Future Generation Computer Systems 138 (2023) 270–279

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

pRIblast: A highly efficient parallel application for comprehensive lncRNA–RNA interaction prediction

Iñaki Amatria-Barral *, Jorge González-Domínguez, Juan Touriño

Universidade da Coruña, CITIC, Computer Architecture Group, Spain

ARTICLE INFO

Article history:
Received 4 April 2022
Received in revised form 22 July 2022
Accepted 23 August 2022
Available online 30 August 2022

Keywords:
Bioinformatics
lncRNAs
High Performance Computing
Parallel Computing
MPI
OpenMP

ABSTRACT

Long non-coding RNAs (lncRNAs) play a key role in several biological processes and scientists are constantly trying to come up with new strategies to elucidate their functions. One common approach to characterize these sequences consists in predicting their interactions with other RNA fragments. Nevertheless, the high computational cost of the bioinformatics tools developed for this purpose prevents their application to large-scale datasets. This paper presents *pRIblast*, a highly efficient parallel application for comprehensive lncRNA–RNA interaction prediction based on the state-of-the-art *RIblast* tool, which has been proved to show superior biological accuracy compared to other counterparts in previous experimental evaluations. Benchmarking on a multicore CPU cluster shows that *pRIblast* is able to compute in a few hours analyses that would need more than three months to complete with the original *RIblast* algorithm, always achieving the same level of prediction accuracy. Furthermore, this novel application can process large input datasets that cannot be processed with the former tool. *pRIblast* is free software publicly available to download at https://github.com/UDC-GAC/pRIblast under the MIT license.

1. Introduction

With the development of next generation sequencing technologies, scientists have witnessed amazing discoveries with regards to RNA biology. For instance, it has been found out that most RNA in the human genome does not translate into protein. This is the so called non-coding RNA (ncRNA), whose discovery has drastically changed the way biologists approach genetics. Among them, long non-coding RNAs (lncRNAs), classically defined as transcripts longer than 200 nucleotides, have gained special attention because their dysfunction is associated with severe diseases, such as diverse types of cancer [1,2], cardiac pathologies [3], preeclampsia [4], Parkinson's disease [5], or SARS-CoV-2 [6]. However, most lncRNAs are still poorly characterized and scientists are still searching for new strategies to elucidate their function.

As lncRNAs work by being assembled with other proteins or RNAs, the identification of their interaction partners is a powerful approach for inferring their function. Hence, fast and reliable computational prediction of interacting lncRNA–RNA pairs is an indispensable technique to further progress in long non-coding RNA characterization. Nevertheless, as prediction models become more sophisticated (and thus results more accurate), the computational cost associated to lncRNA–RNA interaction prediction grows exponentially and prevents its application to large-scale datasets.

This paper presents *pRIblast*, a highly efficient parallel application for comprehensive lncRNA–RNA interaction prediction. It is based on *RIblast* [7], a highly accurate (as proven in [8]) and publicly available tool specifically developed for this purpose. *pRIblast* makes use of High Performance Computing (HPC) technologies to accelerate the analyses on supercomputing facilities and multinode computing clusters. Concretely, it includes Message Passing Interface (MPI) routines and OpenMP pragmas. Furthermore, it has been thoroughly optimized and it can run the interaction prediction algorithm over huge datasets that would have never been processed with *RIblast*. As a consequence, *pRIblast* obtains the same level of prediction precision as its sequential counterpart and achieves speedups of up to 128x on a 256-core cluster, therefore completing a three-month-long analysis in just a few hours.

The main contributions of this work are:

- *pRIblast*, the only tool in the state of the art capable of predicting lncRNA–RNA interactions on multicore clusters and supercomputing facilities.

* Corresponding author.
E-mail addresses: i.amatria@udc.es (I. Amatria-Barral), jgonzalezd@udc.es (J. González-Domínguez), juan@udc.es (J. Touriño).

https://doi.org/10.1016/j.future.2022.08.014

Table 1
Publicly available tools in the state of the art to predict RNA–RNA interactions, indicating their search strategy, support for parallel computing on shared-memory (SM) and distributed-memory (DM) systems, and their references.

Tool	Search strategy	Parallel (SM)	Parallel (DM)	Ref.
AccessFold	Alignment	No	No	[19]
ASSA	Alignment & Accessibility	Yes	No	[11]
IntaRNA2	Accessibility	No	No	[20]
IRIS	Complex joint	No	No	[21]
LASTAL	Alignment	No	No	[22]
LncTar	Complex joint	No	No	[23]
RactIP	Complex joint	No	No	[24]
RIblast	Alignment & Accessibility	No	No	[7]
RIsearch2	Interaction only	Yes	No	[25]
RNAup	Accessibility	No	No	[26]
RRP	Alignment & Accessibility	No	No	[12]

- Two data distribution algorithms (named the area sum and the dynamic decomposition) and two multithreading optimizations (the dynamic scheduling and the sorting heuristic) that efficiently balance the workload and improve thread utilization, both statically and at runtime (Section 4.1).
- A database paging mechanism to reduce the memory usage of the interaction prediction algorithm, thus enabling *pRIblast* to process huge datasets (Section 4.2).
- A parallel-aware I/O procedure to decouple the scalability of the tool to that of the disk operations in those scenarios where an optimal workload decomposition does not imply equal computation times, and so processes finish their computations at different time steps (Section 4.3).

The rest of the paper is organized as follows. Related work is presented in Section 2. Section 3 explains the behavior of the original tool *RIblast* as necessary background to understand the parallel implementation of *pRIblast*, which is described in Section 4. Section 5 shows the experimental evaluation, and conclusions are presented in Section 6.

2. Related work

As of today, there exists a great amount of bioinformatics tools developed with the goal of predicting RNA–RNA interactions [9,10]. Table 1 lists the most relevant ones indicating, among other features, the support provided for parallel computing. However, most of them are not designed to work with lncRNAs. According to a recent review [8], those tools that combine information from local alignments and secondary structures, such as *RIblast* [7], *ASSA* [11], or *RRP* [12], obtain the most accurate biological results when working with lncRNAs. Among them, *RIblast* has been chosen as the starting point for our parallel approach because it is more widely used by the scientific community than *ASSA* or *RRP*. Some examples of biological studies that used *RIblast* are: the identification of lncRNAs correlated with the presence of different types of carcinomas (such as papillary thyroid [13], lung [14] or renal cell [15]), the search of coral reef lncRNAs that are differentially expressed when infected by certain algae [16], or the identification of those lncRNAs that are salt-stressed in grapevine roots [17]. Although a novel alternative has been developed after the publication of this review (namely *IntaRNAhelix* [18]), it has not been considered as basis for our work because it is not publicly available and its biological accuracy has not been compared to *RIblast*.

To the best of our knowledge, the only two tools in the state of the art that can be executed in parallel to accelerate the identification of RNA–RNA interactions are *ASSA* [11] and *RIsearch2* [25]. However, as indicated in Table 1, they are limited to shared-memory systems as they only include multithreading support. Therefore, there is no previous work focused on exploiting the hardware available in distributed-memory clusters and supercomputers for this task.

