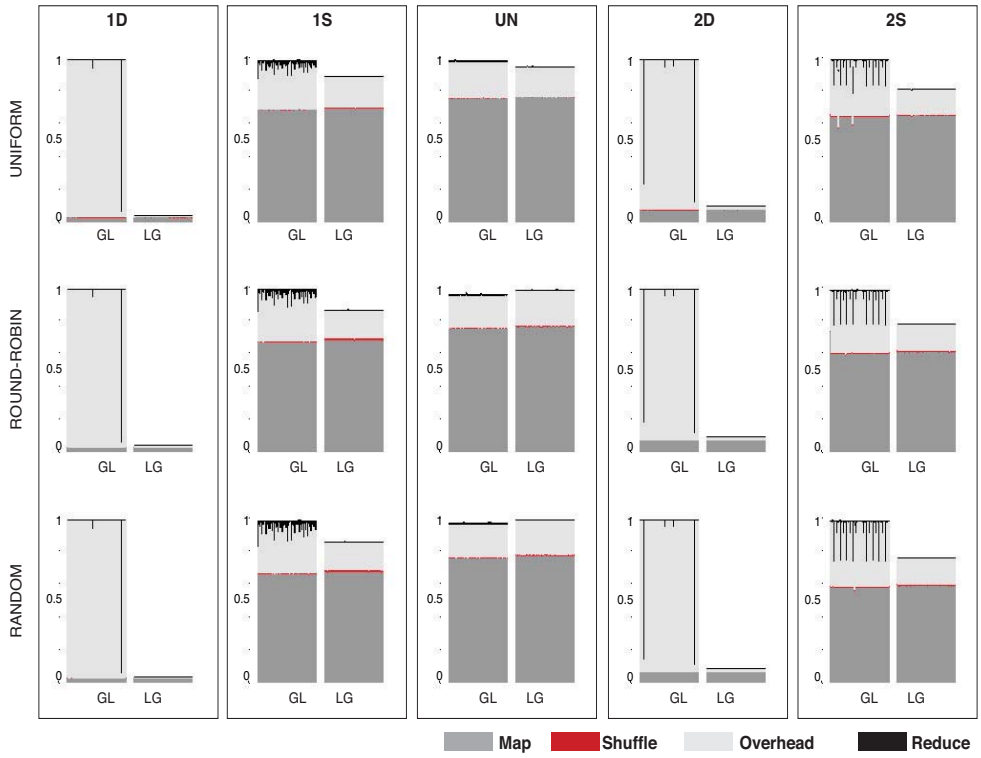


Figure 3.11: Per process times normalized with respect to the longest process per experiment for the Uniform, Round-robin and Random physical distributions.

Large-scale study: Because at the small scale, the GlobalToLocal variation exhibits substantial loss in scalability but the LocalToGlobal variation exhibits constant performance independent of the logical and physical distributions, at the large scale we focus only on LocalToGlobal. We compare and contrast the weak scalability of this search variation on Fusion when the data size increases from 128 GB (50 million ligands) to 2 TB (800 million ligands) while the number of nodes increases from 16 nodes (128 cores) to 256 nodes (2,048 cores). In our study, we use a single MPI process per core and a fixed data size per core of 1 GB. In our tests, we consider the five different logical distributions (i.e., 1D, 1S, 2D, 2S, and UN) and the Round-robin physical distribution.

In our performance analysis, we consider both the total execution time and its time main components (i.e., Map, Shuffle, and Reduce times). Each test is repeated three times; the measurements reported are average values. Figure 3.12 shows the average runtime in seconds and its variation across the three runs when using an increasing number of nodes on Fusion. We observe that as the number of nodes increases, the Map and Reduce times stay constant while the Shuffle times increase and eventually dominate the overall execution time. The Map and Reduce times are computation times. When studying weak scaling, the computation work per node is fixed; thus, we observe a constant behavior. However, the Shuffle times are associated with communication times during which either metadata (in the case of GlobalToLocal) or property aggregates (in the case of LocalToGlobal) are moved across nodes. The MapReduce-MPI library uses MPI_Alltoallv for communication; when the size of data and the number of processes increase, the communication overhead increases. At a smaller scale (i.e., on 16 nodes on Fusion), the execution times do not vary across the different logical metadata distributions; this result is consistent with what we observed on Gordon when using its 64 nodes. At a larger scale (i.e., when using 256 nodes), however, the execution times (mainly Shuffle times) vary with the logical distribution. Specifically, dense metadata (i.e., 1D and 2D) has smaller execution times (and associated Shuffle times). On the other hand, sparse metadata (i.e., 1S, 2S, and UN) has larger execution

times (and associated Shuffle times). This phenomenon is not observed at the small platform scales on Gordon or Fusion with only 16 nodes.

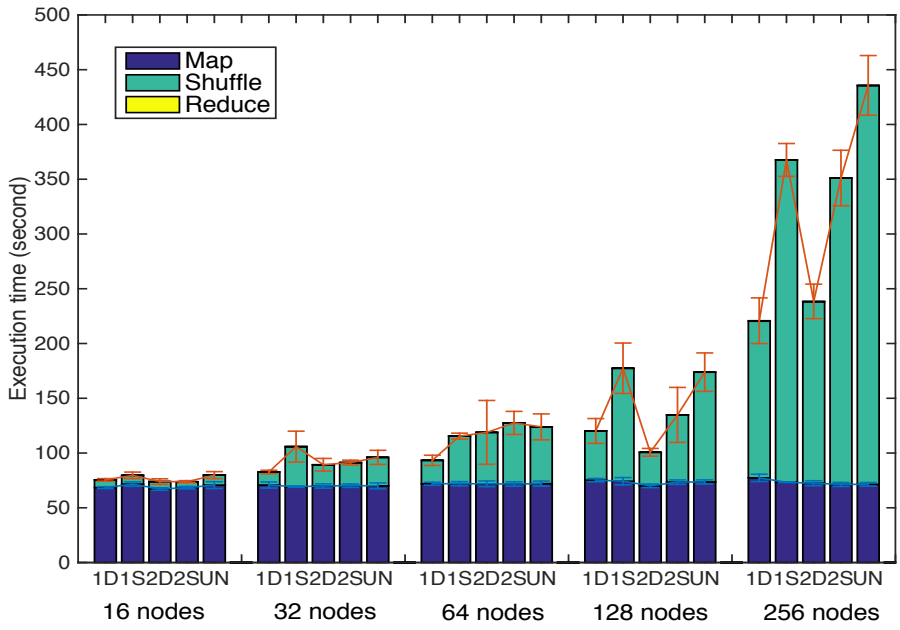


Figure 3.12: Averaged execution times in seconds and their variations cut down in Map, Shuffle, and Reduce times on Fusion with number of nodes ranging from 16 to 256.

To further investigate what causes the runtime variance, we look at the time breakdown of the Shuffle times. Specifically, we measure the average time taken by four consecutive shuffling calls in LocalToGlobal. The selection of four shuffling calls is based on empirical observations in our previous work [29] suggesting that we can represent each metadata point in the N-D space with a key that is 15 digits long (Nkey =15). Consequently, our search along the tree nodes explores four levels of the tree and performs four shuffling calls, as described in detail in Section 3.3.2. Table 3.2 shows the four levels explored in our search for the five logical distributions.

Figure 3.13 shows the average execution times over three runs and the associated variability for the different logical distributions when searching the larger dataset on 256 nodes of Fusion. In the figure, we observe how in the first shuffling call, 1D and 2D

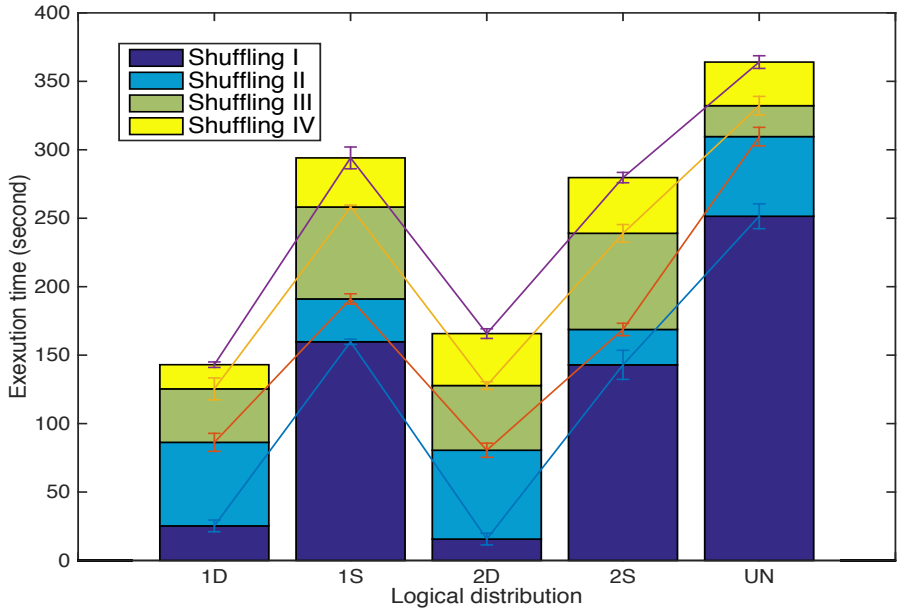


Figure 3.13: Averaged Shuffle times and their variations for the four shuffling calls and the five distributions on 256 nodes of Fusion.

distributions take less time than the 1S, 2S, and UN distributions do. In this shuffle phase, every compute node counts the octants at Level 8. All the octkeys starting with the same 8 digits (e.g., 00000000, 00000001, 00000002) are at the same level, and potentially there are 8^8 many possible octants. In the 1D and 2D distributions, since the metadata populates only a small region of the space, there are fewer octants to count, so the scalar property aggregates that need to be communicated are smaller. In the 1S, 2S, and UN distributions, the metadata populates a larger number of tree nodes; thus, the scalar property aggregates that need to be communicated across nodes are larger. This phenomenon ultimately results in higher Shuffle times. The second, third, and fourth Shuffle phases explore different levels of the tree depending on whether a dense octant is found or not at Level 8. Specifically, in their second Shuffle, 1D and 2D explore Level 12 of the tree (i.e., move down); this exploration results in more octants to explore, larger messages to exchange, and longer Shuffle times compared with the first Shuffle phase at Level 8. However, 1S, 2S, and UN explore Level 4 of the tree (i.e.,

move up); this results in fewer octants to explore, smaller messages to exchange, and shorter Shuffle times compared with the first Shuffle.

In the third shuffling call, 1D and 2D explore Level 10 (i.e., move up); this Shuffle deals with fewer octants compared with the second Shuffle and, ultimately, shorter Shuffle times than at Level 12. This is not the case for 1S and 2S that explore Level 6 (i.e., move down); this level has more octants and thus requires longer Shuffle times compared with the second Shuffle at Level 4. During the third Shuffle, UN explores Level 3; this is associated with fewer octants and shorter Shuffle times than the second Shuffle phase at Level 4. In the fourth shuffling call, 1D and 2D explore Level 9 with its smaller number of octants compared with the previous shuffling calls and consequently requires shorter Shuffle times. 1S and 2S explore Level 5 with fewer octants compared with the third shuffling call and shorter Shuffle times. UN explores Level 2; this level has fewer octants than does the previous shuffling call and requires shorter Shuffle times.

Empirically we observe how, at the large scale, communication times associated with shuffling calls are related to two factors in the search process: (1) the tree level to explore in the shuffling call (i.e., the deeper the level, the more the number of octants and the longer the Shuffle times) and (2) the metadata distribution. For the latter factor, distributions where the metadata populates a smaller space (e.g., 1D and 2D) exhibit Shuffle times that are smaller than distributions where metadata populates a larger space (e.g., 1S, 2S, and UN). Moreover, while at small scales the LocalToGlobal variation of our search does not show any sensitivity to the logical distributions, this situation is no longer true at the larger scales. Thus, this study outlines the limits of our approach for loosely convergent simulations (i.e., 1S and 2S) and for not convergent simulations (i.e., UN); for the datasets generated by these simulations new algorithms are needed to archive scalability at the large scale.

Data scalability: To evaluate the scalability of our method for tightly convergent simulations with growing data sizes (e.g., 1D and 2D), we consider a synthetic dataset of ligand conformations sampled with Docking@Home for the *1dif* ligand-HIV protease

complex. The dataset is built so that it exhibits a strong convergence towards a single, dense cluster (1D). For this dataset, we compare and contrast the execution times of a hierarchical clustering presented in [27] and discussed in Section 3.2.1 versus our N-D clustering [86]. The hierarchical clustering is performed on a dedicated cluster such as Geronimo; the N-D clustering is performed on a larger shared cluster such as Fusion. We use the 3-D metadata as an example since empirically we observed that 3-D and 6-D metadata have similar performance results. The time reported for the hierarchical clustering is the execution time on up to 8 nodes of Geronimo and includes both communication time, in which the distributed dataset is sent to a centralized node through InfiniBand, and analysis times, in which the hierarchical clustering performs comparisons of the conformations' geometries. Data ranges from 0.5 GB (200K ligand conformations) to 4 GB (1.6 million ligand conformations). The time reported for the N-D clustering is the total time for the MapReduce-MPI key steps including Map, Shuffle, and Reduce times on Fusion. The number of Fusion' s nodes ranges from 8 to 256 and the data range from 4 GB (1.6 million ligand conformations) to 2 TB (800 million ligand conformations). Figure 3.14 shows the results of this comparison. In

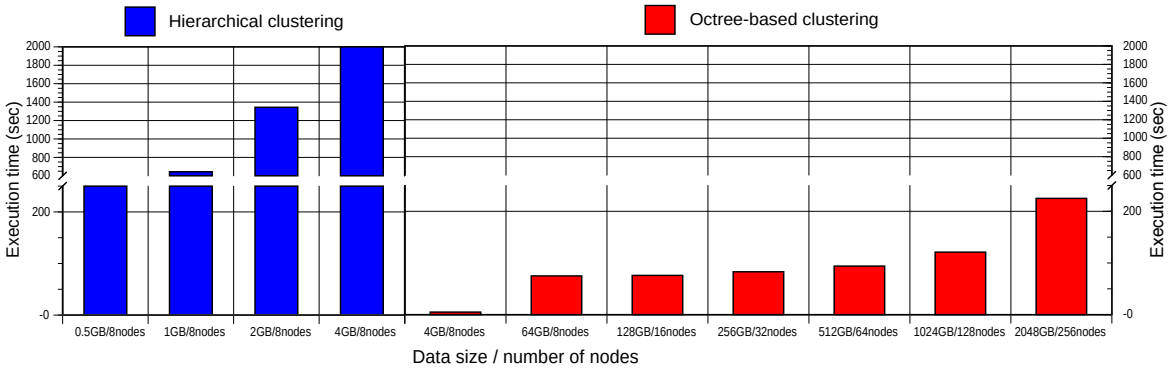


Figure 3.14: Performance comparison of our distributed clustering vs. the hierarchical clustering when the size of data increases.

the figure, we observe that the hierarchical clustering is not able to scale out to more than a dataset of 4 GB (1.6 million ligand conformations) and 8 nodes due to the fact that it needs to move the whole distributed dataset onto one node, and performs

all-to-all comparisons among the conformation records on that node. The type of comparison (i.e., the computation of the root-mean-square deviations among all the conformations) quickly fills the node's memory. A comparisons of the times for the hierarchical clustering for the 4 GB dataset versus the N-D clustering for the same dataset reveals a performance speedup of 400X for our analysis method. Specifically, the same type of analysis on the identical dataset is performed in 1999 seconds by the hierarchical clustering and in 5 seconds by our N-D clustering.

