`GTfold`: A Scalable Multicore Code for RNA Secondary Structure Prediction

Amrita Mathuriya
College of Computing
Georgia Institute of Technology

David A. Bader
College of Computing
Georgia Institute of Technology

Christine E. Heitsch
School of Mathematics
Georgia Institute of Technology

Stephen C. Harvey
School of Biology
Georgia Institute of Technology

August 23, 2008

Abstract

The prediction of the correct secondary structures of large RNAs is one of the unsolved challenges of computational molecular biology. Among the major obstacles is the fact that accurate calculations scale as $O(n^4)$, so the computational requirements become prohibitive as the length increases. Existing folding programs implement heuristics and approximations to overcome these limitations. We present a new parallel multicore and scalable program called GTfold, which is one to two orders of magnitude faster than the de facto standard programs and achieves comparable accuracy of prediction. Development of GTfold opens up a new path for the algorithmic improvements and application of an improved thermodynamic model to increase the prediction accuracy.

In this paper we analyze the algorithm's concurrency and describe the parallelism for a shared memory environment such as a symmetric multiprocessor or multicore chip. In a remarkable demonstration, GTfold now optimally folds 11 picornaviral RNA sequences ranging from 7100 to 8200 nucleotides in 8 minutes, compared with the two months it took in a previous study. We are seeing a paradigm shift to multicore chips and parallelism must be explicitly addressed to continue gaining performance with each new generation of systems. We also show that the exact algorithms like internal loop speedup can be implemented with our method in an affordable amount of time. GTfold is freely available as open source from our website.

1 Introduction

RNA molecules perform a variety of different biological functions including the role of "small" RNAs (with tens or a few hundred of nucleotides) in gene splicing, editing, and regulation.

At the other end of the size spectrum, the genomes of numerous viruses are lengthy single-stranded RNA sequences with many thousands of nucleotides. These single-stranded RNA sequences base pair to form molecular structures, and the secondary structure of viruses like dengue [3], ebola [16], and HIV [17] is known to have functional significance. Thus, disrupting functionally significant base pairings in RNA viral genomes is one potential method for treating or preventing the many RNA-related diseases.

Figure 1: The optimal secondary structure of an HIV-1 virus with 9,781 nucleotides predicted using GTfold in 84 seconds using 16 dual core CPUs. The minimum free energy of the structure is -2,879.20 Kcal/mole.

Viral sequences range in length from about 1,000 to over 1,000,000 nucleotides in the recently discovered virophage. Length of the viral sequences poses significant computational challenges for the current computer programs. Free energy minimization excluding pseudo-knots is a conventional approach for predicting secondary structures. The mfold [20, 11] and RNAfold [9] programs are the standard programs used by the molecular biology community for the last several decades. Recently, other folding programs such as simfold [1] have been developed. These programs predict structures with good accuracy for the RNA molecules

having fewer than 1,000 nucleotides. However, for longer RNA molecules, prediction accuracy is very low.

According to the thermodynamic hypothesis, the structure having the minimum free energy (MFE) is predicted as the secondary structure of the molecule. The free energy of a secondary structure is the independent sum of the free energies of distinct substructures called loops. The optimization is performed using the dynamic programming algorithm given by Zuker and Stiegler in 1981 [21] which is similar to the algorithm for sequence alignment but far more complex. The algorithm explores all the possibilities when computing the MFE structure. There are heuristics and approximations which have been applied to satisfy the computational requirements in the existing folding programs.

One potential approach to improve the accuracy of the predicted secondary structures is to implement advanced thermodynamic details and exact algorithms. However, while the incorporation of these improvements can significantly increase the accuracy of the prediction, it also drastically increases running time and storage needs for the execution. We use shared memory parallelism to overcome the computational challenges of the problem.

We have designed and implemented a new parallel and scalable program called GTfold for predicting secondary structures of RNA sequences. Our program runs one to two orders of magnitude faster than the current sequential programs for viral sequences on an IBM P5 570, 16 core dual CPU symmetric multiprocessor system and achieves comparable accuracy in the prediction. We have parallelized the dynamic programming algorithm at a coarse-grain and the individual functions which calculate the free energy of various loops at a fine-grain. We demonstrate that GTfold executes exact algorithms in an affordable amount of time for large RNA sequences. Our implementation includes an exact and optimized algorithm in place of the usually adopted heuristic option for internal loop calculations, the most significant part of the whole computation. GTfold takes just minutes (instead of 9 hours) to predict the structure of a *Homo sapiens* 23S ribosomal RNA sequence with 5,184 nucleotides. Development of GTfold opens up the path for applying essential improvements in the prediction programs to increase the accuracy of the predicted structures.

The algorithm has complicated data dependencies among various elements, including five different 2D arrays. The energy of the subsequences of equal length can be computed independently of each other without violating the dependencies pattern introduced by the dynamic programming with a set of five tables. Our approach calculates the optimal energy of the equal length sequences in parallel starting from the smallest to the largest subsequences and finally the optimal free energy of the full sequence. We also describe the nature of individual functions for calculating the energy of various loops and strategies for parallelization.

2 Related Work

Several approaches exist for RNA secondary structure prediction. While this paper focuses on free energy minimization, another approach [15] predicts secondary structures by computing the structures of smaller subsequences and using them to rebuild the full structure. Though

this latter approach runs fast, it is an approximation, and can miss the candidates that do not follow the usual behavior. Also, the success of these kinds of approaches is dependent upon the ability of rebuilding methods to identify motifs correctly by consistently combining the substructures into a full structure.

Several distributed memory implementations of RNA secondary structure prediction have been developed; however, they may not be portable to current parallel computers (e.g., [12, 5]) or do not store the tables necessary for finding suboptimal structures (e.g., see [9]). For instance, Hofacker *et al.* [9] partition the triangular portion of 2D arrays into equal sectors that are calculated by different processors in order to minimize the space requirements and data is reorganized after computing each diagonal. Traceback for the suboptimal secondary structures is not possible because it requires the filled tables. In [8], the authors observe that to fold the HIV virus, memory of 1 to 2GB is required, dictating the use distributed memory supercomputers; yet in our work, we demonstrate that this can now be solved efficiently on most personal computers. In our work, for the first time, we give scientists the ability to solve very large folding problems on their desktop by leveraging multicore computing.

Zhou and Lowenthal [19] also studied a parallel, out-of-core distributed memory algorithm for the RNA secondary structure prediction problem including pseudoknots. However, their approach does not implement the full structure prediction but rather studies a synthetic data transformation that improves just one of the dependencies found in the full dynamic programming algorithm.

3 RNA Secondary Structure

RNA molecules are made up of A, C, G, and U, nucleotides which can pair up according to the rules in {(A,U), (U,A), (G,C), (C,G), (G,U), (U,G)}. Nested base pairings result into 2D structures called secondary structures. There are 3D interactions among the elements of the secondary structures which result into 3D structures called tertiary structures. Pairings among bases form various kinds of loops, which can be classified based on the number of branches present in them. Nearest neighbor thermodynamic model (NNTM) provides a set of functions and sequence dependent parameters to calculate the energy of various kinds of loops. The free energy of a secondary structure is calculated by adding up the energy of all loops and stacking present in the structure.

Figure 2 shows an MFE secondary structure predicted by GTfold of a sequence with 79 nucleotides. Various loops annotated in the figure are named as hairpin loops, internal loops, multiloops, stacks, bulges and external loops. Loops formed by two consecutive base pairs are called stacks. Loops having one enclosed base pair and one closing base pair are called internal or interior loops. Internal loops with length of one side as zero are called bulges. Loops with two or more enclosed base pairs and one closing base pair are called multiloops or multibranched loops. The open loop which is not closed by any base pair is called an external or exterior loop.

