Parallel RNA Sequence-Structure Alignment

Tong Liu and Bertil Schmidt
School of Computer Engineering, Nanyang Technological University, Singapore 639798,
pg04221219@ntu.edu.sg, asbschmidt@ntu.edu.sg

Abstract

With the growing number of known RNA genes efficient and accurate computational analysis of RNA sequences is becoming increasingly important. Stochastic context-free grammars (SCFGs) are used as a popular tool to model RNA secondary structures. However, algorithms for aligning an RNA sequence to an SCFG are highly compute-intensive. This has so far limited applications of SCFGs to relatively small problem sizes. In this paper we present the design of a parallel RNA sequence-structure alignment algorithm. Its implementation on a PC cluster leads to significant runtime savings. This makes it possible to compute sequence-structure alignments of even the largest RNAs such as SSU rRNAs and LSU rRNAs in reasonable time.

1. Introduction

Sequence alignment is the most important basic operation in computational molecular biology. Several alignment tools are well established for DNA and protein sequence analysis. These include BLAST [2], FASTA [16], HMMER [1], and ClustalW [21] to name just a few. Besides DNA and proteins there are also many biologically important macromolecules that are composed of RNA such as transfer RNA and ribosomal RNA. Alignment of RNA sequences is important to identify RNA gene families and RNA motifs [17]. Since the number of known RNA genes is expanding rapidly, fast and sensitive RNA sequence analysis is of high importance to research in this area [20].

Unfortunately, many RNA families only exhibit a relatively low degree of primary sequence similarity but rather conserve a consensus base-paired secondary structure. Techniques that generally work well for protein and DNA sequence analysis are therefore generally not well suited for studying RNA. Stochastic context-free grammars (SCFGs) have been introduced in computational biology to address this problem [9,18]. SCFGs provide a general probabilistic framework for RNA families that models both primary sequence consensus as well as information present in the secondary structure. Applications of SCFG-based RNA models are searching databases for homologous RNA sequences with high sensitivity and building multiple RNA sequence alignments. In the context of biological sequence analysis, SCFGs can be seen as an extension of profile Hidden Markov Models (HMMs) to cope with secondary structures [7]. RFAM [11] is a public available database of SCFGs representing non-coding RNA families, which is comparable to the PFAM database of profile HMMs representing protein families [4].

Applications of SCFG-based RNA models require the alignment of an RNA sequence to a model. Even though polynomial-time dynamic programming algorithms exist for calculating the optimal alignment, the runtime is still very high. The corresponding computational complexity for a sequence of length L and a model with K states is $O(K \cdot L^2 + B \cdot L^3)$, where B is the number of bifurcation states in the model. This high computational alignment cost has limited the use of SCFG-based RNA models to small structural RNAs, e.g. transfer RNA genes [14]. However, the alignment runtime on sequential computers is too high for many larger RNAs of interest such as ribosomal RNA (rRNA).

One approach to speed up this time consuming procedure is to introduce heuristics in the alignment algorithms [5,13]. The main drawback of this solution is that the more efficient the heuristics, the worse is the quality of the result. Another approach to get high quality results in a reasonable time is to use parallel processing. In this paper we present an efficient parallel algorithm for aligning an RNA sequence to an SCFG model. The algorithm has been implemented on a PC cluster using C and MPI. The implementation achieves a speedup of 16 using 20 processors. This makes it possible to align long subunit rRNAs (LSU rRNAs) using SCFGs in approximately 15 minutes, which requires 4 hours on a single Pentium 4.

Parallelization of dynamic programming alignment algorithms in computational biology has been done several times before, e.g. [3,10,19]. Previous approaches focused on parallelizing two-dimensional dynamic programming algorithms such as the Smith-Waterman and the Viterbi algo-

Table 1. The seven state types for modelling RNA with corresponding production rules. Each overall production probability is the independent product of an emission probability e_v and a transition probability t_v. Both are position dependent parameters that depend on state v. For example, a P-state v generates two characters a and b (one of 16 possible base-pairs) with probability $e_v(a, b)$ and transits to one of several possible new states Y with probability $t_v(Y)$. B-states split into two new S-states with probability 1 and E-states terminate derivations.

State type	Description	Production	Emission	Transition
P	pair emitting	$P \rightarrow aYb$	$e_v(a, b)$	$t_v(Y)$
L	left emitting	$L \rightarrow aY$	$e_v(a)$	$t_v(Y)$
R	right emitting	$R \rightarrow Ya$	$e_v(a)$	$t_v(Y)$
B	bifurcation	$B \rightarrow SS$	1	1
S	start	$S \rightarrow Y$	1	$t_v(Y)$
D	delete	$D \rightarrow Y$	1	$t_v(Y)$
E	end	$E \rightarrow \varepsilon$	1	1

rithm. The solution presented in this paper is more complex since it involves a three-dimensional dynamic programming algorithm with different recurrence relations for the individual states of the grammar.

