# Foldalign 2.2: multithreaded structural RNA alignment

Daniel Sundfeld[1,3] Jakob Havgaard[1,2] Alba C. M. A. de Melo[3]
Jan Gorodkin[1,2]

[1]Center for non-coding RNA in Technology and Health

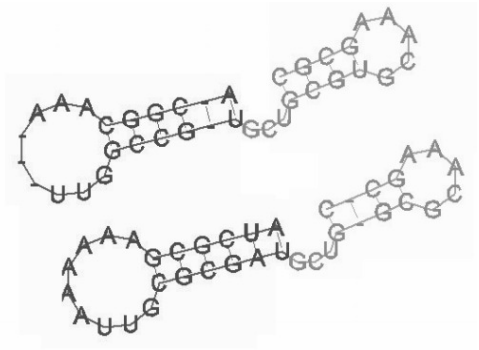
[2]IKVH
University of Copenhagen

[3]Department of Computer Science
University of Brasilia

19/02/2015 - Winterseminar

# Foldalign

```
A-CGGCAAA---UUGGCCG-UGCUGCGUGCAAAGCGC
((((((.........)))))) ...((((.....))))
AUCGCGAAAAAAUUGCGCGAUGCUG-GCGCAAAGC-C
```



*

Algorithms for simultaneously aligning and folding:

- **the probabilistic methods**: usually based on Stochastic Context Free Grammars, that use statistical learning to creates a solution based on previously known answers;
- **Sankoff-like methods**: use some method of minimization of the free energy to calculates how elements of an RNA structure contribute to free energy.

*M. Ziv-Ukelson, I. Gat-Viks, Y. Wexler, and R. Shamir. A faster algorithm for rna co-folding. In Proceedings of the 8th International Workshop on Algorithms in Bioinformatics, pages 174–185, 2008.

# Foldalign

The time complexity $O(L^6)$, where L is the sequence length of the sequences, makes the Sankoff algorithm prohibited for long sequences. Foldalign use several constraints to reduce the time requirement:

- The final alignment is limited to $\lambda$ nucleotides, and the maximum length difference between any two subsequences being aligned is limited to $\delta$ nucleotides;
- Subalignments with score bellow to a threshold are eliminated.