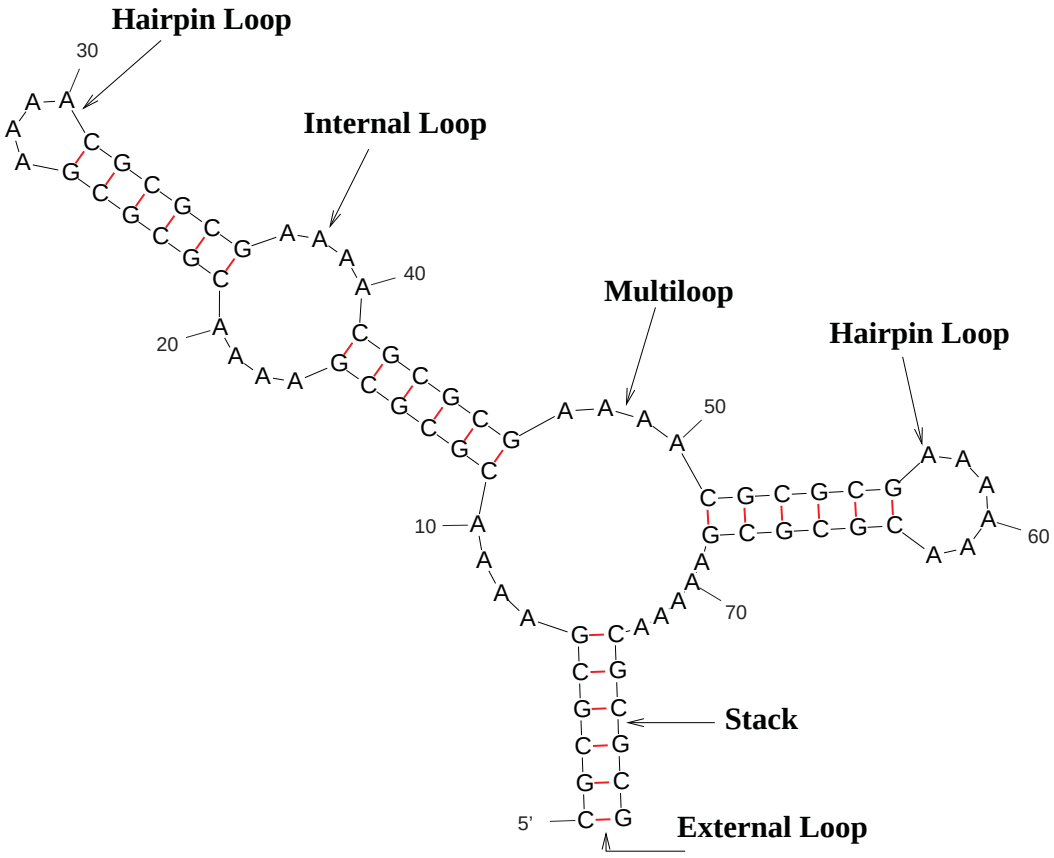

Figure 2: A sample RNA secondary structure with 79 nucleotides.

4 Thermodynamic Prediction Algorithm

Prediction of secondary structures with the free energy minimization is an optimization problem like the Smith-Waterman local alignment algorithm. There is a well-defined scoring function which can be optimized via dynamic programming, and structures achieving the optimum can be found through traceback. However, while sequence alignment can be performed with one table and a relatively simple processing order, RNA secondary structure prediction requires five tables with complex dependencies. Each class of loop has a different energy function which is dependent upon the sequence and parameters. For the internal loops and multiloops with one or more branches, all enclosed base pairs need to be searched which makes the loop optimal for the closing base pair.

The algorithm can be defined with recursive minimization formulas. Simplified recursion formulas are reproduced here from [10] for convenience. Pseudocode of our algorithm that implements thermodynamically equipped recursion formulas is presented in Appendix A. Consider an RNA sequence of length N, free energy $W(N)$, and index values i and j which vary over the sequence such that $1 \leq i < j \leq N$. The optimal free energy of a subsequence from 1 to j is given with the following formula:

$$W(j) = \min\{W(j-1), \min_{1 \leq i < j}\{V(i,j) + W(i-1)\}\} \tag{1}$$

In Eq. (1), $V(i,j)$ is the optimal energy of the subsequence from i to j, if it forms a base pair (i,j). It is defined by the following equation.

$$V(i,j) = \min \begin{cases} eH(i,j), \\ eS(i,j) + V(i+1, j-1), \\ VBI(i,j), \\ VM(i,j) \end{cases} \tag{2}$$

Eq. (2) considers loops that a base pair (i,j) can close. The $eH(i,j)$ function returns the energy of a hairpin loop closed by base pair (i,j). Function $eS(i,j)$ returns the energy of a stack formed by base pairs (i,j) and $(i+1, j-1)$. $VBI(i,j)$ and $VM(i,j)$ are the optimal free energies of the subsequence from i to j in the case when the (i,j) base pair closes an internal loop or a multiloop, respectively.

$$VBI(i,j) = \min_{i < i' < j' < j}\{eL(i,j,i',j') + V(i',j')\} \tag{3}$$

where, $i' - i + j - j' - 2 > 0$.

The formulation of the multiloop energy function has linear dependence upon the number of single stranded bases present in the multiloop. The standard is to introduce a 2D array WM to facilitate the calculation of VM array. Eq. (4) and (5) shows calculations of $WM(i,j)$ and $VM(i,j)$ respectively.

$$WM(i,j) = \min \begin{cases} V(i,j) + b, \\ WM(i, j-1) + c, \\ WM(i+1, j) + c, \\ min_{i < k \leq j}\{WM(i, k-1) + WM(k, j)\} \end{cases} \tag{4}$$

$$VM(i,j) = \min_{i+1<h\leq j-1}\{WM(i+1,h-1) + WM(h,j-1) + a\} \quad (5)$$

These minimization formulas can be implemented recursively as well as iteratively. We implement an iterative formulation of the algorithm in GTfold, described later in this paper. The implementation uses various 1D and 2D arrays corresponding to $W(j)$ and $V(i,j)$, $VBI(i,j)$, $VM(i,j)$, $WM(i,j)$ values. Also we use $calcW(j)$, $calcV(i,j)$, $calcVBI(i,j)$, $calcVM(i,j)$ and $calcWM(i,j)$ functions to calculate the values of $W(j)$, $V(i,j)$, $VBI(i,j)$, $VM(i,j)$ and $WM(i,j)$ array elements.

4.1 Parallelism

The dynamic programming algorithm is computationally intensive both in terms of running time and storage. Its space requirements are of $O(n^2)$ as it uses four 2D arrays named $V(i,j)$, $VBI(i,j)$, $VM(i,j)$ and $WM(i,j)$ that are filled up during the algorithm's execution. The main issue is running time rather than memory requirements. For instance, GTfold has a memory footprints of less than 2GB (common in most desktop PCs) even for sequences with 10,000 nucleotides.

The filled up arrays are traced in the backwards direction to determine the secondary structures. The traceback for a single structure takes far less time than filling up these arrays. Time complexity of the dynamic programming algorithm is $O(n^3)$ with the currently adopted thermodynamic model. The two indices i and j are varied over the entire sequence, and every type of loop for every possible base pair (i,j) is calculated. This results in the asymptotic time complexity of $O(n^2)\times$ maximum time complexity of any type of loop for a base pair (i,j).

Computations of internal loops and multiloops are the most expensive parts of the algorithm. We can see from Eq. (3) that, in the calculation of $VBI(i,j)$, all possible internal loops with the closing base pair (i,j) are considered by varying indices i' and j' over the subsequence from $i+1$ to $j-1$ such that $i' < j'$. This results in the overall time complexity of $O(n^4)$. To avoid large running time, a commonly used heuristic is to limit the size of internal loops to a threshold k usually set as 30. This significantly reduces running time from $O(n^4)$ to $O(k^2n^2)$. The heuristic is adopted in most of the standard RNA folding programs.

Lyngsø *et al.* [10] suggest that the limit is a little bit small for predictions at higher temperatures and give an optimized and exact algorithm for internal loop calculations which has the time complexity of $O(n^3)$ with the same $O(n^2)$ space. The algorithm searches for all possible internal loops closed by base pair (i,j). Practically, this algorithm is far slower than the heuristic. Choosing one of the options is a tradeoff of running time versus accuracy. In GTfold we provide an option for the user to select the heuristic or internal loop speedup algorithm. Also, our parallelization scheme is valid for both the options.

Thermodynamics of multiloops are still not understood fully, but improvements continue to be made. Searching for an optimal multiloop closed by a base pair (i,j) requires searching for all enclosed base pairs which makes the loop optimal. To make the multiloop energy function feasible to compute, it may be approximated in $O(n^3)$ time. This function has linear

dependence upon the number of single stranded bases in the multiloop. Time complexity of the algorithm to implement a relatively more realistic multiloop energy function having logarithmic dependence upon the single stranded bases in the loop is exponential. Also, many other advanced thermodynamic details such as coaxial dangling energies are not implemented in the multiloop energy calculations during the optimization, as it significantly increases the running time.

Both running time and spaces needs are expected to increase with the use of better thermodynamic models. While memory requirements can be satisfied with today's high-end servers with 256GB or more memory, running time will continue to play as a major prohibitive factor in solving these grand challenge problems. Our parallelization approach in GTfold is designed to keep all these factors in consideration.

