IMPLEMENTATION IMPROVEMENTS TO AN RNA PSEUDOKNOT PREDICTION ALGORITHM

by

YUNZHOU WU

(Under the Direction of Liming Cai)

ABSTRACT

The general problem of RNA pseudoknot prediction is computationally intractable. Most existing algorithms require the worst case CPU time $O(N^6)$ and RAM space $O(N^4)$ even for restricted pseudoknot categories. Such resource requirements make it infeasible to predict pseudoknots for RNA sequences of even a moderate length. This research work demonstrates two implementation techniques, memory mapping and parallel computation, to reduce resource usage in the algorithms by taking advantage of the ways the matrix is organized and computed. The techniques are applied to an automated and dynamic programming-based pseudoknot prediction algorithm we recently developed. Our experiments shows that savings in memory usage of approximately 97% and nearly 10-fold speedup (using 16 parallel processors) are achieved for the pseudoknot prediction tests conducted. Most existing RNA prediction algorithms are dynamic programming-based and evolved from the CYK algorithm for stem-loop prediction; thus our techniques are general and applicable to these algorithms as well.

INDEX WORDS: RNA pseudoknot prediction, dynamic programming, memory mapping, parallel computation, stochastic grammar modeling

IMPLEMENTATION IMPROVEMENTS TO AN RNA PSEUDOKNOT PREDICTION ALGORITHM

by

YUNZHOU WU

B.Engg., Jianghan Petroleum Institute, China, 1994

M.Engg., China University of Mining and Technology, China, 1997

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2003

IMPLEMENTATION IMPROVEMENTS TO AN RNA PSEUDOKNOT PREDICTION ALGORITHM

by

YUNZHOU WU

Major Professor: Liming, Cai

Committee: Russell L. Malmberg
Eileen T. Kraemer

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
May 2003

DEDICATION

To my beloved family

ACKNOWLEDGEMENTS

I am pleased to have this dedicated page to express my gratitude to my major professor Dr. Liming Cai for his invaluable guidance and warm encouragement throughout my degree program, especially his selfless help in both my academic research and my life.

I would also like to thank Dr. Russell L. Malmberg for his patient direction and kind assistance on my thesis research. My sincere thankfulness should also go to Dr. Eileen T. Kraemer for serving on my thesis committee.

I also appreciate the faculty, staff, and graduate students of UGA for their cooperation and help.

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENTS v

LIST OF TABLES viii

LIST OF FIGURES ix

CHAPTER

1 INTRODUCTION 1

2 A STOCHASTIC GRAMMATICAL MODELING OF RNA PSEUDOKNOTS 9

2.1 Parallel Grammars to Specify Pseudoknots 10

2.2 Technical Observations 13

3 PSEUDOKNOT PREDICTION ALGORITHM 15

3.1 The Algorithm 16

3.2 Computational Complexity of Algorithm 20

4 IMPROVEMENT OF SPACE USAGE 23

4.1 Memory Mapping 25

4.2 Further improvement in the memory usage: modifying grammar writing style 28

5 PARALLEL COMPUTATION FOR THE PREDICTION ALGORITHM 29

5.1 Parallel computing system and techniques 29

5.2 Data Dependency and Analysis of parallelism 33

5.3 Implementation of Parallelism 34

5.4 Further improvement of algorithm 35

6 SUMMARY 40

REFERENCES 43

LIST OF TABLES

Page

Table 2.1: PCGS productions for two base-paired regions 12

Table 2.2: Derivation of base-paired regions aacg and cguu 12

Table 2.3: Master grammar G_0 describing a pseudoknotted structure 13

Table 4.1: Comparison of memory usage between methods with and without memory-mapping technique 27

Table 4.2: A grammar model compliable to Chomsky style and a modified grammar 28

Table 5.1: Running time and speedup for approach 1 with different number of processors 34

Table 5.2: When Dij value approaching to the limit N…… 36

Table 5.3: For some Dij's, the total number of iterations cannot be evenly divided by the thread count… 37

Table 5.4: Time (seconds) comparison between two approaches for some special *dij* values..... 38

Table 5.5: Time (seconds) comparison between two approaches when *dij* is approaching to the limit 63(the sequence length) 38

Table 5.6: Overall performance comparison between two parallelization approaches..... 39

LIST OF FIGURES

Page

Figure 2.1: A pseudoknot in the consensus structure of β-proteobacteria 11

Figure 2.2: Derivation tree of a pseudoknotted structure..... 12

Figure 3.1: A part prediction result generated by our algorithm..... 20

Figure 5.1: Parallel computation model adopted by openMP..... 31

Figure 5.2: The order of filling matrix M..... 32

CHAPTER 1

INTRODUCTION

The importance of RNA molecules in cells is recognized by the key roles they play in transcription and translation. It is well-known that RNA molecules carry protein-coding information in their base sequences, and that some RNA molecules, such as ribozymes, can also serve as catalysts in reactions. The function of an RNA molecule is determined by the exact sequence in which their nucleotide bases are ordered, with the linear sequence of bases forming the primary structure of the RNA molecule. The interactions between bases separated by long or short distances allow the RNA molecule to fold up in space to form its secondary structure; the sets of stacked base pairs are called double *helices*, which might be present in sequences in a parallel or nested manner called *stem loops*, or in a crossing manner called *pseudoknots*.

Pseudoknots are functionally important in a number of RNAs. They have been found participating in translation [Felden *et al*., 2001], viral genome structure [Paillart *et al*., 2002], and ribozyme active sites [Tanaka *et al*., 2002]. For example, pseudoknot structures present in coding regions can stimulate ribosomal frame-shifting and translational read-through during elongation [Kontos *et al*., 2001]. Pseudoknots in noncoding regions can initiate translation either by being part of so-called internal ribosomal entry site(IRES) in the 5’ noncoding region or by forming translational enhancers in the 3’noncoding region [Deiman and Pleij,1997].

A number of computational approaches have been developed to predict RNA secondary structures. Most of these methods can be categorized as thermodynamic modeling or stochastic grammar modeling. Thermodynamics modeling uses a model that describes how to assign free energies to legal secondary structures; the secondary structure of an RNA sequence is predicted as the structure of least free energy [Tinoco *et al*., 1973; Nussinov *et al*., 1980; Zuker *et al*.,1981; Turner *et al*., 1988; Mathews 1999]. The thermodynamic method has resulted in a few widely used software systems, such as MFOLD [Zuker *et al*.,1981] and Viena [*http://www.tbi.univie.ac.at/~ivo/RNA/*] that can predict non-pseudoknotted structures for any given single RNA sequence at more than 70% accuracy. This type of modeling has generality but lacks flexibility in describing RNA structures as compared to stochastic grammar approaches. Stochastic models based on Chomsky context–free grammar to describe the nested and parallel patterns of interactions between nucleotide bases have been very successful for predicting stem-loops in RNA sequences [Eddy and Durbin,1994; Lari and Young, 1990; Sakakibara *et al*., 1994]. In a stochastic context-free grammar (SCFG), each production is associated with a probability parameter computed from a set of RNA training sequences. A structure prediction algorithm can be automatically generated from each specifically trained structure model. SCFG is more flexible than other modeling approaches in the sense that it allows specific description of structure patterns and automated generation of prediction algorithms. Moreover, unlike other approaches that focus on specialized algorithms for solving the single-sequence structure prediction problem, SCFG can also be used to develop profiles for multiple alignment, consensus structure prediction, and structural homology recognition in database searches [Eddy and Durbin,1994; Brown, 2000;

Durbin *et al*., 1998]. It has even been shown that in general minimum energy based modeling approaches such as Zuker's can be converted to SCFG style [Rivas and Eddy, 2000; Durbin *et al*., 1998].

However, stochastic modeling of RNA pseudoknots based on grammar systems still remains a big challenge in the bioinformatics community because of the additional complexity imposed by crossing patterns of base pairings in pseudoknots. Theoretically, pseudoknots require a context-sensitive grammar that is more complex than CFG and clumsy to manipulate. Some attempts have been made to model pseudoknots in the spirit of SCFG. Brown and Wilson [1996], for example, have introduced a prediction model based on the intersection of two separate SCFGs to describe two crossing helices of a pseudoknot. This approach, however, is heuristic and does not ensure optimality. Uemura *et al*.' s tree adjoining modeling approach seems too complicated to implement [Uemura *et al*., 1999]. After introducing a thermodynamic-based dynamic programming algorithm for pseudoknot prediction [Rivas and Eddy, 1999], Rivas & Eddy [2000] presented a grammar model to describe the legal structures identified by their algorithm. This grammar is based on auxiliary semantic symbols to reorder the strings generated by an otherwise context-free grammar. As described in [Rivas and Eddy, 2000], this grammar approach makes it possible to develop fully probabilistic modeling of pseudoknots. However, the rules of reordering appear hard to implement.