The study of the weak scalability on Fusion when the number of conformations increases from 25 million ligands (64 GB) to 800 million ligands (2 TB) and the number of nodes increases from 8 nodes (64 cores) to 256 nodes (2,048 cores), reveals that our analysis of the pharmaceutical dataset of interest scales up to 500X in data size. Specifically, the hierarchical clustering can deal with up to 4 GB of data without encoring in substantial slow down due to memory swap, while the N-D clustering can deal with up to 2 TB of data without encoring into major slow down due to the iterative exchange of densities in the MapReduce-MPI shuffle phase.

3.5 Accuracy

3.5.1 Platform

The accuracy tests are executed on the Geronimo cluster at the University of Delaware; the cluster is described in Section 3.4.1. Geronimo mounts the real datasets generated by Dockign@Home and thus provides easy access to the data generated at runtime.

3.5.2 Datasets

To assess the accuracy of our proposed method, we run docking trials on Docking@Home for 23 protein-ligand complexes for HIV protease (an aspartic acid protease protein), 21 protein-ligand complexes for Trypsin (a serine protease protein), and 12 protein-ligand complexes for P38alpha kinase (a serine/threonine kinase protein) over five years. Figure 3.15 shows the three proteins. For each protein-ligand complex,

we considered all the ligand conformations sampled with Docking@Home. The number of ligand conformations sampled for each complex is different, depending on the number of jobs successfully returned from the volunteer hosts and the number of conformations returned per job. On average, each complex contains around 210,000 ligand conformations.

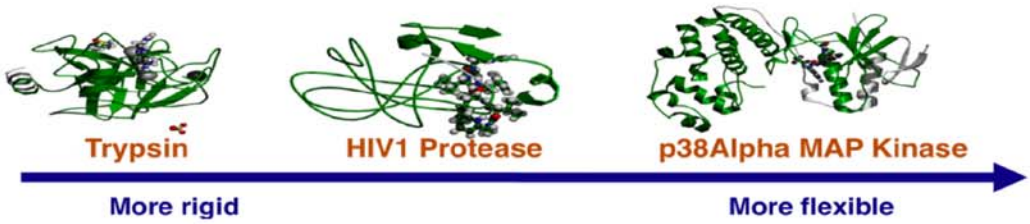


Figure 3.15: Three proteins whose results from the Docking@Home datasets are analyzed for this accuracy study.

HIV protease (HIV PR) is a protein in the HIV virus that is essential for its replication in human cells. While building a new HIV virus inside the human cell, HIV PR cleaves some newly synthesized viral protein in a particular fashion. The cleaved pieces are required in order to build a mature HIV virus. HIV PR has been considered a therapeutic target for preventing HIV proliferation. A drug that can bind to the active site of HIV PR or deactivate it will eventually stop the replication of HIV virus in human cells. Such a drug is called a protease inhibitor. Several protease inhibitors (e.g., saquinavir, ritonavir, indinavir, and nelfinavir) are available for treating HIV infection [5].

Trypsin is a protease that breaks down other proteins in the digestive system. Recent studies suggest that inhibitors of Trypsin can have potential application in breast cancer treatment. In this disease, Trypsin-like proteases activate protease-activated receptor 2 (PAR2), a protein in the tumor cell membrane. When activated, PAR2 causes the degradation of the extracellular matrix, resulting in the spread of the tumor cell from one place to the other (metastasis). Drugs can act as inhibitors by

deactivating the Trypsin-like protease and are, therefore, potential agents capable of stopping the spread of breast cancer [24].

P38alpha is the most flexible protein among the three proteins considered. P38alpha is also known as SAPK2a and MAPK14. It is involved in the regulation of cellular stress responses as well as the control of proliferation and survival of many cell types. Several promising compounds that inhibit P38alpha are being investigated as potential therapies for arthritic and inflammatory diseases [81].

3.5.3 Results and discussion

We study three key aspects in our accuracy tests: (1) the accuracy of our methods versus an accurate but not scalable method (i.e., our previous work based on probabilistic hierarchical clustering [27]) and a scalable but not accurate method (i.e., the naïve selection approach based on only the lowest conformation energy [27]); (2) the impact of the different mappings on the accuracy when using our method; and (3) the impact of the different density thresholds on our N-D search and selection.

For each protein, its accuracy is the number of complexes with captured near-native conformations over the total number of complexes for that protein. Note that a near-native conformation has an RMSD from the experimentally observed conformation that is smaller than or equal to two angstroms (Å). For our clustering, we apply the three mapping methods described in Section 3.3 (i.e., 3-D, 3-Dlog, and 6-D mappings) to the real Docking@Home data and reduce the ligand geometries into metadata points. We compare and contrast the three mappings as they allow us to progressively capture a richer range of aspects related to the ligand geometry and location in the protein pocket. We consider two criteria when selecting the density threshold (i.e., the minimum number of metadata points in the tree nodes selected by the N-D search). The first criterion is to set the density threshold to a fixed number equal to 500 metadata points. In other words, we always select the densest tree node with at least 500 ligand conformations (i.e., octant in the case of the 3-D and 3-Dlog mappings or 6-D subspace in the case of the 6-D mapping). The second criterion is to set the density

threshold equal to 0.5% of the number of ligands conformations in each protein-ligand complex dataset. For example, when the dataset contains 100,000 ligand conformations, the density threshold for this dataset is 500 points. When the dataset contains 500,000 ligand conformations, the density threshold for this dataset is 2,500 points. In both cases, we capture a near-native conformation if the arithmetic median of the conformations associated with the metadata point in the selected tree node is below or equal to two Å. The use of the median is preferred as the accuracy metric over the mean because it is less affected by extreme values, although the majority of our overall results are not very sensitive to whether the median or mean is used for selection [36].

For the probabilistic hierarchical clustering, the distance metric used to cluster each ligand is the RMSD of its atom coordinates versus all the other ligands already in the cluster. If a simulation converges, then the largest cluster with lower internal variance is likely the cluster that contains more near-native conformations. We capture a near-native conformation if the centroid of the selected cluster is a near-native conformation [27]. For the energy-based approach, we consider 100 D@H conformations selected based on their lowest energy versus the same crystal structure, which we denote as the naïve approach. Here, we identify the near-native conformation if the arithmetic median of the lowest energy conformations is below or equal to two Å.

When using our method, the probabilistic hierarchical clustering, or the energy-based approach, we perform the clustering and selection of the near-native candidates without using any information on the crystal structures available for the complexes. The crystal structures play an important role only in the validation phase when, for each complex, we calculate the RMSD of the clustering candidate with respect to its crystal structure. Table 3.3 summarizes the accuracy of the three approaches: (1) our method (N-D clustering) with a fixed density threshold equal to 500 and with density threshold equal to 0.5% of the dataset, (2) the probabilistic hierarchical clustering, and (3) the energy-based approach.

Clustering methods: N-D, hierarchical, energy-based: When comparing our methods with the two more traditional methods in Table 3.3, we observe how, for

all three proteins (i.e., HIV protease, Trypsin, and P38alpha), our N-D clustering always gives better accuracy. In particular, for the HIV protease, our clustering (using 3-D, 3-Dlog, or 6-D mappings and the two density threshold selection criteria) captures all the 23 near-native conformations (100%); the probabilistic hierarchical clustering method captures 20 of the 23 near-native conformations (87.0%); and the energy-based approach is able to identify only 8 of the 23 near-native conformations (34.8%). For Trypsin, our clustering captures 17 of the 21 near-native conformations (81.0%); the probabilistic hierarchical clustering method captures 16 of the 21 near-native conformations (76.2%); and the energy-based approach identifies only 5 of the 21 near-native conformations (23.8%). For the P38alpha kinase, our clustering captures 10 of the 12 near-native conformations (83.3%); the probabilistic hierarchical approach captures 6 of the 12 near-native conformations (50.0%); and the energy-based approach identifies 1 of the 12 near-native conformations (0.8%).

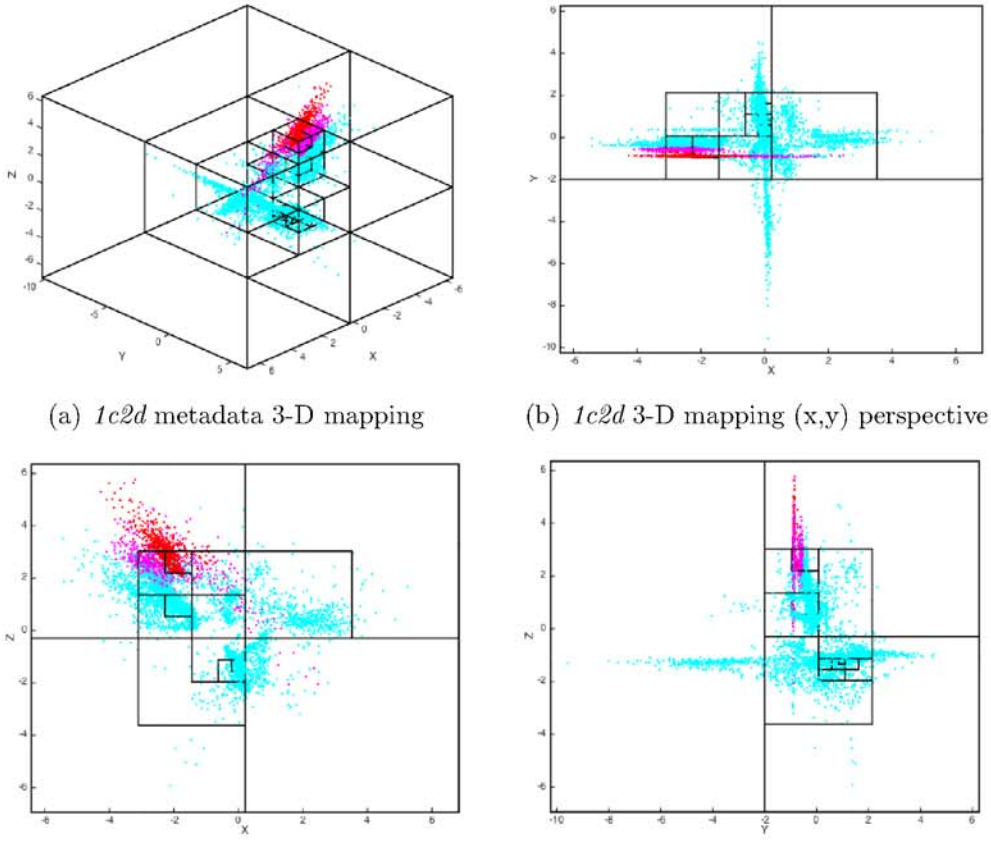
Metadata mapping: 3-D vs. 3-Dlog vs. 6-D: When mapping the ligand conformation into metadata using the three mapping techniques, we find no clear winner between the 3-D and the 6-D mappings when considering the HIV and P38alpha proteins, as shown in Table 3.3. In particular, for the HIV protease, the 3-D and 6-D mapping methods capture 21 of the 23 near-native conformations (91.3%) and 23 of the 23 near-native conformations (100%) using a density threshold equal to 500, respectively, and capture both 100% near-native conformations using a density threshold equal to 0.5% of the dataset, respectively. For the P38alpha protease, the 3-D and 6-D mapping methods capture 9 of the 12 near-native conformations (75%) and 10 of the 12 near-native conformations (83.3%) using a density threshold equal to 500, respectively, and capture both 83.3% near-native conformations using a density threshold equal to 0.5% of the dataset, respectively. These overall tendencies can be due to the fact that the ligands docked in these proteins' pockets are long and with a high degree of freedom. We note that the docking pockets considered in Docking@Home are relatively small and known a priori. If this were not the case (e.g., we deal with large docking regions and we do not know the exact location where the ligand conformation

is docked), then we would expect that a 6-D mapping was the right approach to pursue with our method. When dealing with a relatively rigid protein such as Trypsin, however, we observe that the ligands are relatively small and rigid (with very low degrees of freedom). In this case, the 3-Dlog mapping method achieves better accuracy than the 3-D and 6-D methods achieve. In particular, for the Trypsin protease, the 3-Dlog mapping method captures 16 of 21 (76.2%), and 17 of 21 (81.0%) near-native conformations using the two density threshold selection criteria.

The reduced flexibility of ligand conformations in the pocket explains why the 3-Dlog mapping works well for ligands docking in the Trypsin protease but not for ligands docking in the other two proteins. The ligands docked into Trypsin are very small and with limited flexibility as the protein itself is rigid. When a small and rigid ligand conformation is in a near-vertical position in a pocket, its slope is very large. If the conformation position slightly changes, the slope also changes significantly, because of the projections. In the case of Trypsin, some near-native conformations may have similar shapes and be in near-vertical positions, and their slopes may differ to the extent such that the mapping may not result in a dense enough subspace containing the metadata. By taking the log of the slopes, we slow the slope differences when dealing with vertical ligand conformations.

As an example, Figure 3.16 and Figure 3.17 show the 3-D metadata obtained by using the 3-D mapping and the 3-Dlog mapping respectively for ligand *1c2d* from protein Trypsin using 10,000 ligand conformations. Figure 3.16(a) shows the set of metadata obtained by using 3-D mapping. The points in cyan have RMSD less than 8 Å but larger than 2 Å, the points in pink have RMSD less than 2 Å but larger than 1 Å, and the points in red have RMSD less than 1 Å. The black subspaces denote the subspaces with densities larger than the 0.5% threshold. The deepest black subspace contains the ligand conformations identified by our clustering. As shown in Figure 3.16, many of the near-native metadata points (i.e., red points) have larger absolute coordinates values (i.e., in a near-vertical position). The red metadata points are spread out in the subspace to the extent that they do not form a dense-enough

cluster. Similar results using the 3-Dlog mapping method are shown in Figure 3.17. This time the red metadata points have much smaller absolute coordinates values, and the metadata obtained using the 3-Dlog mapping are less sensitive to changes when in the near-vertical position. Hence they form a denser subspace that our method can more easily identify.



(a) *1c2d* metadata 3-D mapping

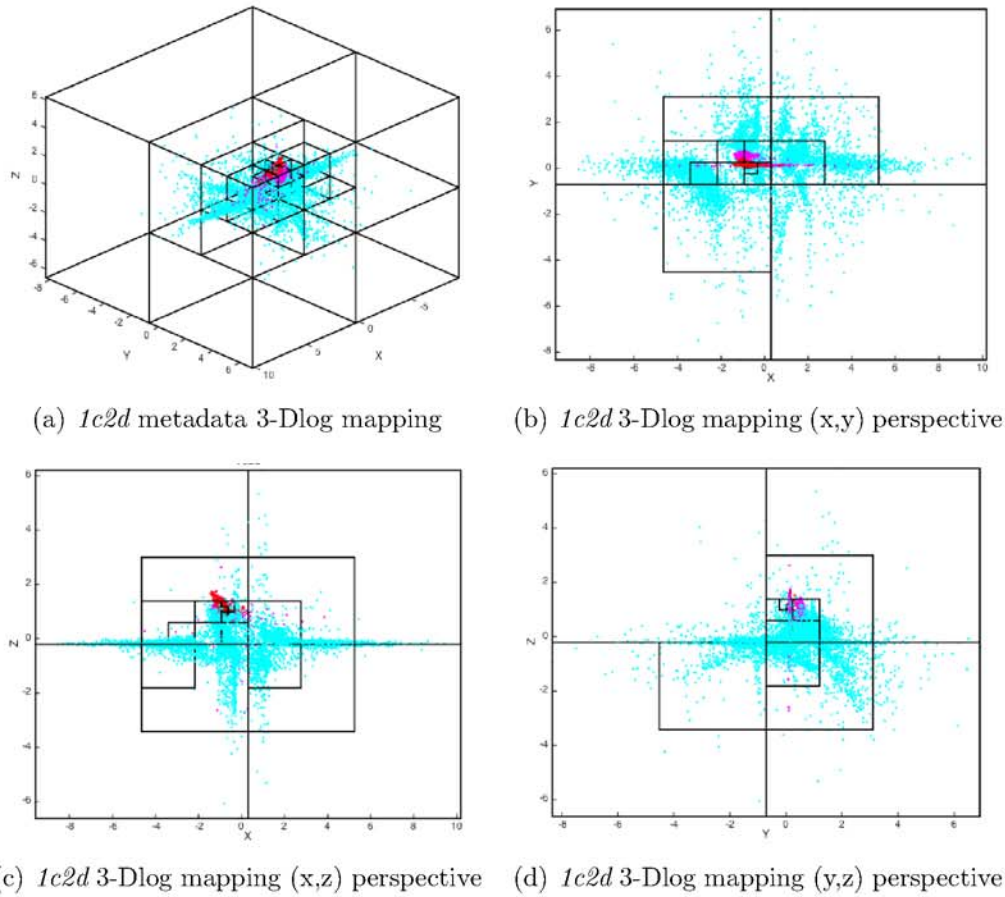
(b) *1c2d* 3-D mapping (x,y) perspective

(c) *1c2d* 3-D mapping (x,z) perspective

(d) *1c2d* 3-D mapping (y,z) perspective

Figure 3.16: Metadata for ligand *1c2d* using 3-D mapping (a), from the (x,y) perspective (b), the (x,z) perspective (c), and the (y,z) perspective (d).