5 GTfold

5.1 Dependencies and Access Patterns

Figure 3 shows a general ij plane. A valid base pair is defined as (i, j) where $j > i$. Thus, only the upper right triangle is valid. Secondary structures can have only nested base pairings, meaning if there are two base pairs (i, j) and (i', j') such that $i < i' < j$ then the constraint $i < i' < j' < j$ is also satisfied. This assumption of nested base pairings results in the general dependency of an element (i, j) on the elements in the Δ ABC as shown in Figure 3. To find the optimal loop formed by a base pair (i, j), we need to search for all enclosed base pairs over the subsequence from $i + 1$ to $j - 1$. In the case of internal loops we need to search for one enclosed base pair while for a multiloop we need to search for more than one base pair. In this fashion, the computation of all types of loops for an element (i, j) follows the above dependency pattern.

The speedup algorithm for internal loop calculations follows the same general technique but its access pattern differs. It updates the elements outside the dependency triangle Δ ABC shown in Figure 3 for the element A. It is an optimized algorithm to reduce the space complexity. We define a *small* internal loop as a loop in which at least one of the sides is less than a constant c. The algorithm scores these small internal loops as special cases by applying a function derived from the general internal loop energy function. If an enclosed base pair (i_p, j_p) is better than another candidate of the enclosed base pair (i_{p1}, j_{p1}) for the closing base pair (i, j), then the enclosed base pair (i_p, j_p) will also be better than (i_{p1}, j_{p1}) for the closing base pairs of the form $(i - b, j + b)$, where b is a positive integer such that $1 \leq b \leq min\{i - 1, N - j\}$. This algorithm uses the best enclosed base pair (i_p, j_p) for the closing base pair (i, j) to evaluate all internal loops closed by the base pairs of the form $(i-b, j+b)$ at the same time. This way, at element (i, j), the elements of the form $(i-b, j+b)$ are also accessed. The access pattern of this algorithm is shown in Figure 4 excluding the calculation of special cases.

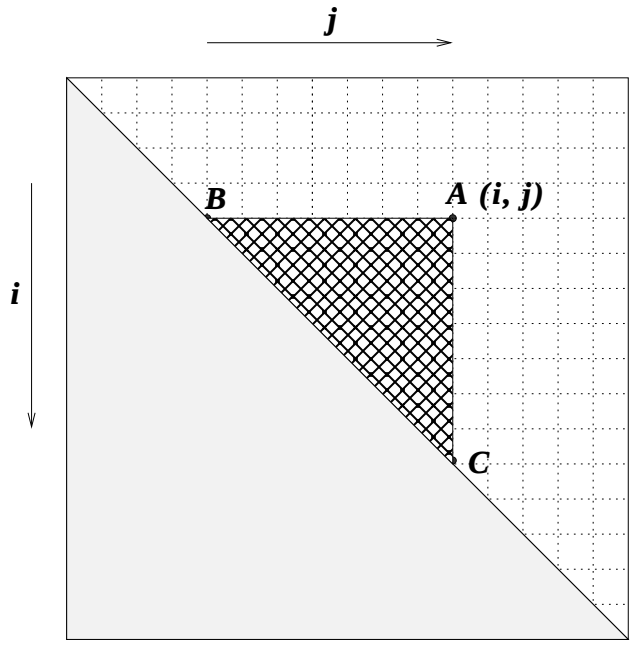

Figure 3: The implicit dependency of point A on the elements present in the triangle ABC

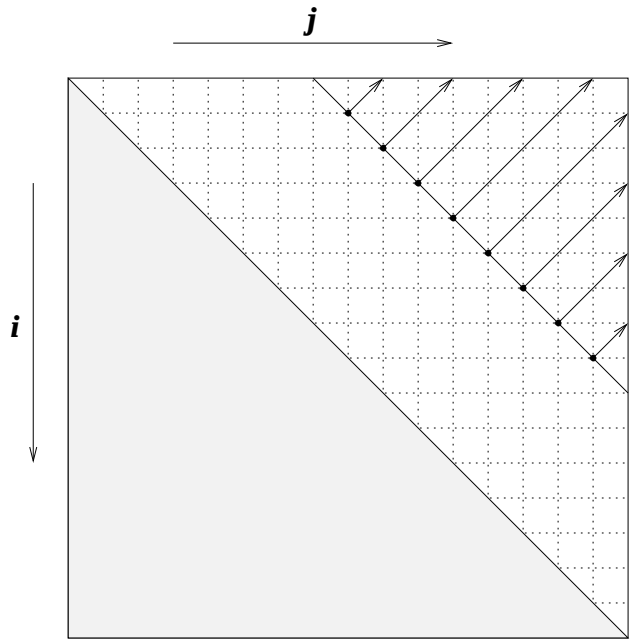

Figure 4: The access pattern of $VBI(i, j)$ for the internal loop speedup algorithm

5.2 Approach

In the region of the general 2D ij plane corresponding to $j > i$, a point (i, j) corresponds to the computation of energy of the subsequence from i to j. The dependency pattern shown in Figure 3 allows the calculation of all the elements existing on a line $j - i = k$ to be independent of each other, where k is any integer from the set $\{0, 1, 2, \ldots, N-1\}$. This way the computation of the line $j - i = k$ can be performed in parallel, and the whole space can be computed by considering subsequent lines from $k = 0$ to $k = N - 1$. Note that the points on one of the lines corresponds to the equal length subsequences.

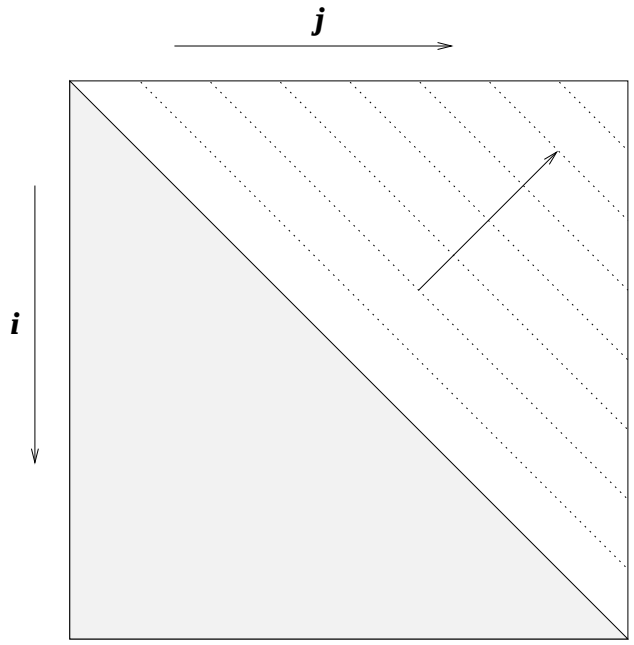

Figure 5: Showing the pattern of computation implemented in GTfold

Algorithm 1 arranges the nested *for* loops to compute in the manner described above. The first *for* loop runs for different lines starting from $j = i$ to $j = i + N - 1$ and the second *for* loop calculates all the points on one line in parallel. Figure 5 shows the sequence of these computations. This parallelization strategy is suitable for future improvements to the thermodynamic model or to optimizations for computing the various energy functions. This coarser level of parallelism enables us to exploit more concurrency while offering compatibility for possible future improvements.

There are other orderings of the computation that cover the whole space without violating the dependency pattern. One way is to compute the elements column-wise, starting from $j = 1$ to $j = N$. On one column the computation is done for the increasing values of $j - i$, i.e. from row i to row 1. A second way is to compute the elements row-wise, starting from $i = N$ to $i = 1$. On one row the computation is done for the increasing values of $j - i$, i.e. from column $j = i$ to column $j = N$. These two ways achieve a higher degree of spatial locality but they are inherently sequential.

```
input  : Sequence of Length N
output: Optimal Energy of the sequence
begin
    for b ← 0 to N do
        #pragma omp parallel for schedule (guided)
        for i ← 1 to N − b do
            j ← i + b;
            calcVBI(i, j);
            calcVM(i, j);
            calcV(i, j);
            calcWM(i, j);
        end
        calcW(b + 1);
    end
    return W(N);
end
```

Algorithm 1: Main function to compute the secondary structure of an RNA sequence

Parallelism at individual functions

Parallelism can also be exploited at the finer level of individual functions which compute the energies for the various kinds of loops for a closing base pair (i, j). The general pattern of different functions for calculating the energy of these kinds of loops is the same except for the function that computes internal loop energies using the speedup algorithm. The general pattern is to consider various possible options of the corresponding type of loop and select the option that gives the minimum energy. In simplified terms, this pattern of calculation performs minimization over several possible values. These types of calculations are easily done in parallel by assigning equal-sized chunks of minimization work to all threads, collecting the results, and taking the global minimum over all values.