In our research, we have introduced a new grammar model for RNA pseudoknots based on the *Parallel Communicating Grammar System* [Paun and Santean, 1990]. A PCGS consists of a number of Chomsky grammars (called components) that are parallel and synchronized with one another. Each component can query sequences generated by

other components and multiple components can make queries at the same time. Such a communication protocol allows more than one grammar components to work together and be more powerful than a single component of the same type. In particular, a pseudoknot can be described by such a grammar system in which a context-free grammar generates a stem loop while a pair of synchronized components generates the crossing helix. Technically, the stochastic version of the grammatical modeling of pseudoknots can be as simple as an SCFG containing a special query symbol as a nonterminal. As with SCFG, our new modeling of pseudoknots allows top-down grammatical descriptions of RNA pseudoknot structures in the same manner as an SCFG does for stem-loops. The new model also leads to an automated bottom-up dynamic programming algorithm for pseudoknot structure prediction [Cai *et al*, 2003].

In our pseudoknot prediction algorithm, a matrix M is used to evaluate, for every subsequence x[*i..j*] and every nonterminal X, the maximum probability for x[*i, j*] to admit the substructure specified by X. The computation is done in the spirit of the CYK algorithms [Cocke, Younger and Kasami] for stem-loop prediction. Due to the complexity imposed by the crossing double helices forming pseudoknots, our algorithm needs to distinguish three categories of substructures: traditional stem-loops, pseudoknots, and those containing a potential pseudoknot base pairing region, called P-*structures*. In particular, for dynamic programming-based probability computation, not only the range [*i..j*] of each subsequence but also the range of each potential region contained with the subsequence is considered. This results in the requirements of $O(N^6)$ for running time and $O(N^4)$ for memory space, where N is the length of sequence. The main space consumption by the algorithm is the two matrices: a five-dimensional M and a four-

dimensional M1. The former is for storing the maximum probability for each subsequence x[*i*..*j*] (which may or may not contain a potential region x[*r*..*s*]) to admit the substructure specified by each nonterminal X. The latter is used for the maximum probability of forming a crossing helix by each pair of two subsequences.

This thesis work seeks to improve the algorithm performance for both memory usage and running time. It has been shown that general problem of RNA pseudoknotted structure prediction is computationally intractable [Akustu 2000; Lyngso *et al*., 2000]. Even for restricted pseudoknot categories, most existing prediction algorithms, including ours, require a worst case CPU time $O(N^6)$ and memory space $O(N^4)$, making it unrealistic to predict pseudoknots for RNA sequences of even a moderate length. So it is very important to explore approaches to reduce the resource requirements to make these algorithms practical. There have been a few algorithmic approaches developed for improving the efficiency of dynamic programming algorithms, including those solving biological problems. Most of these improvements take advantage of certain assumptions about the optimization function, such as convexity, concavity and sparsity, and usually a speedup factor of (N/log N) is realized. For example, Eppstein *et al*. [1991] used heuristic techniques in minimum energy-based stem-loop prediction algorithms and improved the space complexity with a little sacrifice of time. However, these techniques are unlikely to drastically reduce the computational resource requirements by the pseudoknot prediction algorithm due the intractability of the problem. On the other hand, in relevant research to date there are few implementation techniques developed for improving the efficiency of the prediction algorithms based on dynamic programming.

We have introduced two implementation techniques: *memory mapping* and *parallelism* for improving the efficiency of the pseudoknot prediction algorithm. We have been able to compress effectively the memory space used in this algorithm with a memory mapping technique that implements the actually used portions of matrices M1 and M. In particular, let N be the sequence length, and S_{max} and S_{min} be the maximum and minimum length of a base-paired helix, respectively, then the actually used portion of M1 has the size $[(N-S_{min}+1)(S_{max}-S_{min}+1)]^2$. For each nonterminal without a potential region, the actually needed matrix space in M is N(N+1)/2, while for each nonterminal containing a potential region, the required memory in M has the size N(N+1)(N+2)(N+3)/24. The technique maps the actually used portions of M1 and M to a one-dimensional array via the use of an offset mapping function. Since in general less than half of the nonterminals contain a potential region, the implemented algorithm needs only $1/48^{th}$ of the memory required without memory mapping. That is, the mapping technique allows us to predict sequences about 2.46 times longer than we could without the technique.

The computation of matrix M is also highly parallelizable. In particular, the matrix is computed diagonal by diagonal, starting from the major diagonal. Entries with indices (*i*, *j*) are computed in the increasing order of index distance Dij=*j*-*i*+1; the computation of an entry depends only on the entries with short index distances. Ideally, the computations for entries with the same index distance could be evenly partitioned and distributed to a number of parallel processors. Technically, to avoid the scenario in which the unexpected delay of one processor may cause another one to be idle, the algorithm employs many threads working on entries with the same index distance. For all threads, the sequential loop corresponding to index *i* is decomposed so that more than one iteration can be

executed at the same time. We have implemented a dynamic task distribution across processors so that those working on threads that complete early may be assigned more work through this load-balancing strategy. We refer to this method as *approach 1*.

We have implemented the algorithm with approach 1 via the openMP technique on the shared memory computer system SGI Origin 2000. As expected, the approach saves significant time. However, when the index distance approaches the limit N (the length of input sequence), not all processors are used. We have invented a better work partition strategy to overcome this drawback. Since for the computation at any index distance, each processor loops sequentially through all relevant nonterminals at each iteration of *i*, the partition for parallel work can be done at the level of nonterminals while still allowing all co-working threads to synchronize at the same index distance. We refer to this method as *approach 2*. The benefits of this new strategy are two-fold. First, the work can be partitioned more evenly to avoid unnecessary delay among threads. Secondly, even at the last index distance (Dij = N), all processors can be fully utilized because task assignment is based on different nonterminals instead of loop variable *i*. The parallelism techniques allow us to achieve a nearly 10-fold speedup (using 16 processors) for the conducted pseudoknot prediction tests conducted on real tmRNA data.

Most pseudoknot prediction algorithms are developed along two different lines of modeling: thermodynamics and stochastic grammars. Actually, both kinds of algorithms are dynamic programming-based, or more specifically, are CYK algorithm-like. Our implementation techniques can reduce computational resource consumption by taking advantage of the ways the matrix is organized and computed. While most existing prediction algorithms that evolved from the stem-loop prediction algorithm are all

variants of the dynamic programming-based CYK algorithm, our implementation strategies should be applicable to other dynamic programming matrix computation-based algorithms for pseudoknot prediction as well.

This thesis is organized as follows. In chapter 2, we introduce a new stochastic grammar model for pseudoknot prediction based on PCGS; the details of the pseudoknot prediction algorithm generated by this stochastic grammar model are described in chapter 3. In an effort to improve the efficiency of the algorithm, we present two implementation techniques that we have developed: *memory mapping* and *parallelism*, in chapter 4 and chapter 5, respectively. In chapter 6, we summarize our work and discuss the applicability of our model and the implementation techniques.

CHAPTER 2

A STOCHASTIC GRAMMATICAL MODELING OF RNA PSEUDOKNOTS

In RNA molecules, the hydrogen bonds between nucleotide bases cause the RNA molecules to fold in space from their linear sequences to the spatial secondary structures; the interaction between these nucleotides leads to different helices in RNA molecules. While ordinary RNA stem-loop structures can be viewed as nesting patterns of base-pairing nucleotides and can be modeled by SCFG [Eddy and Durbin, 1994; Sakakibara *et al*., 1994], RNA pseudoknots require crossing patterns of base-pairing that cannot be modeled by SCFG. Formally, pseudoknots require Chomsky context-sensitive grammars, which are much more complex than CFG as well as clumsy to implement. There have been some attempts to model pseudoknots in the spirit of CFG grammars, but these approaches are either too difficult to manipulate or unable to ensure global optimality.

Our new grammar modeling approach [Cai *et al*., 2003] for RNA pseudoknots is based on the *Parallel Communicating Grammar System* (PCGS) [Paun and Santean 1990; Cai, 1995, 1996] Such a grammar system consists of a set of Chomsky grammars (called *component*s, one of which is called the *master*) that can rewrite in parallel and communicate with each other synchronously under certain communication protocols. More concretely, one grammar component can query the sequences generated by other components and many components can make queries at the same time. PCGSs whose components are of a certain type are provably more powerful than a single Chomsky

grammar of the same type. Theoretically, a pseudoknot structure can be described as a normal stem-loop embedded with a crossing helix, which could be done by a CFG grammar and a few regular grammars designed to synchronously generate crossing double helices of the pseudoknot structure.

2.1 Parallel grammars to specify pseudoknots

To understand how the PCGS specifies pseudoknots, we first need a proper definition of an RNA pseudoknotted structure.

Given an RNA sequence, a *base region* is a subsequence of the sequence. Two *base-paired regions* are two base regions that form a double helix. The two base regions are also said to *contribute* to the helix.

Definition 2.1: Given a subsequence *t* in sequence *s*, a *potential region* in subsequence *t* is a base region within *t* that doesn't contribute to any helix in *t* but does contribute to a helix in *s*.

Definition 2.2: A sequence *t* is a *P-structure* if it contains a potential region, and *a non-trivial* P-structure if the potential region is between two base-paired regions.

Definition 2.3: A sequence s is a *pseudoknotted structure* if it contains two non-overlapped subsequences t_1,and t_2 such that:

1. t_1 and t_2 are both P-structures;
2. at least one of t_1 and t_2 is a non-trivial P-structure; and
3. two potential regions contained(respectively) in t_1 and t_2 form a helix.

Figure (2.1) shows one of the pseudoknots contained in the consensus structure of a β-proteobacteria family studied by Felden *et al*. [2001] and how it is described based on our definition of pseudoknotted structure.