The rest of this paper is organized as follows. Section 2 provides a description of SCFG models of RNA. The dynamic programming algorithm for aligning a sequence to a model is described in Section 3. The new parallel algorithm is introduced in Section 4. Experimental results are presented in Section 5. Section 6 concludes the paper.

2. SCFG models of RNA

Context-free grammars (CFGs) are well-known structures in formal language theory. Mathematically, a CFG G is described by a 4-tuple (Σ, V, S, P), where

- Σ is a finite set, called the alphabet, e.g. $\Sigma = \{a, c, g, u\}$ for RNA
- V is a set of non-terminals (also called states)
- P is a set of production rules $\alpha \longrightarrow \beta$, where α is a non-terminal and β can be any string of non-terminals and terminals (terminals are characters from the alphabet plus the empty string ϵ)
- $S \in V$ is a special start non-termninal

A string $s \in \Sigma^*$ can be derived in G from a non-terminal U, if U can be rewritten to s by a sequence of production rules. A CFG G = (Σ, V, S, P) generates a language over , denoted by $L(G)$ as follows: A string $s \in \Sigma^*$ is in $L(G) \Longleftrightarrow s$ can be derived from S. An SCFG G is a CFG in which all production rules are assigned probabilities such that the sum of the probabilities of all possible production rules from any given non-terminal is one [7].

For modelling RNA secondary structures we are using a specific SCFG architecture that has only seven different types of states (non-terminals). The states with associated production rules are shown in Table 1. Figure 1 illustrates an example how they can be used to model an RNA sequence family with annotated secondary structure multiple alignment. P-states are used to model base-paired columns, L-states model single-stranded columns wherever possible, R-states model bulges on the 3' side of a stem, and B-states model splits into multiple stems or multi-branch loops.

The SCFG in Figure 1 is flexible enough to tolerate mismatches to the consensus sequence by replacing specific base assignments in the parse tree with symbol emission probabilities from one of the 16 possible pairwise nucleotide combinations (for P-states) or one of the 4 possible singlet nucleotides (for L- and R-states). However, it does not allow for insertions and deletions. In order to account for insertions and deletions, states are extended to nodes as follows: P-states to $MATP$-nodes, L-states to $MATL$-nodes, R-states to $MATR$-nodes, B-states to BIF-nodes, S-states to $ROOT$-, $BEGL$- or $BEGR$-nodes, and E-states to END-nodes. Figure 2a shows this expansion for the model in Figure 1. Each node-type itself consists of a group of states with particular properties (see Table 2). Match states (MP, ML, MR) account for the conserved consensus columns of an alignment. Insert states (IL, IR) account for insertions relative to the consensus. Delete states (DEL) emit nothing and allow for the possibility of deletions relative to the consensus. States are connected to certain other states by state transitions. All node-types have fixed state transition structures, which are displayed in Figure 2b. The resulting SCFG-architecture for modelling RNAs is called Covariance Model (CM) [9].

3 Alignment Algorithm

An RNA sequence can be aligned to a CM to determine the probability that the sequence belongs to the modelled family. The most probable parse tree generating (or emit-

