MPI-dot2dot: A Parallel Tool to Find DNA Tandem Repeats on Multicore Clusters

Jorge González-Domínguez
Universidade da Coruña, CITIC, Computer Architecture Group, Spain

José M. Martín-Martínez
Universidad de Granada, Institute Carlos I for Theoretical and Computational Physics, Spain

Roberto R. Expósito
Universidade da Coruña, CITIC, Computer Architecture Group, Spain

Abstract

Tandem Repeats (TRs) are segments that occur several times in a DNA sequence, and each copy is adjacent to other. In the last few years, TRs have gained significant attention as they are thought to be related with certain human diseases. Therefore, identifying and classifying TRs has become a highly important task in bioinformatics in order to analyse their disorders and relationships with illnesses. *Dot2dot*, a tool recently developed to find TRs, provides more accurate results than the previous state of the art, but it requires a long execution time even when using multiple threads. This work presents *MPI-dot2dot*, a novel version of this tool that combines MPI and OpenMP so that it can be executed in a cluster of multicore nodes and thus reduces its execution time. The performance of this new parallel implementation has been tested using different real datasets. Depending on the characteristics of the input genomes, it is able to obtain the same biological results as *Dot2dot* but more than 100 times faster on a 16-node multicore cluster (384 cores). *MPI-dot2dot* is publicly available to download from `https://sourceforge.net/projects/mpi-dot2dot`.

Tandem Repeat; High Performance Computing; MPI; OpenMP; Bioinformatics

1 Introduction

A Tandem Repeat (TR) is defined in genomics as a certain number of repetitions formed by one or more bases (motif) that appear adjacent to each other. TRs are mutations generated during the DNA duplication process when a certain fragment of the sequence is replicated more than once. These structures are very common in eukaryote genomes and are considered biologically relevant. For instance, they are related to gene expression, evolution and a wide range of human diseases [15, 34]. Moreover, the identification of those proteins which are mainly based on TRs is key as they can be artificially designed [10, 35].

All these reasons have led to an increasing research interest in finding and characterizing TRs. Scientists consider that TRs are present in biologic functions that are still unknown, so much more analyses looking for TRs characterization must be made. Thanks to Next Generation Sequencing (NGS) technologies [23] the amount of available genetic and genomic data has drastically increased during the last years. Furthermore, there is a rich abundance of bioinformatics tools that can be used to detect and characterize TRs [21, 24], and newer ones continue to emerge. However, the most sensible and accurate tools usually require long computational time to analyze those large biologic datasets which can be generated nowadays through NGS technologies. Consequently, many scientists limit the size of these datasets in order to obtain the results in an affordable time, which can lead to miss interesting conclusions.

Dot2dot [12] is a recently published tool focused on detecting Short Tandem Repeats (STRs), which are related to some diseases as spinal and bulbar muscular atrophy [19], autism [33], or bipolar disorder [32]. According to the experimental results presented in [12], results provided by *Dot2dot* are more accurate than those of alternative tools present in the state of the art. In fact, this tool has already been used in real biologic experiments which have led to interesting conclusions [17, 20]. However, its main drawback is that it requires long computational time to analyze large NGS datasets, even though it provides parallel support through multithreading. These high computational requirements might force the scientists either to use a faster but less accurate tool, or to work with smaller datasets (less biologic information). Both alternatives might lead to miss interesting TRs.

In this paper we present *MPI-dot2dot*, a parallel tool for detecting STRs that provides the following contributions over the state of the art:

- Up to our knowledge, this is the first parallel application that can exploit modern High Performance Computing (HPC) multicore clusters to accelerate the search of TRs.

- The parallel approach used in *MPI-dot2dot* combines Message Passing Interface (MPI) [1] processes, which allow to work on different nodes of a distributed-memory cluster, and OpenMP [7] threads to reduce the memory overhead within each cluster node.

- *MPI-dot2dot* reduces the memory requirements of the dot plots in order to be able to analyze large datasets.

- The biological results provided by *MPI-dot2dot* are highly accurate, as they are identical to those of *Dot2dot*.
- This tool is publicly available to download and use under an open-source license from `https://sourceforge.net/projects/mpi-dot2dot`.

The rest of the paper is organized as follows. Section 2 presents previous works related to the procedure of finding TRs. Section 3 presents as background some concepts about the original *Dot2dot* tool that are necessary to understand the goal of this work and the implementation of our method. Our parallel implementation is described in Section 4. Section 5 provides the experimental evaluation. Finally, conclusions and future work lines are presented in Section 6.

2 Related Work

There has been a great effort for many years in the development of tools to search, identify and characterize TRs [21, 24], most of them focused on STRs. All these applications can be divided into the following three classes:

- Methods that perform an exhaustive search of all possible TRs. The main drawback of this approach is its high computational cost, which makes it unfeasible in most scenarios, as the number of possible combinations grows exponentially with the length of the motif. For instance, the main algorithm in *mreps* [18] follows the exhaustive approach to find all the repetitions that fulfill certain mathematical properties, but the results are then processed with an heuristic to provide only those with a biologically relevant representation. This approach is more common when searching specifically for microsatellites [2, 31], as their short length makes the exhaustive search still affordable.
- Algorithms based on a dictionary. This approach starts from a set of seeds which are later extended into strings that are searched in the sequence. These algorithms are suitable for those scenarios where only a limited list of predefined patterns must be found. Some examples of this type of tools are *TROLL* [6], *STAR* [9], and *BWtrs* [29].
- *ab-nitio* algorithms. They are non-exhaustive methods that use advanced mathematical techniques to improve the results of the search. Unlike the previous class, they do not require any previous knowledge about the input sequences. One of the most traditionally employed *ab-nitio* tool is *TRF* [4], which models the patterns that are repeated according to their similarity and the frequency of their differences. *TRF* then uses statistical criteria to select the proper patterns. Other examples of more modern *ab-nitio* tools are *tandemSWAN* [5], which is based on local autocorrelation analysis; *TRStalker* [28], which follows a multiphase algorithm where the candidates

are sorted according to a score and only the best ones reach the next phase; *MsDetector* [13] and *ULTRA* [27], based on hidden Markov models; *TAREAN* [26], with a graph-based sequence clustering phase followed by a reconstruction step from the most frequent k-mers; and *TR-ESA* [14], based on enhanced suffix arrays. Moreover, there even exist methods to find TRs on concrete scenarios such as datasets with noisy long reads [16].