As a validation, we examine the crystal structure of the *1c2d* ligand. Figure 3.18(a) shows the crystal structure of the ligand inside the protein pocket when we look at it from the z axis. Similarly, Figure 3.18(b) and 3.18(c) show the same crystal structure inside the protein pocket when we look at it from the x and y axis, respectively. We observe that the crystal structure is in a near-vertical position when



(a) *1c2d* metadata 3-Dlog mapping

(b) *1c2d* 3-Dlog mapping (x,y) perspective

(c) *1c2d* 3-Dlog mapping (x,z) perspective

(d) *1c2d* 3-Dlog mapping (y,z) perspective

Figure 3.17: Metadata for ligand *1c2d* using 3-Dlog mapping (a), from the (x,y) perspective (b), the (x,z) perspective (c), and the (y,z) perspective (d).

we look at it from the z axis. This results in a linear regression line with a relatively large slope when projecting the ligand conformation onto the 2-D (x, y) plane, hence a large value in the z coordinate of the 3-D metadata as shown in Figure 3.16(c).

Static vs. variable density threshold: When dealing with different thresholds for the selection of the densest subspace (i.e., 500 points or a variable-density threshold equal to 0.5%), we observe that using a variable-density threshold achieves better accuracy than using a fixed threshold equal to 500 for each individual dataset, as shown in Table 3.3. In particular, for the HIV protease, when using a variable density, the 3-D, 3-Dlog, and 6-D achieve 100%, 87.0%, and 100% accuracy, respectively; when using a fixed threshold, the 3-D, 3-Dlog, and 6-D achieve 91.3%, 73.9%, and 100%

(a) Crystal structure of *1c2d* from (x, y) perspective (b) Crystal structure of *1c2d* from (y, z) perspective (c) Crystal structure of *1c2d* from (z, x) perspective

Figure 3.18: Crystal structure of *1c2d* from different perspectives.

accuracy, respectively. For the Trypsin protease, when using a variable density, the 3-D, 3-Dlog, and 6-D achieve 71.4%, 81%, and 66.7% accuracy, respectively; when using a fixed threshold, the 3-D, 3-Dlog, and 6-D achieve 57.1%, 76.2%, and 61.9% accuracy, respectively. For the P38alpha protease, when using a variable density, the 3-D, 3-Dlog, and 6-D achieve 83.3%, 66.7%, and 83.3% accuracy, respectively; when using a fixed threshold, the 3-D, 3-Dlog, and 6-D achieve 75%, 58.3%, and 83.3% accuracy, respectively. This can be easily motivated by the large set of ligand conformations in our datasets and the need to adapt the thresholds to these large numbers.

3.6 Summary and Future Work

Summary: This chapter presents the application of the general data analysis method to the clustering problem in which we identify the geometries of ligand conformations and predict their probabilities of well-docking into a protein pocket based on their geometries. We discuss variations of the general method, including three ways to perform data reduction of the ligand conformations into metadata (i.e., 3-D, 3-Dlog, and 6-D mapping) and two ways to perform the metadata space search in parallel (i.e., GlobalToLocal and LocalToGlobal).

We evaluate the performance of our method on 64 nodes of Gordon at the San Diego Supercomputer Center and on 256 nodes of Fusion at Argonne National Laboratory, considering fifteen scenarios by combining five different clustering configurations and three physical distributions of the data. At a smaller scale of 64 nodes of Gordon, our method shows that when exchanging smaller messages of scalar property aggregates in the LocalToGlobal variation, the execution time is not sensitive to metadata distributions in the datasets: it varies only 3.4% across the five logical distributions. The GlobalToLocal variation, on the other hand, is sensitive to the metadata distributions. At a larger scale of 256 nodes of Fusion, our method shows that the data shuffle stage dominates the execution time, and the Shuffle time is affected by the metadata distribution as well as the level of trees explored in the N-dimensional tree. From the point of view of the scalability, our method is approximately 400X faster in execution and can analyze approximately 500X larger datasets compared with the hierarchical clustering, because it does not require any direct movement and comparisons of ligand conformations.

The accuracy results on 56 ligands docking in 3 proteins (i.e., HIV, Trypsin, and P38alpha) show that our method can achieve 100%, 81.0%, and 83.3% clustering accuracy, respectively, whereas the hierarchical-based clustering achieves 87.0%, 76.2%, and 50.0% clustering accuracy and the energy-based scoring achieve only 34.8%, 23.8%, and 0.8% accuracy.

Future work: Future work lies in two directions: (1) investigate how to automatically choose the best mapping method (i.e., 3-D, 3-Dlog, and 6-D) for individual ligand based on various features of the ligand and the protein pocket, and (2) study how to exchange and aggregate the partial densities more efficiently in the LocalToGlobal variation at the extreme scale when the metadata is not tightly converged.

For the first research direction, we empirically observed that for relatively rigid proteins such as Trypsin, and for ligands such as *1c2d* whose crystal structure is in near-vertical position in the x, y, or z dimensions inside the docking pocket, the 3-Dlog mapping method tends to produce results with higher accuracy. However, for

relatively flexible proteins such as P38alpha, and for relatively long and flexible (with high degrees of freedom) ligands, the 3-D and 6-D mapping methods achieve better accuracy. Moreover, when considering docking pockets that involve large regions for which the exact docking location is not known a priori, the 6-D mapping method can potentially give higher accuracy because it takes into consideration the ligand positions in the docking pocket as well as the ligand shape and orientation. Future work should study automatic techniques to quantify the accuracy of the 3-D, 3-Dlog, and 6-D mappings for various types of ligands and protein pockets at runtime and without the scientist's intervention.

For the second research direction, in our large-scale performance study, we observed that the Shuffle times, in which the partial densities of subspaces (i.e., scalar property aggregates) are communicated, is affected by the logical distributions of the metadata. In other words, when data loosely converge, the Shuffle times are responsible for performance decays. Moreover, at the largest scale (i.e., 2,048 compute cores), the Shuffle times take up more than half of the total execution time. Future studies should investigate how to minimize the communication times associated with the shuffling calls. One possibility is to consider both the metadata distribution and the topology of the nodes when shuffling calls are performed. Instead of exchanging densities among nodes based on the N-D subspace identifiers, we envision a shuffling phase that exchanges densities among topologically close nodes. Another possibility is to design a more efficient and scalable library than MapReduce-MPI to communicate data among compute nodes using advanced parallel computing techniques such as overlapping communication with computation and multithreading.

Table 3.1: Total MapReduce times in seconds across processes broken down into distinctive components: Map (M), Shuffle (S), Overhead (O), Reduce (R), and Total (T).

Uniform										
	1D		1S		UN		2D		2S	
	GL	LG	GL	LG	GL	LG	GL	LG	GL	LG
M	2.64E+05	2.68E+05	2.65E+05	2.68E+05	2.65E+05	2.68E+05	2.65E+05	2.68E+05	2.64E+05	2.69E+05
S	2.35E+04	2.65E+03	1.40E+03	2.74E+03	1.32E+03	1.82E+02	9.88E+03	2.71E+03	2.63E+03	1.56E+03
O	7.77E+06	5.86E+04	1.01E+05	6.18E+04	6.69E+04	5.39E+04	3.07E+06	5.91E+04	1.26E+05	5.45E+04
R	8.63E+03	6.95E+00	1.64E+03	8.38E+00	1.72E+03	1.32E+00	6.50E+03	7.98E+00	2.06E+03	6.43E+00
T	8.06E+06	3.29E+05	3.69E+05	3.33E+05	3.35E+05	3.22E+05	3.35E+06	3.30E+05	3.95E+05	3.25E+05
Round-robin										
	1D		1S		UN		2D		2S	
	GL	LG	GL	LG	GL	LG	GL	LG	GL	LG
M	2.64E+05	2.69E+05	2.64E+05	2.70E+05	2.65E+05	2.68E+05	2.64E+05	2.69E+05	2.65E+05	2.68E+05
S	1.91E+04	3.65E+03	1.33E+03	5.04E+03	1.47E+03	2.50E+03	1.04E+04	3.85E+03	2.68E+03	3.98E+03
O	8.49E+06	7.10E+04	1.13E+05	5.62E+04	6.02E+04	6.55E+04	3.30E+06	6.18E+04	1.51E+05	6.04E+04
R	9.35E+03	1.06E+01	1.90E+03	1.31E+01	1.93E+03	6.16E+00	7.09E+03	1.11E+01	2.56E+03	1.13E+01
T	8.78E+06	3.43E+05	3.80E+05	3.31E+05	3.28E+05	3.36E+05	3.59E+06	3.35E+05	4.21E+05	3.33E+05
Random										
	1D		1S		UN		2D		2S	
	GL	LG	GL	LG	GL	LG	GL	LG	GL	LG
M	2.66E+05	2.69E+05	2.65E+05	2.69E+05	2.65E+05	2.69E+05	2.66E+05	2.69E+05	2.64E+05	2.69E+05
S	1.94E+04	3.73E+03	1.35E+03	4.92E+03	1.25E+03	2.53E+03	1.02E+04	3.77E+03	3.17E+03	3.98E+03
O	8.63E+06	6.16E+04	1.14E+05	5.72E+04	6.09E+04	6.28E+04	3.62E+06	5.67E+04	1.66E+05	6.28E+04
R	9.52E+03	1.06E+01	2.03E+03	1.30E+01	1.96E+03	6.12E+00	7.81E+03	1.09E+01	2.99E+03	1.07E+01
T	8.93E+06	3.35E+05	3.82E+05	3.32E+05	3.29E+05	3.34E+05	3.90E+06	3.30E+05	4.37E+05	3.36E+05

Table 3.2: The four levels explored in our tree search for the five logical distributions when representing each point with a 15-digit key.

Logical	Shuffle			
Distribution	I	II	III	IV
1D	8	12	10	9
1S	8	4	6	5
2D	8	12	10	9
2S	8	4	6	5
UN	8	4	3	2

Table 3.3: Comparison of the number of hits for different scoring approaches: our clustering with density threshold equal to 500 and equal to 0.5% of the total number of points, a probabilistic hierarchical clustering, and an energy-based scoring method.

Protein	N-D Clustering with 500 Points			N-D Clustering with 0.5% Points			Hierarchical	Min. Energy
	3-D	3-Dlog	6-D	3-D	3-Dlog	6-D		
HIV	21/23(91.3%)	17/23(73.9%)	**23/23(100.0%)**	**23/23(100.0%)**	20/23(87.0%)	**23/23(100.0%)**	20/23(87.0%)	8/23(34.8%)
Trypsin	12/21(57.1%)	**16/21(76.2%)**	13/21(61.9%)	15/21(71.4%)	**17/21(81.0%)**	14/21(66.7%)	**16/21(76.2%)**	5/21(23.8%)
P38alpha	9/12(75.0%)	7/12(58.3%)	**10/12(83.3%)**	**10/12(83.3%)**	8/12(66.7%)	**10/12(83.3%)**	6/12(50.0%)	1/12(0.8%)
All	42/56(75.0%)	40/56(71.4%)	**46/56(82.1%)**	**48/56(85.7%)**	45/56(80.4%)	47/56(83.9%)	42/56(75.0%)	14/56(25.0%)

Chapter 4

CLUSTERING OF PROTEIN FOLDING TRAJECTORIES

Protein folding simulations search for trajectories leading to conformations close to the native (folded) protein structure originating from an unfolded conformation. During the folding process, the protein changes its conformations into what are called meta-stable and transition stages. In a meta-stable stage, consecutive protein conformations have similar geometric shapes and thus display small structural variations. In a transition stage, consecutive protein conformations change from one meta-stable stage to another and thus exhibit large structural variations. The study of folding trajectories includes intra-trajectory analysis, which aims to identify meta-stable and transition stages, and inter-trajectory analysis, which studies the ability of multiple trajectories to explore overlapping folding space and eventually converge to the same conformation. Traditionally, both intra-trajectory and inter-trajectory analysis methods follow a centralized approach that moves the trajectory datasets to one centralized node and processes the data only after the simulations are complete.

The third analysis problem in this thesis is to identify recurrent patterns in the folding trajectories locally, without moving the trajectory datasets. To this end, we consider each protein conformation in the trajectory in isolation and extract its geometric shape as a single 3-D metadata point consisting of the three largest eigenvalues of the atoms' distance matrix. A hierarchical fuzzy c-means clustering is performed on subsets of local metadata points to map a trajectory's subsegment into meta-stable and transition stages. The performance and accuracy study of the method are performed on the 207 GB folding trajectory dataset of the protein HP-35 NleNle (i.e., a variant of the villin headpiece subdomain containing alpha helix) and 5.4 TB folding trajectories

of the protein BPTI (i.e., bovine pancreatic trypsin inhibitor containing both alpha helix and beta sheet).

The rest of this chapter is organized as follows. Section 4.1 gives background information on protein folding trajectories, the intra- and inter-trajectory analysis problems, and the Parallel MATLAB framework. Section 4.2 reviews the related work on traditional approaches for trajectory analysis and other big data analysis problems in MapReduce. Section 4.3 presents the methodology. Sections 4.4 and 4.5 present the performance scalability and accuracy evaluation on the NleNle and BPTI datasets, respectively. Section 4.6 summarizes the research results and discusses future work.

4.1 Background

4.1.1 Protein folding trajectories

Protein folding simulations generate trajectories that reveal ensembles of diverse structures as well as folding kinetics involved in actual protein folding process. During the folding process, the protein changes its conformations into what are called meta-stable and transition stages. In a meta-stable stage, consecutive protein conformations have similar geometric shapes and thus display small structural variations. In a transition stage, consecutive protein conformations change from one meta-stable stage to another and thus exhibit large structural variations.

With the power of high-performance computing platforms, folding trajectories are generated in large scale on large distributed memory systems. For a typical trajectory dataset, each trajectory is divided into consecutive frames. Each frame contains a fixed number of consecutively sampled protein conformations during the simulation. Each frame is generated and stored separately on the node of the distributed memory system. Ideally, the analyses of such trajectory datasets should be in a distributed fashion as are the generation and storage of the datasets.

