SP-fold – Speculative Parallelization for Parallel Algorithm of RNA Secondary Structure Prediction on Multicore

Narendra Chaudhari
Department of Information Technology
Dr Babasahab Ambedkar College of Engg. and Research, Nagpur, India

Anjali Mahajan
Department of Computer Science & Engineering
Priyadarshini Institute of Engineering & Technolog, Nagpur, India

ABSTRACT
Bioinformatics is faced with accelerating increase of data set sizes originating from powerful high-throughput measuring devices. Extensive computational power is the basic requirement for solving problems in bioinformatics. One of the key solutions to time-efficient data processing is the proper implementation of computational intensive tasks using parallel technology. A large number of cores is combined into a single chip to improve the overall performance of the multi-core processors. This depicts the current trends in processor architecture.

This paper proposes a new software-only speculative parallelization scheme for implementing RNA Secondary Structure Prediction algorithm in parallel. The scheme is developed after a systematic evaluation of the design options available. It is also shown to be efficient, robust and to outperform previously proposed schemes used for parallel implementation of RNA Secondary Structure Prediction.

Categories and Subject Descriptors
D.1.3 [Software]: Concurrent programming-Parallel programming.

General Terms
Parallel algorithm, Bioinformatics,

Keywords
RNA Secondary Structure Prediction, Speculative Parallelization

1. INTRODUCTION
Algorithms based on Thermodynamic models for RNA Secondary Structure Prediction have been parallelized in the past. Those algorithms were developed for specific computer architectures that were available over two decades ago. These previous parallel implementations are not compatible with current architectures because since then, parallel computing technology has changed substantially[12].

The current architectures includes multi-core processors, memory hierarchies with several levels of cache, and the availability of enormous memory. Therefore, the need for new parallel implementations to take full advantage of these new platforms has emerged[5].

The structure having the minimum free energy (MFE) is defined to be the secondary structure of the molecule. This is as per the thermodynamic hypothesis. There are distinct substructures called loops. The independent sum of the free energies of the loops together contributes to the free energy of a secondary structure. The dynamic programming algorithm given by Zuker and Stiegler in 1981[16] has been used to perform the optimization. This algorithm is similar to the algorithm for sequence alignment but far more complex. The algorithm is designed in such a way that it explores all the possibilities when computing the MFE structure. To satisfy the computational requirements in the existing folding programs the use of heuristics and approximations is done.

In this paper, the new algorithm for implementation of parallel execution of RNA secondary structure prediction by Speculative Parallelization for modern multi-core computers is proposed. We have used speculative parallelization technique for converting the sequential loops to a form suitable for parallel execution. Code sections are aggressively executed in parallel with speculative parallelization[2]. Hardware schemes require modifications to the processors and memory system. They are fast but expensive. Software schemes require no extra hardware but can be inefficient sometimes. The proposed scheme is novel. It is implemented using carefully tuned data structures, proper selection of synchronization policies, and scheduling mechanisms

We are successful in reducing the prediction time for long RNA sequences on modern parallel platforms, in comparison to sequential algorithms which still take many minutes to produce a single MFE structure.

Our program SP-fold has been compared with two of the most widely-used sequential programs, UNAfold [10] and RNAfold [1][8].

2. RNA SECONDRY STRUCTURE
An RNA strand consists of a sugar phosphate backbone. To this backbone the four bases cytosine (C), guanine(G), adenine (A), and uracil(U) are attached. Each strand has two chemically distinct ends, starting known as the 5' end and concluding known as 3' end. Simple Watson-Crick base pairing involves the bonding of C with G and A with U via three or two hydrogen bonds, respectively. Additionally, pairs can form between G and U called as wobble. A secondary structure describes which bases are paired for a given RNA strand. The secondary structure of a strand of length n is a set of pairs (i,j) where i and j are in the range [1,...,n] . The pair (i,j) represents a pairing between the i^{th} and j^{th} bases in the strand. The bases in the strand are indexed from 1 to n. It starts at the 5' end. In a secondary structure, each base has at most one partner[6]. The substructures called stems or helices have base pairs which are most often found stacked onto other base pairs. The unpaired bases interspersed in stems are known as internal loops or bulges. Loops occurring at the ends of stems are called hairpins. The loops from which more

than two stems originate are known as multi-branched loops, or simply multiloops. Fig. 1 shows a standard display of a secondary structure for an RNA strand, in which the stems and loops are apparent. This particular secondary structure is pseudoknot-free. The structure is pseudoknot free if it does not have any two base pairs (i,j) and (i',j') where i<i'<j<j'. We are dealing with pseudoknot free structures.