As explained in the previous section, *MPI-dot2dot* is based on *Dot2dot*, which can also be categorized as an *ab-nitio* tool as it relies on an efficient data representation and uses heuristics in the search. More details about this tool can be found in Section 3 and in [12]. One important feature of *Dot2dot* is its multithreading support to reduce its runtime on parallel shared-memory systems.

Besides *Dot2dot*, we can find in the literature other parallel implementations for TR search. For instance, GPUs have been used to accelerate the analyses of TRs either thanks to linear algebra parallel routines [30] or during the sequencing procedure [8]. FPGAs have also been used to search for imperfect TRs, and to significantly improve the performance when compared to the best CPU algorithm [22]. Nevertheless, up to our knowledge, there are no available tools to accelerate the search of TRs using several nodes of an HPC cluster.

3 Background: Dot2dot

Dot2dot is a recently developed *ab-nitio* application based on heuristics to discover TRs. It provides results with better precision than other seven state-of-the-art algorithms [12]. *Dot2dot* accepts as input multisequence files obtained through NGS technologies either with FASTA or FASTQ format, and allows multithreading execution to reduce its linear computational cost. Moreover, this tool is able to discover imperfect TRs and to apply five different filtering levels, which can be selected by the user through a configuration file.

3.1 Dot plots

The name *Dot2dot* comes from the concept "dot plot", which is the basis of the tool. Dot plots are widely used in bioinformatics to compare two sequences and identify regions with high similarity. A dot plot is a bidimensional matrix where two sequences are represented in the rows and columns, respectively, and each cell receives a different colour depending on the similarity between their nucleotides. An example of a dot plot can be seen in Figure 1.

Dot2dot searches TRs in the sequences of the input dataset one by one. It creates one symmetric dot plot per sequence, where both the rows and columns are related to the same sequence. In this type of dot plots the main diagonal is always in the same colour, and there are a set of parallel secondary diagonals which form certain patterns that can be used to identify TRs. The amount of secondary diagonals represents the number of repetitions, while their length is related to the motif size. *Dot2dot* uses an efficient data representation of the

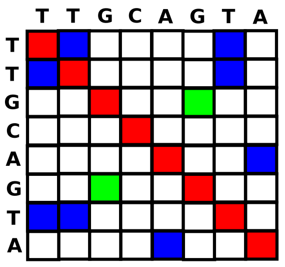

Figure 1: Example of a dot plot.

symmetric dot plots, instead of using just a bidimensional array with quadratic memory requirements. More information can be found in [12].

3.2 Search methodology

Algorithm 1 illustrates the general behavior of a *Dot2dot* execution to find the TRs of all the sequences included in an input file. The main loop in Line 3 performs the same work over different sequences: it reads the sequence (Line 4), creates the dot plot (Line 5), initializes some data structures that are necessary to represent the TRs (Lines 6 to 8), performs the proper work to discover TRs (Lines 9 to 16), and finally writes the TRs found in the corresponding output file (Line 17).

The main work for each sequence consists in searching candidate TRs. This is performed through two nested loops, which look for these candidates in every position of the sequence (Line 9) and for all the allowed motif lengths (Line 11). The best candidate for each position is selected (Line 13), but it is only included in the output list when it does not overlap to previously selected TRs (Lines 14-16).

3.3 Multithreading approach

As previously explained, *Dot2dot* performs the same work over all the sequences contained in a certain input file. The original *Dot2dot* tool includes a multithreading implementation using POSIX threads (pthreads) [25] where each thread searches for TRs on different input sequences.

Although the work done for each sentence is independent, threads must share the access to the input and output files. *Dot2dot* protects those accesses with two critical sections, one for reading the input and another one for writing the output. Moreover, the threads wait for their proper turn before writing so that the output results are sorted as in the input file (i.e. TRs related to the $i-th$ sequence in the input file must be in position i in the output file). Each critical section is implemented with a condition variable and its respective mutex. Once one thread arrives to the critical section, it checks whether the condition variable indicates its turn. Otherwise, the thread is blocked. Once one thread leaves

```
1  Input: Dataset with $N$ sequences, $minL$ and $maxL$ as minimum and
   maximum motif length
2  Output: Output file $out$ with the information of the found TRs
3  for $0 < i < N$ do
4      Read sequence $S_i$
5      $DP_i$ = CreateDotPlot($S_i$)
6      $len$ = Length($S_i$)
7      $finalTRs$ = emptyList
8      $prevTR$ = empty
9      for $0 < j < len$ do
10         $candidateTRs$ = emptyList
11         for $minL < k < maxL$ do
12             Add the TRs found with $SearchTR(DP_i, j, k)$ to $candidateTRs$
           end
13         $currentTR = Best(candidateTRs)$
14         if $currentTR$ does not overlap $prevTR$ then
15             Add $currentTR$ to $finalTRs$
16             $prevTR = currentTR$
           end
       end
17     Write $finalTRs$ into $out$
   end
```

Algorithm 1: Pseudo-code of the method applied in *Dot2dot* to find the TRs of all the sequences in a single multisequence file.

the critical section, it updates the turn in the condition variable and wakes up the other threads, which then check whether the new turn corresponds to them before continuing or being blocked again. As exposed by the authors in the supplementary material of [12], this approach for thread synchronization using turns has a negative impact on the overall performance when the number of threads increases and the length of the sequences presents high variability, which is a very common scenario in genomics datasets.