4.1.2 Intra- and inter-trajectory analysis

Molecular dynamics (MD) simulations generate large datasets of folding trajectories of biomolecules such as proteins. The trajectories explore a large conformational space of protein structures and thus provide information for investigating the structure-function relationship, which is essential in protein folding studies. By looking at the groups of conformations that share similar geometric features and the transitions between these groups, scientists can learn more about the protein folding process. Such knowledge has many practical applications including a better understanding of protein misfolding and of how to speed drug design for certain related diseases [22].

The study of folding trajectories includes intra-trajectory analysis, which aims to identify meta-stable and transition stages, and inter-trajectory analysis, which studies the ability of multiple trajectories to explore overlapping folding space and eventually converge to the same conformation. The analyses typically rely on clustering techniques and can reveal important information about structural ensembles and the pathways a protein can go through during its folding process.

4.1.3 Parallel MATLAB

As a high-level scripting language, MATLAB has been considered an inefficient language for high-performance computing(HPC) [52]. Recently, however, because of the advances in its memory model and its finer programming granularity, Parallel MATLAB has emerged as a suitable language for HPC, as suggested by the HPC community in a spring 2013 meeting at Argonne National Laboratory [53]. Today, several supercomputers support Parallel MATLAB, including Gordon, the SDSC cluster used for our tests.

In Parallel MATLAB, users can launch parallel jobs, comprising a set of tasks, in what is called a client session. This session usually runs on the local machine used by the programmer. Launched jobs are submitted to a cluster or supercomputer to execute the tasks on its processors. Each MATLAB session running a task on a processor is called a worker. MATLAB provides the user with the functionality to define jobs

and tasks. A job represents an abstraction of the application to be solved through the execution of multiple tasks. Tasks are blocks of code with specific functionalities. For example, consider an application performing matrix multiplication over very large matrices. In this case, a job is the matrix multiplication, and tasks are identical sets of operations performed over partitions of data in which the multiplication was decomposed. As another example, consider a classifier exploiting functional parallelism. In this case, the tasks are different processing units including the preprocessing of the data and the training module. To showcase that our method does not depend on a particular MapReduce library or framework, we implemented our method in Parallel MATLAB.

4.2 Limits of Current Practice

4.2.1 Traditional methods for trajectory analysis

When performed in a distributed fashion, trajectory analysis targets simple properties such as the number and position of ion molecules that permeate a channel. For example, Tu et al. proposed a parallel framework called HiMach to analyze long MD trajectories [80]. HiMach's scalable and user-friendly MapReduce-style programming interface exclusively analyzes simple statistical data of long trajectories in which the analysis operation is naturally parallel. In contrast, our work considers more sophisticated geometric features of protein conformations on large datasets in which the analysis operation is traditionally considered as naturally centralized.

Another group of research efforts focuses on various statistical information of folding trajectories. Work in this direction includes Ensign et al. [26] and Shaw et al. [69]. Ensign et al. investigated the heterogeneity in folding trajectories by looking at selected characteristics such as the RMSDs of the alpha helices between sampled protein conformations and the energy-minimized native structure. Shaw et al. studied the atomic-level of protein structure dynamics by looking at the RMSDs as well as other statistics of sampled protein conformations characterizations (e.g., computational f-values for selected mutants, survival probability distributions for the internal water

molecules of a protein). In our work, we focus on the more complex geometric shape for groups of conformations in the trajectory without making reference to the native structure.

More sophisticated trajectory analyses, such as the intra-trajectory and inter-trajectory analysis considered in this thesis, are traditionally centralized, target small protein molecules, and work for short folding trajectories. Algorithmically the analysis requires constructing a centralized dissimilarity matrix using all the trajectory data, reducing the dimensionality of the matrix, and then clustering the low-dimensional matrix. The work of Best el al. and Phillips et al. follows this method [58, 7]. In general the centralized nature of the algorithms in Best el al. and Phillips et al. make their analysis inefficient when dealing with large proteins and long trajectories. Our work differs from these approaches in three ways. First, it considers and supports the analysis for sophisticated protein conformations; we look at the protein HP-35 NleNle (i.e., a variant of the villin headpiece subdomain) with 35 amino acids, whereas Best et al. focused on a single molecule with only three amino acids. Second, we use the real folding trajectories of the protein HP-35 NleNle for validation, in contrast to the synthetic folding trajectories used in the work of Phillips et al. Third, our work exhibits high scalability, whereas the work of both Best el al. and Phillips et al. does not.

4.2.2 Other big data analysis problems

More general big data analysis problems can be categorized as small analytics on big volumes of data and big analytics on big volumes of data [82]. Small analytics refers to running simple count, sum, min, max and average operations. The operations are usually data parallel and extract simple statistical information from large amounts of data. For these problems, there are multiple research efforts using in-situ analysis to improve performance. For example, Bennett et al. introduced a hybrid approach that combines in-situ and in-transit processing for extreme-scale scientific analysis such as topological analysis, descriptive statistics, and visualization [6]. Their approach requires dealing with the in-transit overhead, however, whereas our method enables

purely in-situ data analysis without the overhead of in-transit operations. The work of Lakshminarasimhan et al. is another relevant example [44]. Their work combines in-situ data encoding and index generation techniques exclusively for efficient query processing. Arguably, this approach reduces the time to generate indexes and thus indirectly reduces the time to analysis. Our work, however, enables efficient in-situ analysis for geometric shape analysis without the need for indexing.

Big analytics refers to data clustering, regression, machine learning, and other more complex operations. For these big data analyses, several research efforts have focused on improving the performance of the analysis methods by reducing data movement across nodes. For example, as discussed in Section 3.2.2, Cordeiro et al. optimized a density-based clustering approach by finding a trade-off between disk I/O and network communication [18]. The three algorithms presented in their work rely on local clustering of subregions and the merging of local results into a global solution. The algorithm requires iterative data reading from disk. In contrast, our method performs only a single-pass analysis on the whole dataset. The work of Liu et al. is another example of big analytics methods with the same goal of reducing data movement [48]. Their work caches sub-results of tasks for later use in order to reduce data movement; therefore, it is more efficient when a large number of tasks share the same subtasks and use the same sub-results. On the other hand, our work considers how to reduce data movement in the context of a single task (i.e., the intra- and inter-analysis for the whole dataset). To the best of our knowledge, our work is the first to rethink and redesign the analysis method on large-scale protein folding trajectories in order to provide a one-pass, distributed online, in-situ data analysis.

4.3 Methodology

The method of identifying recurrent patterns in folding trajectories of proteins consists of two steps. Given a trajectory composed of frames generated in a distributed fashion and containing up to 400 consecutively folding protein conformations,

our method first extracts relevant features from each conformation into a local meta-data representation. In our case, the relevant feature is the local knowledge on the geometric conformation of the protein structure. Second, the method clusters each conformation to either a meta-stable or a transition stage, using the local knowledge on the conformational features of the structures in the frame only, and then rebuilds the global knowledge (i.e., stages and structural phase space explored) of the given trajectories through a reduction operation [85]. Figure 4.1 shows an overview of our approach.

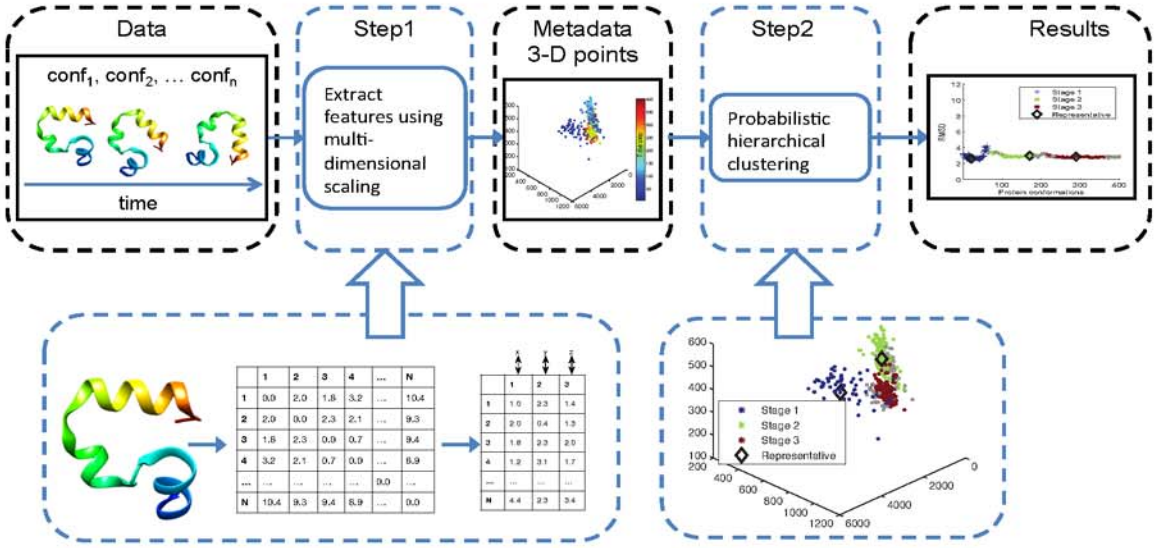
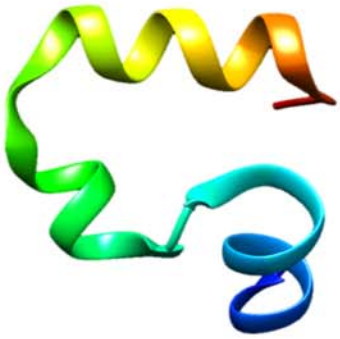


Figure 4.1: Overview of the clustering of protein folding trajectories.

4.3.1 Extraction of conformational features

Our method extracts the shape of each single folding conformation into a meta-data representation that preserves the relevant information contained in the data but reduces the overall data size. Rather than working with the atom coordinates of the complete protein molecule, our method first represents each protein conformation using an $N * N$ distance matrix (DM), where N is the number of backbone atoms in the protein. More specifically, the matrix contains the distance from each backbone

atom to the other backbone atoms in the same protein conformation. Figure 4.2(a) shows an example of a folding conformation for the protein HP-35 NleNle (i.e., a variant of the villin headpiece subdomain) taken from folding trajectories generated by Folding@Home [26]. Figure 4.2(b) shows part of the corresponding distance matrix representing the same conformation in a different format. The representation of each protein conformation is transformed from a set of $3 * M$ floating-point numbers (where 3 identifies the coordinates in the Cartesian space of each atom and M is the number of atoms in the entire protein, with $M > N$) to an $N * N$ matrix of floating-point numbers.



	1	2	3	4	...	N
1	0.0	2.0	1.8	3.2	...	10.4
2	2.0	0.0	2.3	2.1	...	9.3
3	1.8	2.3	0.0	0.7	...	9.4
4	3.2	2.1	0.7	0.0	...	8.9
...	...	...	...	...	0.0	...
N	10.4	9.3	9.4	8.9	...	0.0

(a) (b)

	x	y	z
	1	2	3
1	1.0	2.3	1.4
2	2.0	0.4	1.3
3	1.8	2.3	2.0
4	1.2	3.1	1.7
...	...	...	...
N	4.4	2.3	3.4

(c)

Figure 4.2: One conformation of the villin HP-35 protein (a); part of its distance matrix using only its backbone atoms in the conformation (b); and three eigenvectors and the associated eigenvalues capturing and synthesizing the conformation geometry (c).

Once the method has mapped each conformation to a distance matrix, it applies classical multidimensional scaling (MDS) to each distance matrix separately. MDS reduces the $N * N$ symmetric distance matrix into a lower-dimensional matrix while maintaining the original information on the protein conformation (i.e., the distance of each backbone atom to all the other backbone atoms). For simplicity and ease of human interpretation, we applied MDS to reduce the data dimensionality to an $N * 3$ matrix. MDS also generates a set of three $N * 1$ column vectors (q_1, q_2, q_3), and each eigenvector comes with its eigenvalue $(\lambda_1, \lambda_2, \lambda_3)$. Each eigenvalue represents the

amount of variations in the data associated with the corresponding eigenvector [8]. In this case we use the information contained in the leading eigenvalues to summarize the conformational features of the protein at a given time in the folding process. Thus, we are able to reduce the protein dimensionality into a single point in the 3-D Euclidean space. In other words, the three eigenvalues represent the variance or curves of the backbone atoms with respect to each other in the protein's alpha helices or beta sheets. Figure 4.2(c) shows the three eigenvalues and eigenvectors obtained for the matrix in Figure 4.2(b). The figure also shows the reduced representation of the protein conformation into the 3-D point. In summary, the method maps each protein conformation from $3*M$ atom's coordinates to $3*1$ floating-point numbers. It performs the mapping for each conformation in a frame separately from the other mapped conformations in a concurrent manner.

By mapping geometries into single 3-D points and working on the points rather than the trajectory frames, we hypothesized that close 3-D points in the new three-dimensional space were generated by similar protein conformations in the original folding trajectory. We empirically validated this hypothesis using the whole villin dataset consisting of 451 trajectories. Each trajectory contains around 30K protein conformations. For each trajectory, we performed a two-sided Pearson correlation coefficient test on the following two variables: $RMSD_{dm}$ and $RMSD_{3D}$. $RMSD_{dm}$ is the RMSD between the distance matrix of the crystal structure and the distance matrix of each protein conformation in the whole trajectory. Similarly, $RMSD_{3D}$ is the RMSD between the 3-D point of the crystal structure and the 3-D point of each protein conformation in the whole trajectory. As discussed in Section 3.3.1, Pearson coefficients can fall in the range of [-1, 1]. A value close to -1 means that there is a strong negative linear correlation between the two variables in a test. A value close to 1 means that there is a strong positive linear correlation between the two variables in a test. More specifically, a value equal to or larger than 0.7 denotes a strong linear correlation, and a value between 0.5 and 0.7 denotes a moderate linear correlation [73]. In our case, if the 3-D points that are close together have similar protein conformations, we should observe

a linear correlation between the two variables $RMSD_{dm}$ and $RMSD_{3D}$. As shown in Table 4.1, in 341 of the 451 trajectories (i.e., 75.6%), the $RMSD_{3D}$ and $RMSD_{dm}$ are strongly correlated. In 78 of 451 (i.e., 17.3%), the two variables are moderately correlated. In all, we empirically observed that in 92.9% of the trajectories, our method is effective in maintaining the conformational information.