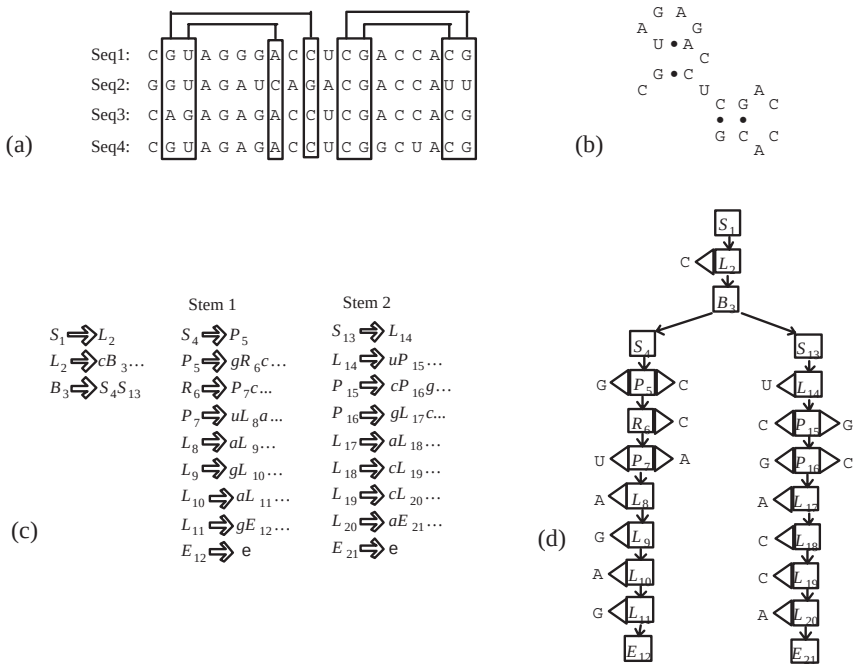

Figure 1. (a) An ungapped multiple alignment of an RNA family with 4 sequences. The annotated secondary structure of this family is represented by the boxed base-paired positions in the alignment and base-paired partners are connected by lines. (b) Corresponding consensus secondary structure. (c) Production rules of an SCFG modelling this structure using the state types from Table 1. For clarity, only one of the possible productions is shown for each state (the production corresponding to the consensus sequence of the multiple alignment). (d) Parse tree for the consensus sequence in which RNA nucleotides are assigned to the SCFG-states. Parse trees can be used to display sequences emitted (or generated) by the SCFG.

Table 2. Basic node-types of a CM with corresponding states.

Node	$MATP$	$MATL$	$MATR$	$BIFUR$	$ROOT$	$BEGL$	$BEGR$	END
States	MP, ML, MR	ML	MR, D	BIF	BEG	BEG	BEG	E
	DEL, IL, IR	DEL, IL	IR		IL, IR		IL	

ting) the sequence determines the similarity score. The CYK/inside algorithm computes this score by dynamic programming (DP) as follows [12]. Given is an input sequence $x = x_1 \ldots x_L$ of length L and a CM G of length K with states numbered in preorder traversal. The CYK/inside algorithm iteratively calculates a 3-dimensional DP matrix $M(i, j, k)$ for all $i = 1, \ldots, j + 1, j = 0, \ldots, L, k = 1, \ldots, K$. $M(i, j, k)$ is the log-odds score of the most likely CM parse tree rooted at state k that generates the subsequence $x_i \ldots x_j$. The matrix is initialized for the smallest subtrees and subsequences, i.e. subtrees rooted at E-states and subsequences of length 0. The iteration then proceeds outwards to progressively longer subsequences and larger CM subtrees. Computation of $M(i, j, k)$ for $1 \leq i \leq j + 1, 0 \leq j \leq L, 1 \leq k \leq K$ is given by the following recurrences, where $d = j - i + 1$, $S(k)$ is the type of state k, $C(k)$ is the set of states that k can transit to, $e_k(x_i)$ is the log-odds score of the emission probability of character x_i in state k, $t_k(\gamma)$ is the log-odds score of the transition probability from state k to state γ. The recurrences show that calculation of $M(i, j, k)$ is state-type dependant. For instance, if $S(k) = MP$, then k emits x_i, x_j and transits to one of its children states $C(k)$. However, the score of the optimal parse tree rooted in that generates $x_{i+1} \ldots x_{j-1}$ has already been calculated in the DP-matrix cell $M(i + 1, j - 1, \gamma)$. The maximum over all possible choices of child states is then taken to compute $M(i, j, k) : e_k(x_i, x_j) + \max_{\gamma \in \mathbf{C_k}} [M(i+1, j-1, \gamma) + t_k(\gamma)]$.

$$M(i,j,k) = \begin{cases} \max\limits_{\gamma \in \mathbf{C_k}} \left[M(i,j,\gamma) + logt_k(\gamma) \right] & if\ S(k) \in \{DEL, BEG\} \\ e_k(x_i, x_j) + \max\limits_{\gamma \in \mathbf{C_k}} \left[M(i+1, j-1, \gamma) + t_k(\gamma) \right] & if\ S(k) \in MP\ and\ d \geq 2 \\ e_k(x_i) + \max\limits_{\gamma \in \mathbf{C_k}} \left[M(i+1, j, \gamma) + t_k(\gamma) \right] & if\ S(k) \in \{ML, IL\}\ and\ d \geq 1 \\ e_k(x_j) + \max\limits_{\gamma \in \mathbf{C_k}} \left[M(i, j-1, \gamma) + t_k(\gamma) \right] & if\ S(k) \in \{MR, IR\}\ and\ d \geq 1 \\ \max\limits_{i-1 \leq mid \leq j} \left[M(i, mid, k_{left}) + M(mid+1, j, k_{right}) \right] & if\ S(k) \in BIF \\ 0 & if\ S(k) \in E \\ -\infty & otherwise \end{cases}$$

