Parallel construction of RNA databases for extensive lncRNA–RNA interaction prediction

Iñaki Amatria-Barral
CITIC, Universidade da Coruña
A Coruña, Spain
i.amatria@udc.es

Jorge González-Domínguez
CITIC, Universidade da Coruña
A Coruña, Spain
jgonzalezd@udc.es

Juan Touriño
CITIC, Universidade da Coruña
A Coruña, Spain
juan.tourino@udc.es

ABSTRACT

Long non-coding RNA sequences (lncRNAs) have completely changed how scientists approach genetics. While some believe that many lncRNAs are results of spurious transcriptions, recent evidence suggests that there exist thousands of them and that they have functions and regulate key biological processes. For the experimental characterization of lncRNAs, many tools that try to predict their interactions with other RNAs have been developed. Some of the fastest and more accurate tools, however, require a slow database construction step prior to the identification of interaction partners for each lncRNA. This paper presents a novel and efficient parallel database construction procedure. Benchmarking results on a 16-node multicore cluster show that our parallel algorithm can build databases up to 318 times faster than other tools in the market using just 256 CPU cores. All the code developed in this work is available to download at GitHub under the MIT License (https://github.com/UDC-GAC/pRIblast).

CCS CONCEPTS

• **Applied computing** → **Bioinformatics**; • **Computing methodologies** → **Massively parallel and high-performance simulations**;

KEYWORDS

Bioinformatics, lncRNA–RNA, High-Performance Computing, Parallel Computing, MPI, OpenMP

ACM Reference Format:
Iñaki Amatria-Barral, Jorge González-Domínguez, and Juan Touriño. 2023. Parallel construction of RNA databases for extensive lncRNA–RNA interaction prediction. In *The 38th ACM/SIGAPP Symposium on Applied Computing (SAC '23), March 27-April 2, 2023, Tallinn, Estonia.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3555776.3577772

1 INTRODUCTION

Long non-coding RNAs (lncRNAs) are classically defined as transcripts longer than 200 nucleotides without any protein-coding capacity. lncRNAs play integral roles in several biological processes. Indeed, lncRNA dysfunction is associated with many severe diseases, such as Parkinson's [5] and several types of cancer [4, 12].

The functional characterization of lncRNAs is key in order to trace the genetic origin of such diseases. Because lncRNAs work by being assembled with other proteins or RNAs [7], the computational identification of RNA interacting partners for each lncRNA has proved to be a powerful approach for inferring their functions [10, 13]. Furthermore, according to a recent review [3], tools combining information from local alignments and RNA secondary structure, such as RIblast [6] or ASSA [2], obtain the most accurate lncRNA–RNA interacting pairs. Nevertheless, as these algorithms implement the most sophisticated prediction models, their computational costs grow exponentially and prevent their application to large-scale RNA datasets.

To overcome this obstacle, we developed pRIblast [1], which is a parallel and highly efficient tool for the comprehensive prediction of lncRNA–RNA interactions on multicore clusters. pRIblast is based on RIblast [6], a highly accurate and widely used tool developed for such purpose [9, 11]. That is, by definition, pRIblast and RIblast are exact algorithms and they obtain the same level of prediction accuracy. However, pRIblast dramatically accelerates the prediction procedure and completes the lncRNA–RNA analyses up to 128 times faster on a 256-core cluster (i.e., pRIblast can complete a three-month-long analysis in just a few hours). Moreover, pRIblast can process huge datasets that would run out of memory with the former application. Still, pRIblast requires the same database construction step as RIblast prior to the lncRNA–RNA interaction prediction per se. So, the construction of such database has now become a bottleneck. Although it is possible to run the lncRNA–RNA search step in parallel and benefit from lower prediction times, the prediction process is now stuck with a sequential and costly process of building an RNA database for later use in the lncRNA–RNA pair search step.

As a consequence, this work presents a parallel algorithm for the construction of RNA databases to be used alongside pRIblast to predict extensive lncRNA–RNA interactions at unrivalled speeds. The new algorithm uses *de facto* standard High-Performance Computing technologies to exploit cluster and supercomputing architectures: Message Passing Interface (MPI) and OpenMP. As it will be shown in Section 3, our novel approach is able to achieve superlinear speedups (318x) when running over a real and representative RNA dataset on a 16-node multicore cluster (256 CPU cores in total).