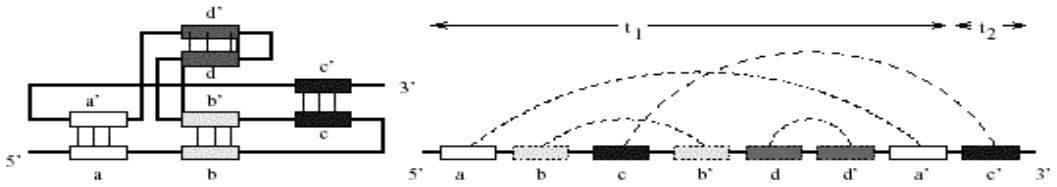

Figure 2.1: (a) a pseudoknot in the consensus structure of β-proteobacteria (b) as a pseudoknotted structure where c in t_1 and c' in t_2 are two potential regions.

Before we show how to model pseudoknotted structures with PCGS, we examine how to model two base-paired regions with synchronized grammar components. Table 2.1 shows a simple example of modeling a base-paired region with three regular grammar components: G1, G2 and G3. In this example, G1 starts querying G2, then G1 and G2 query G3 at the same time. Each step of querying is carried out synchronously. A querying grammar would copy the string derived by the queried grammar; each grammar returns to its start symbol after being queried. For example, G1, G2 and G3 collaborate to generate two base-paired regions. Table 2.2 shows a parallel derivation of the two base-paired region **aacg** and **cguu**. The first few steps of the parallel derivation are as follows. In G1, start symbol S1 derive Q2, which means querying G2. At the same time in G2, the start symbol S2 derives T, which can be recognized by G1, allowing G1 to copy the derived result T of G2 to G1 in the second step. Meanwhile, G2 returns to its start status S2 after being queried. In the afterward two steps, in G1 T derives T1 which in turn derives Q3 (querying G3); in G2 S2 derives T which in turn also derives Q3 (querying G3). So the nonterminal A derived in G3 will be copied to G1 and G2 at the same time respectively in the next step.

Table 2.1: PCGS productions for two base-paired regions

G1:	G2:	G3:
$< S1 > \rightarrow < Q2 >$	$< S2 > \rightarrow < T >$	$< S3 > \rightarrow < A >$
$< T > \rightarrow < T1 >$	$< T > \rightarrow < Q3 >$	$< S3 > \rightarrow < C >$
$< T1 > \rightarrow < Q3 >$	$< A > \rightarrow < Q3 > u$	$< S3 > \rightarrow < G >$
$< A > \rightarrow a < Q3 >$	$< C > \rightarrow < Q3 > g$	$< S3 > \rightarrow < U >$
$< C > \rightarrow c < Q3 >$	$< G > \rightarrow < Q3 > c$	$< S3 > \rightarrow < H >$
$< G > \rightarrow g < Q3 >$	$< U > \rightarrow < Q3 > a$	
$< U > \rightarrow u < Q3 >$	$< H > \rightarrow < H2 >$	

Table 2.2: derivation of base-paired regions **aacg** and **cguu**

$< S1 >$	$< S2 >$	$< S3 >$
$\Rightarrow < Q2 >$	$\Rightarrow < T >$	$\Rightarrow < A >$
$\Rightarrow < T >$	$\Rightarrow < S2 >$	$\Rightarrow < A >$
$\Rightarrow < T1 >$	$\Rightarrow < T >$	$\Rightarrow < A >$
$\Rightarrow < Q3 >$	$\Rightarrow < Q3 >$	$\Rightarrow < S3 >$
$\Rightarrow < A >$	$\Rightarrow < A >$	$\Rightarrow < A >$
$\Rightarrow a < Q3 >$	$\Rightarrow < Q3 > u$	$\Rightarrow < S3 >$
$\Rightarrow a < A >$	$\Rightarrow < A > u$	$\Rightarrow < C >$
$\Rightarrow aa < Q3 >$	$\Rightarrow < Q3 > uu$	$\Rightarrow < S3 >$
$\Rightarrow aa < C >$	$\Rightarrow < C > uu$	$\Rightarrow < G >$
$\Rightarrow aac < Q3 >$	$\Rightarrow < Q3 > guu$	$\Rightarrow < S3 >$
$\Rightarrow aac < G >$	$\Rightarrow < G > guu$	$\Rightarrow < H >$
$\Rightarrow aacg < Q3 >$	$\Rightarrow < Q3 > cguu$	$\Rightarrow < S3 >$
$\Rightarrow aacg < H >$	$\Rightarrow < H > cguu$	$\Rightarrow < A >$
$\Rightarrow aacg < H >$	$\Rightarrow < H2 > cguu$	

To model a pseudoknot with a PCGS, basically, a context-free grammar component G0 is used to describe a stem-loop together with G1, G2 and G3 that describe two base-paired regions to form a double helix crossing with the stem-loop to form a pseudoknot. Table 2.3 and Figure 2.2 give the master grammar G0 and a derivation tree of the pseudoknot, respectively. In Table 2.3, nonterminal <Pk> stands for pseudoknot, and <NTP-struct > for non-trivial P-structure:

Table 2.3 Master grammar G_0 describing a pseudoknotted structure

G0: < S0 > $\rightarrow$ < Pk >
< Pk > $\rightarrow$ < P-struct >< NTP-struct >
< P-struct > $\rightarrow$ < primary > < Q1 > < primary >
< NTP-struct > $\rightarrow$ < primary > < P-helix > < primary >
< P-helix > $\rightarrow$ a < P-helix > u
< P-helix > $\rightarrow$ c < P-helix > g
< P-helix > $\rightarrow$ g < P-helix > c
< P-helix > $\rightarrow$ u < P-helix > a
< P-helix > $\rightarrow$ a < NTP-struct >
< P-helix > $\rightarrow$ < Q2 >

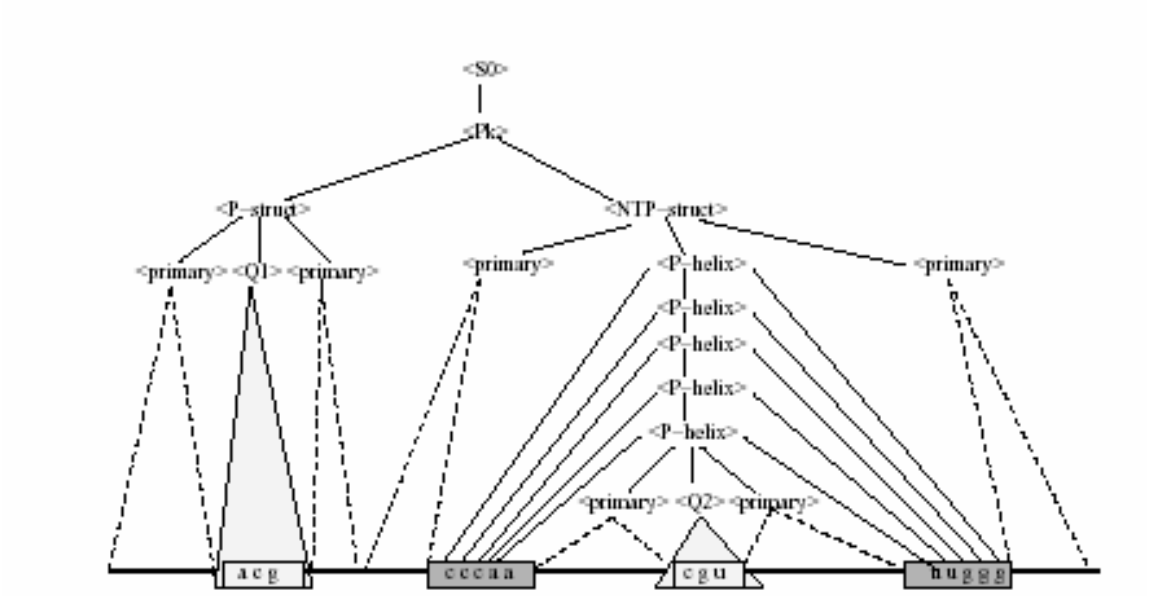

Figure 2.2 The derivation tree of a pseudoknotted structure.

2.2 Technical Observations:

The PCGS description for RNA pseudoknotted structures has lead to the following technical observations:

(1) A pseudoknot structure of an RNA sequence can technically be specified by a single CFG allowed to use special query symbols as non-terminals. These query

symbols do not have rewriting rules but are simple macros corresponding to potential base paired regions. Only one query symbol is needed for the most restricted case of pseudoknots. In our grammar model, a simple pseudoknot can be formed by a stem-loop with a potential base pairing region, e.g. <P-stem >, and a non-structured subsequence containing another potential base pairing region, e.g. < P-primary >. More complex pseudoknotted structures can be formed by employing more than one query symbol for substructures with potential base pairing regions.

(2) The stochastic version of our grammar model is as simple as a SCFG. The only difference lies in associating probability parameters with query symbols because there are no rewriting rules for these special nonterminals. Unlike a deterministic PCGS model, where the probability of two potential base pairing regions forming a crossing helix is described by two regular components that synchronously generate them respectively; in our grammar, we associate probability parameters with query symbols as nonterminals. The probability of the stem formed by two potential base regions corresponding to <Q> symbols is determined by a complementary alignment between the two base regions. Therefore the probability parameters for query symbols could be represented in a 5x5 matrix that describes a probability distribution for all possible base pairs among four nucleotide bases and the gap.