The same HPC technologies (MPI processes and multithreading) used within *pRIblast* have been successfully applied to other bioinformatics problems. The workload and data distribution approach is different depending on the characteristics of the application. In some cases, such as the construction of genetic networks [27] or the subtomogram alignment [28], the workload is similar for all instances or sequences and it is sufficient to apply a static distribution with the same amount of data for each process. But it is not the common scenario in bioinformatics, and more sophisticated distributions must be employed when the amount of work is irregular for each instance or sequence. One option consists in maintaining the static data distribution at the process level and using a dynamic scheduling for multithreading, as in the case of multiple sequence alignment [29]. However, this approach is not enough to obtain high scalability on large clusters and supercomputers, as will be shown in the experimental evaluation of Section 5. Some bioinformatics applications apply a dynamic distribution also at the process level, but always using a master–worker paradigm where one master process reads the input data and sends fragments to the other processes (the workers) as soon as they have finished the previous tasks [30,31]. The main drawback of this approach is the bottleneck introduced in the network due to the master–worker communications. As will be explained in Section 4.1.3, *pRIblast* overcomes this problem with a completely dynamic distribution (without a single master) using MPI one-sided routines which, up to our knowledge, has never been applied to the bioinformatics domain. An efficient parallel I/O approach, also based on one-sided routines, as the one presented in Section 4.3, has not been found in the state of the art either.

3. Background: RIblast

RIblast [7] is a widely used tool for extensive lncRNA–RNA interaction analysis. The key idea behind the *RIblast* algorithm is the utilization of a seed-and-extension heuristic, which is broadly adopted in sequence homology search tools. It also makes use of suffix arrays, memoization, and an approximate RNA energy model to predict lncRNA–RNA interactions.

The *RIblast* execution consists of two major steps: database construction and RNA interaction search. The second step is generally the most critical one, because different batches of input query sequences (the lncRNAs) may be run against the same target RNA dataset (the database) . Consequently, it is the target for parallelization in the current version of *pRIblast*.

In the database construction step *RIblast* starts by calculating approximate accessible energies for each sequence in the target

```
1  procedure RNAInteractionSearch()
2      db := loadTargetDB()
3      seqs := loadQuerySeqs()
4      for seq in seqs do
5          suff := constructSuffixArray(seq)
6          acc := calculateAccessibility(seq)
7          seeds := seedSearch(suff, acc, db)
8          hits := extendSeeds(seeds, suff, acc, db)
9          saveHits(hits)
10     end for
11 end procedure
```

Fig. 1. Pseudocode of the *RIblast* RNA interaction search step.

RNA dataset. Accessible energies are later used in the RNA interaction search step to find promising seed regions. Second, the target RNA sequences are reversed and concatenated to build a suffix array. Third, search results of short substrings are exhaustively precalculated (again, to be later used in the RNA interaction search step). And finally, the approximate energies, the concatenated sequences, the suffix array, and the search results of short substrings are stored in a set of binary and text files that comprise the resulting target database.

In the RNA interaction search step *RIblast* first calculates approximate accessible energies and constructs a suffix array for every lncRNA in the input query dataset. Second, seed regions with a particularly low interaction energy are found using the suffix arrays and the accessibility energies of the queries and target sequences in the database. Third, interactions are extended from seed regions, and finally, interactions that fully overlap are removed prior to storing the final results in an output text file.

Fig. 1 shows a high-level pseudocode of the algorithm used in *RIblast* for the RNA interaction prediction step. Note that the seed-and-extension procedure (lines 7 and 8), which is the most computationally demanding one, accounts for more than 95% of the total runtime because their computational complexity is quadratic with respect to the number of sequences. They are also the reason for *pRIblast* to focus its performance optimization efforts on this second step of the algorithm. Not only it is the step most often executed in real biological analyses, but also the one that requires the largest runtime.

4. pRIblast implementation

pRIblast provides a solution to the high computational cost of lncRNA–RNA interaction prediction by effectively exploiting the hardware available in multicore clusters and supercomputing facilities. It obtains the same biological results as *RIblast*, therefore achieving the same level of prediction accuracy. As its predecessor, *pRIblast* provides a command-line interface utility and is available to the scientific community at GitHub[1] under the MIT license.

As previously stated, the second step of the *RIblast* algorithm has been accelerated within *pRIblast*, since it implements the most demanding functions behind the computational prediction of interacting lncRNA–RNA pairs: the *seedSearch* and *extendSeeds* functions shown in Fig. 1. Specifically, a two-level parallelization procedure has been developed.

On the one hand, the workload is distributed among several MPI processes launched on different nodes of a computing cluster. MPI is the *de facto* standard to develop parallel programs running on distributed-memory systems by providing portable, efficient, and flexible message-passing routines. On the other hand, each MPI process spawns several OpenMP threads to take advantage of all the CPU cores within each node. OpenMP provides a scalable and portable application programming interface for shared-memory systems. This hybrid parallel implementation based on HPC standards is flexible enough to be adapted to any multicore cluster or supercomputer. It can also exploit all the characteristics of the ever-evolving modern CPU architectures such as vector instructions, thread affinity, and HyperThreading, which allows to schedule several threads into the same CPU core to minimize stall cycles, hence improving performance.

4.1. Data distribution

The main objective of *pRIblast* is to efficiently distribute the workload (essentially the lncRNA sequences) among a set of MPI processes, and spawn OpenMP threads that run in parallel to accelerate the prediction of lncRNA–RNA interacting pairs. Indeed, the data distribution function used to assign sequences to processing units (i.e., processes or threads) is key to achieve maximum performance. But, because *pRIblast* works using a seed-and-extension heuristic, some sequences may be considered more promising by the tool (i.e., the *seedSearch* function finds more seeds), and thus successive steps to predict interactions will be computationally more expensive than others. This tricky situation is therefore the main topic in this section: how to optimally distribute sequences among processes if they produce different amounts of work.

Three different approaches to assign sequences to processing units are proposed: a pure block distribution, an area sum algorithm, and a dynamic decomposition scheme. They range from less to more sophisticated, and they may be used in different scenarios to ensure that *pRIblast* achieves maximum performance.

4.1.1. Pure block distribution

The first data distribution approach developed within *pRIblast* is the pure block procedure. It is a classical data decomposition scheme and it is useful when the workload is evenly balanced among all the sequences of the input datasets. It is quite straightforward: MPI routines are used to divide the input batch of query sequences among a set of processes in chunks of size

$$S = \left\lceil \frac{\#\text{seqs}}{\#\text{procs}} \right\rceil \tag{1}$$

and then OpenMP threads are spawned within each process to compute its assigned sequences in parallel.

Nevertheless, as discussed earlier, the workload may not be evenly balanced among the sequences of the input datasets, meaning that this decomposition scheme would not achieve good speedups in virtually any scenario. So, two thread-level improvements are applied to minimize the impact of the variable workload: a dynamic OpenMP scheduling policy and a sorting heuristic.