2 PARALLEL IMPLEMENTATION

In order to construct a database, pRIblast starts by calculating approximate accessible energies for each sequence in the target RNA dataset. These energies are later used in the RNA interaction search step to find promising lncRNA–RNA pairs. Second, the target RNA sequences are reversed and concatenated (i.e., encoded). Third, the

SAC'23, March 27–April 2, 2023, Tallinn, Estonia

ACM ISBN 978-1-4503-9517-5/23/03.
https://doi.org/10.1145/3555776.3577772

encoded sequences are used to build a suffix array and search results of short substrings are exhaustively precalculated (again, these results, alongside the encoded sequences and the suffix array, are later used to find RNA interactions). And fourth, the approximate energies, the encoded sequences, the suffix array, and the search results are stored in a set of binary and text files that comprise the resulting database.

The calculation of the accessible energies and the encoding of the RNA sequences, which are the most computationally demanding steps behind the construction of the RNA databases in pRIblast (they account for more than 99.5% of the runtime), are independent procedures and may be run in parallel. The construction of the suffix array and subsequent search of short substrings, however, is a strictly sequential process that cannot be parallelized. Therefore, we proposed the following MPI + OpenMP algorithm to build RNA databases in parallel:

(1) The set of RNA sequences is distributed among several MPI processes.
(2) Every MPI process spawns OpenMP threads and runs the calculation of the accessible energies and the encoding procedure in parallel.
(3) Once all the sequences have been encoded and their energies have been calculated, one process gathers all the encoded data and computes the suffix array and search of short substrings.
(4) All processes save the results they computed to construct the resulting RNA database.

We developed three distinct data decomposition approaches to assign sequences to MPI processes: the pure block procedure, the heap decomposition, and the dynamic distribution of sequences. The pure block procedure assigns sequences to processes dividing the input dataset into chunks with the same number of RNAs. However, because input sequences are different in length and the computational complexity of the encoding and energy procedures are functions of the length of the sequences, this procedure fails to balance the workload among the processes (i.e., one process may receive 10 sequences that sum 100 units of work, while another one may receive 10 sequences totalling only 20 units of work). To overcome this scenario, we developed the heap decomposition, which takes into account the fact that larger RNAs imply more computation to evenly distribute the sequences with the same length among all the MPI processes. Hence, making it so that all processes will ideally compute the same units of work. Figure 1 exemplifies this scenario, showing how the heap distribution evenly balances the workload when two processes (P_0 and P_1) are used to compute a set of sequences L (each element represents the length of a sequence) and we assume one unit of work t per character in the sequences. Furthermore, we developed one last scattering approach: the dynamic distribution of sequences. It assigns sequences to processes at runtime to minimize the penalty in execution time that arises when two sequences equal in length produce different amounts of work.

Once sequences have been distributed among the MPI processes, each one spawns OpenMP threads and runs the encoding and energy procedures in parallel. To optimize the execution time of the threads, we developed the sorting heuristic. It sorts the sequences

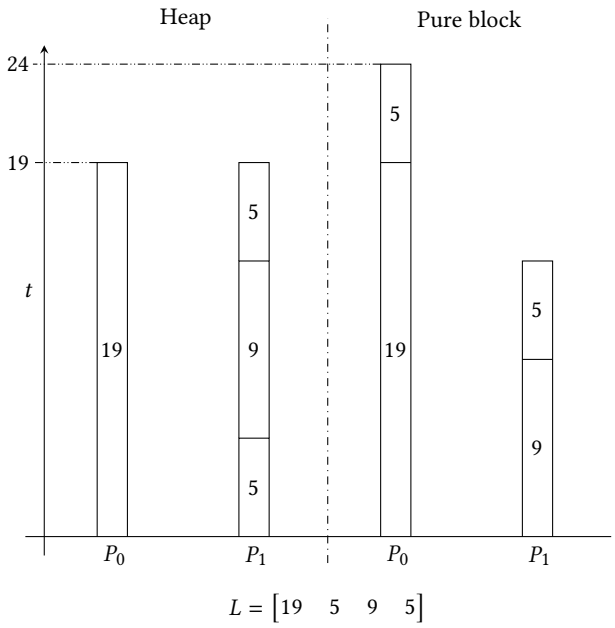

Figure 1: Comparison between the heap distribution of sequences (left) and the pure block decomposition

assigned to each process in descending order of their length (i.e., longer sequences first), so that some threads can process the RNAs that are expected to account for most of workload first, while others can process a large number of smaller RNAs. Figure 2 shows how the sorting heuristic achieves better performance when two threads (T_0 and T_1) process a chunk of sequences L (again, L shows the length of the sequences and we assume that work is proportional to the length of the RNAs). In a real scenario, the heuristic alone improved significantly the processing of the largest dataset featured in Section 3 (Homo), increasing the speedup in about 100 points (from 61.19 to 167.48) using 256 cores.