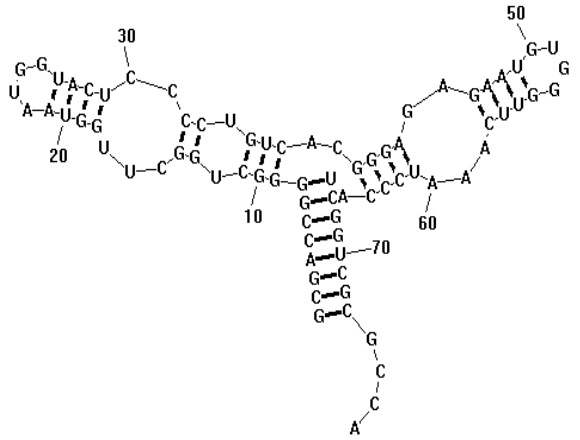

Figure 1 Secondary Structure for RNA Strand

To model the full structure of the molecule, the information on the secondary structure of an RNA molecule can be used[4].

RNA secondary structure prediction is the problem of determining the most stable structure for a given sequence. We measure stability in terms of the free energy of the structure. Thus we want to find a structure of minimal free energy which we will also call an optimal structure. Nearest neighbor thermodynamic model (NNTM) provides a set of functions and sequence dependent parameters to calculate the energy of various kinds of loops. The energy of a secondary structure is assumed to be the sum of the energies of the loops of the structure and furthermore the loops are assumed to be independent, that is, the energy of a loop only depends on the loop and not on the rest of the structure [3].

2.1 Secondary Structure Prediction Algorithm

The prediction of secondary structures with the minimum free energy is an optimization problem. The optimum scores calculations for the structure is done through Traceback Algorithm. The prediction algorithm requires five tables with complex dependencies. Each class of loop has a different energy function. All enclosed base pairs are required to be searched for calculating the energies of the internal loops and multiloops with one or more branches. This makes the loop optimal for the closing base pair.

Recursive minimization formulas are used to define the algorithm. Let, N be the length of an RNA sequence, it's free energy denoted by FE(*N*), and i, j are the index values such that $1 \le i < j \le N$. The optimal free energy of a subsequence of the RNA sequence from the index values 1 to *j* is given with the following formula[1]:

$$FE(j) = \min\left\{FE(j-1),\ \min_{1\le i<j}\{OE(i,j)+FE(i-1)\}\right\} \qquad (1)$$

The term OE(i,j) denotes the optimal energy of the subsequence from i to j in equation (1). It forms a base pair (i, j). It is defined by the following formula.

$$OE(i,j) = \min\begin{cases} EH(i,j), \\ eS(i,j) + OE(i+1,j-1), \\ OEI(i,j), \\ OEM(i,j) \end{cases} \qquad (2)$$

A base pair (*i*, *j*) closes many loops which are considered by equation (2). The function E*H*(*i*, *j*) calculates the energy of a hairpin loop closed by base pair (*i*, *j*). Function e*S*(*i*, *j*) calculates the energy of a stack formed by base pairs (*i*, *j*) and ($i + 1$, $j - 1$). OE*I*(*i*, *j*) and OE*M*(*i*, *j*) are the optimal free energies of the subsequence from *i* to *j*. OEI(i,j) denotes the optimal free energy closed by (*i*, *j*) base pair for an internal loop and OEM(i,j) for a multiloop.

$$OEI(i,j) = \min_{i<i'<j'<j}\{eL(i,j,i'j') + (i',j')\} \qquad (3)$$

Where, i'-i+j-j'-2>0

The multiloop energy function is formed. It is linearly dependent upon the number of single stranded bases present in the multiloop. A 2-Dimensional array denoted by, *OEM2*, is used to facilitate the calculation of *OEM* array. The calculations of *2OEM* (*i*, *j*) and *OEM* (*i*, *j*) are shown by formulae (4) and (5) respectively.

$$OEM2(i,j) = \min\begin{cases} OE(i,j) + b, \\ OEM2\,(i,j-1) + c, \\ OEM2(i+1,j) + c, \\ \min_{i<k\le j}\{OEM2(i,k-1) + OEM2(k,j)\} \end{cases} \qquad (4)$$

$$OEM(i,j) = \min_{i+1<h\le j-1}\{OEM2(i+1,h-1), OEM2(h,j-1) + a\} \qquad (5)$$

We have implemented these minimization formulas recursively.