Table 4.1: Number of trajectories with Pearson coefficient (co) in range for the 451 trajectories

Condition	$co \geq 0.7$	$0.7 > co \geq 0.5$	$0.5 > co$
Count	341/451 (75.6%)	78/451 (17.3%)	32/451 (7.1%)

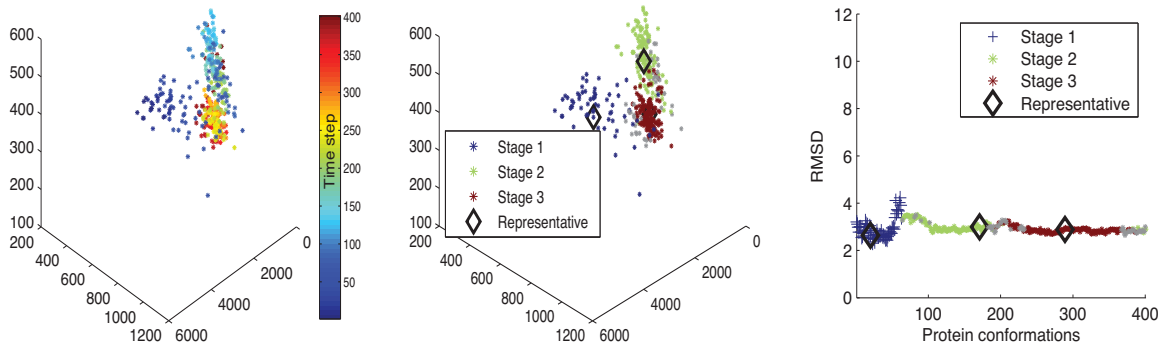
4.3.2 Clustering of meta-stable and transition stages

As the folding evolves, the protein changes between meta-stable and transition stages. Each frame composing a trajectory may contain up to 400 protein conformations that can be clustered into one or more of these two stage categories. Rather than clustering the 400 conformations based on their N atomic coordinates, our method works on the 400 3-D points generated above, each of which represents a protein conformation, and then computes the probability that each point belongs to a meta-stable or transition stage within a frame. Probabilities are computed by using a fuzzy clustering technique. More specifically, the method uses a hierarchical fuzzy c-means clustering to classify the points into meta-stable and transition stages for each local frame. By taking all the 3-D points in one frame as input, the method partitions the points into two stages using the fuzzy c-means algorithm [14]. From here each point has a certain probability of belonging to one of two initial stages. The points that do not strongly belong to either stage (probability < 0.6) are removed. Our method then performs a Welch's t-test with p-value less than 0.01 to assess the equality of the two stages. If the two stages are equal according to the t-test, then the classification process finishes having identified one single stage within the frame. If the two stages are not equal, then

the stage with more points inside is selected as the new input. The method iterates using the new input throughout the process by mapping the selected points into two new stages, removing those points that do not strongly belong, and testing the equality of the two stages. For each identified stage, the method selects the 3-D point with the highest probability to be the representative of this stage and renames it as such. The method also computes a within-cluster or intra-cluster variance, which is basically the squared distance from each point in the stage to the representative [54]. The method then uses the within-cluster variance to automatically classify stages into meta-stable or transitional. Empirically we observed that when the variance is less than 100, the stage can be classified as meta-stable; otherwise, the stage can be classified as transitional. As a result, we partition the frame conformations into stages by using only the generated 3-D points.

Figure 4.3(a) shows the 400 3-D points in one frame of the folding villin. The points are colored to change from blue to red to show the simulation advancing from time step 1 to time step 400. Figure 4.3(b) shows the three stages (clusters) for the same frame identified by our method; in this figure, all points belonging to the same color correspond to a single stage. The method also selects one conformation within each cluster as a representative of the associated stages (black diamond marks in the figure). Figure 4.3(c) maps back the three stages to the 400 conformations in terms of the root mean square deviation (RMSD) of each folding conformation from the folded protein observed with crystal structure techniques. More specifically, the x-axis represents the 400 protein conformations as the simulation steps proceed, and the y-axis represents the RMSD of each conformation with respect to the crystal structure. The asterisks mean that the given stage is a meta-stable stage, while the crosses mean that the given stage is a transition stage.

In order to rebuild the global knowledge of the trajectory stages (intra-trajectory analysis) as well as to explore overlapping structural phase space of all the trajectories (inter-trajectory analysis), the local knowledge collected with the two steps described above is ultimately collected into a global knowledge on a single node by using a



(a) (b) (c)

Figure 4.3: 3-D points in one frame sorted based on the time generation of the associated conformation (a); the three clusters and representatives identified by our method (b); and the mapping of the stages back to the folding conformation RMSDs from native protein conformation (c).

reduction operation. In intra-trajectory analysis, the global knowledge comprises the representatives for each stage in each frame and the stage classifications indicating whether it is a meta-stable or a transition stage. In inter-trajectory analysis, the global knowledge comprises the representative 3-D point of each stage in all the frames of all the trajectories. The reduction operation enables scientists to reconstruct a global knowledge of the trajectories from the metadata with limited data storage and movement.

4.3.3 Integration of the clustering algorithm into Parallel MATLAB

We implemented the MapReduce-style framework in Parallel MATLAB. As recall from Section 4.1.3, Parallel MATLAB supports the concept of job and tasks. A job consists of a set of tasks that can be executed in parallel. Users submit the job to the distributed memory system (e.g., the Gordon supercomputer), and the set of tasks run in parallel on nodes of the system.

In our implementation, a MATLAB job is the intra- and inter-trajectory analysis on the whole trajectory dataset. The job comprises identical tasks executing in

parallel for the three consecutive functional modules in our method (i.e., the conformational feature extraction, the per-frame stage classification, and the rebuilding of a global knowledge). When in execution, each worker performs the task on its own local data. There is a reduction communication at the end of the tasks, in which representatives' metadata of stages are sent to one node to rebuild the recurrent stages that the trajectory goes through. Note that the representatives information is significantly smaller comparing to the original trajectory dataset.

In our implementation, we take advantage of data parallelism whenever possible. For this purpose we use the SPMD constructions provided by MATLAB to enable workers to automatically execute code blocks within SPMD keywords in parallel; each worker operates on its local data assuring the one-pass, distributed online characterization of an in situ analysis.

4.4 Performance

4.4.1 Platform

The hardware platform used in this work is Gordon, which was also used for the tests on the ligand datasets discussed in Section 3.4.1. Gordon provides support for Parallel MATLAB execution. We use 16 Gordon compute nodes (i.e., 256 cores) for our performance and accuracy tests.

4.4.2 Datasets

The dataset we are working with is the protein folding trajectories of a variant of the villin headpiece subdomain (HP-35 NleNle) that was generated by Folding@Home using the volunteer computing paradigm (i.e., a distributed memory system) [26]. The villin protein contains 35 amino acids and 576 atoms. The whole dataset is available to the public and consists of 451 trajectories, starting from nine initial conformations. Each trajectory is segmented into multiple frames, which are each generated on independent computers. The dataset has 19,483 frames, each containing 400 consecutively folding conformations or snapshots. The whole dataset is 203 GB, and each frame is

approximately 11 MB in size. In order to emulate the condition in which frames are distributed in a scattered way across the nodes of Gordon, the dataset is distributed on the compute nodes in a round-robin fashion.

4.4.3 Results and discussion

We study two important performance aspects of our method: first, whether it meets the requirements of the in-situ analysis compared to traditional methods, and second, whether it scales as both the data size and the number of cores grow.

Proposed distributed clustering vs. sequential clustering of Phillips: To assess whether our method meets the requirements of the in-situ analysis, we compare its performance with that of the traditional approach adopted in Phillips' work, which is the most recent and complete study of intra-trajectory analysis [58]. To ensure a fair comparison, we rigorously implemented the algorithm presented in Phillips' work in Parallel MATLAB and exploited all the parallelism available in the algorithm. We compared the two methods in terms of execution times, memory usage, and size of data exchanged among nodes (i.e., the three main criteria that an in-situ analysis is expected to meet). Because Phillips performs only intra-trajectory analysis on single trajectories, we selected the longest trajectory in the dataset and considered its 50 frames and 20,000 protein conformations. Phillips' method moves the 50 trajectories' frames to one single node and builds a dissimilarity matrix containing the distance between each pair of the protein conformations in the whole trajectory. The method reduces the dimensionality of the dissimilarity matrix and performs the k-mean clustering on the lower dimensional matrix for different values of k. Since k is not known a priori and the selection of the initial centroids can affect the accuracy of the clustering results, Phillips proposes to use four values of k equal to 3, 5, 10, and 15 and perform each clustering with 10 random initial centroids. To maximize the parallelism in Phillips' method, we ran each k value with a set of random centroids in parallel by one MATLAB worker. The 40 partial results are ultimately reduced to a single node. In our method, the same trajectory is processed by 50 MATLAB workers in parallel, and each worker operates

on one of the 50 frames. The partial results are then reduced to a single node as in Phillips' method. We repeated the runs for both methods ten times and reported the average values observed.

The performance results are shown in Figure 4.4, which includes the execution times and shows how our method runs the intra-trajectory analysis in 41.5 seconds in contrast to the 10,116 seconds (approximately 3 hours) that Phillips' method required to analyze the same trajectory. Phillips' method spends most of the execution time working on the real data. The most time-demanding phases of Phillips' algorithm are building the matrix to store all the atom coordinates for the entire trajectory (approx. 23% of the average execution time); constructing the dissimilarity matrix (approx. 51% of the average execution time); and computing the normalized graph Laplacian of the dissimilarity matrix (approx. 24% of the average execution time). Our method has a shorter execution time since it generates and analyzes metadata. Figure 4.4(b) shows that our method uses 6.9 MB per core to store the local data needed for the analysis while the traditional method uses approximately 16 GB per core. Our method uses significantly less memory than the traditional method does, since we kept only the distance matrix of each conformation and the corresponding 3-D points of the local frame in memory. In contrast, Phillips' method keeps the distance matrix, the centralized dissimilarity matrix, and the dimensional-reduced matrix of the entire trajectory in memory. Figure 4.4(c) shows that our method moves 4.4 KB local results to a single node in the final reduction step to rebuild the global knowledge, whereas the traditional method moves 539 MB of data to each core performing a single k-mean clustering with one set of k randomly selected centroids (i.e., the node receives 50 frames each of approximately 11 MB). Our method significantly reduces the size of communication since it uses only the local reduced results (i.e., a selection of the 3-D points) to rebuild the global knowledge. Overall, this evaluation shows that our method meets the requirements of in-situ analysis by reducing the execution time of more than 2 orders of magnitude, using 4 orders of magnitude less memory, and reducing the size of the moved data more than 3 orders of magnitude. Similar behaviors have been

observed when performing the intra-trajectory analysis for the other trajectories in the dataset of interest.

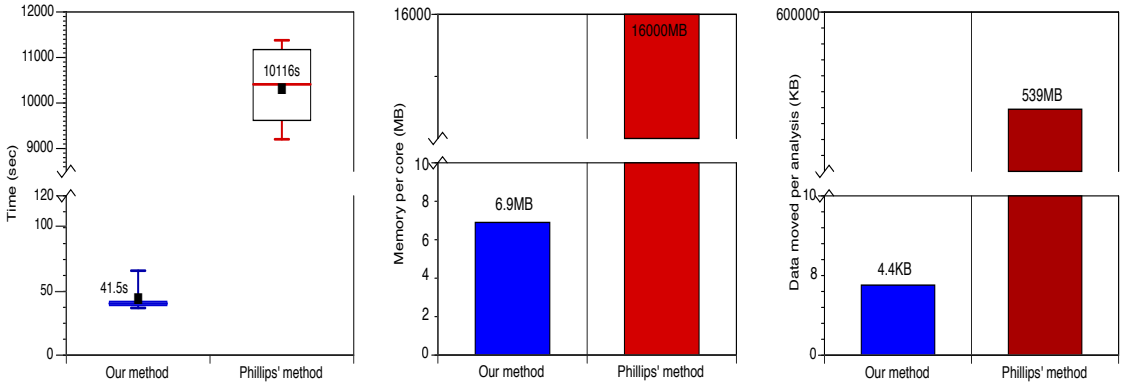


Figure 4.4: Comparison of our method with the traditional method proposed by Phillips in terms of execution time (a), memory usage per core (b), and data moved for the analysis (c).

Study of weak scalability: To assess whether our method scales for a large dataset, we use all of the 203 GB folding trajectories of the villin HP-35 dataset. The scalability study consists of the inter-trajectory analysis for the multiple trajectories to explore overlapping folding spaces. We measured the communication and computation times of this analysis and evaluated the weak scalability of our method as the size of the distributed memory system (i.e., the number of cores used on Gordon) and the size of the simulation (i.e., the size of the dataset) grow. We fixed the size of trajectory data per MATLAB worker (Gordon core) to 76 frames (i.e., 846 MB). The dataset was distributed on the compute nodes by using a round-robin fashion to simulate the actual distribution resulting from a simulation on a distributed memory system. We repeated the runs three times and reported the average results.

Table 4.2 shows the observed weak scalability. We observed a linear scalability and a parallel efficiency above 90% using up to 128 workers (cores). As the input data size grows, the size of local results that need to be sent to the core with the global knowledge grows proportionally. The number of cores involved in the communication

also increases. Both factors result in the increasing trend in the communication time and a slight decrease of the efficiency on 256 cores. Overall, by taking advantage of local data and communicating only a small amount of local results, our method provides a scalable method for in-situ analysis for big folding trajectory datasets generated on large distributed memory systems.

Table 4.2: Weak scalability study of our method on the 203 GB villin dataset on Gordon

Num. of cores	16	32	64	128	256
Input data (GB)	12.5	25.5	51	101.5	203
Exchanged data (KB)	86.4	182.3	370.7	745.7	1498.4
Computation (sec)	1996.3	1997.5	1991.4	1960.3	2002.4
Communication (sec)	34.9	50.6	80.6	188.4	402.5
Total time (sec)	2031.2	2048.2	2072.5	2149.3	2407.3
Parallel efficiency (%)	n/a	99.2	98.0	94.5	84.4

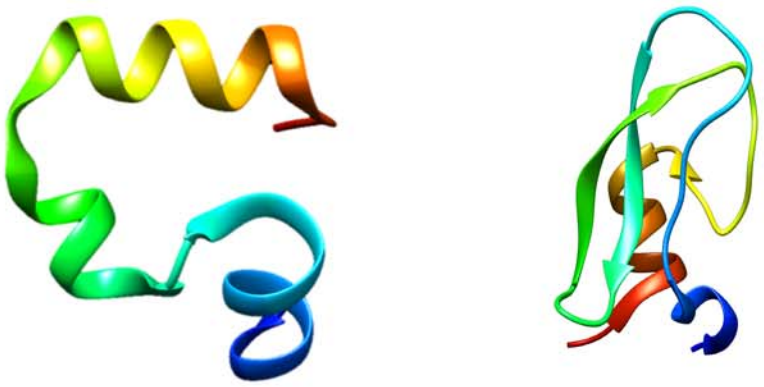
4.5 Accuracy

4.5.1 Platform

The accuracy tests are run on the same platform as the performance tests (as described in Section 4.4.1).

4.5.2 Datasets

Two folding trajectory datasets of two proteins are used in the accuracy tests. One is the trajectory for a smaller protein HP-35 NleNle generated by Folding@Home [26]. This protein contains 35 amino acids and 576 atoms and forms 3 alpha helices, as shown in Figure 4.5(a). The other one is the trajectory for a larger protein BPTI generated by the Anton supercomputer of D. E. Shaw [69]. This protein contains 58 amino acids and 909 atoms and forms 2 beta sheets and 2 alpha helices, as shown in Figure 4.5.



(a) (b)

Figure 4.5: Crystal structure for the protein NleNle (a) and BPTI (b).

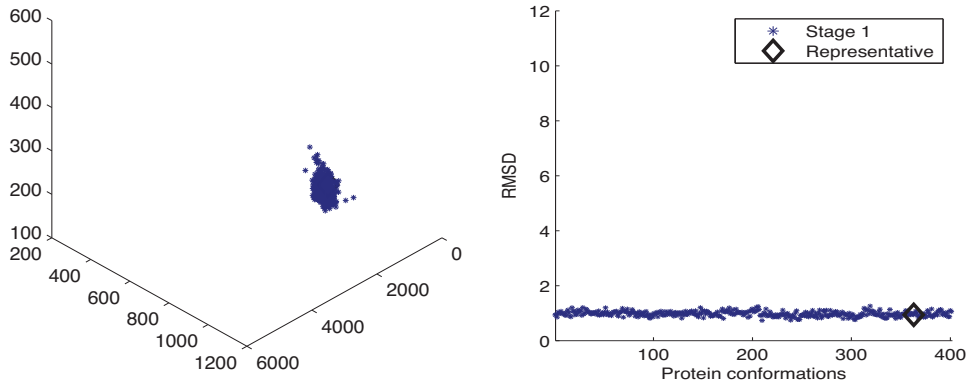
4.5.3 Results and discussion

We empirically study the accuracy of our method for both intra-trajectory analysis and inter-trajectory analysis on the smaller protein HP-35 NleNle and the larger protein BPTI.