4 Implementation

MPI-dot2dot is a novel tool to accelerate the search of TRs, which provides exactly the same output results as *Dot2dot* (and, thus, its high accuracy), but at significantly lower runtime thanks to exploiting the computational capabilities of multicore clusters. These parallel computers can be defined as distributed-memory systems that consist of several nodes interconnected through a network. Each node contains a memory module and provides several CPU cores for computation (see an example in Figure 2).

MPI-dot2dot uses the same configuration mechanism as *Dot2dot* in order to simplify its adoption by those biologists that already know how to use the original tool. Both tools require by command line the paths to the input and

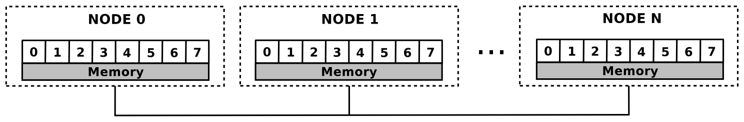

Figure 2: Example of a multicore cluster with N nodes, each one containing eight CPU cores.

output files, as well as to an additional configuration file where other optional parameters can be set (for instance, the maximum and minimum motif length). More information about this configuration file can be found in the reference manual that is available in the same public repository as the source code of the *MPI-dot2dot* tool[1].

4.1 Reduction of memory requirements for dot plots

As mentioned in Section 3.1, *Dot2dot* uses a novel data representation for the dot plots instead of a simple two dimensional array [12]. Name N the length of the sequence whose symbols must belong to the finite alphabet Σ, then this representation only needs $|\Sigma|$ vectors of length N for the values and an additional vector pointer with also length N. It means that the size of the dot plot for a certain sequence S is:

$$mem_{dp}(S) = |\Sigma| \cdot N \cdot sizeof(float) + N \cdot sizeof(ptr) = \\ (|\Sigma| \cdot sizeof(float) + sizeof(ptr)) \cdot N$$

As in most architectures *float* and pointer sizes are 4 and 8 bytes, respectively, the previous formula can be simplified to:

$$mem_{dp}(S) = (|\Sigma| \cdot 4 + 8) \cdot N$$

However, the memory requirements can be too high even with this efficient implementation, especially when increasing the number of simultaneous threads: each thread in *Dot2dot* works with a different sequence, with its respective dot plot, and the dot plots of all threads must be kept in memory at the same time. Prior to the parallel implementation, we have analyzed this data structure in order to reduce the memory requirements in *MPI-dot2dot*.

The main function used in both tools (*SearchTR()* in Algorithm 1) requires dot plots whose values are weights between 0 and 1 for each combination of base pairs. In *Dot2dot*, these weights are represented with simple-precision real numbers using the *float* datatype (4 bytes per weight). Nevertheless, the precision provided by one *float* is not necessary as the algorithm never uses more than two decimal points (100 possible values between 0.01 and 0.99). Instead, *MPI-dot2dot* uses the *char* datatype (1 byte) to represent these 100 points, reducing the total memory requirements almost by a factor of four:

$$mem_{dp}(S) = (|\Sigma| + 8) \cdot N$$

[1] `https://sourceforge.net/projects/mpi-dot2dot`

4.2 MPI parallelization

MPI-dot2dot includes MPI directives in order to search for TRs on distributed-memory systems. MPI is established as a de-facto standard for message-passing, and provides a portable, efficient and flexible mechanism to exploit this kind of parallel systems. A parallel MPI program consists of several processes with associated local memory. Each process can directly access to its local memory, but data communication must be performed if one process needs information stored in a remote memory module. The performance overhead generated by these inter-process communications heavily depends on the underlying hardware, especially on the latency and bandwidth of the interconnection cluster network (see Figure 2).

MPI-dot2dot distributes the sequences of the input file among the processes, and the process that is in charge of a certain sequence performs all the work related to it. Concretely, the input file is divided into NP groups of contiguous sequences, being NP the number of processes of the parallel program. The MPI implementation is flexible enough to work with any number of processes $NP < N$ (N represents the number of sequences in the input file).

Algorithm 2 gathers the general behavior of the MPI parallelization in *MPI-dot2dot*. The execution starts with the process established as root reading the whole input file to know the length of each sequence (Lines 4-6). This information is necessary in order to decide which sequences will be assigned to each process. Concretely, two options were implemented:

- A block distribution that assigns the same number of sequences to each process. Its main advantage is the simplification of the preprocessing step, as the root process only needs to know the total number of sequences of the input file. However, as it does not take into account the length of the sequences, it can cause workload imbalance in the common scenario where such length is highly variable.

- A balanced distribution that assigns a similar number of bases among the processes. In this case, the number of sequences per process can be significantly different, but the workload is better balanced than in the previous approach. The main drawback is that it requires the root process to completely read the input file in order to know the length of each sequence.

The function to distribute the data (Line 7) stores in two arrays the position in the input file of the first sequence assigned to each process, and the number of sequences to analyze, respectively. Then, each process obtains the information of the distribution related to it with the MPI collective *scatter* (Lines 8-9). Collectives are MPI communication routines that involve several processes. They are designed by the MPI developers in order to adapt themselves to the architecture of the cluster and then provide good performance.

