Bioinformatics, 32(8), 2016, 1238–1240
doi: 10.1093/bioinformatics/btv748
Advance Access Publication Date: 24 December 2015
Applications Note

Structural bioinformatics

Foldalign 2.5: multithreaded implementation for pairwise structural RNA alignment

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

[1]Center for Non-Coding RNA in Technology and Health, IKVH, University of Copenhagen, Frederiksberg, Denmark and [2]Department of Computer Science, University of Brasilia, Brasília, DF, Brazil

*To whom correspondence should be addressed.

Associate Editor: Ivo Hofacker

Received on 23 July 2015; revised on 14 December 2015; accepted on 16 December 2015

Abstract

Motivation: Structured RNAs can be hard to search for as they often are not well conserved in their primary structure and are local in their genomic or transcriptomic context. Thus, the need for tools which in particular can make local structural alignments of RNAs is only increasing.

Results: To meet the demand for both large-scale screens and hands on analysis through web servers, we present a new multithreaded version of Foldalign. We substantially improve execution time while maintaining all previous functionalities, including carrying out local structural alignments of sequences with low similarity. Furthermore, the improvements allow for comparing longer RNAs and increasing the sequence length. For example, lengths in the range 2000–6000 nucleotides improve execution up to a factor of five.

Availability and implementation: The Foldalign software and the web server are available at http://rth.dk/resources/foldalign

Contact: gorodkin@rth.dk

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Recent research points towards an increasing awareness of structured RNAs in genomic and transcriptomic sequences (Gorodkin *et al.*, 2010; Westhof and Romby, 2010). However, the tools needed for structural analysis, such as pairwise local structural RNA alignments, are not yet fully developed.

Foldalign (Havgaard *et al.*, 2007) is a tool that explicitly carries out local pairwise structural alignment of RNA sequences based on the Sankoff algorithm (Sankoff, 1985). Even though tools like CMfinder can carry out local RNA structure alignment on multiple sequences (Yao *et al.*, 2006), the pairwise problem is still of key interest. A range of other methods for RNA structural alignments focusing more on the global alignment (Dowell and Eddy, 2006; Knudsen and Hein, 2003) have been proposed, but only a few efforts were made to parallelize these methods (Fu *et al.*, 2014; Sukosd *et al.*, 2011).

The relevancy of a parallel version of the pairwise Sankoff algorithm is underpinned by its time complexity of $O(L^6)$, where L is the sequence length. This makes it prohibitive for long sequences, but Foldalign uses several heuristics: a maximum length of the alignment, λ, and a maximum difference, δ between any two subsequences being aligned. This reduces the time complexity to $O(L^2\lambda^2\delta^2)$. However, runtime and memory is further substantially improved using several other heuristics like limiting the multiloop calculation and pruning of the alignment score, for details see Havgaard *et al.* (2007). All these heuristics can be used in a parallel version of the algorithm, which we address here, providing new opportunities for both large-scale analysis as well as case-based analyses through a web interface.

2 Implementation and results

The core Foldalign algorithm has six nested loops (Supplementary Section 1) which are subject for parallelization. The first and second loops have the values $i = L_1, L_1 - 1, \dots, 1$ and $k = L_2, L_2 - 1, \dots 1$, where L_1 and L_2 are the lengths of the sequences S_1 and S_2. The current multithreading model is based on the fact that many of the cells can be calculated in parallel. Foldalign divides the dynamic programming matrix in long term memory (LTM), for cells that can only be part of a multibranch loop rule and short term memory (STM), for others. While processing (i, k), the algorithm only writes to STM cells with coordinates (i, k), $(i-1, k)$, $(i, k-1)$, $(i-1, k-1)$ and LTM cells with coordinates (i, k).