The speedup algorithm for internal loop calculations can be parallelized only for special cases. The computations of general internal loops using the extension principle are inherently sequential. However the generally adopted heuristic option for internal loops has the general minimization pattern, is easily parallelizable, and uses two nested *for* loops which results in a complexity of $O(k^2)$ for a particular (i, j). Multiloop calculations also follow the general minimization pattern for the VM and WM arrays, have $O(n)$ time complexity for an element (i, j), and are amenable to parallelization as well.

5.3 Implementation Details

We use OpenMP [13] to implement shared memory parallelism. All the subsequent diagonals are considered with the upper *for* loop and parallelism is implemented by applying

an OpenMP *for* loop pragma over the inner *for* loop to parallelize the computation on the diagonal in consideration as shown in Algorithm 1. The guided scheduling strategy works best for this parallelization. This is because there may not be equal amounts of work for every point on the diagonal. If the bases i and j are not able to make a pair then it is not necessary to compute for the whole calculation. In this case, for the heuristic option of internal loop calculations, only $WM(i, j)$ needs to be calculated. And for the speedup algorithm $VBI(i, j)$ also needs to be calculated with $WM(i, j)$.

We explore the function level parallelism for the last few diagonals by deciding a threshold variable A, such that the parallelism is implemented at the higher level for the diagonals up to $j - i = A$ and implement only the function level parallelism starting from the diagonal $j - i = A + 1$ to $j - i = N - 1$. This facilitates the use of more threads to exploit more parallelism at the time when there are not enough points on the diagonals. However this technique did not give us a performance advantage.

Cache locality

For this algorithm the ratio of computation to the memory accesses is low. Energy of a secondary structure is calculated by adding up the energies of various loops present in the structure. Energy of a sequence is the sum of various energy terms of which some are read directly from the energy tables and others are calculated by the program. Therefore, large cache sizes and locality in reference for accessing various data elements play an important role in reducing the running time of GTfold. Computing the elements row-wise or column-wise as described in Section 5.2 provides better cache locality than computing the elements on the subsequent diagonals. However, these two ways are inherently sequential.

6 Experimental Results

We have performed several experiments to establish that GTfold runs faster than competing folding programs such as mfold and RNAfold and achieves accuracy comparable with them. For the running time and accuracy comparisons, we are using RNAfold distributed with Vienna RNA Package version 1.7.2 and UNAFold version 3.6, which supersedes mfold.

6.1 Energy and Structure Comparison

To establish the accuracy of GTfold, we compare the structures obtained from GTfold, mfold, and RNAfold, with the correct structures determined with the more reliable method of comparative sequence analysis [6, 7]. Please note that the comparative sequence analysis method requires large data sets for the prediction of secondary structures, and therefore, its application is limited with the availability of the required datasets.

Doshi *et al.* [4] take a phylogenetically diverse dataset of ribosomal RNA sequences and compare the optimal secondary structures predicted using mfold 2.3 and mfold 3.1 with the correct structures. Here we are using the 16S and 23S ribosomal RNA sequences from

Figure 1 and Table 4 of their study [4] for accuracy comparisons and are taken from the Gutell database [2]. We provide a GenBank accession number for each of these sequences. For predicting structures with UNAFold and RNAfold their command line default options are used. Accuracy of the structures is calculated in the same manner as in [4] with one difference. We include non-canonical base pairs from the comparison instead of excluding them. This affects the accuracy of all three programs in the same manner because none of them is able to predict non-canonical base pairs due to the lack of energy parameters for them. The accuracy is calculated as a percentage of correctly predicted base pairs which are also present in the correct secondary structure.

Table 1: Free energy (in Kcal/mole) comparison of GTfold, UNAFold and RNAfold for 16S rRNA sequences

Sequence	Length	GTfold	UNAFold	RNAfold
X00794	1962	-741.90	-722.70	-746.60
X54253	701	-149.00	-141.30	-149.03
X54252	697	-142.50	-137.50	-142.52
Z17224	1550	-564.80	-549.10	-565.12
X65063	1432	-582.00	-570.80	-581.94
X52949	1452	-802.70	-794.50	-804.40
X98467	1295	-487.00	-460.00	-489.31
Y00266/M24612	1244	-325.60	-317.30	-328.80
K00421	1474	-687.00	-682.10	-687.01

Table 2: Accuracy comparison (in percent) of GTfold, UNAFold and RNAfold for 16S rRNA sequences of Table 1

Sequence	GTfold	UNAFold	RNAfold
X00794	30.33	31.65	27.91
X54253	25.67	20.32	25.13
X54252	21.16	21.64	21.16
Z17224	26.03	24.57	24.57
X65063	24.09	22.02	23.83
X52949	15.07	16.08	15.07
X98467	15.56	10.97	16.33
Y00266/M24612	19.19	17.30	18.11
K00421	76.42	75.76	76.42

Table 1 shows the optimal free energy of various 16S ribosomal RNA sequences predicted with GTfold, UNAFold and RNAfold. Table 2 shows the accuracy comparison for the three programs for the sequences of Table 1. Similarly Table 3 shows the optimal free energy obtained using the programs for 23S ribosomal RNA sequences, and Table 4 shows the accuracy comparison for the sequences of Table 3.

Table 3: Free energy (in Kcal/mole) comparison of GTfold, UNAFold and RNAfold for 23S rRNA sequences

Sequence	Length	GTfold	UNAFold	RNAfold
X14386	3105	-791.30	-775.10	-792.65
X54252	953	-180.20	-173.50	-179.71
X52392	1621	-395.80	-389.70	-397.85
J01527	3273	-700.60	-684.60	-702.87
K01868	3514	-1328.50	-1294.30	-1333.95
X53361	4052	-1692.90	-1665.60	-1696.49
X52949	2850	-1707.90	-1689.20	-1709.80
M67497	3029	-1661.10	-1647.10	-1668.12

Table 4: Accuracy comparison (in percent) of GTfold, UNAFold and RNAfold for 23S rRNA sequences of Table 3

Sequence	GTfold	UNAFold	RNAfold
X14386	21.77	18.79	18.32
X54252	23.74	21.92	23.29
X52392	24.44	25.56	24.16
J01527	24.72	30.11	25.14
K01868	20.67	22.01	17.85
X53361	21.54	16.59	15.44
X52949	34.44	31.24	26.69
M67497	64.05	63.6	63.94

The small differences in the optimal free energy scores predicted using the three programs are due to the algorithmic issues and policies related to thermodynamic aspects. Energy comparisons shown in Tables 1 and 4 demonstrate that our energy scores for the various sequences lie in the very small range of these standard programs. Accuracy percentages shown in Tables 2 and 3 establish that GTfold achieves accuracy comparable with UNAFold and RNAfold for the diverse dataset chosen. Accuracy comparisons for various ribosomal sequences show that in general the accuracy of the prediction programs are very low. The prediction accuracy is expected to improve with the inclusion of advanced thermodynamic details that are not presently incorporated due to high computational cost. The development of GTfold facilitates the implementation of these improvements.

6.2 Running Time Comparison

GTfold implements parallelism for shared memory multiprocessor and multicore systems. Running time experiments are performed on an IBM P5-570 server with 16 dual core 1.9 GHz CPUs and 256 GB of main memory with L2 cache of 1.9 MB per CPU. GTfold is compiled with IBM xlC compiler Enterprise Edition 7.0, with -q64 option for the 64 bit com-

pilation, -O3 level of optimization and -qsmp=omp option for OpenMP support. RNAfold and UNAFold are compiled with their default compiler and compilation options. An additional flag -maix64 is set while compiling UNAFold and RNAfold due to the runtime memory limitations on the system for 32-bit compilations.