Validation of the intra-trajectory analysis on the smaller protein: When mapping the points to stages, we know that within a meta-stable stage, points with similar conformations have a similar RMSD with respect to the crystal structures. This results in a stage with a small within-cluster variance. On the other hand, within a transition stage, the conformations change more rapidly since they are transitioning from one meta-stable stage to another. This results in a stage with a large within-cluster variance. Consequently, we hypothesize that by using the within-cluster variance of the points within a single stage, we could separate the meta-stable and transition stages automatically, without comparing atoms across conformations or points across stages. In other words, the method could compute the variance of the points locally within each stage. As a result, no trajectory data movement would be needed, and only a limited amount of memory would be used.

As part of the empirical validation of our hypothesis, we present three case studies representing the three scenarios observed in the 451 trajectories of the villin

dataset. Figures 4.6(a) and 4.6(b) show a scenario in which it is known that the protein conformations within a given simulation frame go through only minor conformational changes. Figure 4.6(a) shows the 3-D points generated by our method. For this case, the variance measured by our method among the points is 25; therefore, the frame is automatically classified as one single meta-stable stage. In Figure 4.6(b), the x-axis represents the 400 protein conformations, and the y-axis represents the RMSD of each conformation in comparison with the crystal structure of the protein. Because the 400 conformations have all close RMSD values, this figure confirms the presence of the single meta-stable stage.

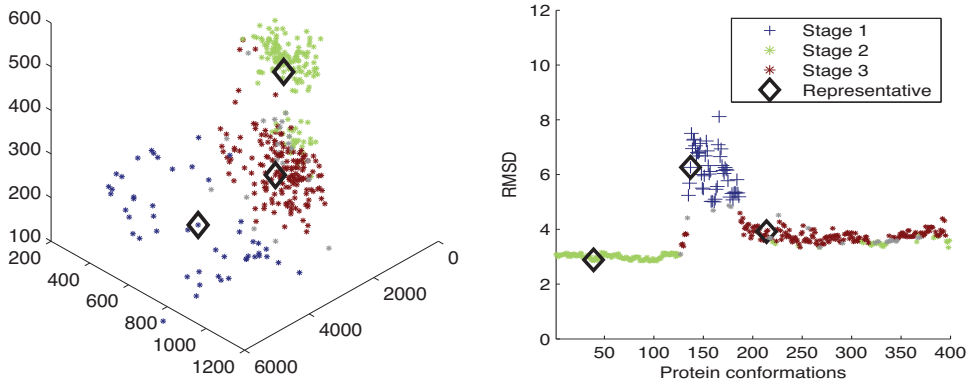


(a) (b)

Figure 4.6: Case study with single meta-stable stage: 3-D points generated by our method and their classification (a) and RMSDs of the conformations from the crystal structure (b).

Figures 4.7(a) and 4.7(b) show the scenario in which it is known that the conformations go through minor conformational changes and then change dramatically to a different conformational structure at the end of the frame. Figure 4.7(a) shows the 3-D points generated by our method. For this case, the method identifies three stages with variances of 75, 170, and 93, respectively. Thus, the frame is automatically partitioned first as one meta-stable stage, second as one transition stage, and third as another meta-stable stage. Figure 4.7(b) shows the 400 conformations and

their RMSDs compared with the crystal structure. This figure confirms how the protein changes its structure through three stages with three different RMSD ranges: two different meta-stable stages denoted by two groups of different-colored asterisks (i.e., green and red), and one transition stage between the two meta-stable stages denoted by the crosses (i.e., blue).

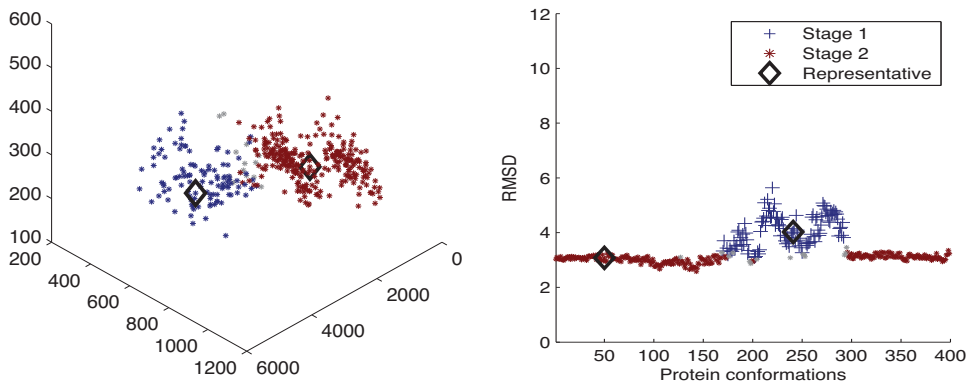


(a) (b)

Figure 4.7: Case study with two meta-stable stages and one transition stage: 3-D points generated by our method and their classification (a) and RMSDs of the conformations from the crystal structure (b).

Figures 4.8(a) and 4.8(b) show the scenario in which it is known that the folding protein goes through a meta-stable stage followed by a transition stage that returns the protein into the original meta-stable stage. Figure 4.8(a) shows the 3-D points generated by our method. According to our method, the points first map conformations that maintain a similar structure and then change drastically but return to the same structure once again at the end of the frame. Figure 4.8(b) shows the 400 conformations and their RMSDs compared to the crystal structure. The RMSDs confirm the transition phase (i.e., from the red asterisks to the blue crosses) and the roll back to the starting structure with similar RMSDs (i.e., from the blue crosses to the red asterisks).

Validation of inter-trajectory analysis on the smaller protein: We also hypothesized that our method could identify whether multiple trajectories explore similar protein conformations and eventually converge to a similar structure. This hypothesis is a

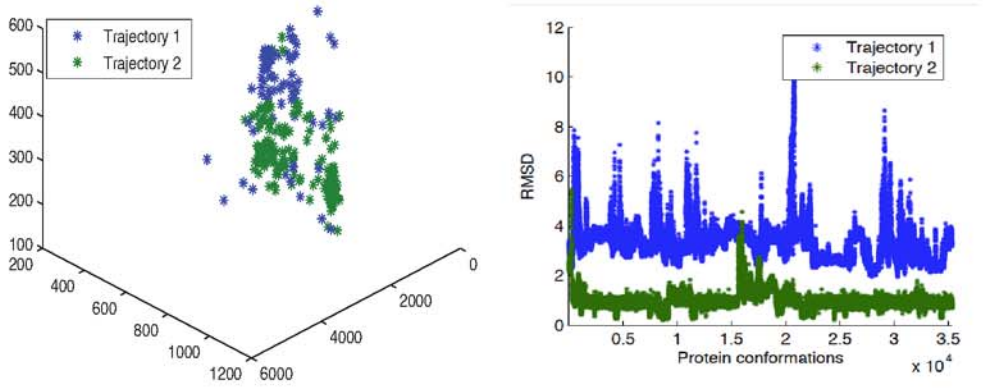


(a) (b)

Figure 4.8: Case study with one meta-stable stage and one transition stage: 3-D points generated by our method and their classification (a) and RMSDs of the conformations from the crystal structure (b).

corollary of the first hypothesis assessing how two close-in-space 3-D points in a single trajectory map two similar protein conformations. Because of the independent manner in which we extract the coordinates of each 3-D point (i.e., the eigenvalues of each single protein conformation) without any reference to other conformations within the trajectory or across trajectories, we can claim that the first hypothesis holds for points across trajectories. Across the villin dataset we observed two relevant case studies. Figure 4.9 depicts the scenario in which two trajectories explore different structures. Each point in Figure 4.9(a) corresponds to a representative of one meta-stable stage, and points with the same color belong to the same trajectory. Figure 4.9(b) shows the RMSDs of the conformations associated to the two trajectories in Figure 4.9(a) from the known crystal structure. By showing how the two trajectories have significantly different RMSDs, Figure 4.9(a) confirms that the 3-D representative points of the trajectories explore different structural spaces.

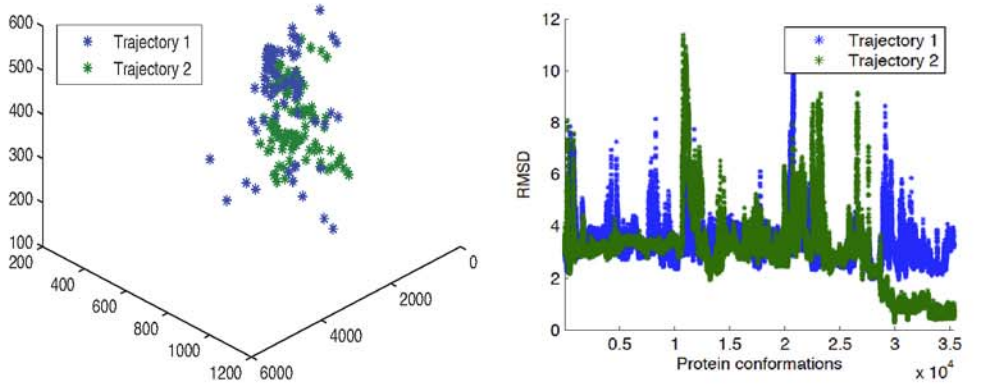
Figure 4.10 presents a different scenario in which it is known that two trajectories explore similar structural space. Figure 4.10(a) presents the results of our method and Figure 4.10(b) the validation in terms of RMSDs of the associated conformations versus the crystal structure. As it is displayed in Figure 4.10(b), the two trajectories have



(a) (b)

Figure 4.9: Example of two trajectories exploring different conformation spaces using our method (a) and by comparing the RMSDs of the conformations from the crystal structure of the protein (b).

similar RMSDs, and thus the 3-D representative points of the two trajectories explore the similar structural space. In general, these two patterns are observed across the 451 trajectories of the villin dataset; the observations support our hypothesis for this small protein.



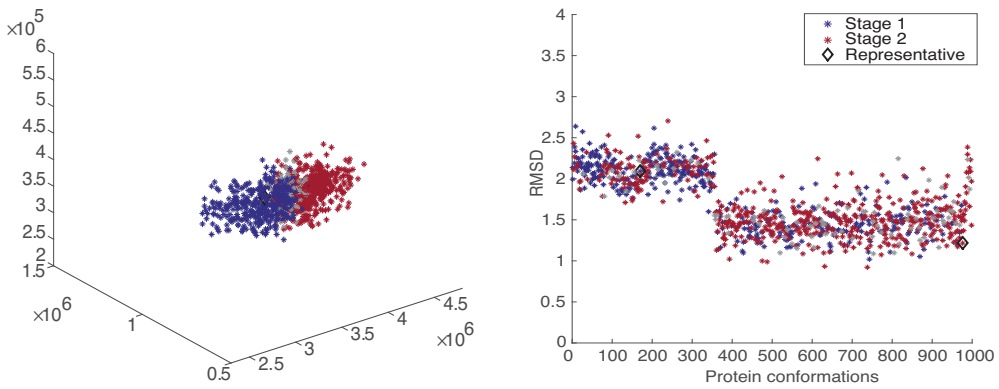
(a) (b)

Figure 4.10: Example of two trajectories exploring the same conformation space using our method (a) and by comparing the RMSDs of the conformations from the crystal structure of the protein (b).

Validation of intra-trajectory analysis on the larger protein: We observe that for

a smaller protein such as HP-35 NleNle, our method accurately captures the geometric shape features and clusters protein conformations into recurrent patterns. Next, we study the accuracy of our method when dealing with a larger and more complex protein structure such as the protein BPTI. The aspect we assess is whether the three eigenvalues of the entire distance matrix can still capture the changes in the folding protein.

Figure 4.11 shows the scenario in which it is known that the conformations go through two meta-stable stages. Figure 4.11(a) shows the 3-D metadata and the two clusters identified by our method. Figure 4.11(b) shows the 1,000 protein conformations and their RMSDs compared with the crystal structure of the protein. We observe that the two clusters identified by our clustering method still match the two meta-stable stages shown in the RMSDs of protein conformations but the classification presents some noise (i.e., the mixture of blue and red metadata shown in Figure 4.11(b)).



(a) (b)

Figure 4.11: Case study with two meta-stable stages: 3-D points generated by our method and their classification (a) and RMSDs of the conformations from the crystal structure (b).

The reasons for the noise are that the BPTI protein is larger in size (i.e., number of amino acids) and has a more complex structure (i.e., two alpha helices and two antiparallel beta sheets) than the NleNle protein. BPTI contains 58 amino acids, compared with the 35 in NleNle. This larger number results in larger entries in the

distance matrix of each protein conformation and thus larger eigenvalues. When using the set of the three largest eigenvalues, one eigenvalue is much larger than the other two, as shown in Figure 4.11(a). When applying our hierarchical clustering algorithm to this set of eigenvalues, the largest one almost always determines how the clusters are formed, which results in the observed noise.

In addition to its larger size, BPTI also has a more complex structure. It contains two alpha helices and two antiparallel beta sheets. As shown in previous research, antiparallel beta sheets are significantly more stable than alpha helices because of the well-aligned H-bonds [38]. Thus, the stable antiparallel beta sheets in the protein structure result in relatively small conformational changes in the protein's geometry compared with NleNle. Specifically, for BPTI, the range of RMSDs of the protein structures in the trajectory is from 1 to 3.5 angstroms (Å), as validated in [69]. For NleNle, on the other hand, the range is from 0.5 to 12 angstroms (Å), as shown in Figure 4.9 and 4.10. Our method represents the geometrical features of a protein conformation by using the three largest eigenvalues obtained from the entire distance matrix. In this case, because of the stable and larger structure of the protein, the 3-D metadata can no longer capture the small changes in the geometrical features accurately. Both observations on the larger protein size and the more complex protein structure suggest that we should look at specific portions of the distance matrix. In particular, we should zoom into the distance matrix and compute eigenvalues associated for sub-matrices of different rigid or flexible sub-structures in the protein.

4.6 Summary and Future Work

Summary: This chapter presents the application of our data analysis method to a clustering problem in which we identify any folding patterns within and across trajectories of a folding protein. To this end, we use our one-pass, distributed method to perform the intra- and inter-trajectory analysis of two protein-folding datasets (i.e., a smaller HP-35 NleNle protein and a larger BPTI protein) generated in a distributed fashion. Our method maps the geometrical shape of the protein conformation at a

specific folding time to the matrix's three main eigenvalues. The computation of the eigenvalues is performed in isolation, without considering past or future conformations of the folding protein or moving all the conformations to a central server. The eigenvalues become the metadata on which the clustering is performed. We integrate the method into a MapReduce-based framework using Parallel MATLAB. We measure the performance of the framework when analyzing the folding trajectory of the small protein called HP-35 NleNle (i.e., a variant of the villin headpiece subdomain) on 256 compute cores of Gordon supercomputer. We compare our framework's performance with a more traditionally used and centralized approach and observe significantly faster runtimes (i.e., two orders of magnitude faster, 41.5 seconds comparing to 3 hours in the traditional method). We also observe three and four orders of magnitude reduction respectively in the size of data movement and memory usage (i.e., 6.9 MB memory usage comparing to 16 GB in the traditional method, 4.4 KB data moved comparing to 539 MB). In addition, our framework presents a linear weak scalability and maintains a parallel efficiency above 90% when using up to 128 cores. The overall results in this thesis support our claim that our method is suitable for in-situ data analysis of folding simulation for small proteins, since we observe that our method executes sufficiently fast, avoids the need for moving trajectory data, and uses a limited amount of memory.