CHAPTER 3

PSEUDOKNOT PREDICTION ALGORITHM

The grammar modeling framework introduced in the previous discussion allows automatic generation of a prediction algorithm from each specified grammar model describing a specific pseudoknot structure. SCFGs use a grammar to generate a string by applying a set of string rewriting rules or productions. The order of productions used to generate the string (input sequence) can be represented in a parsing tree which accordingly can be interpreted as different biological structures such as stem-loops and base pairs. Due to the ambiguity of the set of the grammar system, a given yield (sequence) could be derived by a number of parsing trees corresponding to different structures, and each parsing tree can have a probability calculated from the probabilities assigned to each production of the SCFG grammar. The predicted structure should correspond to the one with the largest probability. An SCFG grammar can model RNA secondary structures such as stem-loops but not pseudoknots.

In our PCGS modeling of pseudoknots, because the auxiliary grammar components generating a crossing helix can be substituted by the pairwise (complementary) alignment between two potential regions, technically we can make our modeling almost the same as a SCFG by allowing the query symbols as special nonterminals. Query symbols can derive any potential subsequence as a contribution to the pseudoknot, while the probability of forming the pseudoknot is computed by the alignment between the

subsequences derived by query symbols. Therefore, we introduce a 5x5 probability matrix that describes the probability distribution in the double helices for all possible base pairs among the four nucleotides and gaps, which are included for the purpose of producing bulges.

3.1 The Algorithm

This automated pseudoknot prediction algorithm is dynamic programming-based and resembles the CYK algorithm. With probabilities associated with productions in the SCFG including special query nonterminals, it calculates the maximum probability for deriving the given input sequence by computing the maximum probabilities of all the substructures adopted by each nonterminal. The algorithm discerns three categories of substructures: traditional stem-loops (called *N-structure*), pseudoknots, and *P-structures* for those containing a potential region. Let x be an input sequence with length N, the maximum probability for subsequence x[*i*..*j*] to admit the substructure specified by nonterminal X is essentially the maximum probability for X to derive subsequence x[*i*..*j*]. The definition can be given as a recurrence. We distinguish three categories of nonterminals (or substructures) in the following definition. We assume that the SCFG is given in the Chomsky Normal Form (CNF).

(1) First, if X specifies an N-structure, then the maximum probability

$$\Pr(X, i, j) = \max \{ \Pr(Y, i, k) \times \Pr(Z, k+1, j) \times \Pr(X \rightarrow YZ)\} \quad (3.1)$$

where the maximization is taken over all possible productions $X \rightarrow YZ$ and all possible k, $i \leq k < j$

(2) If X specifies a pseudoknot, then the probability

$Pr(X, i, j) = \max \{ Pr(Y, i, k, s, t) \times Pr(Z, k+1, j, u, v) \times Pr(X \rightarrow YZ) \times Pr(s, t, u, v)\}$ (3.2)

where the maximization is taken over all possible productions $X \rightarrow YZ$, all possible k, $i \leq k < j$, and all appropriate s, t, u, and v, $i \leq s, t < k$; $k+1 \leq u, v < j$; and where $Pr(s, t, u, v)$ represents the maximum probability of subsequence $x[s..t]$ and $x[u..v]$ forming a crossing helix.

(3) If X specifies a P-structure, then the probability

$Pr(X, i, j , k, l)= \max\{Pr(Y, i, r, k, l) \times Pr(Z, r+1, j) \times Pr(X \rightarrow YZ) \}$

Or

$$\max\{Pr(Y, i, r) \times Pr(Z, r+1, j, k, l) \times Pr(X \rightarrow YZ) \} \quad (3.3)$$

where the maximization is taken over all possible productions $X \rightarrow YZ$ and all possible r, $i \leq r < j$, and all appropriate indexes k and l for potential region $x[k,..l]$ within the P-structure $x[i..j]$. The base case for the recurrence is that $Pr(X, i, i) = Pr(X \rightarrow x[i])$ for every i=1,…, N. The global optimal solution is $Pr(S_0, 1, N)$, where S_0 is the starting symbol in the grammar.

Let Dij be the distance between index i and j. We can compute an optimal solution to $Pr(X, i, j)$ in a ascending order of Dij until $Pr(S_0, 1, N)$ is computed. There is a natural dynamic programming approach to compute the maximum probability of the optimal predicted structure. The detailed algorithm is the following:

Given an input sequence $x = x[1..n]$ of length N, the prediction algorithm must compute the following three matrices.

(1). Matrix M1, such that, for given $i \leq j$ and $k \leq l$, for $j<k$, $M1[i, j, k, l]$ is the maximum probability that two base regions $x[i..j]$ and $x[k..l]$ form a crossing helix.

The computation of each entry M1[*i*, *j*, *k*, *l*] is accomplished by the pairwise complementary alignment between the two base regions x[*i..j*] and x[*k..l*]. The scoring measure between two aligned bases is essentially the probability for them to form a base pair. The score for a bulge or a loop is the measure of aligning all the bases in a bulge (or a loop). The algorithm is similar to the global pairwise alignment algorithm. The required memory space $O(N^4)$ and the running time for the algorithm is $O(N^3)$.

(2). Matrix M, such that for given nonterminal X, $i \le j$, and $k \le l$, where $i \le k$, and $l \le j$, M[X, *i* ,*j*, *k*, *l*] is the maximum probability that X→x[*i.. j*] with exclusion of potential region x[*k..l*]. The computation of this matrix is the core of the prediction algorithm. The maximum probability for sequence x to have the predicted structure is $M[S_0, 1, N, 0, 0]$. According to the probability definition for different nonterminals given in equations (3.1), (3.2) and (3.3), we have the following process to compute matrix M. Let Dij = *j-i+1* be the distance between index *i* and *j*,

(1) Initialization:

$$M[X, i, i, 0, 0] = \Pr[X \rightarrow x[i]] \quad (3.4)$$

For special nonterminal $X \in Q$ that is a query symbol,

$M[X, i, j, k, l] = 1$ with $i = k$, $j = l$

and set all other entries to 0.

Compute entries M[X, *i*, *j*, *k*, *l*], for all $i \le k$, $l < j$, Dij > 1, and all nonterminals X according to the following rule: for all entries with certain Dij value, we use entries with smaller Dij values to compute starting from Dij=1. The computation continues in the ascending order of index distance Dij. From the perspective of the matrix, the

computation is conducted diagonal by diagonal, starting from the major one, where the Dij =1.

The following pseudocode fragment shows the computing procedure of matrix M:

```
//computing is conducted at a certain Dij level each time, in an ascending order of Dij
For ( Dij=2; Dij ≤ N; Dij ++)// Dij indicates the distance between index i and j, inclusive
 {
     for (i=0; i ≤ N-Dij; i++)
        {
            j=i+Dij -1;
          for (X=0; X < tableindex; X++)//loop through all nonterminals
            {
             // computing M[X, i, j, k, l]…
              if (X is N-structure) use equation (3.1) ;
              if( X is Pseudoknot) use equation (3.2);
              if (X is P-structure) use equation (3.3);
            }
        }
 }
```

(3). Matrix S, such that for given nonterminal X, $i \le j$, and $k \le l$, where $i \le k$ and $l \le j$, S[X, *i*, *j*, *k*, *l*] holds the production X→YZ and number *k*, $i \le k < j$, used to obtain the maximum probability M[X, *i*, *j*, *k*, *l*] for which Y→x[*i*..*k*] and Z→x[*k* + 1..*j*] and in the case X being a pseudoknot, the tuple (*r*, s, *u*, *v*), $r < s \le k$ and $k < u < v$, for potential base regions x[*r*..s] and x[*u*..*v*] that form a (crossing) helix. The memory space of S is $O(N^4)$; each entry in S can be computed along with matrix M. More specifically,

whenever we get the optimal substructure of any subsequence, we store the first production deriving this substructure in the matrix S.