Once these communications have finished, all processes start to analyze the sequences assigned to them (Lines 10-24). In this code block, only two minor

```
Input: Execution with $NP$ processes, dataset with $N$ sequences, $minL$ and
  $maxL$ as minimum and maximum motif length
Output: Output file *out* with the information of the found TRs
**if** *I am the root process* **then**
    **for** $0 < i < N$ **do**
        Read sequence $S_i$
        $seqsLen[i]$ = Length($S_i$)
    **end**
    $Dist(seqsLen, numSeqsPerProc, iniSeqPerProc)$
**end**
$MPI\_Scatter(numSeqsPerProc, ..., myNumSeqs, ...)$
$MPI\_Scatter(iniSeqPerProc, ..., myIniSeq, ...)$
**for** $myIniSeq < i < myIniSeq + myNumSeqs$ **do**
    Read sequence $S_i$
    $DP_i$ = CreateDotPlot($S_i$)
    $len$ = Length($S_i$)
    $finalTRs$ = emptyList
    $prevTR$ = empty
    **for** $0 < j < len$ **do**
        $candidateTRs$ = emptyList
        **for** $minL < k < maxL$ **do**
            Add the TRs found with $SearchTR(DP_i, j, k)$ to $candidateTRs$
        **end**
        $currentTR = Best(candidateTRs)$
        **if** *currentTR does not overlap prevTR* **then**
            Add $currentTR$ to $finalTRs$
            $prevTR = currentTR$
        **end**
    **end**
    Write $finalTRs$ into *myOut*
**end**
$MPI\_Barrier()$
**if** *I am the root process* **then**
    **for** $0 < i < NP$ **do**
        Copy *myOut* of process $i$ into *out*
    **end**
**end**
```

Algorithm 2: Pseudo-code of the MPI approach in *MPI-dot2dot*.

modifications arise with respect to Algorithm 1. On the one hand, the loop of Line 10 does not go over all sequences, but only through the ones assigned to the process. On the other hand, the TRs are not written to the final output in Line 24, but to intermediate files (a different output file for each process).

Finally, once an MPI *barrier* guarantees that all processes have finished their work and the TRs are written into their corresponding intermediate output files (Line 25), the root process merges the information of these intermediate files into the final output file of the tool (Lines 26-28). As the work is assigned to the processes in blocks of contiguous sequences, the root process only has to open once each intermediate file, and then it can copy the information into the final output with a single I/O routine. This two-level writing (first, write into intermediate files and, second, copy to the final files) brings with a performance penalization. Nevertheless, some alternatives such as sending the output information to the root process through MPI communications, or directly writing in the final files with a mutex synchronization, would lead to worse performance.

4.3 Hybrid MPI/OpenMP parallelization

In a pure MPI program each process is generally linked to one hardware CPU core. For instance, when executing *MPI-dot2dot* in a cluster such as the one shown in Figure 2, each node would have eight processes working at the same time (one process per core), each of them with its own sequence and dot plot. This approach can lead to an execution error if the memory of the node is not large enough to simultaneously store eight dot plots. The memory requirements per node increase both with the length of the sequences and with the number of processes per node.

For instance, assume a system as the one described by Figure 2 (eight cores per node) and an input dataset with sequences of equal length, whose dot plots require 20 GB of memory each. With the pure MPI parallelization described in the previous subsection we would need to map eight MPI processes to each node in order to exploit the eight available CPU cores. As each process works over a different sequence, each one has to create its own dot plot, which means that the memory requirements per node are $20 \cdot 8 = 160$ GB, which are not usually available on one single node of a cluster.

To overcome this issue, *MPI-dot2dot* includes a hybrid parallel implementation, where each MPI process can launch several threads that can collaborate in the work related to the same dot plot. The main goal of this approach is to alleviate the memory requirements per node. Using again the example of the previous paragraph, this hybrid approach would allow executions with one MPI process per node which launches eight threads. It means that the eight cores of the node would be working but the dot plot of only one sequence is stored in memory at the same time, reducing the memory requirements per node to just 20 GB.

The multithreading support is implemented with OpenMP [7], an API based on directives for platform-independent shared-memory parallel programming. Each MPI process initially only has one CPU core associated, but it is able to spawn an arbitrary number of threads that can be mapped to other cores in the same node. This hybrid parallel approach allows *MPI-dot2dot* to be executed in the cluster of Figure 2 not only with a pure MPI configuration (eight processes per node), but also with intermediate configurations such as four processes per node with two threads each.

As all threads share the resources of their parent MPI process, they can access to the same dot plot data structure. Therefore, *MPI-dot2dot* can exploit the computational resources of the whole node without requiring to store one dot plot per core. However, OpenMP can provoke situations where two or more threads need to access the same data simultaneously (race conditions). The programmer must ensure that data modifications are performed in the correct order, either using mutexes that serialize shared memory accesses or creating copies of the data for each thread (private data). Nevertheless, the use of mutexes can significantly decrease the overall performance of parallel applications, as in the multithreaded version of *Dot2dot* (see Section 3.3).

Lines 11 to 24 in Algorithm 2 represent the computation that different processes must complete for each sequence. The goal of our hybrid approach is to distribute the work for one single sequence among the different threads spawned by each process. Loops are usually the target of the OpenMP directives, assigning different iterations to each thread. Concretely, the multithreaded support in *MPI-dot2dot* is included in the loop represented by Line 16 in Algorithm 2, so that different threads can simultaneously search for candidate TRs starting in a different position of the sequence. As the dot plot is a read-only data structure stored in shared memory, all threads can independently search for these candidate TRs in their assigned positions.

Nevertheless, a dependency among threads arises as all of them must check whether the best candidate of the position overlaps to the previous TRs (Lines 20 to 23 in Algorithm 2): all threads would need to know the best candidate for positions that were analyzed by other threads. To solve this, *MPI-dot2dot* uses a double-checking mechanism that avoids any synchronization among threads. In this approach each thread has its own lists of previous, current and final TRs. In each iteration, each thread only checks the overlapping of its best TR to those included in its private list (i.e., only to those best candidate TRs which were found in positions of the sequence assigned to the same thread). Once all threads have finished their work, there are several final lists (one per thread). The main thread of the process performs a second check among the TRs included in all those lists in order to guarantee that no TR overlaps with a previous one, even though they were found by different threads.