2.2 Speculative Parallelization Algorithm for RNA Secondary Structure Prediction

There are many computations involved in finding the internal loops and multiloops in the proposed algorithm. They are the most expensive parts of the algorithm. This can be noticed from the Secondary Structure Prediction Algorithm described in previous section. In Eq. (3), in the calculation of *OEI*(*i*, *j*), all possible internal loops with the closing base pair (*i*, *j*) are considered. This is done 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 reduce the complexity we are using speculative parallelization technique, which converts the sequential loops to a form suitable for parallel execution. The code sections are aggressively executed in parallel with speculative parallelization.

2.2.1 Speculative Parallelism

The proposed approach to speculative execution is presented here. Consider a pair of sub-computing statements S and S' in a sequential program P. Statements S and S' may represent consecutive iterations of a loop. The execution of S precedes the execution of S', i.e. S →S' during sequential execution. Thus, the values computed during the execution of S should be available during the execution of S'. The goal of speculative execution is to allow the statements to execute in any order. The order of execution of S and S' can be such that by speculatively executing S' while S is still executing, the final values are not affected. During the speculative execution of S', if a data value is read before it has been computed by S, the dependency is violated[17]. Then misspeculation occurs. Misspeculation produces wrong results. So, the values computed during speculative execution of S' must be discarded and S' must be executed again. On the other hand, if misspeculation does not occur, the execution time of the program is potentially reduced due to parallel execution of the consecutive statements. Of course, speculative execution is only beneficial if the misspeculation does not occur frequently.