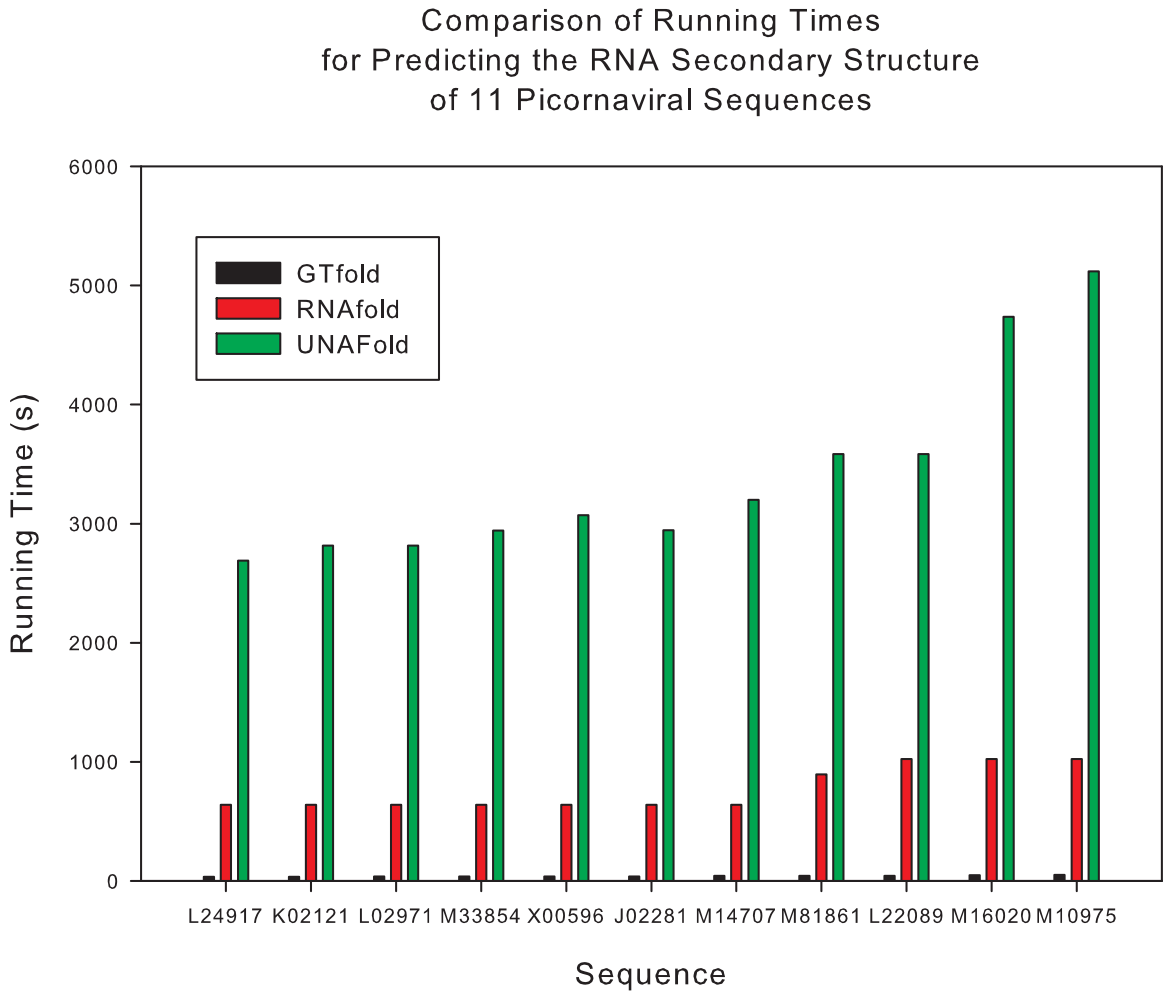

Figure 6: Comparison of running times for predicting the RNA secondary structures of 11 picornaviral sequences. The sequences are arranged in increasing order of length from 7124 to 8214 nucleotides.

Palmenberg and Sgro in 1997 [14] investigated the optimal and suboptimal secondary structures of 11 picornaviral RNA sequences using mfold version 2.2. The length of the sequences varies from 7124 to 8214 nucleotides. They report that each sequence required 5-7 days of CPU time using a modern workstation so that all 11 sequences took 2 to 3 months of time. In stark comparison, GTfold finishes the execution of this set of sequences in approximately 8 minutes using 32 threads.

In Figure 6 we compare the running time of GTfold with 32 threads, UNAFold, and

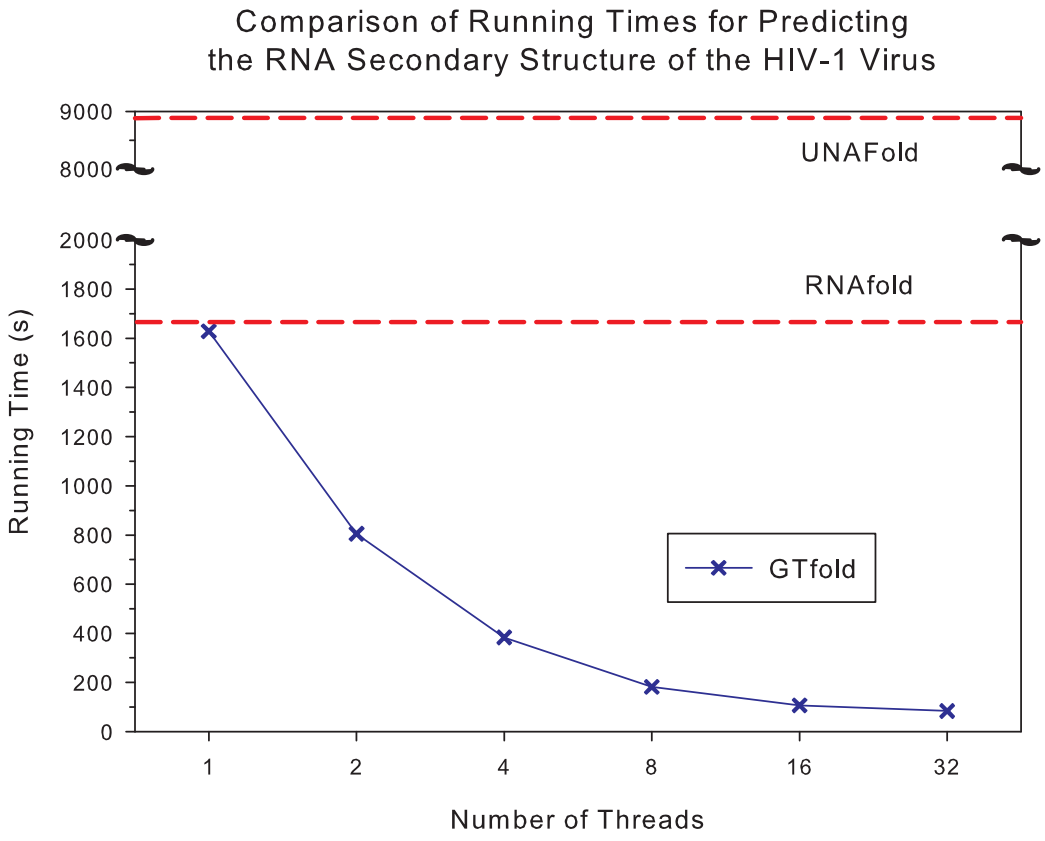

Figure 7: Comparison of running times for predicting the RNA secondary structure of the HIV-1 virus. The dashed horizontal lines represent the sequential running time of UNAFold and RNAfold.

RNAfold, for all the picornaviral sequences and using the same machine. We can see that GTfold runs one to two orders of magnitude faster than the standard the sequential programs UNAFold and RNAfold.

Figure 7 compares the running time of GTfold, UNAFold, and RNAfold, for an HIV-1 sequence (accession number Z11530) with 9,781 nucleotides. The secondary structure predicted with GTfold of the viral sequence is shown in Figure 1. All three programs implement the heuristic option and limit the internal loop size to 30. It is clear from the graph that GTfold with one thread performs much better than UNAfold and is comparable with RNAfold. The running time of GTfold decreases with the increasing number of processors. Even with two threads GTfold runs 2.06 times faster than RNAfold. GTfold folds the entire HIV viral sequence in 84 seconds with 32 threads in comparison to UNAFold and RNAfold which take approximately 2.4 hours and 27 minutes, respectively.