The dynamic scheduling makes threads not to get assigned a fixed chunk of the block of sequences. Rather, threads take one sequence at a time and ask for a new one once they have finished its computation. No responsibility is therefore assigned in advance to any thread, and so all of them work together to compute their block of sequences as fast as they can.

As for the sorting heuristic, it is based on the assumption that the probability of finding seeds in a sequence is a function of its length. So, sorting the block of assigned sequences in length descending order leads to optimal execution times in multithreading environments. But note that this is a rather bold assumption. The number of seeds that are generated for a certain sequence is not only a function of its length. Biological factors, such as the free energy released by the interaction of

[1] https://github.com/UDC-GAC/pRIblast.

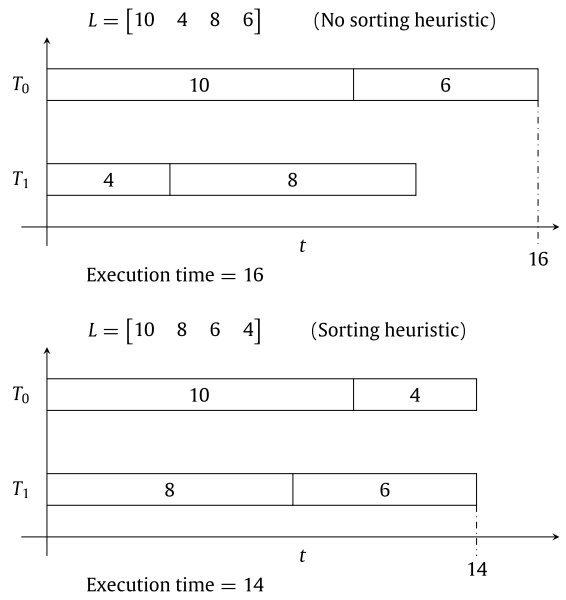

Fig. 2. Example to illustrate that the sorting heuristic (bottom) produces optimal results if the workload is a direct function of the lncRNAs' lengths and a dynamic scheduling policy is applied.

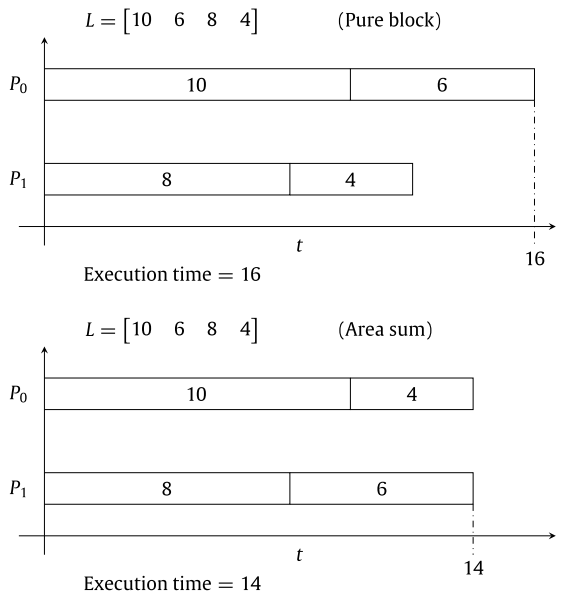

Fig. 4. Comparison between the pure block decomposition scheme (top) and the area sum algorithm (bottom).

```
 1  procedure RNAInteractionSearch()
 2      db := loadTargetDB()
 3      // apply the block decomposition
 4      mySeqs := blockDecomposition()
 5      // sort the block of sequences
 6      sort(mySeqs, reverse=True)
 7      // dynamic scheduler within the block of sequences
 8      #omp parallel for schedule(dynamic)
 9      for seq in mySeqs do
10          suff := constructSuffixArray(seq)
11          acc := calculateAccessibility(seq)
12          seeds := seedSearch(suff, acc, db)
13          hits := extendSeeds(seeds, suff, acc, db)
14          saveHits(hits)
15      end for
16      mergeHits()
17  end procedure
```

Fig. 3. Pseudocode of the pure block *pRIblast* RNA interaction search step.

two RNA segments, play an important role, too. Therefore, it is possible to over or underestimate the cost of computing a sequence. Anyway, although not perfect, the sorting heuristic proved to significantly speed up the pure block distribution in all preliminary experiments.

Fig. 2 shows an example with the difference in theoretical runtime between using the dynamic scheduling policy alone and adding the sorting heuristic to process an input vector L of lncRNA sequences with two threads (T_0 and T_1). L represents the lengths in characters of the sequences to compute, and the interaction prediction time of each lncRNA is estimated assuming that sequence length and workload are directly proportional (i.e., for each character in the sequence string we assume one unit of time t).

To sum it all up, Fig. 3 shows a high-level description of the pure block *pRIblast* RNA interaction search algorithm. It looks like Fig. 1 back in Section 3, except for lines 4, 6, 8 and 16. Lines 4 and 6 implement the block distribution procedure and the sorting heuristic, respectively. Line 8 spawns OpenMP threads to execute the loop iterations in parallel using the dynamic scheduling policy. And finally, line 16 reconstructs the final output file now that sequences have been distributed among a set of MPI processes. For now, we assume that the final output file is built sequentially after all processes finish the interaction prediction loop. However, a parallel-aware I/O procedure to speed up this file reconstruction will be presented later in Section 4.3.

4.1.2. Area sum distribution

Preliminary tests showed that the pure block decomposition was a rather simple, yet effective, data distribution approach. However, it soon fell short in scalability as execution time optimization could only be made at thread level by means of the dynamic OpenMP scheduling policy and the sorting heuristic. Or put differently, as the pure block approach fails to address the problem of variable workload, it relies on two clever but limited techniques to accelerate the prediction of interacting lncRNA–RNA pairs. And therefore, as datasets become larger and sequences more diverse, performance starts to dramatically degrade because blocks produce increasingly different amounts of work (i.e., seeds). To overcome this scenario, a more informed data decomposition scheme was designed, the area sum algorithm, which addresses the problem of variable workload both at process and thread level.

Assuming again that the workload is proportional to the lengths of the sequences (i.e., the probability of finding seeds increases with the lncRNA length), the area sum distribution states as follows: assign to each process a number of sequences so that their lengths sum up to the value

$$S = \left\lfloor \frac{\sum_{i=1}^{n} \text{length(seqs[i])}}{\#\text{procs}} \right\rfloor \tag{2}$$

In other words, this approach tries to provide a similar amount of characters to each process so that they all produce a similar amount of seeds, even though this may mean that they receive a different amount of sequences.

An $O(np)$ algorithm (where n is the number of sequences and p the number of processes) was developed to implement the area sum data distribution procedure. It distributes lncRNAs in a greedy-like fashion, and its operation is exemplified in Fig. 4 for

two processes (P_0 and P_1), where it is compared with the pure block algorithm discussed in the previous subsection. This figure helps to understand what exactly the S parameter represents here, which is no more than a "bucket size". In this sense, the area sum procedure carefully fills up the buckets so that all of them have a similar amount of workload, unlike the pure block procedure, which does not take into account the length of the sequences and fails to evenly distribute the workload at process level.