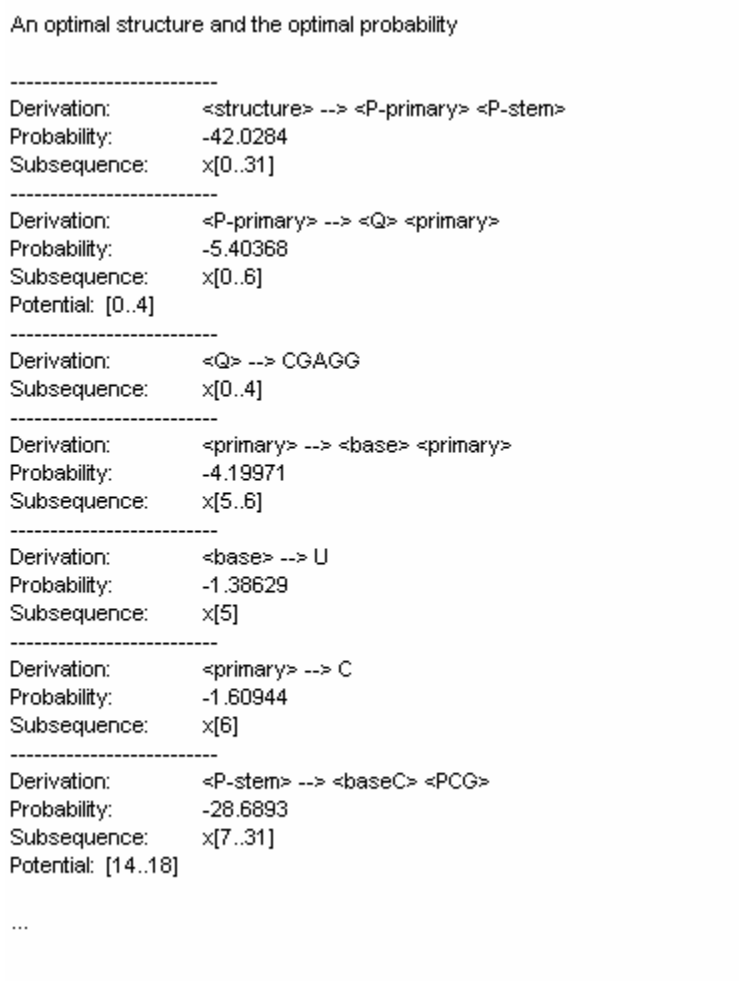

Figure 3.1 A part of a prediction result generated by our algorithm

3.2 Computational Complexity of Algorithm

A prediction example is shown in Figure 3.1. In theory, like most pseudoknot prediction algorithms developed by others, our algorithm takes $O(N^6)$ for CPU running time and $O(N^4)$ for memory space. These complexities can be derived from the dynamic program

used to compute the major matrix M. It is obvious that the memory space of matrix M is $O(N^4)$. We can analyze the running time of computing matrix M. In our algorithm, nonterminals of different categories have different time complexities. First, if nonterminal X specifies N-structure, according to equation (3.1), entry M[X, *i*, *j*, 0, 0] stores the probability of X deriving subsequence x[*i*..*j*]. The number of such entries is N^2 and each entry take O(N) time to compute; so the time required to compute all such entries with nonterminal X is $O(N^3)$. Secondly, if nonterminal X specifies a pseudoknot, according to equation (3.2), the number of entries is also N^2. There are five variables *k*, s, *t*, *u*, *v*, all with the possible range from 1 to N in equation (3.2). Because subsequences x[s..*t*] and x[*u*..*v*] are dedicated to forming a crossing helix, in real RNA molecules, the lengths of two base paired regions could not be too different. Therefore, *v* could not vary too much once s , *t*, and *u* are fixed, thus each entry for X can still take $O(N^4)$ time to compute. The total computing time for pseudoknot nonterminal X will be $O(N^6)$. Finally, if nonterminal X specifies a P-structure, the number of entries for X is $O(N^4)$. According to equation (3.3), each entry can take O(N) time to compute and the running time for P-structure nonterminal X is $O(N^5)$. To summarize, computing matrix M can take $O(N^4)$ memory space and O (N^6) running time.

The general problem of RNA pseudoknot prediction is NP-hard [Lyngso and Pederson, 2000]. There have been a few algorithmic approaches developed for improving the efficiency of dynamic programming algorithms, including those solving biological problems [Eppstein *et al.*, 1991; Waterman *et al*., 1976; Miller and Myers, 1988]. Most of these improvements take advantage of certain assumptions about the optimization

function, such as convexity, concavity and sparsity, and usually a speedup factor of (N/log N) is realized.

More recently, Eddy presents a memory efficient algorithm [Eddy, 2002] for the problem of aligning a sequence to an RNA profile defined with the Covariance model [Eddy and Durbin, 1994]. The algorithm takes advantage of characteristics of the alignment problem that allows him to employ a divide-and-conquer approach used by Myers and Millers [Myers and Miller, 1988]. However, although the alignment problem is very close to the prediction problem, this nice technique does not seem to be applicable to the prediction problem with stochastic grammar models. The intractability of the prediction problem seems to be inherent and no approach seems to be able to drastically improve the efficiency of its algorithms. For this reason, we examine implementation techniques.

CHAPTER 4

IMPROVEMENT OF SPACE USAGE

For most prediction algorithms, RNA structure prediction is memory-consuming work with $O(N^2)$ for stem-loops and $O(N^4)$ for pseudoknots. In particular, our prediction algorithm computes a four dimensional matrix M1 and a five dimensional matrix M. In this chapter, we introduce an implementation technique to improve the usage of memory space by the algorithm.

We first analyze in detail the memory requirement for an input RNA sequence in the prediction algorithm. We need to define a few variables for the convenience of our following discussion:

N: the sequence length

gnpNo: the number of nonterminals without potential regions

gpNo: the number of nonterminals including potential regions

minStem: the minimum length of a stem

maxStem: the maximum length of a stem

the size of matrix M1 is N^4, the size of M is $(gnpNo+gpNo)*N^4$.

Actually, this memory space is not fully used by the algorithm. We examine matrix M1 first. In our algorithm, M1 stores the probabilities of any two potential base pairing regions, with each entry M1[*i, j, k, l*] subject to the following constraints:

minStem < | *j* – *i* | <maxStem, and

minStem < | $l - k$ | < maxStem.

Actually matrix M1 is a sparse matrix due to the constraints applied to the length of stems/helices. The actually needed memory space can be determined from the following formula:

$((N - \text{minStem} + 1 + N - \text{maxStem} + 1) * (\text{maxStem} - \text{minStem} + 1)/2)^2$ (4.1)

The current usage of matrix M allows us to save even further on memory. In fact, there are two kinds of nonterminals: P-structure for those with a potential region and N-structure for those without. They have different memory requirements. For the latter, indexes i, j are enough for it to represent a substructure. In this situation, also due to the constraint of $i < j$, the actually needed memory is:

$$gnNo * N * (N + 1)/2 \quad (4.2)$$

For P-structure nonterminals, i.e., those containing a potential region, because the constraints $i < j$ and $k < l$, the actually required memory space actually is:

$$gpNo * N * (N + 1) * (N+2) * (N+3)/24 \quad (4.3)$$

To compress the sparse matrices, we have devised a mapping function to map the matrices M and M1 to two linear memory areas with sizes just satisfying the actual space requirements. In the general case, the ratio gnpNo/(gnpNo+ gpNo) is approximately 50%. Considering when N is large enough, memory for N-structure nonterminals could be ignored in the total memory allocation. Then for the space required by P-structure nonterminals, we use only about $N^4/48$ spaces. In other words, by our memory mapping approach the maximum length of sequences that we can process now is around 2.46 times longer than that of sequences processed without memory mapping.

4.1 Memory Mapping

We discuss the details of a few mapping functions we have developed for sparse matrix compression. These mapping functions should not be too complicated because of concerns about speed. In fact, the computation of these functions directly adds a multiplicative factor to the total running time. We have developed concise formulae to project each matrix element to the corresponding cell in the array. We first distinguish two different nonterminals, with and without potential regions and allocate different memory spaces for them. Let *npM* be a one-dimensional array that can store the maximum probability of any subsequence x[*i, j*] admitting the substructure specified by any nonterminal without a potential region, and let *pM* to be a one –dimensional array that can store the maximum probability of any subsequence x[*i, j*] admitting the substructure specified by any nonterminal with a potential region x[*k, l*], where *k* and *l* indicate the range of the potential region. First we give the formula for computing the size of *npM*:

Size of $npM = gnpNo * \sum_{dij=1}^{N} (N - Dij)$

$= gnpNo * (N+1) * N/2$

Where Dij $= j - i + 1$ stands for the distance between index *i* and *j*,

The following is the offset of array element in *npM* for storing the maximum probability of any subsequence x[*i , j*] admitting the substructure specified by any nonterminal X:

$$X * N * (N+1)/2 + (N + N - Dij) * (Dij + 1)/2 + i$$

where X is the sequential number assigned to the nonterminal X, starting from 0.

The size of array *pM* for all nonterminals with potential region is governed by the following formula:

$$gpNo * \sum_{Dij=1}^{N}(N - Dij) * (\sum_{Dkl=1}^{Dij}(Dij - Dkl)) = gpNo * N * (N+1) * (N+2) * (N+3)/24$$

The following is the offset of array element in *pM* for storing the maximum probability of any subsequence x[*i, j*] admitting the substructure specified by any nonterminal with a potential region x[*k, l*]:

$$offset1 = [-3 * Dij^4 + (4N-6) * Dij^3 + (12N+3)Dij^2 + (8N+6)Dij]/24 +$$
$$i * (Dij+2) * (Dij+1)/2 + (2Dij + 3 - Dkl) * Dkl/2 + k - i$$

$$offset = X * N * (N+1) * (N+2) * (N+3)/24 + offset1$$

For matrix M1, considering the minimum and maximum lengths of a stem, the memory space needed is:

$$\sum_{Dij=\min Stem}^{\max Stem}(N - Dij) * \sum_{Dkl=\min Stem}^{\max Stem}(N - Dkl) = [(N - \min Stem + 1 + N - \max Stem + 1) *$$
$$(\max Stem - \min Stem + 1)/2]^2$$

The offset in matrix M1 for the probability of any two potential regions [*i, j*] and [*k, l*]:

$$klLen = (N - \min Stem + 1 + N - \max Stem + 1) * (\max Stem - \min Stem + 1)/2$$
$$offset = (N - \min Stem + N - Dij + 2) * (Dij - \min Stem)/2 + i) * klLen +$$
$$(N - \min Stem + N - Dkl + 3) * (Dkl - \min Stem)/2 + k$$

In our source code, we first allocate memory space for different matrices according to the length of input sequence and the specified grammar model used in the test; then we use the offset computed based on the above formulae to refer to every specific entry in the matrix.