After accessible energies are calculated and sequences are encoded, one process gathers all the encoded data, creates the suffix array and computes the exhaustive search of short substrings. These steps are strictly sequential, and because we want to avoid losing performance waiting for this process to finish computing, we developed an efficient I/O procedure that overlaps computation with disk operations. This procedure works as follows: (1) the process that gathered all the encoded data builds the suffix array and computes the search of short substrings, and then saves all these data to disk; (2) meanwhile, the rest of the processes save their computed energies and encoded sequences one by one into two output files in a pipelined fashion; and (3) once the process responsible for constructing the suffix array finishes its operations, it will save its energies and encoded sequences right away, without stalling its execution, because all other processes will have already finished writing their data. Put in other words, we developed a zero-cost I/O approach, where while one process computes a strictly sequential procedure, all other processes save their results to disk, avoiding stalling execution and therefore not hurting performance.

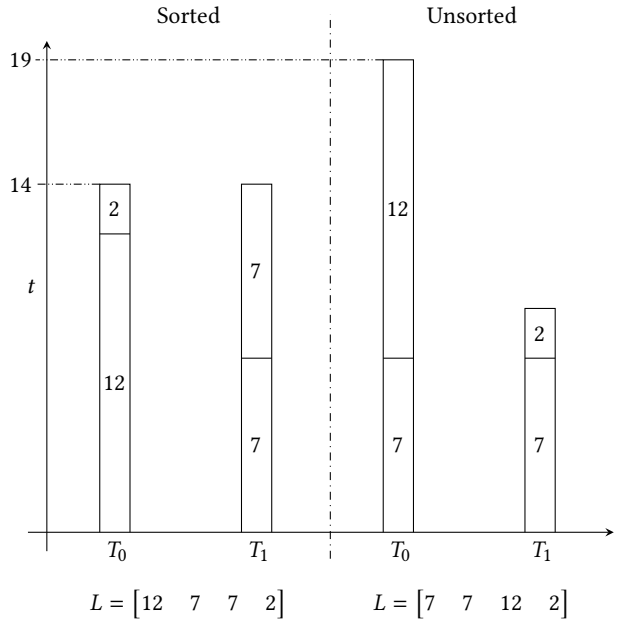

Figure 2: Example to illustrate that the sorting heuristic (left) achieves optimal runtimes

Table 1: Characteristics of the datasets used to evaluate the performance of the parallel database construction algorithm

Dataset	RNAs	Input size	Sequential time	Output size
Ursus	3,899	2.0MiB	837s	17.0MiB
Droso	3,963	4.0MiB	2,497s	39.4MiB
Anas	15,061	18.0MiB	11,682s	195.0MiB
Homo	58,028	76.0MiB	58,562s	839.4MiB

3 RESULTS

To assess the performance of our parallel database construction algorithm, an extensive benchmarking was performed using four real and representative RNA datasets available to download at the Ensembl genome browser [8]. This section presents a performance comparison between the original RNA database construction procedure included in RIblast and our novel parallel algorithm using a 16-node multicore cluster with 256 CPU cores.

Each node in the cluster has two octa-core Intel Xeon E5-2660 Sandy Bridge-EP processors (a total of 16 cores per node with support for HyperThreading) and 64GiB of main memory (DDR3 @ 1600MHz). The nodes are interconnected through a low-latency (1-2μs) and high-bandwidth (56Gbps) InfiniBand FDR network. Regarding the software stack, the GNU Compiler Collection (GCC) v8.3.0 (OpenMP specification 4.5) and OpenMPI v3.1.4 (MPI-3 standard) were the tools used to compile and execute the benchmarks.

Table 1 outlines the characteristics of the RNA datasets: number of RNA sequences, input file size, runtime of the sequential database construction algorithm, and output database size. The datasets were carefully selected from different species, and it is worth to remark that:

Table 2: Runtimes of the novel parallel database construction algorithm using the 16 nodes of the cluster (256 cores) and the best configuration of data distribution, number of threads, and use of HyperThreading (HT)

Dataset	Parallel time	Speedup*	Data distribution	HT
Ursus	5.0s	167.4 (182.6)	Heap (1p16t)	No
Droso	30.4s	82.1 (86.1)	Heap (1p16t)	No
Anas	36.7s	318.3 (256.0)	Dynamic (16p1t)	Yes
Homo	317.5s	184.4 (197.5)	Heap (1p16t)	No

* Maximum theoretical speedup in parentheses

- Two input files with a similar number of sequences (Ursus and Droso) produce different amounts of workload.
- All datasets, except for Anas, are limited in scalability because they contain a sequence so large in comparison to the others that it dominates the parallel runtime and limits the maximum speedup achievable.