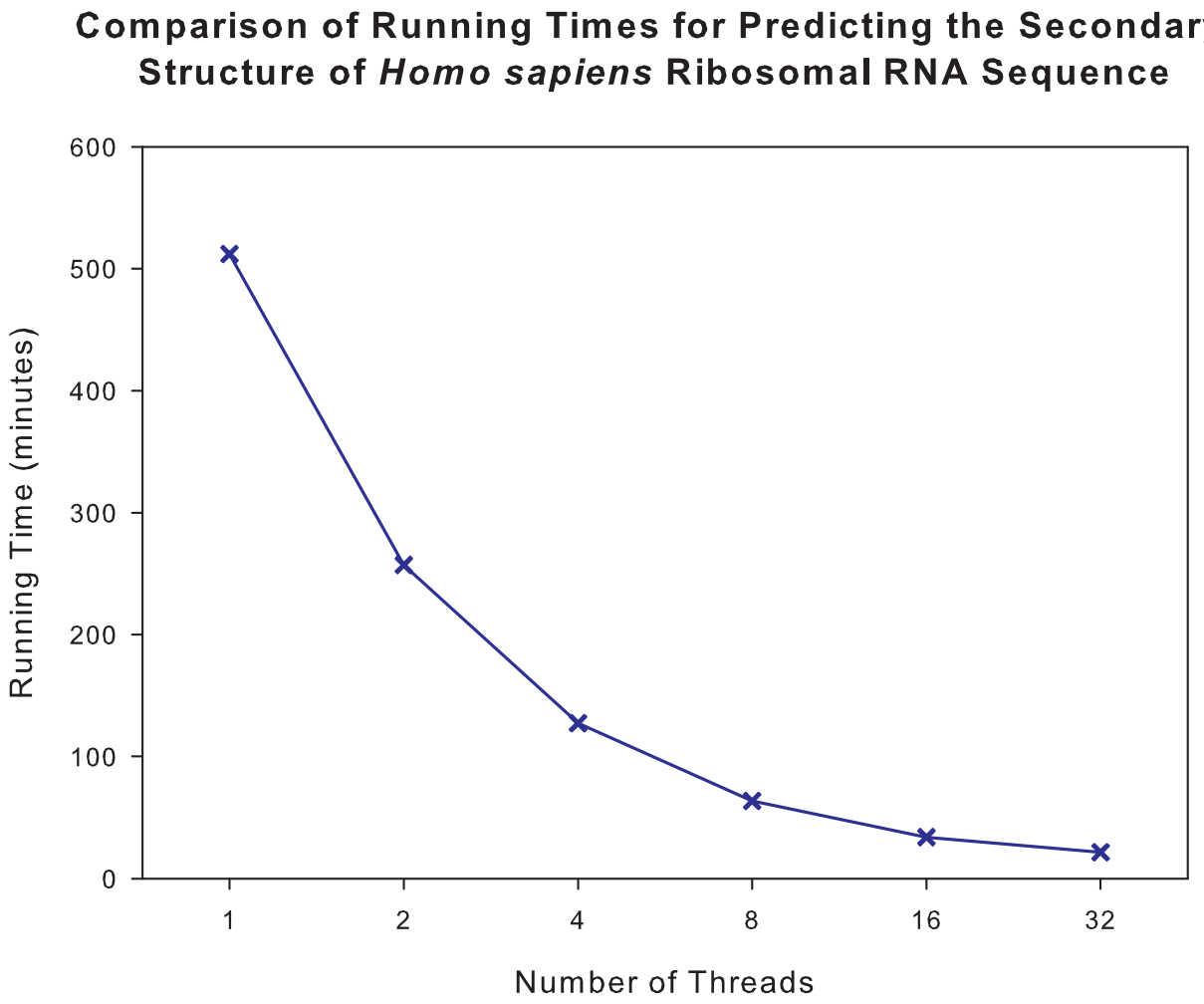

Figure 8: GTfold running time statistics for a *Homo sapiens* 23S ribosomal RNA sequence with accession number J01866/M11167 using the Internal Loop Speedup Algorithm

GTfold also implements an exact algorithm for finding the optimal internal loops called Internal Loop Speedup Algorithm (ILSA). Though internal loops with sizes longer than 30 are observed, they are usually rare. ILSA can catch these exceptional cases occurring with rarity in nature. It is far more expensive to run this algorithm than the commonly used heuristic. Figure 8 shows the running time of GTfold with the varying number of processors for a 5,184 length 23S Ribosomal RNA sequence of *Homo sapiens* with accession number J01866/M11167. GTfold is able to reduce the running time from 512 minutes (approximately 9 hours) to 21.5 minutes by using 32 threads. This way, we show that optimized algorithms such as internal loop speedup algorithm can be executed with GTfold in an affordable time. Please note that UNAFold and RNAfold do not implement the ILSA algorithm and can miss the rare possibilities.

Figure 9 shows the speedup achieved using 2, 4, 8, 16, and 32, threads for GTfold using the internal loop speedup algorithm (ILSA) option for the sequence with accession number J01866/M11167 and the heuristic options with an HIV viral sequence. The maximum speedup achieved in the first and second cases is approximately 23.8 and 19.8, respectively. We have achieved slightly superlinear speedups for 2, 4, and 8, threads in the case of the heuristic option due to the better cache locality when the number of processors is more than one.

7 Conclusions

We have developed GTfold, a parallel and multicore code for predicting RNA secondary structures that achieves 19.8 fold speedups over the current best sequential program for large, important RNA sequences and has accuracy comparable to the existing standard folding programs. We have recognized the computational requirements of the problem and implemented shared memory parallelism for the problem. Development of GTfold solves the problem of prohibitive running time factor for large RNA sequences and will help the molecular biology community to work towards the problem of predicting more accurate secondary structures. We have reduced the running time from approximately 2 months to 8 minutes for folding 11 picornaviral sequences.

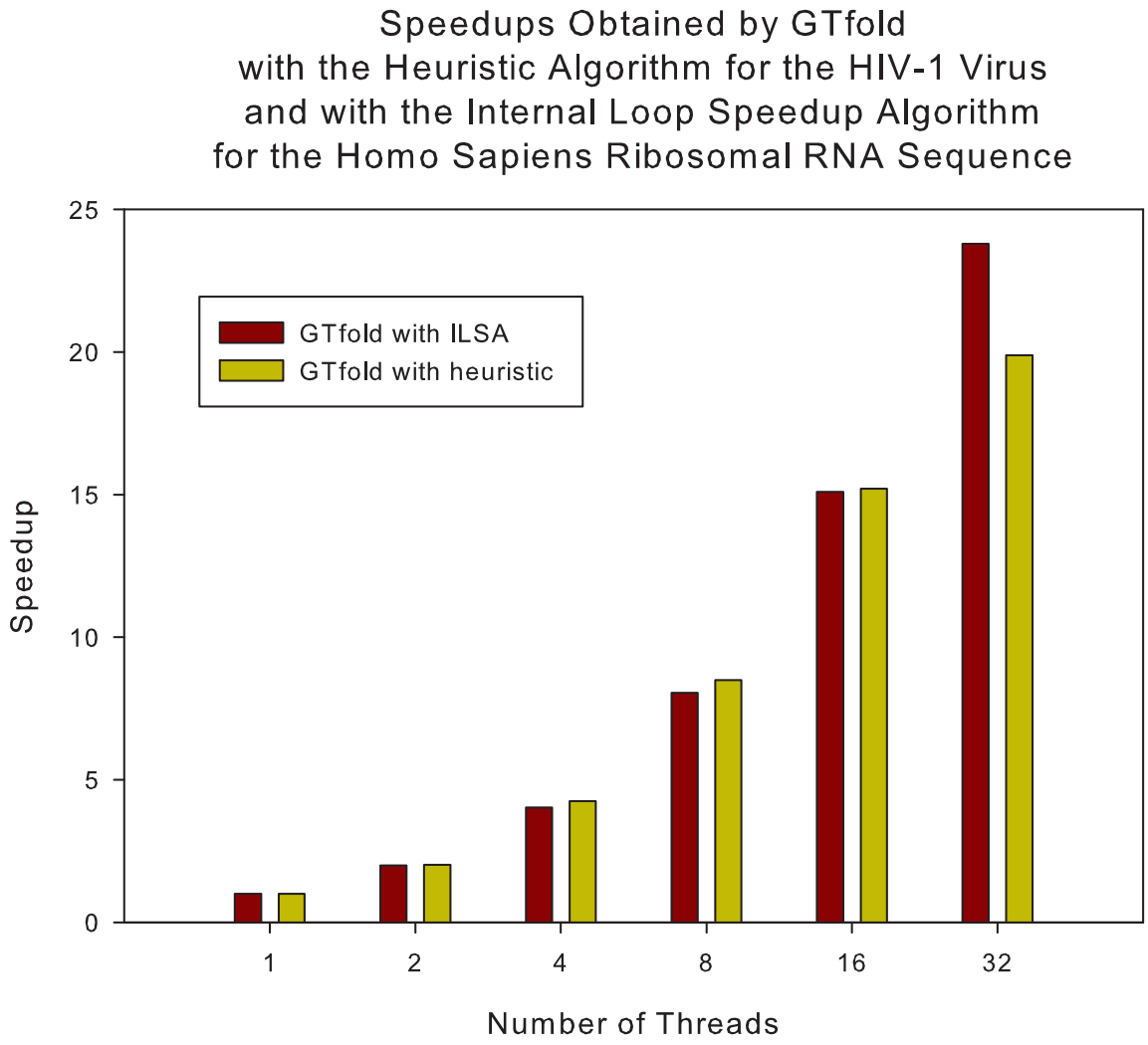

Figure 9: Speedups obtained by GTfold with the heuristic algorithm for the HIV-1 virus and with the internal loop speedup algorithm for the *Homo sapiens* ribosomal RNA sequence. Speedup is with respect to GTfold running on one processor for each series.