Now that processes have assigned blocks of sequences that will ideally produce a similar amount of seeds, OpenMP threads are spawned and the algorithm is run in parallel to accelerate the computation. The area sum algorithm addresses the problem behind the correct assignment of lncRNAs to processes, but sequences within a block may be very different in length. Therefore, the dynamic scheduling policy and the sorting heuristic are once again used to further enhance *pRIblast* performance at thread level.

4.1.3. Dynamic distribution

The area sum algorithm proved to be a step forward towards successfully speeding up the *pRIblast* RNA interaction search procedure. Weighting sequences according to their length makes it possible to correctly balance the workload among processes, leading to significant boosts in performance.

But there still exists one scenario in which the area sum distribution may fall as short as the naive pure block procedure. One process may receive a large amount of sequences considered unpromising by the area sum algorithm and it may turn out that they produce the most workload. As it was previously explained, all the reasoning behind the heuristic knowledge developed within *pRIblast* is based on the assumption that the workload is a direct function of the sequence length, which indeed is not the only factor. Therefore, such a scenario is completely feasible, and in that case the workload would remain extremely unbalanced with the area sum approach.

In order to solve this problem, an additional data decomposition algorithm was built for *pRIblast*, a dynamic data distribution procedure. The idea behind it is to further minimize the time penalty associated with under or overestimating the cost of computing a sequence. So, instead of distributing sequences, the dynamic approach uses MPI one-sided communications to create a shared pool of lncRNA sequences and allow all processes and threads in a cluster to collaboratively solve the computation. Just like the OpenMP dynamic scheduler does at thread level within each process, but now it is done at the whole cluster level. As a consequence, a situation where one process receives a set of lncRNAs that turns out to produce the most workload would never become true. No process would ever have the responsibility of computing a concrete sequence (again, as it was done with the OpenMP dynamic scheduling policy within each block of sequences).

However, this approach increases the amount of communications required to run the *pRIblast* algorithm. Whereas before there was only one point of synchronization (i.e., distributing sequences at the beginning of the computation), now, for each loop iteration run by each thread, an MPI call is needed to pull sequences from the shared pool of lncRNAs. Yet, provided that high performance interconnection networks are used for communications, and taking into account that computing a sequence is far more expensive than an MPI one-sided operation, this overhead can be considered negligible.

Fig. 5 shows a high-level pseudocode of the dynamic *pRIblast* RNA interaction search algorithm. The first most noticeable difference is that data are no longer distributed among processes at the beginning of the computation (line 4), but rather all of them load into memory the whole input set of query sequences. Second, now the sorting heuristic (line 6) is applied globally to the complete set of lncRNA sequences (all processes share the same block of sequences, the input dataset). And third, the pool of shared sequences is implemented using the MPI *Fetch_and_op* operation (line 10). That is, all the threads in the cluster atomically pull and increment a shared index to obtain a reference to the following sequence that has not yet been processed. Therefore, no concrete OpenMP scheduling policy is needed: all threads in the cluster run in parallel and ask the higher level in the hierarchy (the associated MPI process) for sequences to compute. Note that, even though this data distribution algorithm could rely on MPI processing alone, threads are still used to share data through the memory hierarchy, thus reducing memory pressure (e.g., to hold all the database pages in cache), and to allow the use of HyperThreading.

```
1  procedure RNAInteractionSearch()
2      db := loadTargetDB()
3      // do not divide the sequences
4      seqs := loadQuerySeqs()
5      // sort all the sequences
6      sort(seqs, reverse=True)
7      #omp parallel
8      while true do
9          // workload balancing among all threads with MPI
10         idx := Fetch_and_op(one, op=SUM)
11         if idx > length(seqs) then
12             break
13         end if
14         seq := seqs[idx]
15         suff := constructSuffixArray(seq)
16         acc := calculateAccessibility(seq)
17         seeds := seedSearch(suff, acc, db)
18         hits := extendSeeds(seeds, suff, acc, db)
19         saveHits(hits)
20     end while
21     mergeHits()
22 end procedure
```

Fig. 5. Pseudocode of the dynamic *pRIblast* RNA interaction search step.

4.2. Database paging

RIblast writes results to the output file only after all target sequences have been compared against a certain lncRNA query sequence. Therefore, if a target database is too large or a query segment produces a big amount of prediction candidates (i.e., seeds), memory may be filled up before results are finally saved to disk.

pRIblast introduces a very effective optimization to overcome this limitation. It divides the target database into several smaller and independent pages, which, because of their reduced size, will not overflow the available memory space. Put differently, because results can only be written after a sequence has been compared against a target database, *pRIblast* makes predictions against subsets of such database. In this way, once a query sequence has been compared against all subsets of the target dataset, the output file will hold the same results as if the sequence had been compared against the whole target database in one single round.

Because of the nature of the data structures stored in the database, it is not possible to paginate a target dataset with no previous preprocessing. Therefore, the database construction step (see Section 3) has been slightly modified to be able to read chunks of a fixed size later in the RNA interaction search step. In any case, databases built without the tweaked version of the construction step are backwards compatible with the new *pRIblast* workflow.

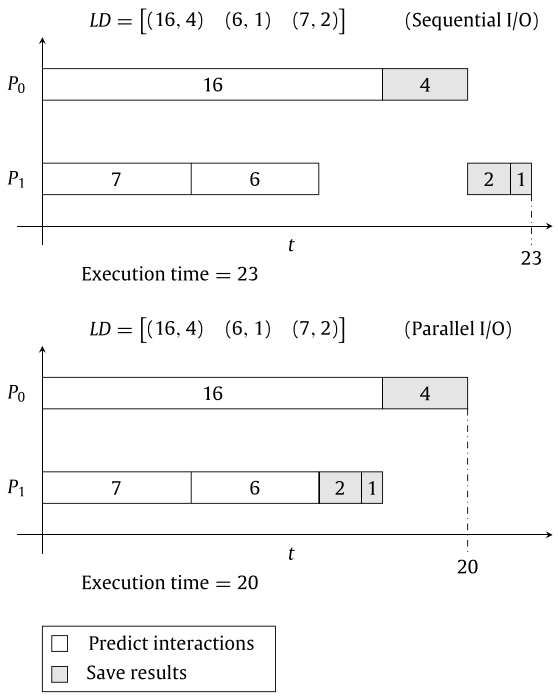

Fig. 6. Time difference between a sequential I/O procedure (top) to reconstruct the *RIblast* output file and the parallel-aware I/O algorithm (bottom) featured within *pRIblast*.

4.3. Parallel-aware I/O

pRIblast can process huge datasets whose results grow to the order of giga and terabytes. Therefore, it is a must to decouple the scalability of the tool to that of the disk operations. In a similar fashion, *pRIblast* is a parallel algorithm, and executing a given dataset with an increasing number of processes dramatically reduces the time spent in the interaction prediction loop. In this case, I/O time becomes comparable to that of the interaction prediction and may compromise the maximum speedup achievable by the novel tool.