This double-check approach brings with certain memory overhead, as *MPI-dot2dot* creates one copy of all the TRs lists per thread. Nevertheless, its impact is almost negligible compared to the amount of memory needed to store the dot plots. Moreover, the performance penalization due to the second check is significantly lower than other alternatives such as including synchronizations between Lines 20 and 21 in Algorithm 2 to guarantee that the best candidate TRs of all threads have been calculated prior to the overlapping check.

The loop of Line 18 in Algorithm 2, which searches for TRs of different length

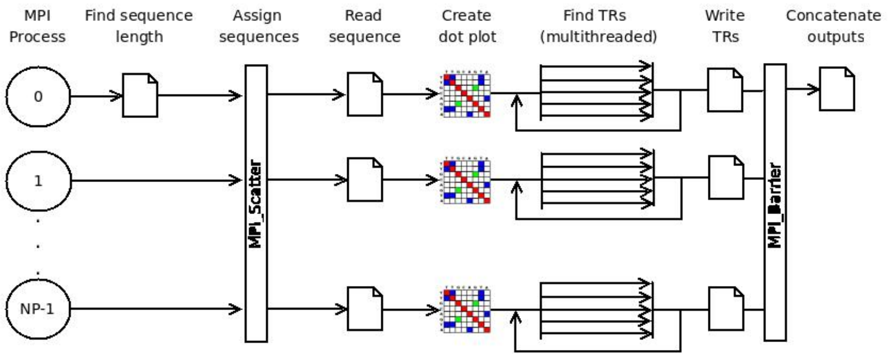

Figure 3: Workflow of *MPI-dot2dot*.

for the same position of the sequence, was discarded as target of parallelization with OpenMP directives as it would require one thread synchronization per position, significantly decreasing performance. Moreover, this loop is generally short (less than 100 iterations) which would reduce the potential benefit of using several threads.

Figure 3 illustrates the approach followed by this hybrid parallel implementation. Although this algorithm presents two sources of potential performance overhead related to I/O functions in the root process, it obtains good performance as communications and synchronizations are minimal.

Finally, remark that the multithreaded approach used in *Dot2dot* (explained in Section 3.3) was discarded as it distributes the sequences among threads, similarly to our MPI parallelization described in Section 4.2. Therefore, each thread would need its own dot plot, and thus the same memory problems exposed at the beginning of this section for the pure MPI approach would also arise.

5 Experimental evaluation

This section presents the experimental evaluation, which is focused on performance in terms of execution time, as the results of *MPI-dot2dot* are identical to those of the original *Dot2dot*, and thus their accuracy has already been proved in [12].

Four datasets, with assemblies of real genomes downloaded from the GenBank in the National Center for Biotechnology Information (NCBI) website [3], are used for evaluation. Their characteristics are shown in Table 1. As previously mentioned, these real datasets contain sequences whose length is highly variable. This table also shows the maximum memory requirements for the original tool and for *MPI-dot2dot*, i.e., the size of the dot plot corresponding to the longest sequence. As explained in Section 4.1, the modification of the datatype in the dot plots reduces the memory requirements of *MPI-dot2dot*.

Table 1: Dataset description.

Organism	Number of sequences	Millions of bases	Max Mem (MB) *Dot2dot*	Max Mem (MB) *MPI-dot2dot*
Picea Glauca (PG)	4,464,856	21,582	8.11	3.13
Picea Engelmannii (PE)	2,394,260	24,943	333.73	128.94
Pinus Lambertiana (PL)	4,253,097	27,602	170.54	65.89
Ambystoma Mexicanum (AM)	125,724	32,396	85,188.98	32,913.92

5.1 Configuration of the experiments

All the experiments were carried out in 16 nodes of the Finis Terrae II supercomputer, installed at the Galician Supercomputing Center (CESGA) [11]. Each node consists of 24 cores (two 12-core Intel Xeon Haswell E5-2680 processors) and 128 GB of memory. They are interconnected through a low-latency and high-bandwidth InfiniBand FDR network (56 Gbps). Regarding software settings, *gcc* version 6.4.0 (which includes support for pthreads and OpenMP) with the -O3 flag was used for the compilation of both *Dot2dot* and *MPI-dot2dot*, while the support for distributed-memory execution is provided by the open-source Open MPI library version 2.1.1.

In order to provide a fair comparison, *Dot2dot* and *MPI-dot2dot* are executed with exactly the same configuration. In general, the by-default parameters were used. The only exception was the maximum motif length, which is changed from 30 (default option) to 50 bases in both tools. Increasing the value of this parameter can provide interesting results in some scenarios, at the cost of longer execution time. Therefore, it makes sense to increase this value in HPC systems.

5.2 Experimental results

The experimental evaluation starts with a comparison of *Dot2dot* and *MPI-dot2dot* in a single node, as the original tool only provides support for parallel computation on shared-memory systems. Figure 4 compares the best runtime that can be obtained by *Dot2dot* on a single 24-core node of the Finis Terrae II with different configurations for *MPI-dot2dot*: only one MPI process and 24 threads (multithreaded); pure MPI execution with 24 processes using the block distribution (block); pure MPI execution with 24 processes using the balanced distribution (balanced); and the best configuration of processes and threads (hybrid). The MPI processes in the hybrid execution use the balanced workload distribution, as can be seen that its performance is significantly higher than using the block distribution in real genomes, whose sequences are very variable in length.