These characteristics allowed us to evaluate comprehensively the scalability of the novel parallel algorithm on scenarios that force it to efficiently distribute the workload.

3.1 Methodology

Each dataset was run using 1, 2, 4, 8, and 16 nodes, accounting for 16, 32, 64, 128, and 256 CPU cores, respectively. In addition, the time spent computing every sequence of each dataset was also measured, meaning that it was possible to compare the behavior of our parallel algorithm to that of a theoretical best (i.e., using an ideal data distribution scheme that works as a perfect scheduler).

Two different configurations of threads per process were also tested for each node setup. These configurations are: 1p16t (i.e., 1 process per node spawning 16 threads each), and 16p1t. This allowed us to conclude: (1) which arrangement of threads per process better suits each data decomposition algorithm, and (2) whether the sorting heuristic makes a difference at the time of computing real datasets.

Finally, it was also tested if the use of HyperThreading yielded any improvement in execution time for any of the configurations described above. HyperThreading allows to schedule several threads into one single CPU core to minimize stall cycles, and thus improve utilization and performance. So, 240 experiments were conducted in total and they were repeated three times to take the median value as the final execution time.

3.2 Experimental results

The experimental results for 16 nodes are summarized in Table 2, which points out the important improvements in execution time when all the cores of the cluster are used to run our parallel database construction algorithm. The times shown correspond to the ones using the best configuration of data distribution scheme, number of threads per process, and use of HyperThreading.

The main conclusions obtained from all the results are:

- The workload is not evenly balanced among the sequences of the datasets (except for Anas), so the scalability of the algorithm is bounded by the execution time of one single

sequence (the longest one) for Ursus, Droso, and Homo. Therefore, because there exists a hard limit to the maximum speedup imposed by these datasets, the important bit is to efficiently use the resources available to achieve the best possible speedup using the minimum hardware. In this sense, the more sophisticated data decomposition approaches developed in this work (i.e., heap-based and dynamic) obtain the best results and are always on par with the theoretical best.

- The heap-based approach and the dynamic decomposition optimally distribute the workload and obtain the best speedups. Indeed, they achieved superlinear speedups for Anas due to the use of the HyperThreading. In contrast, the workload distribution performed by the pure block approach is not efficient, and so it ended being up to two times slower than the other approaches when processing the Ursus dataset (note that it does not show up in Table 2).
- We assert the importance of the sorting heuristic at the time of optimizing the performance of the three distributions. This is specially true for the pure block distribution, where the use of the heuristic significantly impacts the speedup. For instance, processing the Homo dataset differs in about 100 points of speedup for 256 cores (600 seconds in execution time) between using the heuristic or not.
- HyperThreading improves performance in all the scenarios where the execution time is not bounded by one sole sequence. In fact, in these scenarios, the executions with HyperThreading enabled the heap and dynamic distributions to outperform the theoretical best speedup. However, if scalability is bounded, HyperThreading may slightly hurt performance.

Furthermore, this extensive benchmarking allowed us to conclude when and how to use each distribution scheme:

- The pure block distribution serves no purpose in comparison to the heap and dynamic decompositions. Its usage should be limited to benchmarking.
- The heap and dynamic distributions obtain similar results, and their usage should depend on the hardware available to execute the parallel database construction algorithm. So, use the heap distribution if the number of cores per computing node is high to fully exploit the sorting heuristic by spawning one process per node and as many threads as possible. If nodes have not that many cores, use the dynamic distribution instead.
- The use of HyperThreading, in general, is always beneficial, specially if the number of resources is low.

4 CONCLUSIONS

As recent studies have shown that lncRNAs play a key role in severe diseases such as Parkinson's or cancer, the computational identification of lncRNA–RNA interacting pairs has become a hot research topic. As a matter of fact, many applications have been developed with this very purpose. However, some of the fastest and more accurate tools in the market (i.e., pRIblast and RIblast) require a slow database construction step before being able to predict any interaction.

In order to solve this issue, this work presented a novel parallel approach to build RNA databases for the subsequent prediction of lncRNA–RNA interacting pairs. The new algorithm can exploit supercomputing and multicore cluster architectures, and so if it is used in conjunction with pRIblast, they comprise a comprehensive parallel pipeline to predict lncRNA–RNA interacting pairs at unrivalled speeds. Indeed, the results presented in this paper show that it is now possible to build RNA databases up to 318 times faster than with state-of-the-art tools.