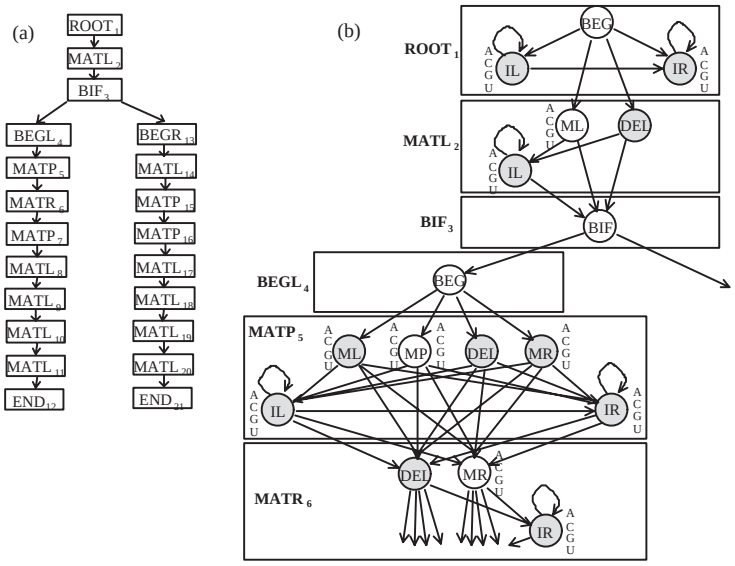

Figure 2. (a) States of the SCFG in Fig. 1 are expanded to nodes. (b) Corresponding internal state-transition structure of the first six nodes. Emission of a character is indicated by ACGU left and right of a state. Each transition and emission has an associated probability (not displayed here).

The recurrence relations for all other state-types can be explained in a similar way.

At the end of the iteration $M(1, L, 1)$ contains the score of the best parse of the complete sequence with the complete model, i.e. the global alignment. The optimal parse tree itself can be reconstructed by a traceback procedure, that follows the maximum scoring path through the DP matrix at each state. The whole algorithm requires $O(L^2 \cdot K)$ memory and $O(K \cdot L^2 + B \cdot L^3)$ time, where B is the number of BIF-states of G.

In order to compute only the optimal alignment score it is not necessary to store the complete three-dimensional DP-matrix and the space-complexity in this case can be easily reduced to $O(L^2 \cdot logK)$. The steep memory requirement of $O(L^2 \cdot K)$ for the traceback procedure is only practical for relatively small sequences and models. It becomes prohibitively large for aligning longer RNAs. However, by using a divide-and-conquer approach the space complexity can be improved to $O(L^2 \cdot logK)$ at the expense of roughly doubling the computation time [8]. The idea is similar to the linear space version of the Needleman-Wunsch algorithm for pairwise sequence alignment [15]. The divide-and-conquer approach identifies DP-matrix cells that must be part of an optimal alignment. This splits the remaining problem into smaller subproblems. These subproblems are then recursively split into even smaller subproblems until the optimal alignment is determined. This method reduces the memory requirement significantly. However, the corresponding runtime is still very high. We have therefore developed an efficient parallel version of this algorithm, which

is described in the next section.

4. Parallel Alignment Algorithm

DP algorithms often exhibit the character of a wavefront computation, that is, each matrix element depends on a set of previously computed elements. Figure 3 shows an example for a triangular two-dimensional matrix. Figure 3b displays the dependency relationship: each matrix element (i, j) is computed from the matrix cells $(i+1, j), (i, j-1), (i+1, j-1)$. The wavefront computation moves along diagonals as depicted in Figure 3a, that is, the shift direction is from southwest to northeast. Depending on the dependency relationship different wavefront shift directions are possible. The dependency relationship for layers of non-BIF states in the CYK algorithm is similar to Figure 3 with the difference that several layers have to be considers: $M(i, j, k)$ is computed from the matrix cells $M(i+1, j, \gamma)$ if $S(k) \in \{ML, IL\}$, $M(i, j-1, \gamma)$ if $S(k) \in \{MR, IR\}$, $M(i+1, j-1, \gamma)$ if $S(k) = MP$ or $M(i, j, \gamma)$ if $S(k) \in \{DEL, BEG\}$ for all $C(k)$.

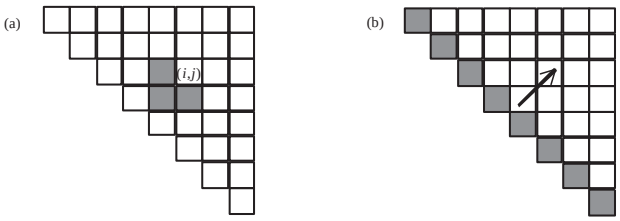

Figure 3. Example of a wavefront computation in a triangular two-dimensional DP matrix. (a) Data dependency: cell (i, j) depends on the cells $(i+1, j), (i, j-1), (i+1, j-1)$. (b) Wavefront shift direction: cells within a diagonal can be computed in parallel.