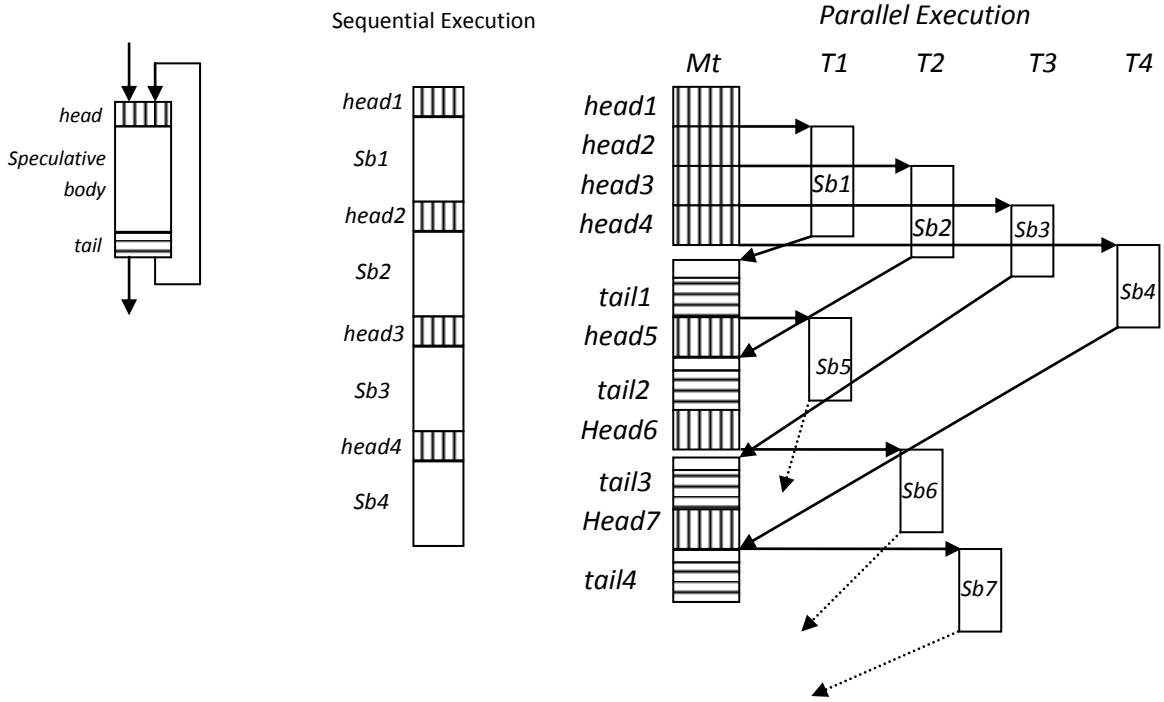

Figure 2. Speculative parallelization of loops

Given a series of dependent computation statements $S_1 \rightarrow S_2 \rightarrow \ldots S_n$, S_1 is scheduled to execute non-speculatively, and S_2 through S_n are scheduled in parallel with S_1 on additional cores. This type of scheduling of S_2 through S_n is called as speculative parallelization.The execution model described above has been used to support speculative execution to execute the sequential RNA Secondary Structure Prediction program on multicore.

2.2.2 Approach

In speculative parallelization systems, speculative thread execution is achieved across several processing cores. These processing cores have private caches. For simplicity, assume that the system is only running the program of interest. The processing core running this sequential program guarantees its correctness. In our model, there are two states of computation of the program- the non-speculative state and the speculative state. These two states are maintained separately from each other.

The non-speculative state of the computation is maintained by the main thread of the parallelized application and multiple parallel threads execute parts of the computation using speculatively-read operand values from non-speculative state. This produces parts of the speculative state of the computation. The computed values are held until they are squashed when a dependency with the program is violated, or committed and used when they are requested by the (non-speculative) program or by other speculative threads[7].

During the execution of speculative thread, the changes made by stores are hold separately, usually in the private cache of the processing core on which the thread is executing, and providing them to dependent loads.

The program locations accessed and modified by the thread are tagged into write and read sets, respectively, along with the modified architected registers. These are held in private caches or auxiliary storage structures until the speculative thread is invalidated or committed. If the speculative thread

had referenced a location during its execution that is modified by the program, it indicates violation of data dependencies. Then, the speculative thread is squashed and its data is discarded [11]. This way of scheduling ensures the isolation of execution of each thread, main or parallel, from execution of all other threads. In case of misspeculation, the reexecution of other parallel threads is not required.

Suppose, we have the computing statements, S →S' which are dependent. We speculate that no dependencies exist from S to S'. We start speculative execution of S' in parallel with S. We are speculating that the variables being used by speculative threads S' will not be modified by S. At the end of the execution of S', we must check to see if S modified any of the variables that were speculatively read by S'. If this is

indeed the case, misspeculation occurs and recovery is performed by re-executing S'. Otherwise speculation is successful and the results computed by S' are copied back to non-speculative state. This is shown in Fig. 2.

Consider a series of dependent computations $S_1 \rightarrow S_2 \rightarrow \ldots S_n$ such that, while S_1 executes non-speculatively, if we speculatively execute S_2 through S_n in parallel with S_1, then the results of S_2 through S_n must be copied to the non-speculative state in-order. This will ensure that if any dependences arise between S_i and S_j (for all $i < j$), they are correctly enforced.

3. LITERATURE SURVEY

3.1 RNA Secondary Structure Prediction

There are several approaches which exist for RNA secondary structure prediction. There are many best known algorithms for predicting the secondary structure of a single input RNA molecule. One approach [13] focuses on free energy minimization secondary structure with respect to a given standard thermodynamic model. They have shown that the task is NP-hard.

A dynamic programming algorithm [16] is described for finding the MFE pseudoknot free secondary structure of a given molecule. The other approach [5] predicts secondary structures by computing the structures of smaller subsequences. These subsequences are then used to rebuild the full structure. The rebuilding methods are based on identify motifs correctly by consistently combining the substructures into a full structure. This ability is the main issue involved for successful execution of such programs.

A very efficient multicore algorithm for RNA Secondary Structure Prediction which uses the medium-grained level is implemented in [9].

Several algorithms of RNA secondary structure prediction have been developed for distributed memory implementations. [14] and [15] are the few examples of this type of implementation.