The conclusions that can be obtained from these results depend on the characteristics of the input datasets (see Table 1). The first group would consists of the *Picea Glauca*, *Picea Engelmannii* and *Pinus Lambertiana* genomes, all of them containing millions of sequences with some thousands of base pairs as average length. All the *MPI-dot2dot* versions are able to exploit the 24 cores

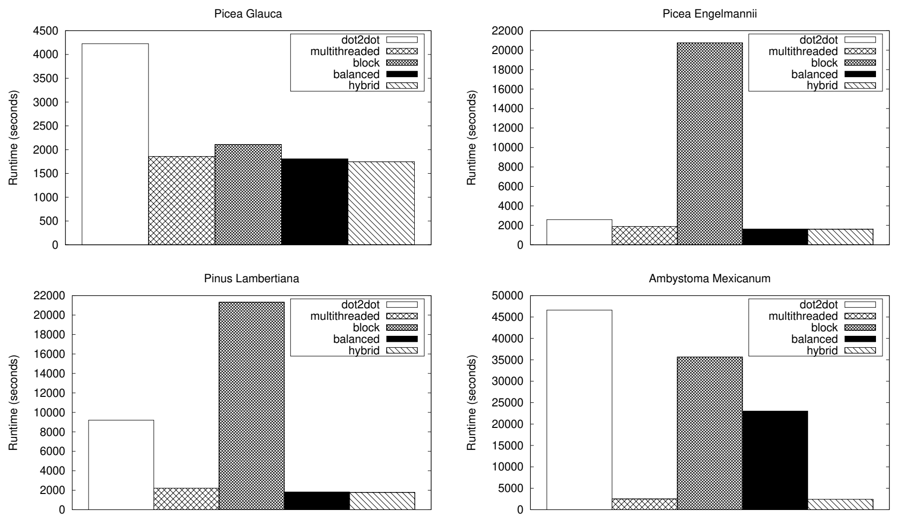

Figure 4: Performance comparison of *Dot2dot* and different configurations of *MPI-dot2dot* on a single node.

of the node, as well as *Dot2dot* with its multithreaded implementation. Several conclusions can be remarked from these datasets:

- The synchronizations required by the inter-sequence multithreaded approach of *Dot2dot*, where complete sequences are assigned to different threads (see Section 3.3), lead to significant performance overhead, especially for the two datasets with more variability in the sequence length (*Picea Glauca* and *Pinus Lambertiana*). This overhead is significantly reduced with the intra-sequence multithreaded version of *MPI-dot2dot*. Consequently, the latter is 2.60 times faster on average, with a maximum speedup of 4.15 for the *Pinus Lambertiana* genome.

- The balanced MPI distribution obtains much better results than the block one that assigns the same number of sequences per process. This behavior was expected as the length of the sequences is highly variable in these real genomic datasets. The results of the block distribution are only competitive for the *Picea Glauca* genome, which presents quite regular lengths, but even in this case they are worse than using the balanced approach.

- The pure MPI implementation that balances workload distribution achieves better performance than any multithreaded approach. Concretely, it is on average 2.99 and 1.13 times faster than the multithreaded versions of *Dot2dot* and *MPI-dot2dot*, respectively, using the 24 cores of one whole node.

- The hybrid MPI/OpenMP implementation with a correct configuration of processes and threads can further improve performance, but its performance gain is not quite high (on average, 2.72%).

On the other hand, the genome of the *Ambystoma Mexicanum* is the largest one but with only 125,000 sequences. Therefore, sequences are significantly longer on average than in the other three datasets. It means that the amount of memory required to create and store the dot plots is also larger. In fact, there is no enough space in the memory of one Finis Terrae II node to store several instances of dot plots. Consequently, *Dot2dot* can only use one thread (and one core for computation), while the pure MPI implementations fail when using more than two processes per node. The intra-sequence multithreaded approach included in the hybrid implementation of *MPI-dot2dot* not only outperforms the inter-sequence one implemented in *Dot2dot*, but also reduces the memory requirements significantly, and thus it can analyze this dataset using 24 threads. Then, the benefit of using *MPI-dot2dot* in this type of datasets is impressive, even when using only one node: while *Dot2dot* requires almost 13 hours to analyze this dataset (it can only exploit one core), our novel tool is able to complete the same work in just 40 minutes by exploiting the whole node (19.07 times faster).

In order to determine the best configuration of processes/threads for the hybrid implementation, previous experiments with different options were executed, whose results are shown in Table 2. The configuration using 12 processes, which only spawn two threads each, obtained the best performance for the three datasets with less average sequence length. This is another proof that the balanced MPI distribution is good enough to efficiently exploit the whole node in those scenarios, as no many threads are required. The only exception is the *Ambystoma Mexicanum* dataset. Again, memory problems arise when analyzing this dataset with several processes in the same node due to the length of some sequences and, consequently, the requirements of their dot plots exceeds the memory available in one node. Concretely, no more than two MPI processes with their corresponding dot plots can be mapped to the same node. The hybrid approach is more beneficial for this type of datasets. For instance, the best results for the *Ambystoma Mexicanum* are obtained using two processes and 12 threads per process.

Although being faster than *Dot2dot* on one node with shared memory, the main advantage of *MPI-dot2dot* is that it can exploit distributed-memory systems to further reduce runtime. Figure 5 shows the evolution of the speedup compared to the best *Dot2dot* execution (when possible, using the 24 cores of one node) when increasing the number of nodes up to 16. *MPI-dot2dot* was executed with the hybrid implementation using the balanced workload distribution and the best processes/threads configuration according to the experiments shown in Table 2. The performance benefit of *MPI-dot2dot* increases with the number of nodes, so that the parallel implementation scales properly with the amount of hardware resources used.