Parallelization of the wavefront computation has been done in different ways depending on the particular parallel architecture being used. On fine-grained architectures, the computation of each cell within a diagonal is parallelized [19]. However, this technique is only efficient on architectures such as systolic arrays, which have an extremely fast inter-processor communication.

In order to parallelize the wavefront computation on coarse-grained architectures like PC clusters it is more convenient to assign a number of adjacent columns to each processor. In order to reduce communication time further, matrix cells can be grouped into blocks. Assuming there are P processors with id's ranging from 0 to $P-1$, then processor i can compute all the cells within a block after receiving the required data from processor $i-1$. Figure 4 shows an

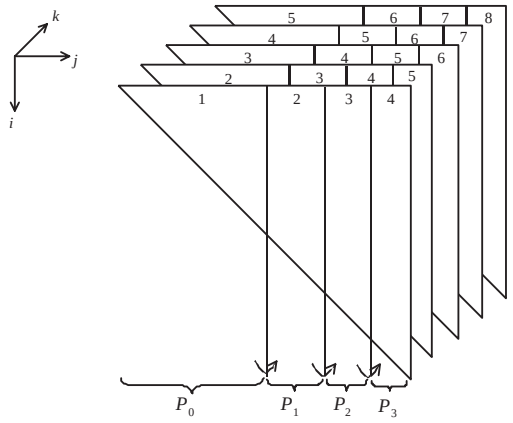

Figure 4. Decomposition of the three-dimensional DP-matrix using four processors $P_0, \ldots, P_3$. The numbers $1-8$ represent consecutive wavefront phases in which cells are computed. In phase j, P_i first receives the right column from $P_i - 1$ which has been computed in phase $j-1$, computes the cells within its own block and then sends the right column to $P_i + 1$.

example of the computation of the CYK algorithm using $P = 4$. After finishing the computation for a layer of the three-dimensional DP-matrix, lower index processors can start with the calculation of the next layer.

Given is an RNA sequence x of length L and a CM G with K states. The three-dimensional DP-matrix can then be partitioned into equal sized areas by assigning the columns to processor i for $i = 0, \ldots, P-1$. This approach is efficient for wavefront computations with an even workload across matrix cells, i.e. each matrix cell is computed from the same number of other matrix cells. We call such a wavefront computation regular. However, for irregular wavefront computation problems, partitioning into equal-sized areas does not work well, since it leads to an uneven workload.

An irregular dependency pattern occurs in the BIF-state layers: $M(i, j, k)$ is computed from all matrix cells $M(i, mid, k_{left})$ and $M(mid+1, j, k_{right})$ for $mid = i-1$ to j. Thus, the load to compute one element in the matrix increases along the shift direction of the wavefront. We call this the load computation density. Figure 5 shows the change of load computation density in a BIF-state layer along the wavefront shift direction by using increasingly blacking shades.

Equal-area column-based matrix decomposition will therefore lead to a poor performance in BIF-state layers, since the workload of processor P_i is higher than the workload of processor P_{i-1}. In order to achieve columnwise decomposition with even workload in BIF-layers, processor

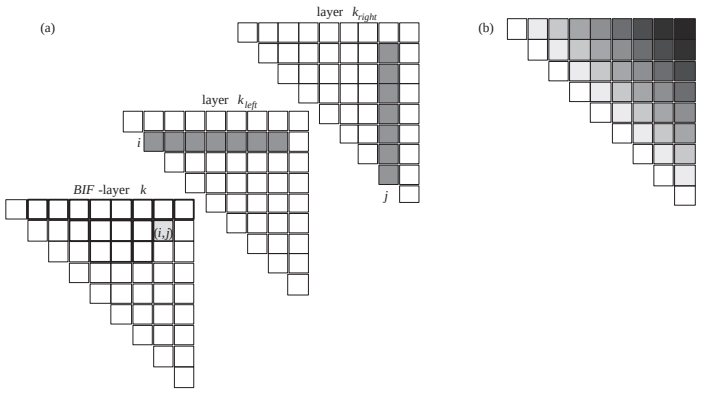

Figure 5. (a) Data dependency in a BIF-layer: Cell $\alpha(i,j,k)$ depends on the cells $\alpha(i,mid,k_{left})$ and $\alpha(mid+1,j,k_{right})$ for all $i-1 \leq mid \leq j$. (b) Load computation density in a BIF-layer increases along the wavefront shift direction.