All the code developed in this work has been merged into the pRIblast tool and released under the MIT License at GitHub (https://github.com/UDC-GAC/pRIblast).

ACKNOWLEDGMENTS

This work was supported by the Ministry of Science and Innovation of Spain (PID2019-104184RB-I00 / AEI / 10.13039 / 501100011033), and by Xunta de Galicia and FEDER funds of the European Union (Centro de Investigación de Galicia accreditation 2019-2022, ref. ED431G 2019/01).

REFERENCES

[1] Iñaki Amatria-Barral, Jorge González-Domínguez, and Juan Touriño. 2023. pRIblast: a highly efficient parallel application for comprehensive lncRNA–RNA interaction prediction. *Future Generation Computer Systems* 138 (2023), 270–279.
[2] Ivan Antonov, Andrey Marakhonov, Maria Zamkova, and Yulia Medvedeva. 2018. ASSA: fast identification of statistically significant interactions between long RNAs. *Journal of Bioinformatics and Computational Biology* 16, 1 (2018), 1840001.
[3] Ivan Antonov, Evgeny Mazurov, Mark Borodovsky, and Yulia Medvedeva. 2019. Prediction of lncRNAs and their interactions with nucleic acids: benchmarking bioinformatics tools. *Briefings in Bioinformatics* 20, 2 (2019), 551–564.
[4] Xiaolin Dong, Xiaoxue He, Aoran Guan, Weikang Huang, Hongping Jia, Yun Huang, Sijin Chen, Zhibo Zhang, Jianpeng Gao, and Hui Wang. 2019. Long non-coding RNA Hotair promotes gastric cancer progression via miR-217-GPC5 axis. *Life Sciences* 217 (2019), 271–282.
[5] Maximilianos Elkouris, Georgia Kouroupi, Alexios Vourvoukelis, Nikolaos Papagiannakis, Valeria Kaltezioti, Rebecca Matsas, Leonidas Stefanis, Maria Xilouri, and Panagiotis K Politis. 2019. Long non-coding RNAs associated with neurodegeneration-linked genes are reduced in Parkinson's disease patients. *Frontiers in Cellular Neuroscience* 13 (2019), 58.
[6] Tsukasa Fukunaga and Michiaki Hamada. 2017. RIblast: an ultrafast RNA–RNA interaction prediction system based on a seed-and-extension approach. *Bioinformatics* 33, 17 (2017), 2666–2674.
[7] Tetsuro Hirose, Yuichiro Mishima, and Yukihide Tomari. 2014. Elements and machinery of non-coding RNAs: toward their taxonomy. *EMBO Reports* 15, 5 (2014), 489–507.
[8] Kevin L Howe, Premanand Achuthan, James Allen, Jamie Allen, Jorge Alvarez-Jarreta, M Ridwan Amode, Irina M Armean, Andrey G Azov, Ruth Bennett, Jyothish Bhai, et al. 2021. Ensembl 2021. *Nucleic Acids Research* 49, D1 (2021), D884–D891.
[9] Zhongxin Jin, Shiwei Gao, Wanyun Ma, Xinning Lyu, Xiaolei Cao, and Yuxin Yao. 2020. Identification and functional prediction of salt stress-related long noncoding RNAs in grapevine roots. *Environmental and Experimental Botany* 179 (2020), 104215.
[10] Daniel Lai and Irmtraud M Meyer. 2016. A comprehensive comparison of general RNA–RNA interaction prediction methods. *Nucleic Acids Research* 44, 7 (2016), e61.
[11] Hangchuan Shi, Yin Sun, Miao He, Xiong Yang, Michiaki Hamada, Tsukasa Fukunaga, Xiaoping Zhang, and Chawnshang Chang. 2020. Targeting the TR4 nuclear receptor-mediated lncTASR/AXL signaling with tretinoin increases the sunitinib sensitivity to better suppress the RCC progression. *Oncogene* 39, 3 (2020), 530–545.
[12] Maria Lina Tornesello, Raffaella Faraonio, Luigi Buonaguro, Clorinda Annunziata, Noemy Starita, Andrea Cerasuolo, Francesca Pezzuto, Anna Lucia Tornesello, and Franco Maria Buonaguro. 2020. The role of microRNAs, long non-coding RNAs, and circular RNAs in cervical cancer. *Frontiers in Oncology* 10 (2020), 150.
[13] Sinan Uğur Umu and Paul P Gardner. 2017. A comprehensive benchmark of RNA–RNA interaction prediction tools for all domains of life. *Bioinformatics* 33, 7 (2017), 988–996.