Empirically we also observe that our framework performs accurate intra-trajectory analysis and inter-trajectory analysis on the smaller protein but fails to capture the meta-stable stages and transition stages for the larger and more complex BPTI protein without adding tangible noise. This observation has resulted in the identification of interesting directions for future work presented below.

Future work: Future work includes research in three directions: (1) investigate new methods to effectively capture the geometrical features of a protein when dealing with more complex protein structures, (2) use stream clustering methods to identify patterns as the trajectories are generated, and (3) apply our method in the field of protein structure refinement in which the structures have only minor conformational changes.

For the first research direction, we observe that using the distance matrix of the whole conformation for large and complex proteins to generate the metadata may result in noisy geometrical clustering of the trajectory stages. One possible approach to address this problem is to look at the part of the distance matrix that corresponds to a specific protein substructure such as a specific alpha helix or a specific beta sheet and to extract metadata only on this partial distance matrix.

For the second research direction, we envision performing our clustering as the protein trajectory is generated in a streaming fashion. The stream clustering can drive intelligent decisions on whether to continue a simulation (e.g., when the protein is going through several transition stages) or to interrupt it (e.g., when the protein is continually in meta-stable stage) without the need for the scientist to review the trajectory. To this end, one should explore and adapt existing and popular stream clustering algorithms such as BRICH and COBWEB [89], [33]. Both algorithms build and update a hierarchical data structure as new data points are generated.

For the third research direction, we anticipate applying our analysis method to the protein structure refinement problem in which protein structures have only minor conformational changes. We should identify which parts of the protein structure are involved in the refinement process, and apply our method by extracting metadata using the corresponding parts of the distance matrix. By considering only the parts of the protein structure that are involved in the refinement process, we should be able to observe the changes that protein structure go through in the refinement process.

Chapter 5

CONCLUSION

This chapter concludes the thesis by summarizing the research results, discussing the limits and opportunities of our method, as well as presenting the broader impact of the thesis.

5.1 Summary

In this thesis, we propose a general method for the classification and clustering analysis of large structural biology datasets on large distributed memory systems. Our method extracts properties and/or features of each data record locally and in parallel, represents the capture properties as concise metadata, and performs a classification or clustering of the metadata without moving the original data to a central server. Our method supports three types of analysis: (1) identification of class memberships from a specific feature or property, (2) identification of features that can be used to predict class memberships, and (3) identification of recurrent patterns in datasets. For each type of analysis, we present a case study in which we apply our method to a specific structural biology dataset to study a relevant scientific problem.

In Chapter 2, we present the first case study in which we use RNA sequences and their secondary structures as a concrete example of datasets for which we want to identify class memberships from a specific feature or property. To this end, we design and implement a MapReduce-based, modularized framework that allows scientists to systematically and efficiently explore the parametric space associated with chunk-based secondary structure predictions of long RNA sequences. The framework cuts a long RNA sequence into chunks and uses powerful prediction programs (e.g., HotKnots, pknotsRG, PKNOTS, and NUPACK) for each chunk's secondary structure prediction.

We note that while these programs work efficiently on single chunks of a sequence, most of the time they do not work on the entire original sequence. We implement our chunk-based framework in Hadoop where chunks' sampling and predictions are associated with map functions while the entire secondary structures are rebuilt from single chunk predictions in reduce functions. We evaluate the framework's performance and accuracy using two datasets of RNA sequences on the Geronimo cluster at the University of Delaware using up to 64 compute cores. We observe that in a first dataset from the RFAM database, when using the HotKnots prediction program on chunks with 79 to 451 bases, our chunk-based framework can deliver scalable performance (i.e., the execution times increase from 49 to 93 seconds). In contrast, the execution times of the same prediction program when entire sequences are predicted as a whole (i.e., no chunking is applied) grow exponentially with the sequences' lengths, ranging from 1 to 16,103 seconds. Similar behaviors are observed with other prediction programs such as pknotsRG, PKNOTS, and NUPACK. In a second dataset from the virus family *Nodaviridae* with sequences' lengths ranging from 1,305 to 3,204 bases, our chunk-based framework can rebuild the longest sequence in less than 20 minutes, while none of the existing prediction programs can handle the 3,204-base sequence as a whole. Our accuracy study using datasets from the RFAM and Pseudobase++ databases outlines how our chunk-based framework obtains structures that are more similar to the structures observed in nature than when no chunking is used.

In Chapter 3, we present the second case study in which we consider structural biology datasets of ligand conformations from protein-ligand docking simulations as an example of datasets for identifying features that can be used to predict class memberships. To this end, we define a method that can efficiently capture the geometries of ligand conformations and predict what conformations dock well into a protein pocket, without moving the conformations to a central server or comparing them with each other. Specifically, for each ligand conformation in the dataset, our method first extracts relevant geometrical properties and transforms these properties into a single metadata point in the N-dimensional (N-D) space. Then, it performs an N-D clustering

on the metadata to search for predominant clusters. Our method avoids the need to move ligand conformations among nodes because it extracts relevant data properties locally and concurrently. By doing so, we transform the analysis problem into a search for densest property aggregates. We integrate the method into the MapReduce-MPI framework where the properties' extractions are performed in the map function of the framework and the search consists of one or multiple map and/or reduce functions. We use the framework for our performance and accuracy study on an 8-node dedicated cluster at the University of Delaware, a 64-node shared cluster at the San Diego Supercomputer Center, and a 256-node shared cluster at Argonne National Laboratory. Our study shows that when using our framework on small computer systems of up to 64 nodes, the performance is not sensitive to data content and distribution. When considering up to 256 nodes at the large scale and with strongly convergent metadata toward a single dense cluster of similar ligand conformations, our framework is approximately 400X faster in execution and can analyze approximately 500X larger datasets compared with a traditional hierarchical clustering based on direct comparisons of ligand conformations. We also demonstrate that our framework captures the geometrical properties of ligand conformations more effectively and predicts near-native ligand conformations more accurately than traditional methods do, including the hierarchical clustering and the energy-based scoring methods. The accuracy results on 56 ligands docking in three proteins (i.e., HIV, Trypsin, and P38alpha) show that our method can achieve 100%, 81.0%, and 83.3% clustering accuracy, respectively, whereas the traditional hierarchical clustering achieves 87.0%, 76.2%, and 50.0% clustering accuracy and the energy-based scoring achieve only 34.8%, 23.8%, and 0.8% accuracy.

In Chapter 4, we present the third case study in which we consider multiple protein folding trajectory datasets sampled from folding simulations as an example of datasets for finding recurrent patterns in datasets. To this end, we define a method that can identify any folding patterns within and across trajectories (i.e., intra- and inter-trajectory, respectively). The one-pass, distributed method also enables in-situ data analysis for large protein folding trajectory datasets by executing sufficiently

fast, avoiding moving trajectory data, and limiting the memory usage. First, the method extracts the geometric shape features of each protein conformation in parallel. Next, it classifies sets of consecutive conformations into metastable and transition stages using a probabilistic hierarchical clustering method. Then, it rebuilds the global knowledge necessary for the intra- and inter-trajectory analysis through a reduction operation. We implement the method in a MapReduce-based framework using Parallel MATLAB in which the shape extractions and classifications are performed as part of the map function and the global knowledge is rebuilt as part of the reduce function. The comparison of our performance results obtained with our framework versus a traditional method based on moving data to a central server that was proposed by Phillips shows the strength of our method. Specifically, the framework can analyze the folding trajectory of a villin headpiece subdomain consisting of 20,000 protein conformations in 41.5 seconds whereas Phillips' method takes approximately 3 hours. Our framework uses 6.9 MB of memory per core while Philips method uses 16 GB per core. Moreover, our framework communicates only 4.4 KB to a central server whereas Phillips' method moves the entire dataset of 539 MB. The overall results in this thesis support our claim that our method is suitable for in-situ data analysis of folding trajectories.

5.2 Limitations and Opportunities

While the overall set of steps to extract knowledge from data is general (i.e., from data to metadata and from metadata to knowledge), the implementation of the individual step depends on the data characteristics and the scientific questions we are trying to answer. For example, the use of projections and linear interpolations for protein folding trajectories has showed poor accuracy because, once folded, the shape of the protein tends to resemble symmetrical round shapes. On the other hand, the use of eigenvalues for simple ligands is not ideal because of the limited number of carbon atoms and the more rigid structure of ligands compared to proteins. Thus we cannot use a single mapping to metadata but we need to understand the nature

of the simulations and the properties embedded in the data before to define creative mapping algorithms that accurately capture only those properties of interest. This limit is actually an opportunity to design new algorithms.

Our overall method assesses the accuracy and performance of the three different analyses empirically on large but still finite datasets. The fact that our results are accurate and efficient, does not exclude that for other datasets we may not be able to reach similar conclusions. An in-depth approach based on quantitatively analyzes of how accurately metadata captures the relevant properties of the original data is a possible future direction.

Even in the best scalability results, we observed that the MapReduce paradigm suffers from performance loss at the extreme scale due to the long data shuffle time. In particular, our performance results for large ligand datasets of 2 TBytes on 256 nodes of Fusion supercomputer show that the data shuffle stage takes more than half of the execution time at this scale. This is due to the fact that the all-to-all communication in the data shuffle stage is expensive, especially when there is load imbalance in the metadata distribution. While we have substantially reduced the size of the data, we have not included the topology of the machine as one of the key tuning factors in the shuffle phase. We expect that a topology-aware data communication, in which the data is aggregated in nodes that are topologically close, can further reduce the impact of the data shuffle stage at the large scale.

5.3 Broader Impact

Cutting-edge distributed memory systems, such as cloud infrastructures and high-end clusters, provide scientists with an efficient and scalable way to generate large-scale distributed datasets by performing various computationally expensive simulations. This massive amount of data results in new challenges for the scientists who have to analyze these data in order to gain scientific insights. This thesis bridges this gap between distributed large-scale data generation and data analyses by providing a general classification and clustering method together with a set of effective algorithms.

From a computer science perspective, by providing a scalable classification and clustering method and the three case studies on the three computational structural biology datasets, this work opens the door to a new generation of distributed analysis methods. These analysis methods are designed from the beginning with the concept of distributed processing and hence ensure performance scalability. Our method is especially attractive to the scientific community because of two features. First, we perform the analyses in a distributed fashion, without moving datasets that are generated and stored locally across a distributed memory system. By avoiding excessive I/O and communication, our method reduces the network bandwidth usage and the energy consumption of the system. Second, our method requires a relatively small amount of memory and is executed sufficiently fast. These two features make our method an ideal candidate for in-situ data analysis, in which we can provide runtime feedback to scientific simulations to ensure and help these simulations make scientifically meaningful progress. Potentially, these features allow the integration of our analysis method into the data generation process of scientific simulations, which bring runtime intelligent to the simulations.

From a scientific perspective, our method provides meaningful information and feedback about the computational simulation's efficiency and accuracy. The accurate results that our method generates help advance the landscape of scientific discoveries by speeding the scientific process such as designing drugs or studying diseases related to protein misfolding. Moreover, the feedback from the accuracy study of our method helps the development of more efficient and accurate mathematical models for scientific simulations.

BIBLIOGRAPHY

[1] *MATLAB and statistics toolbox release 2012b*. The MathWorks Inc., 2012.

[2] R. Abagyan, M. Totrov, and D. Kuznetsov. A new method for protein modeling and design: Applications to docking and structure prediction from the distorted native conformation. *Journal of Computational Chemistry*, 15(5):488–506, 1994.

[3] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[4] D. P. Anderson. BOINC: A system for public-resource computing and storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (GRID)*, pages 4–10, 2004.

[5] K. Backbro, S. Lowgren, K. Osterlund, J. Atepo, T. Unge, J. Hulten, N. M. Bonham, W. Schaal, A. Karlen, and A. Hallberg. Unexpected binding mode of a cyclic sulfamide HIV-1 protease inhibitor. *Journal of Medical Chemistry*, 40(6):898–902, 1997.

[6] J. C. Bennett, H. Abbasi, P. T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, P. Pebay, D. Thompson, H. Yu, F. Zhang, and J. Chen. Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In *Proceedings of the 2012 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–9, 2012.

[7] C. Best and H. C. Hege. Visualizing and identifying conformational ensembles in molecular dynamics trajectories. *Computing in Science Engineering*, 4(3):68–75, 2002.

[8] I. Borg and P. J. F. Groenen. *Modern multidimensional scaling: Theory and applications*. Springer, second edition, 2005.

[9] G. Bouvier, N. Evrard-Todeschi, J. P. Girault, and G. Bertho. Automatic clustering of docking poses in virtual screening process using self-organising map. *Bioinformatics Advance Access*, 26(1):53–60, 2009.

[10] I. Brierley, S. Pennell, and R. J. Gilbert. Viral RNA pseudoknots: Versatile motifs in gene expression and replication. *Nature Reviews Microbiology*, 5(8):598–610, 2007.

[11] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus. CHARMM: A program for macromolecular energy minimization, and dynamics calculations. *Journal of Computational Chemistry*, 4(2):187–217, 1983.

[12] S. W. Burge, J. Daub, R. Eberhardt, J. Tate, L. Barquist, E. P. Nawrocki, S. R. Eddy, P. P. Gardner, and A. Bateman. Rfam 11.0: 10 years of RNA families. *Nucleic Acids Research*, 41(D1):D226–D232, 2013.

[13] B. D. Bursulaya, M. Totrov, R. Abagyan, and C. L. Brooks III. Comparative study of several algorithms for flexible ligand docking. *Journal Of Computer-Aided Molecular Design*, 17(11):755–763, 2003.

[14] R. L. Cannon, J. V. Dave, and J. C. Bezdek. Efficient implementation of the fuzzy c-means clustering algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(2):248–255, 1986.

[15] A. M. Caulfield, L. M. Grupp, and S. Swanson. Gordon: Using flash memory to build fast, power-efficient clusters for data-intensive applications. *ACM SIGARCH Computer Architecture News*, 37(1):217–228, 2009.

[16] M. W. Chang, R. K. Belew, K. S. Carroll, A. J. Olson, and D. S. Goodsell. Empirical entropic contributions in computational docking: Evaluation in APS reductase complexes. *Journal of Computational Chemistry*, 29(11):1753–1761, 2008.

[17] D. S. Chew, M.-Y. Leung, and K. P. Choi1. AT excursion: A new approach to predict replication origins in viral genomes by locating AT-rich regions. *BMC Bioinformatics*, 8(1):163, 2007.

[18] R. L. F. Cordeiro, C. Traina Jr, A. J. M. Traina, J. López, U. Kang, and C. Faloutsos. Clustering very large multi-dimensional datasets with MapReduce. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 690–698, 2011.

[19] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[20] R. M. Dirks and N. A. Pierce. A partition function algorithm for nucleic acid secondary structure including pseudoknots. *Journal of Computational Chemistry*, 24(13):1664–1677, 2003.

[21] R. M. Dirks and N. A. Pierce. An algorithm for computing nucleic acid base-pairing probabilities including pseudoknots. *Journal of Computational Chemistry*, 25(10):1295–1304, 2004.

[22] C. M. Dobson. Protein folding and misfolding. *Nature*, 426(6968):884–890, 2003.

[23] Docking@Home. [ONLINE] http://docking.cis.udel.edu.

[24] F. Dullweber, M. T. Stubbs, D. Musil, J. Sturzebecher, and G. Klebe. Factorising ligand affinity: A combined thermodynamic and crystallographic study of trypsin and thrombin inhibition. *Journal of Molecular Biology*, 313(3):593–614, 2001.

[25] A. Ene, S. Im, and B. Moseley. Fast clustering using MapReduce. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 681–689, 2011.