In the multithreaded version, $t = 1, 2, \dots, N$ threads are created, and each thread works on its own value of i. Every thread sequentially calculates the cells $(i_t, L_2) \rightarrow (i_t, 1)$. When a thread finishes calculating all the i_t cells, it restarts with a new $i_t = i_t - N$ value. Figure 1a shows an example of the STM with one and two threads, respectively. When using multiple threads, a lock is required to avoid race condition, in order to guarantee that position (i, k) is processed only if position $(i+1, k-1)$ has been processed, see Supplementary Section 2. Simultaneous operations at long-term memory are protected by one lock per (i, k). Once a thread finishes, cells that can be part of the multibranch loop are kept in long-term memory, and cells in short term memory are released.

The proposed parallel algorithm was implemented in C++, using POSIX threads (pthreads). The benchmarking was done on a machine with two Intel Xeon E5-2650 processor, each one with 8 cores at 2.00 GHz and 32 GB RAM.

We measured the execution time and memory consumption on randomly generated sequences with varying GC-content and lengths from 2000 nt to 6000 nt. GC-content was fixed in bins in the range from 20% to 60% and G to C and A to U ratios were set to one. In Figure 1b, the average elapsed time and memory consumption are shown for 5 random sequences with 6000 nucleotides, $\delta = 25$ and $\lambda = 1000$. Using 8 threads, the elapsed time is reduced from 4:44 h to 57.1 min (4.98× faster). Using 2 or 4 threads, the reduction is to 2:24 h (1.97× faster) and 1:11 h (3.95× faster). This result is consistent with other lengths (2000–6000), λ values (1000–1500) and GC-Content (20–60%), see Supplementary Sections 3 and 4.

Increasing the number of threads does not considerably increase the memory consumption, because most resources are shared by the threads. This is exemplified in Figure 1b, where 2 threads require only 1.14 times more memory and 8 threads require 1.99 more. We also ran a test with 16 threads, which did not yield any improvement over 8 threads. This is in line with other efforts on parallelizing RNA folding algorithms, see Supplementary Section 3.1. The speedups are also consistent with other sets of sequences with different lengths. See Supplementary Sections 1–3 for tests with other random and real sequence sets, more details about the thread design pattern and synchronization, some optimizations included in our solution, and the simplified recursion function. Due to major changes in the code, we re-executed the tests made in previous versions to evaluate the performance, using datasets with Rfam sequences. In this test, we verified that the localization performance (Havgaard *et al.*, 2007) has not changed in the new version.

Furthermore, the locateHits tool for post-processing Foldalign screens has been reimplemented using C++ and now includes a set of pre-calculated parameters for the P value calculation making it substantially faster than the previous Perl implementation. The scores from the random sequences were used to calculate the extreme value distribution parameters of the Foldalign scores for the sequences in the 20–60% GC-content range (Supplementary Section 4). Previous versions of Foldalign used an interactive method which is statistically unsound or requires the user to run several alignments with random sequences. The webserver has also been updated. The multithreading and new hardware makes it possible for the server to structurally align longer sequences, up to 10 000 nt, with a maximum alignment length of 1000 nt (see Section 5 in the Supplementary Material).

3 Conclusion

We presented a multithreaded version of Foldalign. This new version was carefully designed to keep all the previous program functionalities, such as the dynamical pruning heuristic, limiting the calculation of branch points, and the option to perform local and global alignments. It opens up new possibilities to search for structured RNAs in much longer sequences in reasonable time. One great advantage of the multithreaded version is that it allows for better exploitation of the available hardware when the number of jobs which can run on a given machine is smaller than the number of cores available in the machine due to memory constraints. With the previous version of the algorithm it is necessary to let the extra cores run idle, but now they can be put to use. Foldalign is producing predictions with the same accuracy but in a fraction of the time compared to the previous version. It may produce more accurate predictions than it was previously