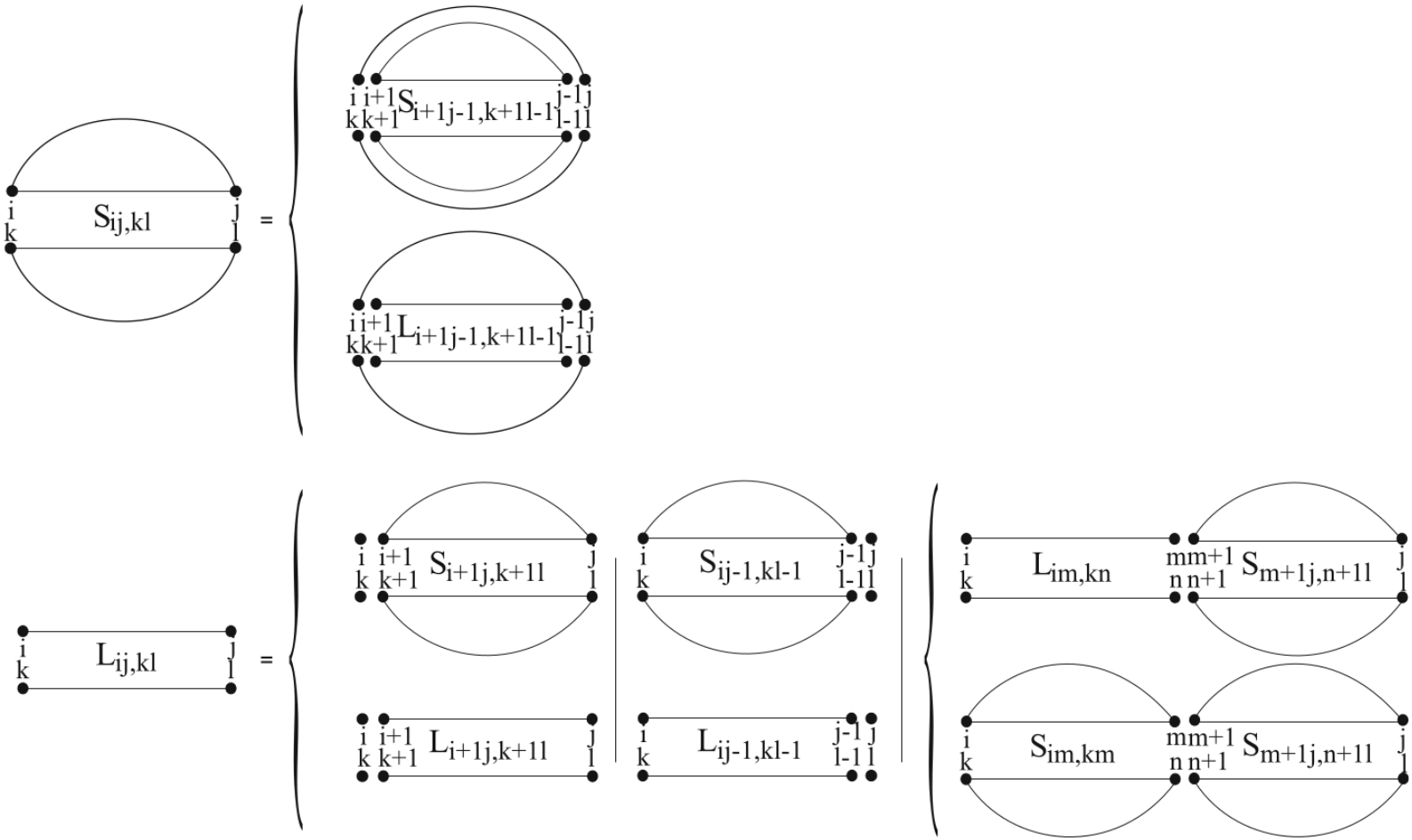
With this constraints, the time complexity is reduced to $O(L^2\lambda^2\delta^2)$

The simplified Foldalign recursion:

$$D_{i,j,k,l} = \max \begin{cases} D_{i+1,j-1,k+1,l-1} + S_{bp}(n_i, n_j, n_k, n_l, \sigma_{bp}) & \text{(a)} \\ D_{i+1,j-1,k,l} + S_{bpil}(n_i, n_j, -, -, \sigma_{bpil}) & \text{(b)} \\ D_{i,j,k+1,l-1} + S_{bpiK}(-, -, n_k, n_l, \sigma_{bpiK}) & \text{(c)} \\ D_{i+1,j,k+1,l} + S_{al}(n_i, n_k, \sigma_{al}) & \text{(d)} \\ D_{i,j-1,k,l-1} + S_{ar}(n_j, n_l, \sigma_{ar}) & \text{(e)} \\ D_{i+1,j,k,l} + S_{gll}(n_i, -, \sigma_{gll}) & \text{(f)} \\ D_{i,j-1,k,l} + S_{glrl}(n_j, -, \sigma_{glrl}) & \text{(g)} \\ D_{i,j,k+1,l} + S_{glK}(-, n_k, \sigma_{glK}) & \text{(h)} \\ D_{i,j,k,l-1} + S_{grK}(-, n_l, \sigma_{grK}) & \text{(i)} \\ \max\limits_{\substack{i<m<j \\ k<n<l}} \{D_{i,m,k,n} + D_{m+1,j,n+1,l} + \sigma_{mbl}\} & \text{(j)} \end{cases} \quad (1)$$

# Foldalign

## Graphical Representantion




J. Gorodkin and W. L. Ruzzo. RNA Sequence, Structure and Function: Computational and Bioinformatic Methods. Humana Press, 2014.