As a solution to this inherent problem in parallel computing, another optimization has been included in *pRIblast*: a scalable, parallel-aware I/O procedure to write results to disk as processes finish the interaction prediction function. That is, as processes may receive a different amount of workload (an optimal decomposition does not mean that all processing units take the same execution time), I/O operations are cleverly hidden in the *pRIblast* parallel execution pipeline to reduce the amount of time that the tool is really stalled waiting for disk operations to complete. Fig. 6 exemplifies this scenario, showing the time difference between using the parallel-aware I/O procedure and a sequential reconstruction of the *pRIblast* output file. Each tuple in the vector *LD* represents the length of a sequence (first number of the pair) and the units of work t that it takes to save its results to disk (second number of the pair). Again, we assume that one character in a sequence string needs one unit of time t. Sequences were optimally assigned to the two processes, and so the important bit is to identify the time difference between using a naive sequential I/O procedure and the parallel-aware I/O procedure built for *pRIblast*.

Compared to a naive I/O scheme where processes write their results one after another once all the computation is completed, the parallel-aware I/O optimization is 45 times faster running on a 256-core cluster using a dataset that produces 436 GB of output data. However, if all processes receive the same amount of workload, this optimization degenerates into a sequential writing of results to disk. But such scenario is very unlikely (again, an optimal decomposition does not mean equal execution times), and so this optimization has proved to speed up and scale the parallel prediction of interacting lncRNA–RNA pairs in all the benchmarks presented in Section 5. This is also the reason to discard the use of MPI-IO routines as a solution to perform parallel I/O in *pRIblast*. As it is extremely unlikely that all processes finish the interaction prediction function at the same time, MPI-IO would only provide marginal performance improvements if the last process to finish the computation shared the storage medium with other processes in the MPI group.

```
1  procedure parallelIO(rank, procs)
2      next := 0
3      count := 0
4      last := Fetch_and_op(rank, op=REPLACE)
5      if last != -1 then
6          Send(rank, to=last)
7          Recv(count, from=last)
8      end if
9      saveResults()
10     count := count + 1
11     if count != procs then
12         Recv(next, from=ANY_SOURCE)
13         Send(count, to=next)
14     end if
15 end procedure
```

Fig. 7. Pseudocode of the parallel-aware I/O optimization to save results as soon as processes finish their computation loop.

The building blocks of this optimization are MPI one-sided communications, MPI point-to-point communications and a globally shared index (which initially holds the special value -1). On top of them, a message queue is created and processes can write their results to a single output file as soon as they finish their computation. Fig. 7 shows the implementation of the parallel-aware I/O procedure using a high-level pseudocode.

First, every process atomically pulls the current value of the shared index and replaces it with its own rank (line 4). If the pulled value is -1 (line 5), it means that the process is the first one to finish the computation and it is free to skip to line 9 and write its results to disk. Otherwise, variable *last* holds the value of the last process in the queue, and therefore the current process must wait until *last* has finished writing its results by calling the blocking *Send* operation (line 6).

Once the first process has finished writing its results, it increments the value of *count* in line 10 (i.e., how many processes have already passed that point in the program). Then, it proceeds to execute lines 11 to 14 to notify the next process in the queue that it can save its results. For that purpose, it first receives the *rank* of the waiting process (line 12) and then updates its *count* value by calling *Send* (line 13). Similarly, the process waiting in the queue executes line 6 to send its rank to the leaving process, and line 7 to receive the updated value of *count*.

All processes will execute this repetitive pattern of send and receive operations to gradually copy their results one by one as soon as they finish their interaction search loop. Once the last process that finishes the computation saves its results, *count* will be equal to the number of processes and lines 12 and 13 will be skipped to finish the execution of the parallel-aware I/O procedure.

5. Experimental evaluation

To assess the performance of *pRIblast*, a set of comprehensive benchmarks were performed using five real, representative

Table 2
Characteristics of the datasets used to assess the performance of the *pRIblast* tool.

Dataset	lncRNAs	RNAs	Output size	*RIblast* execution
Lepi	730	1,256	2.68GB	Successful
Ursus	935	3,899	6.52GB	Successful
Droso	2,266	3,963	13.20GB	Successful
Anser	10,422	10,988	436.22GB	Out of memory
Anas	14,504	15,061	1.56TB	Out of memory

RNA datasets (available to download at the Ensembl genome browser [32][2]). This section presents a performance comparison between the original *RIblast* algorithm and the novel *pRIblast* tool running on a 16-node multicore cluster with a total of 256 CPU cores. Other tools such as *ASSA* or *RRP* (see Section 2) have not been included in the evaluation. The reason is that it is not fair to compare the runtime of tools that follow a different biological approach, and thus do not predict the same lncRNA–RNA interactions.

Each node in the computing cluster has two octa-core Intel Xeon E5-2660 Sandy Bridge-EP processors (16 cores per node with support for HyperThreading) and 64 GB (DDR3 @ 1600 MHz) of main memory. Nodes are interconnected through a low-latency and high-bandwidth InfiniBand FDR network. As for the software stack, the GNU Compiler Collection (GCC) v8.3.0 and the MPI implementation OpenMPI v3.1.4 were the tools used to compile and run the applications (both tools were built with the `-O3` flag enabled).

Table 2 summarizes the characteristics of the RNA datasets. They range from small and easy to compute to very large and completely unbalanced, with sequences that produce so many seeds that the data distribution algorithms are forced to make wise decisions to efficiently distribute the workload. Lepi[3] (*Lepidothrix Coronata*) and Ursus[4] (*Ursus Americanus*) are considered easy to compute, as their scalability is proportional to their size. However, Droso[5] (*Drosophila Melanogaster*), Anser[6] (*Anser Brachyrhynchus*) and Anas[7] (*Anas Platyrhynchos Platyrhynchos*) are considered unbalanced because they are so large and contain sequences so different in length (and thus in amount of work) that they quickly bound the efficiency of the *pRIblast* parallel algorithm. Indeed, the two largest datasets (Anser and Anas) produce so many seeds that *RIblast* runs out of memory when trying to analyze them, whereas *pRIblast* can complete these executions thanks to the database paging mechanism explained in Section 4.2. This shows the power of the optimizations developed within *pRIblast*, and thus makes it an indispensable tool to move forward towards comprehensive computational prediction of lncRNA–RNA interacting pairs.

5.1. Configuration of the experiments

Each dataset was run using 1, 2, 4, 8 and 16 nodes, that is, 16, 32, 64, 128 and 256 cores, respectively. In addition, the time spent computing every sequence of each dataset was also measured. Therefore, it was possible to compare the behavior of the data distribution algorithms to that of a theoretical best one (i.e., using a perfect scheduler that knows *a priori* how long it takes to compute each sequence).

[2] http://ftp.ensembl.org/pub/release-97/fasta.
[3] http://ftp.ensembl.org/pub/release-97/fasta/lepidothrix_coronata/ncrna.
[4] http://ftp.ensembl.org/pub/release-97/fasta/ursus_americanus/ncrna.
[5] http://ftp.ensembl.org/pub/release-97/fasta/drosophila_melanogaster/ncrna.
[6] http://ftp.ensembl.org/pub/release-97/fasta/anser_brachyrhynchus/ncrna.
[7] http://ftp.ensembl.org/pub/release-97/fasta/anas_platyrhynchos_platyrhynchos/ncrna.