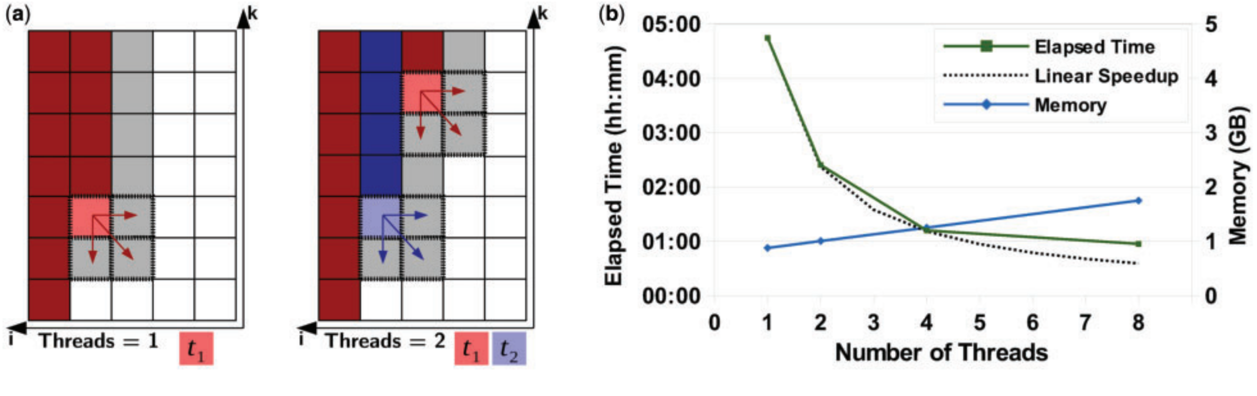

Fig. 1. (**a**) Parallel design example of two sequences. Every cell corresponds to a bidimensional matrix. Red and blue are cells processed by threads t_1 and t_2, respectively. Dark red/blue are cells that have already been processed, light red/blue are cells being processed and white or grey are cells to be processed next. The dashed area represents cells that are being read and written by one thread. (**b**) The Foldalign execution time and memory consumption according to the number of threads. This set contains 5 random sequences with length 6000, GC-content from 40% to 50%, $\delta = 25$ and $\lambda = 1000$. The linear speed up is the ideal speedup, when n threads are used and Foldalign is executed n times faster. With 8 threads, the elapsed time is reduced from 4:44 h to 57.1 min (4.98× faster), while consuming 1.99× more memory

possible, with the much more relaxed constraint parameters δ and λ values. The perspective for this new tool is to contribute significantly to further analysis of structured RNAs in long sequences including specific case analysis through the web server.

Funding

This study was supported by the Innovationfund Denmark, the Danish Research Council for independent research (FTP), the Lundbeck foundation, the Danish Center for Scientific Computing (DCSC, DeiC), and CAPES Foundation, Ministry of Education of Brazil, Brasília - DF (99999.005168/2014-07).

Conflict of Interest: none declared.

References

Dowell,R.D. and Eddy,S.R. (2006) Efficient pairwise RNA structure prediction and alignment using sequence alignment constraints. *BMC Bioinformatics*, **7**, 400.

Fu,Y. *et al.* (2014) Dynalign II: common secondary structure prediction for RNA homologs with domain insertions. *Nucleic Acids Res.*, **42**, 13939–13948.

Gorodkin,J. *et al.* (2010) De novo prediction of structured RNAs from genomic sequences. *Trends Biotechnol.*, **28**, 9–19.

Havgaard,J.H. *et al.* (2007) Fast pairwise structural RNA alignments by pruning of the dynamical programming matrix. *PLoS Comput. Biol.*, **3**, 1896–1908.

Knudsen,B. and Hein,J. (2003) Pfold: RNA secondary structure prediction using stochastic context-free grammars. *Nucleic Acids Res.*, **31**, 3423–3428.

Sankoff,D. (1985) Simultaneous solution of the RNA folding, alignment and protosequence problems. *SIAM J. Appl. Math.*, **45**, 810–825.

Sukosd,Z. *et al.* (2011) Multithreaded comparative RNA secondary structure prediction using stochastic context-free grammars. *BMC Bioinformatics*, **12**, 103.

Westhof,E. and Romby,P. (2010) The RNA structurome: high-throughput probing. *Nat. Methods*, **7**, 965–967.

Yao,Z. *et al.* (2006) CMfinder–a covariance model based RNA motif finding algorithm. *Bioinformatics*, **22**, 445–452.