# Foldalign

## Foldalign algorithm.

1: **for** $i = Length_I \rightarrow 1$ **do**
2: **for** $k = Length_K \rightarrow 1$ **do**
3: **for** $j = 1 \rightarrow \lambda$ **do**
4: **for** $l = (j - \delta) \rightarrow (j + \delta)$ **do**
5: **if** $Ms_{i,j,k,l}$ can be the right part of bifurcation **then**
6: $Ml_{i,j,k,l} = Ms_{i,j,k,l}$
7: **end if**
8: $Expand\_Position(i, j, k, l)$
9: **for** $W_m = 1 \rightarrow (\lambda - j)$ **do**
10: **for** $W_n = (W_m - \delta) \rightarrow (W_m + \delta)$ **do**
11: $Expand\_MB(i, j, k, l, W_m, W_n)$
12: **end for**
13: **end for**
14: **end for**
15: **end for**
16: **end for**
17: **end for**

# Parallel Foldalign

## Parallel algorithm.

The multithreading algorithm is based on the fact that cells in the $i$ loop might be calculated in parallel. Some data dependency exist inside the loop, conditions are added to avoid race conditions.

$N_{threads}$ are created, each one have different $i$ values:

Thread 1:

1: **for** $i = L_I \rightarrow 1$ **step** $N_{threads}$ **do**
2: **for** $k = L_K \rightarrow 1$ **do**
3: (...)

16: **end for**
17: **end for**

Thread 2: (...)

1: **for** $i = (L_I - 1) \rightarrow 1$ **step** $N_{threads}$ **do**
2: **for** $k = L_K \rightarrow 1$ **do**
3: (...)

16: **end for**
17: **end for**

# Foldalign

## Execution sample.

Single Thread:

```
1:  for i = 20 do
2:      for k = 20 do
3:          (...)

16:     end for
17: end for
```

# Foldalign

## Execution sample.

Single Thread:

```
1:  for i = 20 do
2:      for k = 19 do
3:          (...)

16:     end for
17: end for
```

# Foldalign

## Execution sample.

Single Thread:

```
1:  for i = 20 do
2:      for k = ... do
3:          (...)

16:     end for
17: end for
```

# Foldalign

## Execution sample.

Single Thread:

```
1:  for i = 20 do
2:      for k = 1 do
3:          (...)

16:     end for
17: end for
```

# Foldalign

## Execution sample.

Single Thread:

```
1:  for i = 19 do
2:      for k = 20 do
3:          (...)

16:     end for
17: end for
```

# Foldalign

## Execution sample.

Single Thread:

```
1:  for i = 19 do
2:      for k = 19 do
3:          (...)

16:     end for
17: end for
```

# Foldalign

## Execution sample.

Thread 1:

```
 1: for i = 20 step 2 do
 2:     for k = 20 do
 3:         (...)

16:     end for
17: end for
```

Thread 2:

```
 1: for i = 19 step 2 do
 2:     for k = waiting do
 3:         (...)

16:     end for
17: end for
```

# Foldalign

## Execution sample.

Thread 1:

```
 1: for i = 20 step 2 do
 2:     for k = 19 do
 3:         (...)

16:     end for
17: end for
```

Thread 2:

```
 1: for i = 19 step 2 do
 2:     for k = waiting do
 3:         (...)

16:     end for
17: end for
```

# Foldalign

## Execution sample.

Thread 1:

```
 1: for i = 20 step 2 do
 2:     for k = 18 do
 3:         (...)

16:     end for
17: end for
```

Thread 2:

```
 1: for i = 19 step 2 do
 2:     for k = 20 do
 3:         (...)

16:     end for
17: end for
```

# Foldalign

## Execution sample.

Thread 1:

```
 1: for i = 20 step 2 do
 2:     for k = (...) do
 3:         (...)

16:     end for
17: end for
```

Thread 2:

```
 1: for i = 19 step 2 do
 2:     for k = (...) do
 3:         (...)

16:     end for
17: end for
```

# Foldalign

## Execution sample.

Thread 1:

```
 1: for i = 18 step 2 do
 2:     for k = 20 do
 3:         (...)

16:     end for
17: end for
```

Thread 2:

```
 1: for i = 17 step 2 do
 2:     for k = waiting do
 3:         (...)

16:     end for
17: end for
```

# Foldalign

## Execution sample.

Thread 1:

```
 1: for i = 18 step 2 do
 2:     for k = 19 do
 3:         (...)

16:     end for
17: end for
```