$i(0 \leq i \leq P-1)$ needs to process the columns . However, changing the number of assigned columns to each processor for BIF-layers and non-BIF-layers during computation of the three-dimensional CYK DP-matrix would increase the communication overhead. Our final partitioning scheme therefore assigns columns $\sqrt{i/P} \cdot L, \ldots, \sqrt{i+1/P} \cdot L$ to processor i, $i = 0, \ldots, P-1$. The parameter h depends on the number B of BIF-states in G and is determined by the formula . The formula assumes that , which is a reasonable assumption for RNA CMs. The following pseudocode shows the details of the parallel CYK algorithm (without traceback) using the described partitioning scheme with wavefront computation.

The implementation of a parallel space-saving CYK traceback algorithm requires the CYK/inside algorithm as well as the CYK/outside algorithm. The CYK/outside algorithm iteravely calculates DP-matrix $B[i,j,k]$. $B[i,j,k]$ is the log-odds score of of the most likely CM parse tree for a CM generating a sequence $x_1 \ldots x_L$ excluding the optimal parse subtree rooted at state k that accounts for the subsequence $x_i \ldots x_j$. The score of the excluded optimal parse tree can be calculated by the CYK/inside algorithm and is stored in $M[i,j,k]$. Therefore, $argmax_{i,j}(M[i,j,k] + B[i,j,k])$ identifies an index (i,j,k) the optimal parse tree passes through for any state k, assuming that k is in the optimal parse. The space-saving CYK traceback algorithm takes advantage of this fact as follows.

Since we are considering global alignment, any parse tree has to include all BIF- and BEG-states. Hence, by choosing any BIF-state b1 with child states $b1_{left}$ and $b1_{right}$ and calculating: $(i,j,k) = argmax_{i',j',k'}(B[i',j',b1] + M[i',k',b1_{left}] + M[k'+1,j',b1_{right}])$, three cells can be identified that must be used in the optimal alignment: $(i,j,b1)$, $(i,k,b1_{left})$, and

Algorithm 1 Parallel CYK/Inside without traceback

Input: An RNA sequence $x = x_1, \ldots x_L$ of length L, a CM subgraph G of length K with states numbered in preorder traversal. $S(k)$ is the type of state k, $C(k)$ is the set of states that k can transit to , $P(k)$ is the set of parents of state k. $j_p, \ldots j_{p+1}-1$ are the columns of matrix M assigned to processor $p \in \{0, \ldots, np-1\}$ using the described partitioning scheme. np is the number of processors.

Output: Optimal global alignment score $M[1,L,1]$ in the matrix M.

```
FOR all processors p ∈ {0,...,np−1} do in parallel
  FOR k := K down to 1 do flag[k] := false; flag[ ] is local to each processor
  FOR k := K down to 1 do {
    IF (p ≠ 0) then : {
      IF (k = left child of a BIF-state) then
        recv (M[∗, 0 ... j_p − 1, k] from p − 1);
        // blocking receive of columns 0 ... j_p − 1 of layer k, if k is left child of a BIF-state
      IF (S(k) ∈ {MR, IR, MP}) then
       for all γ ∈ C(k) do
        IF (!flag[γ]) then {
          recv (M[∗, j_p − 1, γ] from p − 1);
          // blocking receive of column j_p − 1 from all children layers of k from proc p − 1; avoid receiving same data twice
          flag[γ] := true ; } }
    FOR j := j_p to j_{p+1} − 1 do
     FOR i := j + 1 down to 1 do
       calculate cell M[i, j, k] using the formula of Section 3
    IF (p ≠ np − 1) then {
     IF (k = left child of BIF-state) then
       Isend (M[∗, 0 ... j_{p+1} − 1, k] to p + 1);
       // non-blocking send of columns 0 ... j_{p+1} − 1 of layer k to the processor p + 1
      IF( MR or IR or MP ∈ P(K)) then
       Isend (M[∗, j_{p+1} − 1, k] to p + 1);
       // non-blocking send of column j_{p+1} − 1 of layer k to the processor p + 1 } }
  IF (p = np − 1) then return α[1, L, 1];
}END
```

$(k+1, j, b1_{right})$. This splits the remaining problem into three smaller subproblems, which are then solved recursively using divide-and-conquer.

The parallel space-saving algorithm implementation therefore first calculates the layers $b1_{left}$ and $b1_{right}$ of matrix M by executing the parallel CYK/inside algorithm, where the outer k-loop only has to iterate down to $b1_{left}$. Subsequently, a parallel CYK/outside algorithm is performed to calculate layer $b1$ of matrix B. The parallel CYK/outside algorithm is similar to the parallel CYK/inside algorithm with the main difference that outer k-loop now iterates from 1 up to $b1$. We are choosing the splitting-state $b1$ as the first BIF-state of a CM. This simplifies the computation of the CYK/outside algorithm, since it does not contain any internal BIF-states.