Table 3
Runtime of *pRIblast* on a 16-node cluster (256 CPU cores in total) using the best configuration of algorithm, threads per process and use of HyperThreading. Execution times for *RIblast* are all single-threaded and those marked with * correspond to estimations calculated with *pRIblast*.

Dataset	*RIblast* time	*pRIblast* time
Lepi	2h 6 m 42s	3 m 37s
Ursus	4h 38 m 34s	1 m 59s
Droso	14h 43 m 49s	9 m 55s
Anser	22d 8h 13 m 46s*	3h 56 m 6s
Anas	101d 10h 52 m 55s*	20h 7 m 45s

pRIblast exploits both MPI and OpenMP parallelism. Therefore, for each node setup, five different configurations of number of threads and processes were also tested. These are: 1p16t (i.e., 1 process per node launching 16 threads each), 2p8t, 4p4t, 8p2t and 16p1t. This extensive benchmarking allowed to acquire knowledge about which arrangement of threads per process better suits each data decomposition algorithm and input dataset.

Finally, for all the combinations of the configurations above, it was also tested if using Intel's HyperThreading technology yielded any improvement in execution time.

5.2. Results

The summary of the benchmarking results is presented in Table 3 and Fig. 8. Table 3 highlights the significant improvement in execution time when all the 256 cores of the cluster are used to run *pRIblast*. The runtimes shown for *pRIblast* are the ones using the best configuration for each dataset (i.e., distribution algorithm, number of threads per process, and use of HyperThreading). *RIblast* times are all single-threaded, since it does not provide parallel processing capabilities. Also, the *RIblast* times of Anser and Anas correspond to estimations because, as mentioned before, *RIblast* runs out of memory when processing these datasets. Times were estimated by (1) saving in a file the runtime that the different processes need in *pRIblast* to analyze each sequence; and (2) adding up all these runtimes similarly to what would happen in a sequential execution, where the sequences would be computed one by one.

Fig. 8 shows a scalability analysis of the three *pRIblast* data decomposition algorithms. The graphs show the speedups obtained by each algorithm without HyperThreading using the best threads-per-process configuration. The graphs include two additional curves: one for the theoretical best speedup, and another one for the best speedup of the tool using HyperThreading. The theoretical value was calculated with respect to the number of physical cores, since it is not possible to simulate the behavior of the HyperThreading. This is one of the reasons why it is possible to outperform the theoretical best speedup. Also, for the Anas dataset it was only possible to get results when running *pRIblast* with 256 cores. Any other configuration with fewer cores exceeded the execution timeout of the cluster used for this evaluation. With all these experimental results, it was possible to extract the following conclusions:

- As the workload is not balanced among the sequences of the RNA datasets, the scalability of *pRIblast* is always bounded by the execution time of the most computationally demanding sequence (the one that produces the highest number of seeds). Note the flattening in all the curves, including the theoretical best. Although parallel applications usually offer better speedups for larger datasets, in this case a large dataset may also increase the probability of containing a sequence more promising than the others (i.e., a sequence that produces many more seeds), thus limiting the exploitable

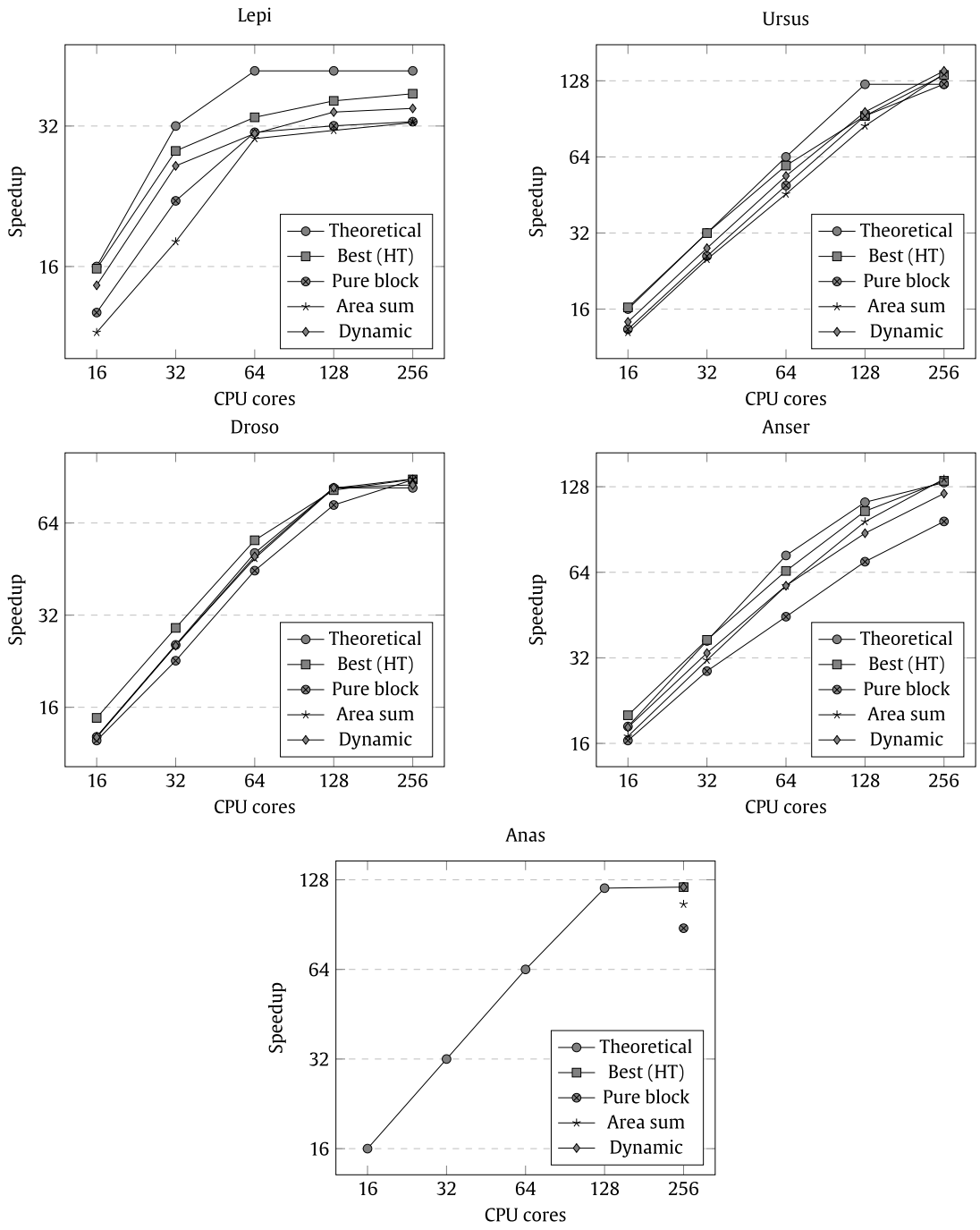

Fig. 8. Speedups of the three different data decomposition algorithms included in *pRIblast* without HyperThreading. The speedups are compared with a theoretical best and the best execution of the tool with HyperThreading enabled.

parallelism. This is the case when comparing the results for Ursus and Droso. Even though Droso is larger than Ursus, it exhibits worse speedup because it becomes increasingly limited in scalability as there is one single sequence that dominates the runtime.