Table 3 summarizes the performance improvement of *MPI-dot2dot* over *Dot2dot*. As can be seen, our tool is significantly faster than the original one for all

Table 2: Runtime (in seconds) in one node of the Finis Terrae II supercomputer obtained by the hybrid parallel implementation of *MPI-dot2dot* with balanced workload distribution and using different configurations of threads and processes. Symbol "-" means that the execution did not finish due to memory problems. The best configuration for each dataset is highlighted in bold letter.

Config.	*PG*	*PE*	*PL*	*AM*
1P - 24Th	1,857	1,886	2,214	2,512
2P - 12Th	1,831	1,810	1,957	**2,444**
4P - 6Th	1,864	1,688	1,852	-
8P - 3Th	1,779	1,615	1,794	-
12P - 2Th	**1,746**	**1,598**	**1,768**	-
24P - 1Th	1,808	1,625	1,824	-

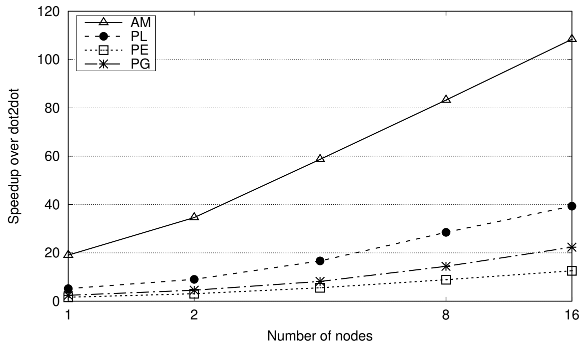

Figure 5: Speedups of the best *MPI-dot2dot* configuration compared to *Dot2dot*. The baseline for the *Ambystoma Mexicanum* is the runtime of the original tool using only one core, as it fails when using multiple threads due to the high memory requirements.

datasets, even when using the same hardware (one node of the Finis Terrae II supercomputer). The performance difference is more remarkable for datasets with long sequences, where *Dot2dot* cannot be executed with multiple threads due to its high memory requirements. Moreover, *MPI-dot2dot* can be executed on 16 nodes of the supercomputer, proving that the MPI implementation can further reduce runtime. For instance, it only needs around 7 minutes to find the TRs of the *Ambystoma Mexicanum* genome, while *Dot2dot* requires almost 13 hours.

Finally, remark that the output of *MPI-dot2dot* and *dot2dot* was identical for all the experiments carried out during this experimental evaluation, which proves that the accuracy of the parallel version is as high as the original tool.

Table 3: Best runtime of both tools. *Dot2dot* was executed on one whole node except for the *Ambystoma Mexicanum* dataset, when only single-core execution was possible. *MPI-dot2dot* was executed for a varying number of nodes.

	Dot2dot	*MPI-dot2dot*				
		1 node	2 nodes	4 nodes	8 nodes	16 nodes
PG	1h 10m	29m 6s	15 m 31s	8m 37s	4m 53s	3m 8s
PE	43m	26m 38s	14m 1s	7m 48s	4m 51s	3m 26s
PL	2h 33m	29m 28s	17m 1s	9m 12s	5m 23s	3m 54s
AM	12h 57m	41m 44s	22m 27s	13m 14s	9m 20s	7m 9s

6 Conclusions and future work

It is believed that the identification of TRs can have great positive impact in the diagnostic and treatment of genetic diseases. Tools to efficiently find these TRs on large genomic datasets are thus required. This work presented *MPI-dot2dot*, a parallel application that obtains the same biologic results as the previously tested *Dot2dot* tool, but at significantly reduced runtime thanks to fully exploiting the hardware of modern multicore clusters.

MPI-dot2dot is based on a hybrid MPI/OpenMP parallel implementation. On the one hand, the MPI routines allow the exploitation of distributed-memory systems thanks to a balanced workload distribution that assigns similar number of bases per MPI process. On the other hand, the OpenMP directives are included within the function that searches for TRs in a certain sequence, significantly reducing the memory requirements compared to using several processes in the same node.

The experimental evaluation was performed on 16 nodes of the Finis Terrae II supercomputer (a total of 384 cores) using four datasets with real genomes and different characteristics. *MPI-dot2dot* is faster than *Dot2dot* in all scenarios, even using the same hardware resources. Our experiments also determined that *MPI-dot2dot* is more beneficial for datasets with long sequences, where *Dot2dot* can only be executed with one thread due to memory problems. For instance, the original tool needed almost 13 hours to analyze the *Ambystoma Mexicanum* genome, while *MPI-dot2dot* was able to find the same TRs in only 7 minutes using 16 nodes of the supercomputer (i.e., 108 times faster).

As future work, we will study the possibility of using Big Data processing frameworks such as Hadoop or Spark in order to further accelerate the search of TRs in different types of distributed-memory systems. Moreover, we will try to develop an autotuning technique to provide information in advance about the possible best combination of MPI processes and OpenMP threads depending on the characteristics of both the hardware and the input dataset.

Acknowledgements

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 (Centro de Investigación de Galicia accreditation 2019-2022 and Consolidation Program of Competitive Reference Groups, under Grants ED431G 2019/01 and ED431C 2021/30, respectively). The authors would like to thank the Galician Supercomputing Center (CESGA) for providing access to the Finis Terrae II supercomputer.