8 Acknowledgments

This work was supported in part by National Institutes of Health grant NIH NIGMS R01 GM083621, and by NSF Grants CNS-0614915 and DBI-04-20513. Additionally, the research of Christine E. Heitsch, Ph.D., is supported in part by a Career Award at the Scientific Interface (CASI) from the Burroughs Wellcome Fund (BWF). We acknowledge visiting students Gregory Nou, Sonny Hernández, and Manoj Soni, who helped with the implementation of GTfold.

References

[1] M. Andronescu, R. Aguirre-Hernandez, A. Condon, and H.H. Hoos. RNAsoft: a suite of RNA secondary structure prediction and design software tools. *Nucleic Acids Research*, 31(13):3416–3422, 2003.

[2] J.J. Cannone, S. Subramanian, M.N. Schnare, J.R. Collett, L.M. D'Souza, Y. Du, B. Feng, N. Lin, L.V. Madabusi, K.M. Müller, N. Pande, Z. Shang, N. Yu, and R.R. Gutell. The Comparative RNA Web (CRW) Site: an online database of comparative sequence and structure information for ribosomal, intron, and other RNAs. *BMC Bioinformatics*, 3(1), 2002.

[3] K. Clyde and E. Harris. RNA secondary structure in the coding region of dengue virus type 2 directs translation start codon selection and is required for viral replication. *J. Virol.*, 80(5):2170–2082, 2006.

[4] K.J. Doshi, J.J. Cannone, C.W. Cobaugh, and R.R. Gutell. Evaluation of the suitability of free-energy minimization using nearest-neighbor energy parameters for RNA secondary structure prediction. *BMC Bioinformatics*, 2004.

[5] M. Fekete, I.L. Hofacker, and P.F. Stadler. Prediction of RNA base pairing probabilities on massively parallel computers. *J. Computational Biology*, 7(1-2):171–182, 2000.

[6] P.P. Gardner and R. Giegerich. A comprehensive comparison of comparative RNA structure prediction approaches. *BMC Bioinformatics*, 5(140), 2004.

[7] R.R. Gutell, J.C. Lee, and J.J. Cannone. The accuracy of ribosomal RNA comparative structure models. *Curr. Opin. Struct. Biol.*, 12(3):301–310, 2002.

[8] I.L. Hofacker, M.A. Huynen, P.F. Stadler, and P.E. Stolorz. Knowledge discovery in RNA sequence families of HIV using scalable computers. In *Proc. of the 2nd Int'l Conf. on Knowledge Discovery and Data Mining*, Portland, OR, 1996.

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

[10] R.B. Lyngsø, M. Zuker, and C.N.S. Pedersen. Internal loops in RNA secondary structure prediction. In *Proc. of the 3rd Ann. Int'l Conf. on Computational Molecular Biology (RECOMB)*, pages 260–267, 1999.

[11] D.H. Mathews, J. Sabina, M. Zuker, and D.H. Turner. Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. *J. Mol. Biol.*, 288:911–940, 1999.

[12] A. Nakaya, K. Yamamoto, and A. Yonezawa. RNA secondary structure prediction using highly parallel computers. *Bioinformatics*, 11(6):685–692, 1995.

[13] OpenMP Architecture Review Board. *OpenMP Application Program Interface*, 3.0 edition, May 2008.

[14] A.C. Palmenberg and J.-Y. Sgro. Topological organization of picornaviral genomes: Statistical prediction of RNA structural signals. *Sem. Virol.*, 8:231–241, 1997.

[15] M. Taufer, T. Solorio, A. Licon, D. Mireles, and M.-Y. Leung. On the effectiveness of rebuilding RNA secondary structures from sequence chunks. In *7th IEEE Int'l Workshop on High Performance Computational Biology (HiCOMB)*, pages 1–8, 2008.

[16] M. Weik, J. Modrof, H-D Klenk, S. Becker, and E. Mühlberger. Ebola virus VP30-mediated transcription is regulated by RNA secondary structure formation. *J. Virol.*, 76(17):8532–8539, 2002.

[17] E.M. Westerhout, M. Ooms, M. Vink, A.T. Das, and B. Berkhout. HIV-1 can escape from RNA interference by evolving an alternative structure in its RNA genome. *Nucleic Acids Research*, 33(2):796–804, 2005.

[18] S. Wuchty, W. Fontana, I.L. Hofacker, and P. Schuster. Complete suboptimal folding of RNA and the stability of secondary structures. *Biopolymers*, 49(2):145–165, 1999.

[19] W. Zhou and D.K. Lowenthal. A parallel, out-of-core algorithm for RNA secondary structure prediction. In *Int'l Conf. on Parallel Processing (ICPP)*, pages 74–81, 2006.

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

[21] M. Zuker and P. Stiegler. Optimal computer folding of large RNA sequences using thermodynamic and auxiliary information. *Nucleic Acids Research*, 9(1):133–148, 1981.

A Pseudocode

The algorithm of predicting the MFE structures can be divided into two steps. The first part is to find the optimal free energy score that can be achieved by a possible secondary structure of the sequence. The second step is called traceback in which we trace in backward direction to find the structure that corresponds to the MFE score. Note that there may be one or more secondary structures corresponding to this score based upon the sequence contents. In this section we provide the pseudocode of the algorithm for the fill step. The algorithm for optimal and complete suboptimal traceback has been described by Wuchty et al. [18]. The fill step algorithm is a dynamic programming algorithm for predicting nested RNA secondary structures given by Zuker and Stiegler in [21].

Minimization formulas which define the algorithm, are recursive in nature; therefore the order of computation of various functions for a particular element (i, j) is important. For example $WM(i, j)$ uses the value of $V(i, j)$, so $calcWM(i, j)$ should be called after $calcV(i, j)$. Similarly, $calcVBI(i, j)$ and $calcVM(i, j)$ should be called before $calcV(i, j)$. $calculate(N)$ function call calculates the MFE score for the whole sequence, where N is the sequence length. Computation of other functions $calcVBI(i, j), calcWM(i, j), calcVM(i, j), calcV(i, j)$ and $calcW(j)$ is performed as shown in algorithms 3, 4, 5, 6 and 7 respectively. Also pseudocode for internal loop calculations using the heuristic option is given algorithm 8 and using the speedup algorithm is given in algorithms 9, 10 and 11.

```
input  : Sequence of Length N
output: Optimal Energy of the sequence
begin
    readEnergyTables();
    readSequence();
    for j ← 1 to N do
        for i ← 1 to N do
            VBI(i, j) ← INF;
            VM(i, j) ← INF;
            V(i, j) ← INF;
            WM(i, j) ← INF;
        end
        W(j) = INF;
    end
    for b ← 4 to N do
        for i ← 1 to N − b do
            j ← i + b;
            calcVBI(i, j);
            calcVM(i, j);
            calcV(i, j);
            calcWM(i, j);
        end
        calcW(b + 1);
    end
    return W(N);
end
```

Algorithm 2: Calculate(int N)

```
input  : Base indices i and j
output: VBI(i, j)
begin
    for i_p ← i + 1 to j − 2 do
        for j_p ← i_p + 1 to j − 1  &  j_p > i_p do
            VBI(i, j) ← MIN( VBI(i, j), eL(i, j, i_p, j_p) + V(i_p, j_p));
        end
    end
    return VBI(i, j);
end
```

Algorithm 3: Naive $calcVBI(i, j)$

input : base indices i and j
output: $WM(i,j)$

begin
  $WM_{ij} \leftarrow V(i,j)+\texttt{auPen}(i,j)+b$;
  $WM_{idj} \leftarrow V(i+1,j)+\texttt{dangle-3'}(j,i+1,i)+\texttt{auPen}(i+1,j)+b+c$;
  $WM_{ijd} \leftarrow V(i,j-1)+\texttt{dangle-5'}(j-1,i,j)\ +\ \texttt{auPen}(i,j-1)+b+c$;
  $WM_{idjd} \leftarrow V(i+1,j-1)+\texttt{dangle-3'}(j-1,i+1,i)$
  $+\texttt{dangle-5'}(j-1,i+1,j)\ +\ \texttt{auPen}(i+1,j-1)+b+2c$;
  $WM(i,j) \leftarrow \texttt{MIN}(WM_{ij}, WM_{idj}, WM_{ijd}, WM_{idjd})$;
  for $h \leftarrow i$ **to** $j-1$ **do**
    $WM(i,j) \leftarrow \texttt{MIN}(WM(i,j), WM(i,h)+WM(h+1,j))$;
  end
  $WM(i,j) \leftarrow \texttt{MIN}(\ WM(i+1,j)+c, WM(i,j-1)+c, WM(i,j))\ ;$
  return $WM(i,j)$;
end