Thread 2:

```
 1: for i = 17 step 2 do
 2:     for k = waiting do
 3:         (...)

16:     end for
17: end for
```

# Foldalign

## Execution sample.

Thread 1:

```
 1: for i = 18 step 2 do
 2:     for k = 18 do
 3:         (...)

16:     end for
17: end for
```

Thread 2:

```
 1: for i = 17 step 2 do
 2:     for k = 20 do
 3:         (...)

16:     end for
17: end for
```

# Parallel Foldalign

## Parallel algorithm.

Other mechanisms are necessary to the final implementation:

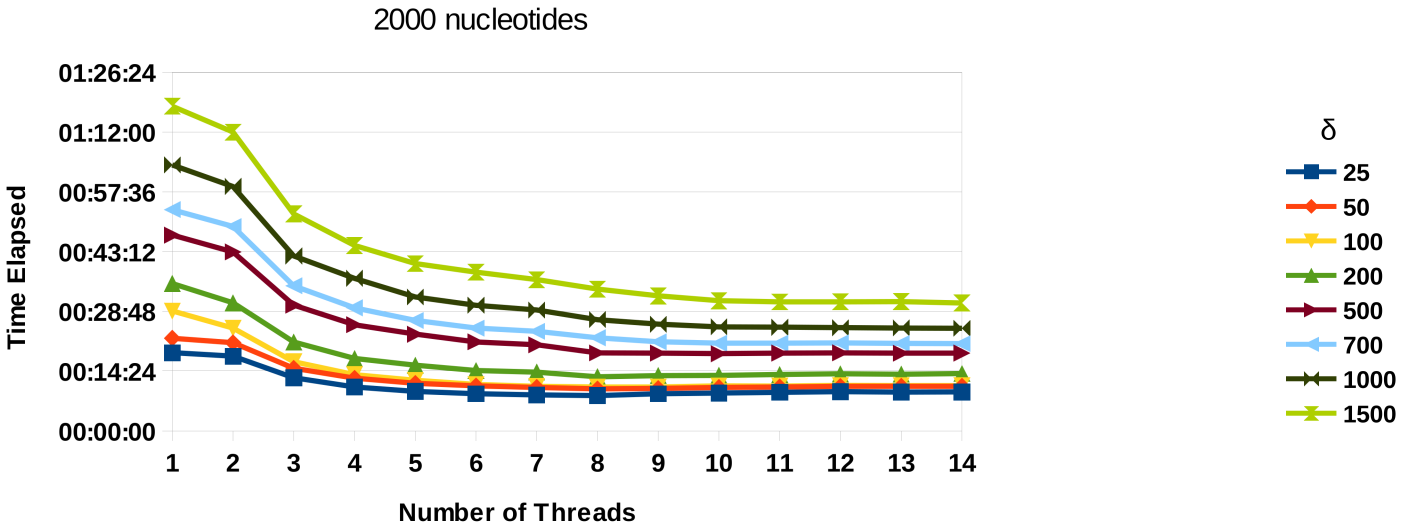
- **Threadpoll:** To avoid thread creation overhead, a fixed number of threads are created and each thread wait for a $i$ value.
- **Protection:** The global memory need protection mechanisms to avoid simultaneous writes.
- and other implementation details...