3.2 Speculative Parallelization

With the aim to extract parallelism from sequential code, lot of research has been done on thread level speculation (TLS)[16]. Hardware support is needed for executing threads in TLS. It also requires a multithreading processor which has the ability to detect misspeculation and recover the computation state. The hardware features are expensive and not easily available in commercial processors. [17] proposed a process based runtime model that enables speculative parallel execution. This method executes Potentially Parallel Regions on multiple cores.

In comparison to TLS techniques, our approach is implemented purely in software. There is no requirement for hardware to detect misspeculation. The software solutions are practical.

4. RESULT

The results of execution speedups obtained using our SP-fold speculative parallelization algorithm applied to RNA Secondary Structure Predication problem are presented. We are interested in the accuracy and run time measurement of our implementations. In this experiment, first the baseline which is the execution time of the sequential algorithms such as UNAfold[10] and RNAfold[1] is measured. Then, the execution time of our algorithm with different numbers of parallel threads is considered. It is found that SP-fold runs faster than UNAfold and RNAfold and achieves accuracy comparable with them. The main loops in the SP-fold contain definite loop dependencies which are caused by commutative statements. A multiple speculative code segment is created from single loop iteration. This is the basic requirement of parallelization. These segments are intervened by statements involving definite loop dependences.

It is noticed that the performance of our algorithm with one parallel thread is slightly worse than the RNAfold. That is due to the overheads involved in speculations. In fact, if only one core is available, we can force the main thread to perform all computations rather than spawning parallel threads.

Fig. 3 shows the comparison of the running time of SP-fold, UNAFold, and RNAfold, for an HIV-1 sequence with 9,781 nucleotides. SP-fold with threads more than one performs much better than UNAfold and RNAfold. It is seen that the running time of SP-fold decreases with the increasing number of threads. It is far better than the other two programs.

The number of cores is the main parameter for a parallel algorithm. The execution time of SP-fold on different number of cores is compared. SP-fold on *n* cores is referred as SP-fold(*n*). The results of execution time for UNAfold and RNAfold with SP-fold(*n*) on *n* = 1, 2, 4, 8, 16, 32 cores are given in Figure 4.1 and 4.2. The programs were run to predict Minimum Free Energy structure for different length subsequences of the HIV-1 genome. It is observed that the performance of SP-fold is always better than UNAfold. It always runs faster than UNAfold.