Both parallel implementations use deallocation of layers as described in Section 3. After completion of the parallel inside and outside algorithm, $argmax_{i',j',k'}(B[i', j', b1] + M[i', k', b1 + left] + M[k' + 1, j', b1_{right}])$ can be calculated by a BIF-state type of computation followed by a maximum reduction operation in the root processor. The root processor then initiates the parallel calculation of each of the three smaller subproblems. If a remaining subproblem is small enough, the corresponding traceback is computed sequentially within a single processor.

5. Performance Evaluation

We have implemented the presented parallel alignment algorithms using C and MPI and experimentally evaluated its performance on a PC cluster with 10 nodes running Linux. Each node consists of two Intel-Xeon CPUs with a clock rate of 2GHz and 1GByte of RAM. The nodes are connected by a 1GBit/s Myrinet switch. The implementation has been evaluated for CMs of different sizes and sequences of different lengths (see Table 3). Tables 4 and 5 show the corresponding runtimes and speedups on our cluster as the number of processors varies for alignment with and without traceback.

Table 3. The five CMs and RNA sequences used for the runtime evaluation of our algorithm. Models and sequences are chosen from four structural RNA types: SRP (signal recognition particle) RNA, RNase P, SSU rRNA, and LSU rRNA. The CMs have been constructed from annotated secondary structure multiple alignments from RDP [6].

Structural RNA type	SRP RNA	RNase	SSU rRNA1	SSU rRNA2	LSU rRNA
Sequence Length	301	596	1190	1545	2904
CM Length (K)	927	1725	4122	4789	9023
BIF-states	4	7	13	30	65

Table 4. Runtimes in seconds of the parallel CYK inside algorithm without traceback for the five CM-sequence pairs of Table 3. The speedup compared to a single processor is also reported.

#Processors	1	2	4	8	16	20
SRP RNA	3.1	1.9	1.2	1.1	-	-
		(1.6)	(2.6)	(2.9)	-	-
RNase P	7.0	3.8	1.9	1.4	-	-
		(1.8)	(3.6)	(5.1)	-	-
SSU rRNA1	516	303	138	68	39	32
		(1.7)	(3.8)	(7.6)	(13.2)	(16.1)
SSU rRNA2	747	411	195	103	55	46
		(1.8)	(3.8)	(7.2)	(13.6)	(16.2)
LSU rRNA	9936	5363	2620	1410	730	602
		(1.8)	(3.8)	(7.0)	(13.6)	(16.5)

Table 5. Runtimes in seconds of the parallel space-saving CYK algorithm with traceback for the five CM-sequence pairs of Table 3. The speedup compared to a single processor is also reported.

#Processors	1	2	4	8	16	20
SRP RNA	10.1	7.3	3.4	1.94	-	-
		(1.4)	(2.9)	(5.2)	-	-
RNase P	12.5	7.85	4.2	2.4	-	-
		(1.6)	(2.9)	(5.2)	-	-
SSU rRNA1	783	505	243	125	63	54
		(1.5)	(3.2)	(6.2)	(12.4)	(14.2)
SSU rRNA2	1119	746	352	182	95	79
		(1.5)	(3.17)	(6.14)	(11.8)	(14.1)
LSU rRNA	15768	9793	4778	2530	1457	1058
		(1.6)	(3.3)	(6.2)	(12.6)	(14.9)

A multiple sequence alignment (MSA) can be constructed by iteratively aligning each sequence to a trained CM. This technique has been previously applied to build MSAs of relatively short tRNA sequences with high accuracy [9]. Parallelism makes it possible to extend this method to longer sequences. We have used our parallel program with traceback to construct an MSA of 60 Ecoli SSU rRNA sequences and one of 8 Ecoli LSU rRNA sequences. The procedure takes around 1.5 hours on our PC cluster inclusive model training, which would have taken almost one day on a single Pentium 4. The quality of the produced alignments is assessed by comparing it to a trusted alignment from the Ribosomal database (RDB) [6] (see Table 6). Table 6 also shows a comparison to the accuracy of alignments produced by HMMER [1] and ClustalW [21].

Table 6. Comparison of the accuracy of MSAs of 60 Ecoli SSU rRNA sequences and 6 Ecoli LSU rRNA sequences produced by three different programs. Parallel CYK and HMMER start by training a CM and a Hidden Markov Model from a set of 30 SSU rRNA test sequences and 20 LSU rRNA test sequences with a known secondary structure. ClustalW is a progressive mutltiple alignment method and does not require any training. The accuracy is then obtained by comparing the computed MSAs to trusted MSAs of the corresponding sequences from RDB [6].