Table 4.1 Comparison of memory usage between methods with and without memory-mapping technique, where the grammar model adopted has 8 nonterminals with a potential region and 7 nonterminals without.

Approach \ sequence Length	40	80	100	120
Memory usage without memory-mapping	317M	5G	12G	26G
Memory usage with memory-mapping	12M	161M	389M	799M
Percentage of Memory saved	96%	97%	97%	97%

4.2 Further improvement in the memory usage: modifying grammar writing style

Although we can drastically improve the memory usage in our algorithm through the memory-mapping technique, it is clear that two factors decide how far we can go in saving memory with our implementation technique. The number of nonterminals without a potential region, and particularly the nonterminals with a potential region are multipliers in memory usage expression (1), (2), and (3). To ease implementation, our grammar model adopts the Chomsky Normal Form (CNF) grammar style as most stochastic grammar models do, which introduces many unnecessary nonterminals. To further improve memory usage, we can modify our algorithm to accept less restricted production styles compared to CNF such that the memory usage in our algorithm will be curtailed. As an example, we show in Table 4.2 one of the grammar models we used in the test and its modified version that will be accepted in our future algorithm. In this example, the original grammar consists of 14 nonterminals, 7 of which contain a potential region. After the modification, the number of noterminals decreases to 6 and those with a potential region to 3. Generally, more than half of memory space, and accordingly more than half of running time, can be saved with less restricted grammar rules.

Table 4.2 (a) a grammar model compliable to CNF; (b) a modified grammar.

Each nonterminal is put in '< >', in particular, nonterminals with potential region start with 'P'

<PK>: <P-primary> <P-stem>

<P-primary>: <Y> <P-primary>

| <Q><Y>

<Y>: <base><Y>

| A | C | G | U

<base>: A |C |G |U

<P-stem>:<baseA><PAU>

|<baseC><PCG>

|<baseG><PGC>

|<baseU><PUA>

|<Q><Y>

<PAU>: <P-stem><baseU>

<PCG>: <P-stem><baseG>

<PGC>: <P-stem><baseC>

<PUA>: <P-stem><baseA>

<baseA>: A

<baseC>: C

<baseG>: G

<baseU>: U

<PK>: <P-primary> <P-stem>

<P-primary>: <Y> <P-primary>

| <Q><Y>

<Y>: <base><Y>

| A | C | G | U

<base>: A |C |G |U

<P-stem>: A <P-stem> U

|C <P-stem> G

|G <P-stem> C

| U <P-stem> A

|<Q><Y>

CHAPTER 5

PARALLEL COMPUTATION FOR THE PREDICTION ALGORITHM

The pseudoknot prediction algorithm has the worst case computational time $O(N^6)$ for filling the core four-dimensional matrix M. In addition, the extra overhead imposed by the memory mapping technique is about 5 times that without the memory mapping implementation. To ensure that our program can handle longer RNA sequences while overcoming the running time barrier, we resort to parallelizing the computation of the prediction algorithm.

5.1 Parallel computing systems and techniques

In contrast to sequential computing on a single CPU processor, parallel computing involves partitioning and distributing computation work across more than one processor in an effort to improve the time used for the computation. In general, parallel computing system architectures fall into two categories according to two different memory organization strategies: distributed memory systems and shared memory systems.

In a distributed memory system, each processor has its own local memory that may not be accessed by other processors in the system. Communications between processors are carried out via the technique of message-passing. These type of systems are highly scalable while programming on them is difficult. The most popular programming techniques involving message-passing including Message Passing Interface(MPI) [Gropp

et al., 1994] and Parallel Virtual Machine (PVM) [Geist *et al*., 1994]. In contrast, in a shared-memory system, all processors share a centralized common memory. Communications between processors are carried out through a shared memory. It is easy to program on this type of architecture. However, it is hard to scale up because when the number of processors increases to a certain extent, the delay in access to the shared memory by processors will become increasingly severe and the system efficiency will drastically drop. The programming techniques on such a platform include openMP [openMP website], Pthread [Lewis *et al*., 1998], and others. In the following, we introduce in detail the two different parallel computing techniques: openMP(Open specification for Multi-Pocessing) and MPI (message passing interface), which apply to shared memory system and distributed memory systems respectively.

openMP Technique: An application program interface (API) that may be used to explicitly direct *multi-threaded and shared memory parallelism*, openMP as a standard started in 1997 when newer shared memory machine architectures started to become prevalent. It was jointly defined and endorsed by a group of major computer hardware and software vendors such as Intel, IBM, Compaq/Digital, HP, Sun and Silicon Graphics etc. Its compiler directive–based programming style granted users great convenience to parallelize their sequential code without losing its powerful parallelization capability. It is easy to use and portable to many platforms.

OpenMP works on shared memory systems with multi-thread support and adopts a parallel execution model called “Fork and Join”. The parallelizing starts with a thread of control (master thread) at the beginning of a parallel region; the control thread then spawns (i.e., forks) many working threads to perform the parallelized job. These threads

then join back to the master thread after their work and only the master thread continues on. Figure[5.1] shows a graphic representation of the model.

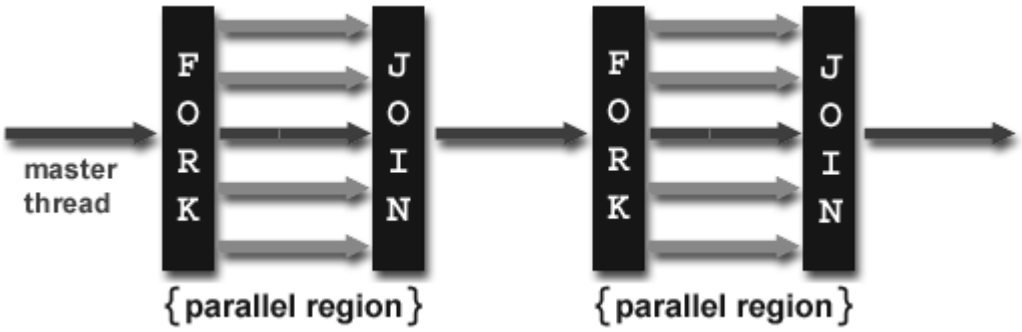

Figure 5.1 parallel computation model adopted by openMP

MPI technique: MPI is a library specification for message-passing, proposed as a standard by a group of vendors, software engineers, and users. MPI allows the coordination of a program to run as multiple processes in a distributed memory environment. Yet it is flexible enough to also be used in a shared memory system. It is available on a wide variety of parallel computers such as the IBM SP2, the Silicon Graphics Origin 2000, or a cluster of workstations (homogenous or heterogeneous) over a local network. The standardization of the MPI library is one of its most powerful features. This means that programmers can write code containing MPI subroutines and function calls that will work on any machine on which the MPI library is installed without having to make changes in the code. So far, MPI supports programming with Fortran and C/C++. Its library provides a set of communication routines for data transfer among processors.

5.2 Data Dependency and Analysis of parallelism

The computation of the core matrix M by the pseudoknot prediction algorithm is performed diagonal by diagonal, starting from the major diagonal (see Figure 5.2). That is, let index distance Dij = *j-i* + 1, the algorithm computes elements of M based on the value of Dij, from the smallest to the largest. In particular, the computation starts on elements with Dij = 1, then continues on elements with higher Dij value until Dij =N. In each step, elements associated with a certain Dij value depend only on those elements associated with a smaller Dij, but not on each other. According to this type of data dependency, the computation of elements with the same Dij value can be independent of each other and can be parallelized across a number parallel processors. In other word, for each index distance Dij, a number of processors can work in parallel for the computation of matrix elements as long as no processors go further to index distance (Dij +1) until all processors finish the work for Dij.

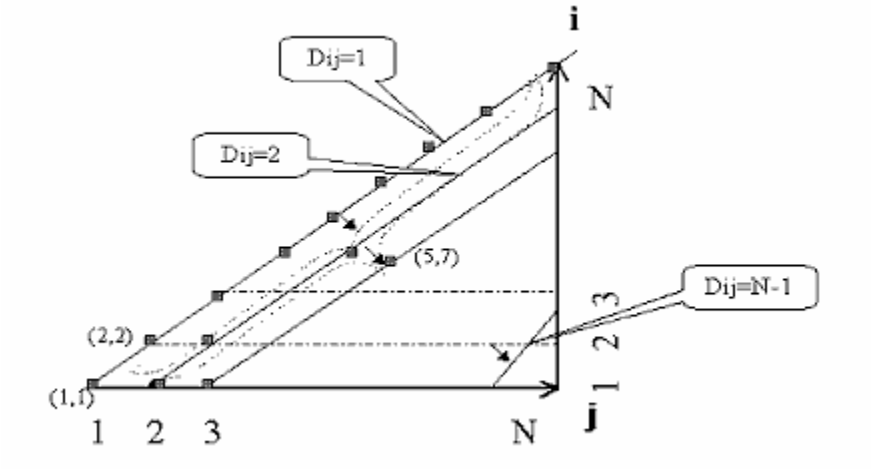

Figure 5.2 The order of filling matrix M

5.3 Implementation of Parallelism

Given the data dependency in the algorithm, parallel computation for the algorithm is likely to lead to a speedup for the running time. We attempted to implement the parallelism with the MPI technique in a distributed memory system, but the idea was given up because of the serious trade off between the overhead cost posed and the efficiency achieved by this method. We realized that the only parallelizable portion of the algorithm is the process of filling the data on the same diagonal (those with the same index distance) whose computations are independent of each other. Each computation needs all previously computed data (entries with lower index distances) and each processor doing the computation should be able to access the computed results from all other processors. If the parallel computation were to be implemented on a distributed memory system, the data sharing would have to rely on communications among processors since every processor can have access to only its local memory. Such communications would be very frequent because at each step, every processor would have to collect data that have been computed in the previous steps. Such huge communications overhead would deteriorate the performance of the algorithm. Employing a number of processors might even result in worse performance than that for running sequential code.