Algorithm 4: $calcWM(i,j)$

input : base indices i and j
output: $VM(i,j)$

begin
  $VM_{ij} = VM_{idj} = VM_{ijd} = VM_{idjd} = INF$;
  // a, b, c are multiloop offset, helix penalty and free base penalty.
  for $h \leftarrow i+2$ **to** $j-1$ **do**
    $VM_{ij} \leftarrow$ MIN($VM_{ij}, WM(i+1,h-1) + WM(h,j-1)$);
  end
  for $h \leftarrow i+3$ **to** $j-1$ **do**
    $VM_{idj} \leftarrow$ MIN($VM_{idj}, WM(i+2,h-1) + WM(h,j-1)$);
  end
  $VM_{idj} \leftarrow VM_{idj}$+dangle-5'$(i,j,i+1)$+$c$;
  for $h \leftarrow i+2$ **to** $j-2$ **do**
    $VM_{ijd} \leftarrow$ MIN($VM_{ijd}, WM(i+1,h-1) + WM(h,j-2)$);
  end
  $VM_{ijd} \leftarrow VM_{ijd} +$ dangle-3'$(i,j,j-1)$ + c;
  for $h \leftarrow i+3$ **to** $j-2$ **do**
    $VM_{idjd} \leftarrow$ MIN($VM_{idjd}, WM(i+2,h-1) + WM(h,j-2)$);
  end
  $VM_{idjd} \leftarrow VM_{idjd}$ + dangle-5'$(i,j,i+1)$ + dangle-3'$(i,j,j-1)$ +$2c$;
  $VM(i,j) \leftarrow$ MIN($VM_{ij}, VM_{idj}, VM_{ijd}, VM_{idjd}$);
  $VM(i,j) \leftarrow VM(i,j) + a + b +$ auPen(i,j);
  return $VM(i,j)$;
end

Algorithm 5: $calcVM(i,j)$

input : base indices i and j
output: $V(i,j)$
begin
  $V(i,j) \leftarrow$ MIN(eH(i,j), eS(i,j) +$V(i+1,j-1), VBI(i,j), VM(i,j)$);
  return $V(i,j)$;
end

Algorithm 6: $calcV(i,j)$

input : base index j
output: Optimal Energy of the sequence from 1 to j, $W(j)$

begin
  for $i \leftarrow 1$ **to** $j-1$ **do**
    $wim1 \leftarrow \texttt{MIN}(0, W(i-1))$
    $W_{ij} \leftarrow V(i,j)+\texttt{auPen}(i,j)+wim1;$
    $W_{idj} \leftarrow V(i+1,j)+\texttt{dangle-3'}(j,i+1,i)+\texttt{auPen}(i+1,j)+wim1;$
    $W_{ijd} \leftarrow V(i,j-1)+\texttt{dangle-5'}(j-1,i,j)+\texttt{auPen}(i,j-1)+wim1;$
    $W_{idjd} \leftarrow V(i+1,j-1)+\texttt{dangle-3'}(j-1,i+1,i)+\texttt{dangle-5'}(j-1,i+1,j)+\texttt{auPen}(i+1,j-1)+wim1;$
    $W(j) \leftarrow \texttt{MIN}(W(j), W_{ij}, W_{idj}, W_{ijd}, W_{idjd});$
  end
  $W(j)=\texttt{MIN}(\ W(j), W(j-1));$
  return $W(j)$;
end

Algorithm 7: $calcW(j)$

input : Base indices i and j
output: $VBI(i,j)$
begin
  $maxloop \leftarrow 30;$
  for $i_p \leftarrow i+1$ **to** $i+maxloop+1$ **do**
    for $j_p \leftarrow j-maxloop-1+(i_p-i-1)$ **to** $j-1$ & $j_p > i_p$ **do**
      $VBI(i,j) \leftarrow \texttt{MIN}(\ VBI(i,j),\ \texttt{eL}(i,j,i_p,j_p)\ + V(i_p,j_p));$
    end
  end
  return $VBI(i,j)$;
end

Algorithm 8: $calcVBI(i,j)$ using the heuristic

input : Indices i and j
output: $VBI(i,j)$

begin
$c \leftarrow 3$;
// bases i_p and j_p forms the interior base pair
// Case 1: When first side < c
for $i_p \leftarrow i+1$ **to** $i+c$ **do**
  for $j_p \leftarrow i_p+1$ **to** $j-1$ **do**
    $VBI(i,j) \leftarrow$ MIN($VBI(i,j)$,eL$(i,j,i_p,j_p)+V(i_p,j_p)$);
  end
end
// Case 2: When first side >= c, and second side < c
for $i_p \leftarrow i+c+1$ **to** $j-2$ **do**
  for $j_p \leftarrow j-c$ **to** $j-1$ & $j_p > i_p$ **do**
    $VBI(i,j) \leftarrow$ MIN($VBI(i,j)$,eL$(i,j,i_p,j_p) + V(i_p,j_p)$);
  end
end
// case 3: General Cases - includes three subcases
// case 3.1: When both sides=c
$i_p = i+c+1$;
$j_p = j-c-1$;
$VBI(i,j) \leftarrow$ MIN($VBI(i,j)$, eL$(i,j,i_p,j_p)+V(i_p,j_p)$);
extend1(i,j,i_p,j_p);
// case 3.2: When First side=c+1, second side=c
$i_p = i+c+2$;
$j_p = j-c-1$;
$E_1 \leftarrow$ eL$(i,j,i_p,j_p)+V(i_p,j_p)$;
// subcase 3.3: When First side=c, second side=c+1
$i_{p1} = i+c+1$;
$j_{p1} = j-c-2$;
$E_2 \leftarrow$ eL$(i,j,i_{p1},j_{p1})+V(i_{p1},j_{p1})$;
$VBI(i,j) \leftarrow$ MIN$(VBI(i,j),E_1,E_2)$;
if $E_1 > E_2$ **then**
  $i_p = i+c+1$;
  $j_p = j-c-2$;
end
extend2(i,j,i_p,j_p);
return $VBI(i,j)$
end

Algorithm 9: $calcVBI(i,j)$ using the internal Loop Speedup Algorithm

```
input: Variable i, j, i_p, j_p
begin
    i_v ← i + c + 1;
    j_v ← j − c − 1;
    for b ← 1 to MIN(i − 1,N − j) do
        VBI(i − b, j + b) ← MIN(VBI(i − b, j + b), eL(i, j, i_p, j_p) + V(i_p, j_p));
        // Two more options
        if VBI(i − b, j + b) > eL(i, j, i_v + b, j_v + b) + V(i_v + b, j_v + b) then
            i_p ← i_v + b;
            j_p ← j_v + b;
            VBI(i − b, j + b) ← eL(i, j, i_p, j_p) + V(i_p, j_p) ;
        end
        if VBI(i − b, j + b) > eL(i, j, i_v − b, j_v − b) + V(i_v − b, j_v − b) then
            i_p ← i_v − b;
            j_p ← j_v − b;
            VBI(i − b, j + b) ← eL(i, j, i_p, j_p) + V(i_p, j_p) ;
        end
    end
end
```

Algorithm 10: $extend1(i, j, i_p, j_p)$

```
input: Variable i, j, i_p, j_p
begin
    i_v ← i + c + 1;
    j_v ← j − c − 1;
    for b ← 1 to MIN(i − 1,N − j) do
        VBI(i − b, j + b) ← MIN(VBI(i − b, j + b), eL(i, j, i_p, j_p) + V(i_p, j_p));
        // Two more options
        if VBI(i − b, j + b) > eL(i, j, i_v + b + 1, j_v + b) + V(i_v + b + 1, j_v + b) then
            i_p ← i_v + b + 1;
            j_p ← j_v + b;
            VBI(i − b, j + b) ← eL(i, j, i_p, j_p) + V(i_p, j_p)
        end
        if VBI(i − b, j + b) > eL(i, j, i_v − b, j_v − b − 1) + V(i_v − b, j_v − b − 1) then
            i_p ← i_v − b;
            j_p ← j_v − b − 1;
            VBI(i − b, j + b) ← eL(i, j, i_p, j_p) + V(i_p, j_p);
        end
    end
end
```

Algorithm 11: $extend2(i, j, i_p, j_p)$