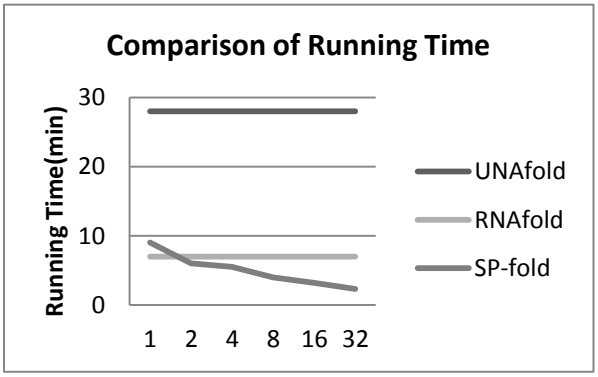

Figure 3. Comparison of Running Time

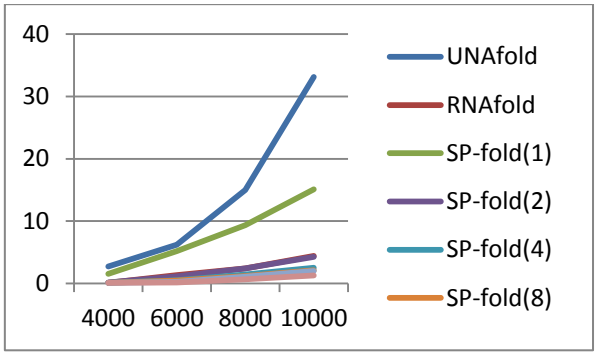

Figure 4.1. Execution Time for different sequences

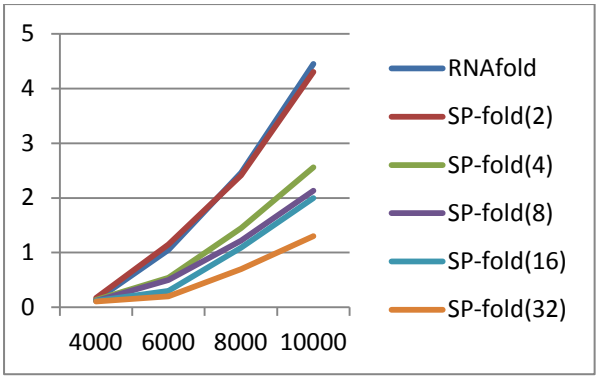

Figure 4.2. Execution Time for different sequences

The execution time of RNA fold is comparable with SP-fold(2). Both of these algorithms compute the Minimum Free Energy structure for the full genome in 7.45 minutes, whereas SP-fold(1) is slower than RNAfold. SP-fold on four or more cores computes Minimum Free Energy structure of full genome in less time than RNAfold.

4.1 Overhead Involved

The proposed technique has overheads involved due to instructions introduced during parallelization. There are 2 types of overheads – Time overhead and Memory overhead. This execution time overhead is measured in terms of the fraction of total instructions executed on each core. The results are based upon an experiment in which we use 32 parallel threads. We break the time overhead into various categories, such as the overhead due to copy operations performed by main thread. It signifies the total number of instructions used for performing coping. It depends on how many variables are needed to be copied. The other type of overhead involves fraction of instructions to be copied, instructions used by parallel threads.

Table. 1. Overheads involved

Overhead due to				
Variable Copy (%)	**Instruction Copy (%)**	**Misspeculation Check (%)**	**Setup (%)**	**Space (%)**
1.23	0.37	1.89	0.48	0.2

The most important overhead is due to misspeculation checking. This uses 1-2% instructions. Apart from this, the last category of time overhead is due to setup operations which involve thread initialization, mapping table allocation and deallocation, etc. The memory overhead depends on the extra space used during speculative parallelization. The Table 1 shows overhead occuring on core during execution of SP-fold.

5. CONCLUSION

A parallel algorithm for RNA Secondary Structure Prediction using Speculative Parallelization for multicore architecture is presented. The proposed algorithm achieves significant speedups over the current best sequential programs and has accuracy comparable to them. Results also illustrate the limitation of the speculative parallelization algorithms. It involves various overheads which can be reduced by reducing overhead involved in misspeculation check.

6. 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] A. Bhowmik, M. Franklin , " A general compiler framework for speculative multithreading." In: SPAA,pp. 99–108 , 2002.

[3] 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

[4] S.R. Eddy, "Computational analysis of conserved RNA secondary structure in transcriptomes and genomes", Technical Report, HHMI Janelia Farm Research Campus, 2013.

[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] J. Gregory Steffan, , C. B. Colohan, , A. Zhai, T.C. Mowry, " A scalable approach to thread-level speculation.", In: ISCA, pp. 1–12 , 2000.

[8] I.L. Hofacker, P.F. Stadler, L.S. Bonhoeffer, M. Tacker, and P. Schuster, "Fast folding and comparison of RNA secondary structures.", Monatshefte f˙ur Chemie, 125:167–188, 1994.

[9] Mathuriya, D. Bader,C. Heitsch, S. Harvey, S,"GTfold: A Scalable Multicore Code for RNA Secondary Structure Prediction.", Technical report, Georgia Institute of Technology 2008.

[10] N.R.Markham, M. Zuker, "UNAFold: Software for Nucleic Acid Folding and Hybridization.", In Bioinformatics: Structure, Function, and Applications, Volume 453 of Methods in Molecular Biology. 2008:3–31.

[11] P . Marcuello, A. González, " Clustered speculative multithreaded processors. ", In: ICS, pp. 365–372,1999.

[12] B.K.Pandey, S.K.Pandey, D.Pandey," A Survey of Bioinformatics Applications on Parallel Architectures" International Journal of Computer Applications (0975 – 8887)Volume 23– No.4, June 2011

[13] Z. Sukosd ,M.S. Swenson, J. Kjems ,C.E. Heitsch, "Evaluating the accuracy of SHAPE directed RNA secondary structure predictions.",Journal of Nuclear Acids Research. 41:2807–16, 2013

[14] 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.

[15] 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.

[16] 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.

[17] C. Zilles, G. Sohi, "Master/slave speculative parallelization.", In: MICRO, pp. 85–96 , 2002.