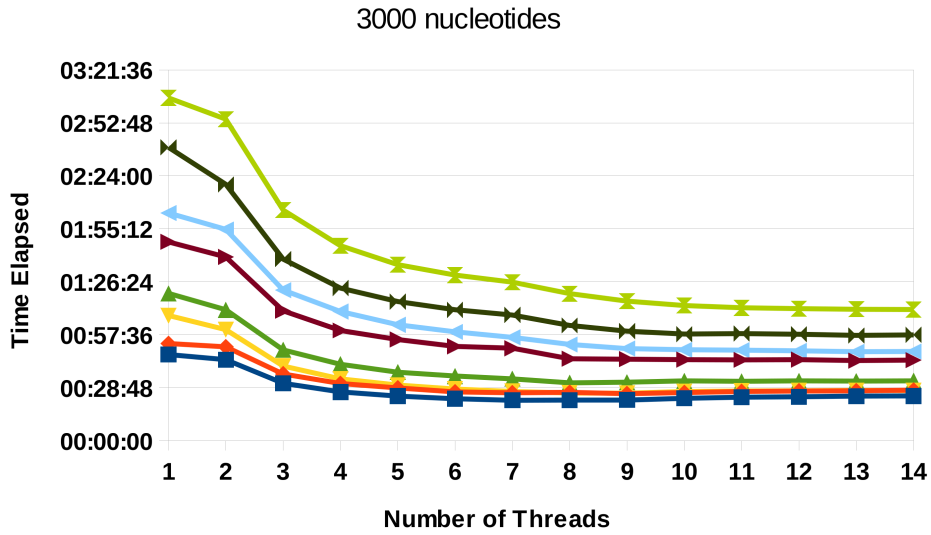
# Results

## Multithreaded Programming: Expectation vs. Reality

# Results

## Random Sequences Performance

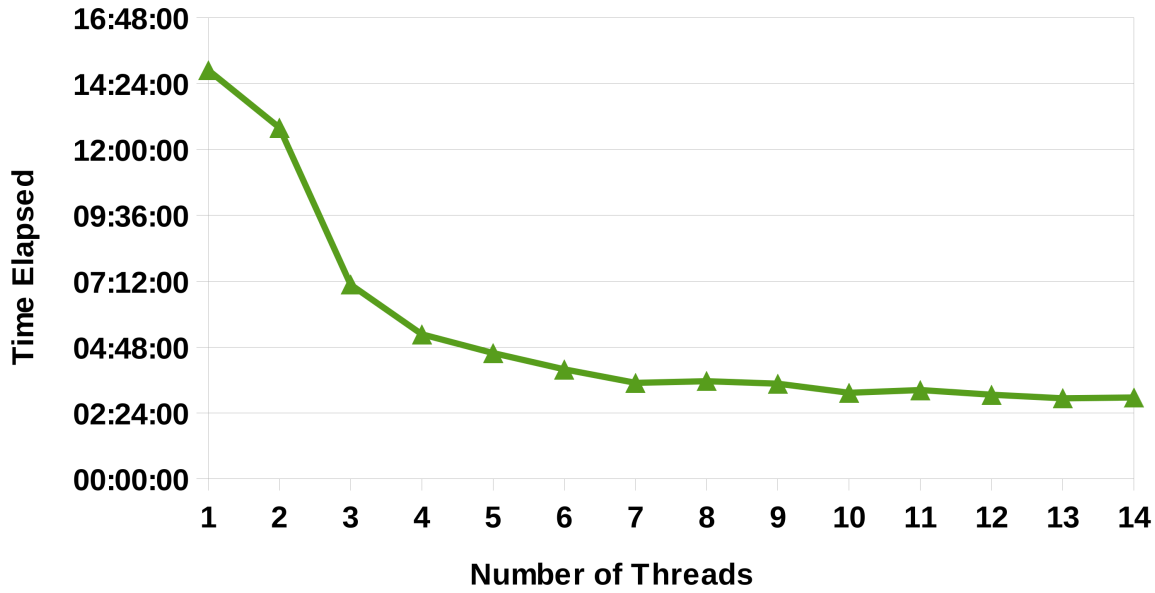


# Results

## Real Sequences Performance

16S Ribosomal RNA from the Gutell website http://www.rna.icmb.utexas.edu/
($|S_1| = 2315, |S_2| = 2019, \lambda = 2019, \delta = 100$).



# Conclusion

A multithreaded version of Foldalign is now available.
The implementation were carefully designed to keep all previous functions, and it open new possibilities to search for structured RNAs in much longer sequences in reasonable time, and in our experiments, it was possible to search 2.5-2.7 times faster with random sequences, and 4.18 times faster with 8 cores, 5.03 times faster with 14 cores.

# Acknowledgements

RTH
CENTER FOR NON-CODING RNA
IN TECHNOLOGY AND HEALTH


Thank you!
Questions?