A more realistic solution is to parallelize the algorithm on a shared memory system. Considering the advantages it can offer, we have chosen the openMP technique to fulfill the parallelization task. In particular, in a shared memory system, processors exchange data through a common memory which each processor can access efficiently. In other

words, there is no difference for a processor to access data generated by itself and that generated by other processors. The technical details on a shared-memory system can also be handled in a natural manner. We can employ many threads working on elements of the matrix M of the same index distance (i.e., those on the same diagonal in Figure 5.2); and all threads are synchronized at the same point before they go further, working on the elements of a higher index distance (i.e., those on the next diagonal). More concretely, we can decompose the second "for loop" in the code given in section 3.1 according to the values of variable *i* such that more than one iteration could be executed at the same time. Dynamic task distribution across processors could also benefit our computation in that fast threads can be assigned more work through this sort of load-balancing strategy. For example, thread 1 may work on iteration (*i*=0, 5, 6) while thread 2 on iteration (*i*=1,3,7), etc.. We refer to this method as *approach 1*.

Basically, through the complier directives provided by openMP, we have implemented this parallelization scheme for the algorithm on the shared memory parallel computer system SGI origin 2000. Table 5.1 shows some speedup results achieved by the parallelized algorithm on tested sequences with average length 64.

Table 5.1: running time and speedup for approach 1 with different number of processors

Proc num	1	4	8	16
Time (mins)	30:03	8:50	5:20	3:33
Speedup	1.00	3.40	5.63	8.46

5.4 Further improvement of algorithm

While our parallel program processed RNA sequences, it also generated a running log, which recorded detailed information regarding the running time of each loop and the time each processor spends on each iteration. Although there is significant speedup by employing *approach 1*, the running log shows two limits to this approach, which are analyzed in the following:

1. When Dij approaches the limit N, only a few processors could be employed in the parallel computation because the smallest work unit assigned to a thread is an iteration corresponding to loop variable *i*,.
2. In most cases, at a given Dij step, the processing time of each iteration for each processor is almost the same. However, there are cases for some special value of Dij, the total number of iterations cannot be evenly divided by the number of processors. In this situation, at least one processor will assume additional computation load of one more iteration than others. All other processors must wait to synchronize with this processor before they can continue the computation at the next index distance.

Tables 5.2 and 5.3 show a portion of a running time log for an input RNA sequence with length 63, where a total of 8 processors (each thread bound to a processor) were employed. In each row of the tables, d*ij* is the value of Dij, *tid* stands for thread identifier (from 0 ~ 7), *i* represents the iteration number assigned to thread *tid*, and followed by the time (in *second*) spent on the iteration *i*., and *dij_time* is the total time spent on specific Dij.

From Table 5.2, we observed the following: because the smallest computing unit assigned to each thread is related to iteration variable *i* for each index distance, for Dij =61, the possible values for iteration parameters are is 0, 1 and 2. Therefore, at most 3 threads could be used (here are threads 0 , 1, and 3). Similarly, 2 threads could be used for for Dij =62 and 1 thread for Dij =63. The remaining threads are idle for these cases.

In Table 5.3, for Dij =54, the loop count corresponding to loop variable *i* is 10. Thread 4 and thread 6 work on two iterations while every other thread works on one iteration for total 8 threads. When each iteration takes almost the same time (here 6.38s) for each thread, other threads have to wait for thread 4 and 6 to finish their additional work. So the time spent is doubled (12.77s). For Dij=55, thread 4 takes an additional iteration while other threads wait for it to be finished. So the total time spent on this index distance is approximately the time for two iterations (13.67s). For Dij=56, each thread takes exactly one iteration, they are almost finished at the same time. Hence the total time is 7.39s.

Table 5.2 When Dij value is approaching the limit N (N is the sequence length, 63 in this case). Although 8 processors are employed, only a few processors (threads) could be utilized because the smallest computation unit is based on each iteration *i* .

dij=61,tid=3,*i*=1,time: 9.98	dij=62,tid=1,*i*=1,time: 10.59	dij=63,tid=0,*i*=0,time: 11.37
dij=61,tid=1,*i*=2,time: 9.98	dij=62,tid=0,*i*=0,time: 10.73	dij=63, dij_time=11.37
dij=61,tid=0,*i*=0,time: 10.11	dij=62,dij_time=10.73	
dij=61,dij_time=10.11		

Table 5.3 For some Dij's, the total number of iterations cannot be evenly divided by the thread count; some threads have to do additional work while others have to wait before synchronization.

dij=54,tid=4,*i*=3,time: 6.38
dij=54,tid=6,*i*=2,time: 6.38
dij=54,tid=5,*i*=1,time: 6.38
dij=54,tid=7,*i*=5,time: 6.39
dij=54,tid=3,*i*=4,time: 6.38
dij=54,tid=1,*i*=6,time: 6.38
dij=54,tid=2,*i*=7,time: 6.38
dij=54,tid=0,*i*=0,time: 6.46
dij=54,tid=4,*i*=8,time: 6.39
dij=54,tid=6,*i*=9,time: 6.38
dij=54,dij_time=12.77

dij=55,tid=4,*i*=1,time: 6.82
dij=55,tid=1,*i*=6,time: 6.83
dij=55,tid=3,*i*=2,time: 6.83
dij=55,tid=2,*i*=5,time: 6.84
dij=55,tid=5,*i*=3,time: 6.84
dij=55,tid=6,*i*=4,time: 6.84
dij=55,tid=7,*i*=7,time: 6.83
dij=55,tid=0,*i*=0,time: 6.92
dij=55,tid=4,*i*=8,time: 6.84
dij=55,dij_time=13.67

dij=56,tid=1,*i*=4,time: 7.31
dij=56,tid=2,*i*=3,time: 7.29
dij=56,tid=5,*i*=1,time: 7.29
dij=56,tid=3,*i*=2,time: 7.30
dij=56,tid=6,*i*=7,time: 7.30
dij=56,tid=7,*i*=6,time: 7.30
dij=56,tid=4,*i*=5,time: 7.30
dij=56,tid=0,*i*=0,time: 7.39
dij=56,dij_time=7.39

To avoid those drawbacks, we have resorted to a better work partition strategy. We have observed that for the computation at any Dij, every processor actually needs to loop sequentially through all relevant non-terminals for each iteration of *i*. Actually we can partition computation work at the non-terminal level and let all co-working threads still synchronize for each index distance. We call this strategy *approach 2*. The benefits of this new strategy are two-fold. First, for the computation at each Dij, an earlier finished processor may calculate the probabilities of nonterminals for the elements belonging to other unfinished processors. Secondly, even when the index distance is at its limit (N), all assigned processors could be fully utilized because task assignment is based on different

nonterminals as well as index distances. This may slightly increase the time spent on each iteration because of the system overhead invoked by this fine-grained partition strategy. But the minor deficiency is definitely compensated for by the aforementioned overall benefits. The running log in Tables 5.4 and 5.5 show that the new parallelization strategy *approach 2* is effective in overcoming the drawbacks of in approach 1.

Table 5.4 Time (seconds) comparison between two approaches for some special Dij values. Here 8 processors are employed.

	Approach 1	Approach 2
Dij=46, iteration count=16+2	10.57	9.14
Dij=47, iteration count=16+1	11.46	9.42
Dij=48, iteration count=16	8.35	9.63
Dij=54, iteration count=8+2	12.77	10.17
Dij=55, iteration count=8+1	13.67	10.3
Dij=56, iteration count=8	7.39	8.92

Table 5.5 Time (seconds) comparison between two approaches when Dij is approaching to the limit 63(the sequence length)

	Approach 1	Approach 2
Dij=59,iteration count=5	8.95	8.21
Dij=60,iteration count=4	9.52	7.72
Dij=61,iteration count=3	10.11	7.85
Dij=62,iteration count=2	10.73	7.48
Dij=63,iteration count=1	11.37	7.83

Table 5.6 Overall performance comparison between two parallelization approaches

Methods\proc num	1	4	8	16
Approach 1(time)	30:03	8:50	5:20	3:33
Approach 2(time)	30:03	8:34	4:57	3:03
Approach 1 (speedup)	1.00	3.40	5.63	8.46
Approach 2 (speedup)	1.00	3.50	6.07	9.85

CHAPTER 6

SUMMARY

The RNA pseudoknotted structure prediction problem remains a challenge in bioinformatics. Interactions between nucleotide bases in an RNA sequence fold it up in space to form a secondary structure such as stem-loops and pseudoknots. Pseudoknots are functionally important in a number of RNAs, and have been found participating in translation, viral genome structure, and ribozyme active sites. It is thus of great interest to resolve the issues in RNA pseudoknot structure determination include pseudoknot prediction for single-RNA.