[26] D. L. Ensign, P. M. Kasson, and V. S. Pande. Heterogeneity even at the speed limit of folding: Large-scale molecular dynamics study of a fast-folding variant of the villin headpiece. *Journal of Molecular Biology*, 374(3):806–816, 2007.

[27] T. Estrada, R. S. Armen, and M. Taufer. Automatic selection of near-native protein-ligand conformations using a hierarchical clustering and volunteer computing. In *Proceedings of the 1st ACM International Conference on Bioinformatics and Computational Biology (BCB)*, pages 204–213, 2010.

[28] T. Estrada, B. Zhang, P. Cicotti, R. S. Armen, and M. Taufer. Accurate analysis of large datasets of protein-ligand binding geometries using advanced clustering methods. *Computers in Biology and Medicine*, 42(7):758–771, 2012.

[29] T. Estrada, B. Zhang, P. Cicotti, R. S. Armen, and M. Taufer. Reengineering high-throughput molecular datasets for scalable clustering using MapReduce. In *Proceedings of the 14th IEEE International Conference on High Performance Computing and Communications (HPCC)*, pages 351–359, 2012.

[30] P. Ferrara, H. Gohlke, D. Price, G. Klebe, and C. L. Brooks III. Assessing scoring functions for protein-ligand interactions. *Journal of Medicinal Chemistry*, 47(12):3032–3047, 2004.

[31] A. Fielding. *Cluster and classification techniques for the biosciences*. Cambridge University Press, 2007.

[32] E. J. Finnegan and M. A. Matzke. The small RNA world. *Journal of Cell Science*, 116(23):4689–4693, 2003.

[33] D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2):139–172, 1987.

[34] M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937.

[35] Apache Hadoop. [ONLINE] http://hadoop.apache.org/.

[36] P. C. D. Hawkins, G. L. Warren, A. G. Skillman, and A. Nicholls. How to do an evaluation: Pitfalls and traps. *Journal of Computer-Aided Molecular Design*, 22(3-4):179–190, 2008.

[37] M. Hefeeda, F. Gao, and W. Abd-Almageed. Distributed approximate spectral clustering for large-scale datasets. In *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, pages 223–234, 2012.

[38] K. A. W. Henzler, D. K. Lee, and A. Ramamoorthy. Determination of alpha-helix and beta-sheet stability in the solid state: a solid-state NMR investigation of]oly(L-alanine). *Biopolymers*, 64(5):246–254, 2002.

[39] I. L. Hofacker, W. Fontana, P. F. Stadler, S. Bonhoeffer, M. Tacker, and P. Schuster. Fast folding and comparison of RNA secondary structures. *Monatshefte für Chemie*, 125(2):167–188, 1994.

[40] D. Hong, A. Rhie, S.-S. Park, J. Lee, Y. S. Ju, S. Kim, S.-B. Yu, T. Bleazard, H.-S. Park, H. Rhee, H. Chong, K.-S. Yang, Y.-S. Lee, I.-H. Kim, J. S. Lee, J.-I. Kim, and J. S. Seo. FX: an RNA-Seq analysis tool on the cloud. *Bioinformatics*, 28(5):721–723, 2012.

[41] A. N. Jain. Bias, reporting, and sharing: Computational evaluations of docking methods. *Journal of Computer-Aided Molecular Design*, 22(3-4):201–212, 2008.

[42] K. N. Johnson, K. L. Johnson, R. Dasgupta, T. Gratsch, and L. A. Ball. Comparisons among the larger genome segments of six Nodaviruses and their encoded RNA replicases. *Journal of General Virology*, 82(Pt 8):1855–1866, 2001.

[43] S. Karlin, A. Dembo, and T. Kawabata. Statistical composition of high-scoring segments from molecular sequences. *Annals of Statistics*, 18(2):571–581, 1990.

[44] S. Lakshminarasimhan, D. A. Boyuka, S. V. Pendse, X. Zou, J. Jenkins, V. Vishwanath, M. E. Papka, and N. F. Samatova. Scalable in situ scientific data encoding for analytical query processing. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing (HPDC)*, pages 1–12, 2013.

[45] B. Langmead, K. D. Hansen, and J. T. Leek. Cloud-scale RNA-sequencing differential expression analysis with Myrna. *Genome Biology*, 11:R83, 2011.

[46] M. S. Lee, M. Feig, F. R. Salsbury Jr., and C. L. Brooks III. New analytic approximation to the standard molecular volume definition and its application to generalized Born calculations. *Journal of Computational Chemistry*, 24:1348–1356, 2003.

[47] H. G. Li, G. Q. Wu, X. G. Hu, J. Zhang, L. Li, and X. Wu. K-means clustering with bagging and MapReduce. In *Proceedings of the 44th Hawaii International Conference on System Sciences (HICSS)*, pages 1–8, 2011.

[48] J. Liu, S. Byna, and Y. Chen. Segmented analysis for reducing data movement. In *Proceedings of the IEEE International Conference on Big Data (BigData)*, 2013.

[49] S. Lorenzen and Y. Zhang. Identification of near-native structures by clustering protein docking conformations. *PROTEINS: Structure, Function, and Bioinformatics*, 68:187–194, 2007.

[50] N. R. Markham and M. Zuker. UNAFold: Software for nucleic acid folding and hybridization. *Methods in Molecular Biology*, 453:3–31, 2008.

[51] A. Matsunaga, M. Tsugawa, and J. Fortes. CloudBLAST: Combining MapReduce and virtualization on distributed resources for bioinformatics applications. In *Proceeding of the IEEE 4th International Conference on eScience (eScience)*, pages 222–229, 2008.

[52] C. Moler. Why there isn't parallel Matlab. *Mathworks Newsletter*, 00:12, 1995.

[53] C. Moler. Parallel MATLAB: From hell no to you bet. *Thirty Years of Parallel Computing at Argonne: A Symposium*, 2013.

[54] E. Mooi and M. Sarstedt. *A concise guide to market research: The process, data, and methods using IBM SPSS statistics*. Springer, 2011.

[55] G. M. Morris, D. S. Goodsell, R. S. Halliday, R. Huey, W. E. Hart, R. K. Belew, and A. J. Olson. Automated docking using a Lamarckian genetic algorithm and empirical binding free energy function. *Journal of Computational Chemistry*, 19(14):1639–1662, 1998.

[56] R. Nussinov and A. B. Jacobson. Fast algorithm for predicting the secondary structure of single stranded RNA. *Proceedings of the National Academy of Sciences of the United States of America*, 77(11):6309–6313, 1980.

[57] E. Perola, W. P. Walters, and P. S. Charifson. A detailed comparison of current docking and scoring methods on systems of pharmaceutical relevance. *Proteins*, 56(2):235–249, 2004.

[58] J. Phillips, M. Colvin, and S. Newsam. Validating clustering of molecular dynamics simulations using polymer models. *BMC Bioinformatics*, 12(1):445–467, 2011.

[59] S. J. Plimpton and K. D. Devine. MapReduce in MPI for large-scale graph algorithms. *Parallel Computing*, 37(9), September 2011.

[60] M. Rarey, B. Kramer, T. Lengauer, and G. A. Klebe. A fast flexible docking method using an incremental construction algorithm. *Journal of Molecular Biology*, 261(3):470–489, 1996.

[61] Z. Rasheed and H. Rangwala. A Map-Reduce framework for clustering metagenomes. In *Proceedings of the 13th IEEE International Workshop on High Performance Computational Biology (HiCOMB)*, pages 549–558, 2013.

[62] J. Reeder, P. Steffen, and R. Giegerich. pknotsRG: RNA pseudoknot folding including near-optimal structures and sliding windows. *Nucleic Acids Research*, 33(suppl 2):W320–324, 2007.

[63] J. Ren, B. Rastegari, A. Condon, and H. H. Hoos. Hotknots: Heuristic prediction of RNA secondary structures including pseudoknots. *RNA*, 11(10):1494–1504, 2005.

[64] E. Rivas and S. R. Eddy. A dynamic programming algorithm for RNA structure prediction including pseudoknots. *Journal of Molecular Biology*, 285(5):2053–2568, 1999.

[65] J. J. Rosskopf, J. H. Upton III, L. Rodarte, T. A. Romero, M.-Y. Leung, M. Taufer, and K. L. Johnson. A 3' terminal stem-loop structure in Nodamura virus RNA2 forms an essential cis-acting signal for RNA replication. *Virus Research*, 150(1-2):12–21, 2010.

[66] D. Sankoff. Simultaneous solution of the RNA folding,alignment and protosequence problems. *SIAM Journal on Applied Mathematics*, 45(5):810–825, 1985.

[67] K. Sato, Y. Kato, M. Hamada, T. Akutsu, and K. Asai. IPknot: Fast and accurate prediction of RNA secondary structures with pseudoknots using integer programming. *Bioinformatics*, 27(13):i85–i93, 2011.

[68] M. C. Schatz. Cloudburst: Highly sensitive read mapping with MapReduce. *Bioinformatics*, 25(11):1363–1369, 2009.

[69] 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, and W. Wriggers. Atomic-level characterization of the structural dynamics of proteins. *Science*, 330(6002):341–346, 2010.

[70] S. S. Shende and A. D. Malony. The TAU parallel performance system. *International Journal of High Performance Computing Applications*, 20(2):287–311, 2006.

[71] C. H. Siederdissen and I. L. Hofacker. Discriminatory power of RNA family models. *Bioinformatics*, 26(18):i453–i459, 2010.

[72] A. Smith, Z. Xuan, and M. Zhang. Using quality scores and longer reads improves accuracy of Solexa read mapping. *BMC Bioinformatics*, 9(1):128, 2008.

[73] G. W. Snedecor and W. G. Cochran. The sample correlation coefficient R. In *Statistical Methods*, pages 175–178. Iowa State Press, 7th edition, 1980.

[74] M. Taufer, R. S. Armen, J. Chen, P. J. Teller, and C. L. Brooks III. Computational multi-scale modeling in protein-ligand docking. *IEEE Engineering in Medicine and Biology Magazine*, 28(2):58–69, 2009.

[75] M. Taufer, M. Crowley, D. Price, A. A. Chien, and C. L. Brooks III. Study of an accurate and fast protein-ligand docking algorithm based on molecular dynamics. *Concurrency and Computation: Practice and Experience*, 17(14):1627–1641, 2005.

[76] M. Taufer, M.-Y. Leung, T. Solorio, A. Licon, D. Mireles, R. Araiza, and K.L. Johnson. RNAVLab: A virtual laboratory for studying RNA secondary structures based on grid computing technology. *Parallel Computing*, 34(11):661–680, 2008.

[77] M. Taufer, A. Licon, R. Araiza, D. Mireles, A. Gultyaev, F. H. D. Van Batenburg, and M.-Y. Leung. Pseudobase++: An extension of PseudoBase for easy searching, formatting, and visualization of pseudoknots. *Nucleic Acids Research*, 37(Database-Issue):127–135, 2009.

[78] V. Thiel, K. A. Ivanov, A. Putics, T. Hertziq, B. Schelle, S. Bayer, B. Weissbrich, E. J. Snijder, H. Rabenau, H. W. Doerr, A. E. Gorbalenya, and J. Ziebuhr. Mechanisms and enzymes involved in SARS coronavirus genome expression. *Journal of General Virology*, 84(Pt 9):2305–2315, 2003.

[79] R. Thiery, K. L. Johnson, T. Nakai, A. Schneemann, J. R. Bonami, and D. V. Lightner. Family Nodaviridae. In *Virus Taxonomy: Ninth Report of the International Committee on Taxonomy of Viruses*, pages 1061–1067. Elsevier Academic Press, 2011.

[80] T. Tu, C. A. Rendleman, D. W. Borhani, R. O. Dror, J. Gullingsrud, M. O. Jensen, J. L. Klepeis, P. Maragakis, P. Miller, K. A. Stafford, and D. E. Shaw. A scalable parallel framework for analyzing terascale molecular dynamics simulation trajectories. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–12, 2008.

[81] Z. Wang, B. J. Canagarajah, J. C. Boehm, S. Kassisa, M. H. Cobb, P. R. Young, S. Abdel-Meguid, J. L. Adams, and E. J. Goldsmith. Structural basis of inhibitor selectivity in MAP kinases. *Structure*, 6:1117–1128, 1998.

[82] What does big data mean. [ONLINE] http://cacm.acm.org/blogs/blog-cacm/155468-what-does-big-data-mean/fulltext.

[83] D. T. Yehdego, B. Zhang, V. K. R. Kodimala, K. L. Johnson, M. Taufer, and M.-Y. Leung. Secondary structure predictions for long RNA sequences based on

inversion excursions and MapReduce. In *Proceedings of the12th IEEE International Workshop on High Performance Computational Biology (HiCOMB)*, pages 520–529, 2013.

[84] B. Zhang, T. Estarda, P. Cicotti, and M. Taufer. On efficiently capturing scientific properties in distributed big data without moving the data - a case study in distributed structural biology using MapReduce. In *Proceedings of the 16th IEEE International Conferences on Computational Science and Engineering (CSE)*, pages 117–124, 2013.

[85] B. Zhang, T. Estarda, P. Cicotti, and M. Taufer. Enabling in-situ data analysis for large protein folding trajectory datasets. In *Proceedings of 28th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 221–230, 2014.

[86] B. Zhang, T. Estrada, P. Cicotti, P. Balaji, and M. Taufer. Accurate scoring of drug conformations at the extreme scale. In *Proceedings of the 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2015.

[87] B. Zhang, D. Yehdego, K. L. Johnson, M.-Y. Leung, and M. Taufer. A modularized MapReduce framework to support RNA secondary structure prediction and analysis workflows. In *Proceedings of the 2012 Computational Structural Bioinformatics Workshop (CSBW)*, pages 86–93, 2012.

[88] B. Zhang, D. T. Yehdego, K. L. Johnson, M.-Y. Leung, and M. Taufer. Enhancement of accuracy and efficiency for RNA secondary structure prediction by sequence segmentation and MapReduce. *BMC Structural Biology*, 13(Suppl 1):S3, 2013.

[89] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 103–114, 1996.

[90] M. Zuker. Mfold web server for nucleic acid folding and hybridization prediction. *Nucleic Acids Research*, 31(13):3406–3415, 2003.

Appendix

RIGHTS AND PERMISSIONS

This is a License Agreement between Boyu Zhang ("You") and Elsevier ("Elsevier"). The license consists of your order details, the terms and conditions provided by Elsevier, and the payment terms and conditions.

Get the printable license.

License Number	3622000646845
License date	May 04, 2015
Licensed content publisher	Elsevier
Licensed content publication	Computers in Biology and Medicine
Licensed content title	A scalable and accurate method for classifying protein–ligand binding geometries using a MapReduce approach
Licensed content author	T. Estrada,B. Zhang,P. Cicotti,R.S. Armen,M. Taufer
Licensed content date	July 2012
Licensed content volume number	42
Licensed content issue number	7
Number of pages	14
Type of Use	reuse in a thesis/dissertation
Portion	figures/tables/illustrations
Number of figures/tables/illustrations	1
Format	both print and electronic
Are you the author of this Elsevier article?	Yes
Will you be translating?	No
Original figure numbers	Fig. 3.
Title of your thesis/dissertation	ENABLING SCALABLE DATA ANALYSIS FOR LARGE COMPUTATIONAL STRUCTURAL BIOLOGY DATASETS ON LARGE DISTRIBUTED MEMORY SYSTEMS SUPPORTED BY THE MAPREDUCE PARADIGM
Expected completion date	May 2015
Estimated size (number of pages)	169
Elsevier VAT number	GB 494 6272 12
Permissions price	0.00 USD
VAT/Local Sales Tax	0.00 USD / 0.00 GBP
Total	0.00 USD