Program	ClustalW	HMMER	Parallel CYK
SSU rRNA	23.9 %	66.8 %	76.0 %
LSU rRNA	26.8%	65.5%	77.8%

6. Conclusions and Future Work

In this paper, we have presented a parallel algorithm for aligning an RNA sequence to an SCFG using a wavefront parallelization technique. Its implementation on a PC cluster using MPI achieves linear speedup. The proposed method allows the construction of MSAs of long RNA sequences such as SSU rRNA and LSU rRNA in reasonable time and with high accuracy. The current implementation only considers global alignment. Our future work will include extending our parallel algorithm to local and semi-global alignments. Its implementation can then be used for SCFG-based RNA sequence database scanning.

References

[1] http://hmmer.wustl.edu

[2] Altschul, S.F., Gish, W., Miller, W, Myers, E.W., Lipman, D.J. Basic Local Alignment Search Tool, Journal of Molecular Biology 215, 403-410 (1990)

[3] Aluru, S., Fuamura, N., Mehrotra, K. Parallel Biological Sequence Comparison using Prefix Computations, Journal of Parallel and Distributed Computing, 63, 264-272 (2003)

[4] Bateman, A., Birney, E., Cerutti, L., Durbin, R., Etwiller, L., Eddy, S.R., Griffith-Jones, S., Howe, K.L. Marshall, M., Sonnhammer, E. The PFAM Protein Families Database, Nucleic Acids Research 30, 276-280 (2002)

[5] Brown, M.P. Small Subunit Ribosomal RNA Modeling using Stochastic Context-Free Grammars, Proceedings ISMB'00, 57-66 (2000)

[6] Cole, J.R., Chai, B, Marsh, T.L., Farris, R.J., Wang, Q., Kulam, S.A., Chandra, S., McGarrell, D.M., Schmidt, T.M., Garrity, G.M., Tiedje, J.M.. The Ribosomal Database Project (RDP-II): previewing a new autoaligner that allows regular updates and the new prokaryotic taxonomy, Nucleic Acids Research 31, 442-443 (2003)

[7] Durbin, R., Eddy., S.R., Krogh, A., Mitchison, G. Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids, Cambridge University Press (1998)

[8] Eddy, S.R. A Memory-Efficient Dynamic Programming Algorithm for Optimal Alignment of a Sequence to an RNA Secondary Structure, BMC Bioinformatics 3:18 (2002)

[9] Eddy, S.R., Durbin, R. RNA Sequence Analysis using Covariance Models Nucleic Acids Research 22, 2079-2088 (1994)

[10] Edmiston, E.W., Core, N.G., Saltz, J.H., Smith, R.M. Parallel Processing of Biological Sequence Comparison Algorithms, Journal of Parallel Programming 17 259-275 (1988)

[11] Griffiths-Jones, S., Bateman, A., Marshall, M., Khanna A., Eddy S.R. RFAM: an RNA family database, Nucleic Acids Research 31, 439-441 (2003)

[12] Lari, K., Young, S.J. Applications of Stochastic Context-Free Grammars using the Inside-Outside Algorithm, Computer Speech and Languages 5, 237-257 (1991)

[13] Lenhof, H.P., Reinert, K., Vingron, M. A polyhedral approach to RNA sequence structure alignment, Journal of Computational Biology 5, 517-530 (1998)

[14] Lowe, T., Eddy, S.R. tRNAscan-SE: a Program for Improved Detection of Transfer RNA genes in Genomic Sequences, Nucleic Acids Research 25, 955-964 (1997)

[15] Myers, E.W., Miller, W. Optimal Alignments in Linear Space, Computer Applications in the Biosciences 4, 11-17 (1988)

[16] Pearson, W.R., Lipman, D.J. Improved Tools for Biological Sequence Comparison, Proc. Natl. Acad. Sci. 4, 2444-2448 (1988)

[17] Rivas, E., Eddy S.R. Noncoding RNA gene detection using comparative sequence analysis, BMC Bioinformatics 2(1), 8-27 (2001)

[18] Sakakibara, Y., Brown, M., Hughey, R., Mian, I.S., Sjolander, K., Underwood, R.C., Haussler, D. Stochastic Context-Free Grammars for tRNA modeling, Nucleic Acids Research 22, 5112-5120 (1994)

[19] Schmidt, B., Schroder, H., Schimmler, M. Massively Parallel Solutions for Molecular Sequence Analysis, 1st International Workshop on High Performance Computational Biology, in Proc. IPDPS'02 (2002)

[20] Storz, G. An expanding universe of noncoding RNAs, Science 296, 1260-1263 (2002)

[21] Thompson, J.D., Higgins, Gibson, T.J. ClustalW: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position specific gap penalties, and weight matrix choice, Nucleic Acids Research 22, 4673-4680 (1994)