- The area sum algorithm and the dynamic decomposition are able to solve the performance issues related to the unbalanced nature of the datasets. These informed data distribution algorithms consistently outperform the pure block procedure, achieving and even surpassing the theoretical best speedup (thanks to better use of the memory hierarchy).
- The dynamic decomposition procedure beats the area sum algorithm only when the number of CPU cores is small and the execution time is not bounded by one sole sequence. On the other hand, when a large number of cores is used the area sum algorithm is faster. It makes sense: with more processes involved the area sum algorithm is less likely to over or underestimate the cost of computing a sequence, and therefore it can perfectly balance the workload with no need for constant communication (as it happens with the dynamic algorithm).
- As for the configuration of processes and threads, the area sum algorithm works best when processes spawn more

threads (i.e., 2p8t or 4p4t). In this case, the area sum approach alone performs relatively well, but it is the sorting heuristic which really makes a difference here, compensating for any under or overestimation made by the distribution procedure. In a similar way, because no OpenMP scheduling is used by the dynamic algorithm and workload balancing relies on repeating MPI communications, this algorithm works better with fewer threads per process (i.e., 8p2t). 16p1t is discouraged since multithreading allows to share data through the memory hierarchy, thus outperforming an MPI-only approach.
- HyperThreading improves the execution time in those scenarios where the computation is not yet bounded by the prediction of one sequence, although it may hurt performance in some cases due to increased memory pressure.

Furthermore, this exhaustive study has allowed to conclude when and how to use each one of the data decomposition algorithms developed within *pRIblast* to achieve maximum speedup. These conclusions are:

- Do not use the pure block algorithm, it was developed only for benchmarking purposes.
- Use the area sum algorithm when there are plenty of computing resources available with respect to the dataset size. Do not enable HyperThreading as it may hurt performance, especially when working with a large number of cores. Use more threads than processes to exploit the sorting heuristic.
- Use the dynamic decomposition procedure if the amount of computing resources is small with respect to the dataset size. HyperThreading will not hurt, and can be really beneficial when using a low number of cores. More processes than threads works best in this case.

6. Conclusions

The computational prediction of interacting lncRNA–RNA pairs is a hot research topic in bioinformatics. Progress in the field may help to further advance in the knowledge of severe diseases, such as cancer or Parkinson's. Therefore, having publicly available tools that can successfully accelerate this task is indispensable to move forward in this direction.

For that purpose, this paper presents the high performance tool *pRIblast*, which is capable of exploiting the hardware available in multicore clusters and supercomputing facilities. *pRIblast* is based on *RIblast*, a highly accurate tool for the prediction of lncRNA–RNA interacting segments, and reduces computation times dramatically. Furthermore, *pRIblast* has been carefully optimized and can take on new challenges that may never have been completed with the original *RIblast* tool. Indeed, the experimental evaluation performed on a 16-node cluster (256 cores in total) showed that *pRIblast* can process two 22- and 101-day real datasets in as little as 4 and 20 h, respectively. But these outstanding results do not only come from the fact that *pRIblast* can run in parallel. The two optimizations developed within the novel tool play a key role at the time of computing these two datasets. The parallel-aware I/O procedure makes it possible to unbound the scalability of the tool from that of the disk operations, thus further improving the speedups achieved when the result files grow to the order of giga or terabytes. And what is more, it would have never been possible to compute these two datasets without the database paging mechanism.

Future work may include: (1) parallelize the *pRIblast* database construction step, even though its computational weight is low compared to that of the RNA interaction search step; (2) develop an approach to automatically choose the most promising data decomposition algorithm depending on the hardware available and the size of the input dataset; and (3) evaluate whether the MapReduce paradigm is a good match to further improve *pRIblast*, as it could provide redundancy, fault tolerance, and automatic workload balancing mechanisms at the expense of translating the prediction algorithm to a different programming language.

CRediT authorship contribution statement

Iñaki Amatria-Barral: Software, Investigation, Validation, Writing – original draft. **Jorge González-Domínguez:** Conceptualization, Supervision, Writing – original draft. **Juan Touriño:** Funding acquisition, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

I have shared all the data and code within the paper.

Acknowledgments

This work was supported by the Ministry of Science and Innovation of Spain (PID2019-104184RB-I00/AEI/ 10.13039/ 5011000 11033), and by Xunta de Galicia and FEDER funds of the European Union (Centro de Investigación de Galicia accreditation 2019–2022, ref. ED431G 2019/01; Consolidation Program of Competitive Reference Groups, ref. ED431C 2021/30). Funding for open access charge: Universidade da Coruña/CISUG.

References

[1] M.L. Tornesello, R. Faraonio, L. Buonaguro, C. Annunziata, N. Starita, A. Cerasuolo, F. Pezzuto, A.L. Tornesello, F.M. Buonaguro, The role of microRNAs, long non-coding RNAs, and circular RNAs in cervical cancer, Front. Oncol. 10 (2020) 150.
[2] X. Dong, X. He, A. Guan, W. Huang, H. Jia, Y. Huang, S. Chen, Z. Zhang, J. Gao, H. Wang, Long non-coding RNA Hotair promotes gastric cancer progression via miR-217-GPC5 axis, Life Sci. 217 (2019) 271–282.
[3] L. Hobuß, C. Bär, T. Thum, Long non-coding RNAs: at the heart of cardiac dysfunction? Front. Physiol. 10 (2019) 30.
[4] M.-T. Moradi, Z. Rahimi, A. Vaisi-Raygani, New insight into the role of long non-coding RNAs in the pathogenesis of preeclampsia, Hypertension Pregnancy 38 (1) (2019) 41–51.
[5] M. Elkouris, G. Kouroupi, A. Vourvoukelis, N. Papagiannakis, V. Kaltezioti, R. Matsas, L. Stefanis, M. Xilouri, P.K. Politis, Long non-coding RNAs associated with neurodegeneration-linked genes are reduced in Parkinson's disease patients, Front. Cellular Neurosci. 13 (2019) 58.
[6] M. Moazzam-Jazi, H. Lanjanian, S. Maleknia, M. Hedayati, M.S. Daneshpour, Interplay between SARS-CoV-2 and human long non-coding RNAs, J. Cellular Molecular Med. 25 (12) (2021) 5823–5827.
[7] T. Fukunaga, M. Hamada, RIblast: an ultrafast RNA-RNA interaction prediction system based on a seed-and-extension approach, Bioinformatics 33 (17) (2017) 2666–2674.
[8] I.V. Antonov, E. Mazurov, M. Borodovsky, Y.A. Medvedeva, Prediction of lncRNAs and their interactions with nucleic acids: benchmarking bioinformatics tools, Brief. Bioinform. 20 (2) (2019) 551–564.
[9] D. Lai, I.M. Meyer, A comprehensive comparison of general RNA-RNA interaction prediction methods, Nucleic Acids Res. 44 (7) (2016) e61.
[10] S.U. Umu, P.P. Gardner, A comprehensive benchmark of RNA-RNA interaction prediction tools for all domains of life, Bioinformatics 33 (7) (2017) 988–996.
[11] I. Antonov, A. Marakhonov, M. Zamkova, Y. Medvedeva, ASSA: fast identification of statistically significant interactions between long RNAs, J. Bioinform. Comput. Biol. 16 (1) (2018) 1840001.
[12] G. Terai, J. Iwakiri, T. Kameda, M. Hamada, K. Asai, Comprehensive prediction of lncRNA-RNA interactions in human transcriptome, BMC Genomics 17 (2016) 12.