Ordinary RNA stem-loops can be viewed as nesting patterns of base-paired nucleotides while pseudoknotted structures require crossing patterns of base pairings. In contrast to stem-loops that can be modeled by SCFG, pseudoknots formally require Chomsky context-sensitive grammars to describe, which are much more complex than CFG, in addition to being clumsy to implement. Our work has been developed from a new stochastic grammar prediction model based on the *Parallel Communicating Grammar System* (PCGS). Technically, this model differs from conventional PCGS grammars in that it uses a complementary alignment of two base pairing regions instead of a set of auxiliary grammars (components) to implement crossing helices in a pseudoknot. The approach has the advantage that it allows a top-down grammatical description of RNA structures in the same manner as an SCFG does, and thus leads to an automated bottom-up dynamic programming algorithm for single-RNA pseudoknot structure prediction.

This thesis also aims at improving the performance of the automated pseudoknot prediction algorithm in terms of both memory usage and running time. The general problem of pseudoknot prediction is difficult. Even for restricted pseudoknot categories, most existing prediction algorithms require the worst case CPU time $O(N^6)$ and memory space $O(N^4)$. Such complexities for prediction algorithms appear to be inherent. It is unlikely that there will be algorithmic approaches that can drastically reduce the computational resource requirements by the problem. Our solution to the performance issue is to adopt implementation techniques. We have developed a *memory-mapping* technique to compress the main matrix used by the algorithm, resulting in saving around 97% memory space for the sequences tested. From both theoretical and practical respects, the memory mapping technique allows us to handle RNA sequences 2.46 times longer than can be done without the mapping technique.

To improve the slow running time of the algorithm caused by the inherently computation-intensive nature of the algorithm and by the overhead posed by the memory mapping technique, we have also developed techniques to parallelize the computation for the algorithm. The algorithm has been parallelized via the openMP technique. It runs on a shared memory computer system and can gain about 10 fold speedup using 16 processors. The parallelism has taken the advantage of the fact that the computing jobs on a diagonal of the matrix are independent of each other. The data are computed in the ascending order of index distance and jobs can be distributed across more than one processor. In our first *approach 1*, we employed many threads working on the entries with the same index distance simultaneously; all parallel computations are synchronized for each index

distance. For this approach, the smallest work unit assigned to each thread is an iteration corresponding to the sequential loop variable for the first index. Such a strategy may cause unnecessary waiting among parallel processors. To overcome unnecessary waiting, we have also introduced another parallelization strategy *approach 2* in which the partition of the computation for processors is at the level of non-terminals. *Approach 2* allows the parallel computation to achieve an extra 10% speedup over the 9-fold improvement on running time already made by *approach 1*.

Most RNA pseudoknot prediction algorithms, including ours, are either thermodynamic-based or stochastic grammar-based; both evolved from the well-known CYK algorithm. The core computation for these algorithm is to fill matrices for energy values or probabilities that resembles the computation in our algorithm. Therefore, the implementation techniques of memory mapping and parallelization that we have developed are general and applicable to other pseudoknot prediction algorithms as well.

REFERENCES

Akutsu, T. 2000. Dynamic programming algorithms for RNA secondary prediction with pseudoknots, *Discrete Applied Mathematics 104*: 45-62.

Brahams, J.P., van den Berg, M., van Batenburg, E., and Pleij, C. 1990. Prediction of RNA secondary structure, including pseudoknotting, by computer simulation. *Necleic Acids Research 18*: 3035-3044.

Brown, M. and Wilson, C., 1996, RNA pseudoknot modeling using intersections of stochastic context-free grammars with applications to database search. *Proceedings of Pacific Symposium on Biocomputing* 1996.

Cai, L., Malmberg, R., and Wu, Y. 2003. Stochastic modeling of RNA pseudoknotted structures: a grammatical approach, *ISMB 2003*.

Cocke, Younger and Kasami , CYK algorithm, internet URL:
http://www.wikipedia.org/wiki/CYK_algorithm

Deiman, B.A.L.M and Pleij, C.W.A.(1997) Pseudoknots: A vital feature in viral RNA. *Seminars Virol.*, 8, 166-175.

Chomsky, N., 1956, Three models for the description of languages. *IRE Transactions on Information Theory 2*: 113-124

Durbin, R., Eddy, S.R., Krogh, A. and Mitchison, G.J. 1998, *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press.

Eddy, S.R. and Durbin, R., 1994, RNA sequence analysis using covariance models. *Nucleic Acids Research 22* :2079-2088.

Eddy, S.R., 2002, A memory-efficient dynamic programming algorithm for optimal alignment of a sequence to an RNA secondary structure. *BMC Bioinformatics volume 3*

Eppstein, D., Gailil, Z., Giancarlo, R., and Italiano, G.F.1991. Efficient algorithms for sequence analysis. *Proceedings of 1991 International Advanced Workshop on Sequences, Italy.*

Felden, B., Massire, C., Westhof, E., Atkins, J.F., and Gesteland, R.F., 2001. Phylogenetic analysis of tmRNA genes within a bacterial subgroup reveals a specific structural signature. *Nucleic Acids Research 29*: 1602-1607.

Geist Al, parallel virtual machine :a users' guide and tutorial for networked parallel computing. *MIT Press*, 1994

Gropp William, Lusk Ewing, Skjellum Anthony, 1994. Using MPI : portable parallel programming with the message-passing interface, *MIT Press*

Konotos, H., Napthine, S., and Brierley, I., 2001 Ribosomal pausing at a frameshifter RNA pseudoknot is sensitive to reading phase but shows little correlation with frameshift efficiency. *Molecular And Cellular Biology 21*: 8657-8670

Lari, K., and Young, S.J. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and languages 5:* 237-257.

Lewis B., Berg J.D., Multithreaded programming with pthreads. 1998 .*Sun Microsystems Press*

Lyngso, R.B. and Pedersen, C.N.S.. 2000. RNA pseudoknot prediction in energy based models. *Journal of Computational Biology 7* : 409-428.

MFOLD software, internet URL:
http://www.biology.wustl.edu/gcg/mfold.html

Myers, E.W. and Miller, W.,1988, Optimal alignments in linear space. *Comput. Applic. Biosci.4*: 11-17

Nussinov, R. and Jacobson, A.B. 1980, Fast algorithm for predicting the secondary structure of single-stranded RNA, *Proc. of the Natl. Academy of Sci. of the USA 77*: 6309-6313.

OpenMP organization, the internet URL:
www.openmp.org

Paillart, J.C., Skripkin, E., Ehresmann, B., Ehresmann, C., and Marquet, R., (2002) In vitro evidence for a long range pseudoknot in the 5'-untranslated andmatrix coding regions of HIV-1 genomic RNA. *Journal of Biological Chemistry*, 277, 5995-6004.

Rivas, E. and Eddy, S.R. 1999, A dynamic programming algorithm for RNA structure prediction including pseudoknots. *Journal of Molecular Biology 285*: 2053-2068.

Sakakibara, Y., Brown, M., Hughey, R., Mian, I.S., Sjolander, K., Underwood, R.C., and Haussler, D. 1994. Stochastic context-free grammars for tRNA modeling. *Nucleic Acids Research 22* : 5112-5120.

Shapiro, B.A. and Wu, J.C. 1996. An annealing mutation operator in the genetics algorithms for RNA folding. *Computer Applications in the Biosciences 12*: 171-180.

Tanaka, Y., Hori, T., Tagaya, M., Sakamoto, T., Kurihara, Y., Katahira, M., and Uesugi, S., 2002 Imino proton NMR analysis of HDV ribozymes: nested double pseudoknot

structure and Mg2 + ion-binding site close to the catalytic core in solution. *Nucleic Acids Research 30*: 766-774

Tinoco, I., Borer, P.N., Dengler, B., Levine, M.D., Uhlenbeck, O.C., Crothers, D.M., and Fralla, J. 1973. Improved estimation of secondary structure in ribonucleic acids, *Nature New Biology 246*: 40-41.

Turner, D.H., Sugimoto, N.,and Freier, S.M.1998. RNA structure prediction. *Annual Review of Biophysics and Biophysical Chemistry 17*:167-192

Uemura, Y., Hasegawa, A., Kobayashi, Y., and Yokomori, T., Tree adjoining grammars for RNA structure prediction, *Theoretical Computer Science 210*: 277-303.

Viena software, the internet URL:
http://www.tbi.univie.ac.at/~ivo/RNA/

Waterman, M.S., Smith, T.F. and Katcher, H. 1976. Some biological sequence metrics. *Advances in Mathematics 20*: 367- 387

Zuker, M. and Stiegler, P. 1981. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic Acids Research 9*: 133-148.

Zweib, C., Wower, I., and Wower, J. 1999, Comparative sequence analysis of tmRNA, *Nucleic Acids Research 27*: 2063-2071.