[13] H. Teng, F. Mao, J. Liang, M. Xue, W. Wei, X. Li, K. Zhang, D. Feng, B. Liu, Z. Sun, Transcriptomic signature associated with carcinogenesis and aggressiveness of papillary thyroid carcinoma, Theranostics 8 (16) (2018) 4345.
[14] X. Wang, G. Li, Q. Luo, J. Xie, C. Gan, Integrated TCGA analysis implicates lncRNA CTB-193M12. 5 as a prognostic factor in lung adenocarcinoma, Cancer Cell Int. 18 (1) (2018) 1–16.
[15] H. Shi, Y. Sun, M. He, X. Yang, M. Hamada, T. Fukunaga, X. Zhang, C. Chang, 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.
[16] C. Huang, D. Leng, S. Sun, X.D. Zhang, Re-analysis of the coral Acropora digitifera transcriptome reveals a complex lncRNAs-mRNAs interaction network implicated in Symbiodinium infection, BMC Genomics 20 (1) (2019) 1–15.
[17] Z. Jin, S. Gao, W. Ma, X. Lyu, X. Cao, Y. Yao, Identification and functional prediction of salt stress-related long noncoding RNAs in grapevine roots, Environ. Exp. Bot. 179 (2020) 104215.
[18] R. Gelhausen, S. Will, I.L. Hofacker, R. Backofen, M. Raden, IntaRNAhelix-composing RNA-RNA interactions from stable inter-molecular helices boosts bacterial sRNA target prediction, J. Bioinform. Comput. Biol. 17 (5) (2019) 1940009.
[19] L. DiChiacchio, M.F. Sloma, D.H. Mathews, AccessFold: predicting RNA–RNA interactions with consideration for competing self-structure, Bioinformatics 32 (7) (2016) 1033–1039.
[20] M. Mann, P.R. Wright, R. Backofen, IntaRNA 2.0: enhanced and customizable prediction of RNA–RNA interactions, Nucleic Acids Res. 45 (W1) (2017) W435–W439.
[21] D.D. Pervouchine, IRIS: intermolecular RNA interaction search, Genome Inform. 15 (2) (2004) 92–101.
[22] S.M. Kiełbasa, R. Wan, K. Sato, P. Horton, M.C. Frith, Adaptive seeds tame genomic sequence comparison, Genome Res. 21 (3) (2011) 487–493.
[23] J. Li, W. Ma, P. Zeng, J. Wang, B. Geng, J. Yang, Q. Cui, LncTar: a tool for predicting the RNA targets of long noncoding RNAs, Brief. Bioinform. 16 (5) (2015) 806–812.
[24] Y. Kato, K. Sato, M. Hamada, Y. Watanabe, K. Asai, T. Akutsu, RactIP: fast and accurate prediction of RNA-RNA interaction using integer programming, Bioinformatics 26 (18) (2010) i460–i466.
[25] F. Alkan, A. Wenzel, O. Palasca, P. Kerpedjiev, A.F. Rudebeck, P.F. Stadler, I.L. Hofacker, J. Gorodkin, RIsearch2: suffix array-based large-scale prediction of RNA-RNA interactions and siRNA off-targets, Nucleic Acids Res. 45 (8) (2017) e60.
[26] U. Mückstein, H. Tafer, J. Hackermüller, S.H. Bernhart, P.F. Stadler, I.L. Hofacker, Thermodynamics of RNA–RNA binding, Bioinformatics 22 (10) (2006) 1177–1182.
[27] J. González-Domínguez, M.J. Martín, MPIGeneNet: Parallel calculation of gene co-expression networks on multicore clusters, IEEE/ACM Trans. Comput. Biol. Bioinform. 15 (5) (2017) 1732–1737.
[28] Y. Lü, X. Zeng, X. Zhao, S. Li, H. Li, X. Gao, M. Xu, Fine-grained alignment of cryo-electron subtomograms based on MPI parallel optimization, BMC Bioinformatics 20 (1) (2019) 1–13.
[29] J. González-Domínguez, Y. Liu, J. Touriño, B. Schmidt, MSAProbs-MPI: parallel multiple sequence aligner for distributed-memory systems, Bioinformatics 32 (24) (2016) 3826–3828.
[30] S. Santander-Jiménez, M.A. Vega-Rodríguez, L. Sousa, Exploiting multilevel parallel metaheuristics and heterogeneous computing to boost phylogenetics, Future Gener. Comput. Syst. 127 (2022) 208–224.
[31] M.V. Shegay, D.A. Suplatov, N.N. Popova, V.K. Švedas, V.V. Voevodin, parMATT: parallel multiple alignment of protein 3D-structures with translations and twists for distributed-memory systems, Bioinformatics 35 (21) (2019) 4456–4458.
[32] K.L. Howe, P. Achuthan, J. Allen, J. Allen, J. Alvarez-Jarreta, et al., Ensembl 2021, Nucleic Acids Res. 49 (D1) (2020) D884–D891.

Iñaki Amatria-Barral received the B.S. in computer science from the Universidade da Coruña (UDC), Spain, in 2021. He also holds an M.S. in High Performance Computing from UDC since 2022. His research interests are related to the acceleration of bioinformatics tools using HPC techniques. His homepage is https://amatria.dev.

Jorge González-Domínguez received the B.S., M.S., and Ph.D. degrees in computer science from the Universidade da Coruña (UDC), Spain, in 2008, 2009, and 2013, respectively. He is currently an Associate Professor with the Department of Computer Engineering, UDC. His main research interests include the development of parallel applications on multiple fields, such as bioinformatics, data mining, and machine learning, focused on different architectures (multicore systems, GPUs, clusters, and so on). His homepage is http://gac.udc.es/~jorgeg.

Juan Touriño is a Full Professor with the Department of Computer Engineering, Universidade da Coruña, where he leads the Computer Architecture Group. He has extensively published in the area of HPC: HPC in Bioinformatics, HPC & Big Data convergence, high performance architectures and networks, HPC programming languages and compilers, parallel algorithms and applications. He is coauthor of more than 170 papers on these topics in international conferences and journals. His homepage is http://gac.udc.es/~juan.