References

[1] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard Version 3.1 (2015). [Online] Available: `http://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf`

[2] Avvaru, A.K., Sowpati, D.T., Mishra, R.K.: PERF: an Exhaustive Algorithm for Ultra-Fast and Efficient Identification of Microsatellites from Large DNA Sequences. Bioinformatics **34**(6), 943–948 (2018)

[3] Benson, D.A., Cavanaugh, M., Clark, K., Karsch-Mizrachi, I., Lipman, D.J., Ostell, J., Sayers, E.W.: GenBank. Nucleic Acids Research **41**(D1), D36–D42 (2012)

[4] Benson, G.: Tandem Repeats Finder: a Program to Analyze DNA Sequences. Nucleic Acids Research **27**(2), 573–580 (1999)

[5] Boeva, V., Regnier, M., Papatsenko, D., Makeev, V.: Short Fuzzy Tandem Repeats in Genomic Sequences, Identification, and Possible Role in Regulation of Gene Expression. Bioinformatics **22**(6), 676–684 (2006)

[6] Castelo, A.T., Martins, W., Gao, G.R.: TROLL-Tandem Repeat Occurrence Locator. Bioinformatics **18**(4), 634–636 (2002)

[7] Dagum, L., Menon, R.: OpenMP: an Industry Standard API for Shared-Memory Programming. Computational Science & Engineering, IEEE **5**(1), 46–55 (1998)

[8] De Roeck, A., De Coster, W., Bossaerts, L., Cacace, R., De Pooter, T., Van Dongen, J., D'Hert, S., De Rijk, P., Strazisar, M., Van Broeckhoven, C., et al.: NanoSatellite: Accurate Characterization of Expanded Tandem Repeat Length and Sequence through Whole Genome Long-Read Sequencing on PromethION. Genome Biology **20**(1), 239 (2019)

[9] Delgrange, O., Rivals, E.: STAR: an Algorithm to Search for Tandem Approximate Repeats. Bioinformatics **20**(16), 2812–2820 (2004)

[10] Doyle, L., Hallinan, J., Bolduc, J., Parmeggiani, F., Baker, D., Stoddard, B.L., Bradley, P.: Rational Design of α-helical Tandem Repeat Proteins with Closed Architectures. Nature **528**(7583), 585–588 (2015)

[11] Galician Supercomputing Center: CESGA. [Online] Available: `https://www.cesga.es`. Last visited: August 2021

[12] Genovese, L.M., Mosca, M.M., Pellegrini, M., Geraci, F.: Dot2dot: Accurate Whole-Genome Tandem Repeats Discovery. Bioinformatics **35**(6), 914–922 (2019)

[13] Girgis, H.Z., Sheetlin, S.L.: MsDetector: Toward a Standard Computational Tool for DNA Microsatellites Detection. Nucleic Acids Research **41**(1), e22–e22 (2013)

[14] Gupta, S., Prasad, R.: Searching Exact Tandem Repeats in DNA Sequences Using Enhanced Suffix Array. Current Bioinformatics **13**(2), 216–222 (2018)

[15] Hannan, A.J.: Tandem Repeats Mediating Genetic Plasticity in Health and Disease. Nature Reviews Genetics **19**(5), 286 (2018)

[16] Harris, R.S., Cechova, M., Makova, K.D.: Noise-Cancelling Repeat Finder: Uncovering Tandem Repeats in Error-Prone Long-Read Sequencing Data. Bioinformatics **35**(22), 4809–4811 (2019)

[17] Kinkar, L., Korhonen, P.K., Cai, H., Gauci, C.G., Lightowlers, M.W., Saarma, U., Jenkins, D.J., Li, J., Li, J., Young, N.D., et al.: Long-Read Sequencing Reveals a 4.4 kb Tandem Repeat Region in the Mitogenome of Echinococcus Granulosus (sensu stricto) Genotype G1. Parasites & Vectors **12**(1), 1–7 (2019)

[18] Kolpakov, R., Bana, G., Kucherov, G.: mreps: Efficient and Flexible Detection of Tandem Repeats in DNA. Nucleic Acids Research **31**(13), 3672–3678 (2003)

[19] La Spada, A.R., Wilson, E.M., Lubahn, D.B., Harding, A., Fischbeck, K.H.: Androgen Receptor Gene Mutations in X-Linked Spinal and Bulbar Muscular Atrophy. Nature **352**(6330), 77–79 (1991)

[20] Li, Z., Li, M., Xu, S., Liu, L., Chen, Z., Zou, K.: Complete Mitogenomes of Three Carangidae (Perciformes) Fishes: Genome Description and Phylogenetic Considerations. International Journal of Molecular Sciences **21**(13), 4685 (2020)

[21] Lim, K.G., Kwoh, C.K., Hsu, L.Y., Wirawan, A.: Review of Tandem Repeat Search Tools: a Systematic Approach to Evaluating Algorithmic Performance. Briefings in Bioinformatics **14**(1), 67–81 (2013)

[22] Martínek, T., Lexa, M.: Hardware Acceleration of Approximate Tandem Repeat Detection. In: Proceedings of the 2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM '10), pp. 79–86 (2010)

[23] McCombie, W.R., McPherson, J.D., Mardis, E.R.: Next-Generation Sequencing Technologies. Cold Spring Harbor Perspectives in Medicine **9**(11), a036798 (2019)

[24] Merkel, A., Gemmell, N.: Detecting Short Tandem Repeats from Genome Data: Opening the Software Black Box. Briefings in Bioinformatics **9**(5), 355–366 (2008)

[25] Nichols, B., Buttlar, D., Farrell, J.P.: Pthreads Programming: A POSIX Standard for Better Multiprocessing, vol. 19 (1996)

[26] Novák, P., Ávila Robledillo, L., Koblížková, A., Vrbová, I., Neumann, P., Macas, J.: TAREAN: a Computational Tool for Identification and Characterization of Satellite DNA from Unassembled Short Reads. Nucleic Acids Research **45**(12), e111–e111 (2017)

[27] Olson, D., Wheeler, T.: ULTRA: a Model Based Tool to Detect Tandem Repeats. In: Proceedings of the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics (BCB '18), pp. 37–46 (2018)

[28] Pellegrini, M., Renda, M.E., Vecchio, A.: TRStalker: an Efficient Heuristic for Finding Fuzzy Tandem Repeats. Bioinformatics **26**(12), i358–i366 (2010)

[29] Pokrzywa, R., Polanski, A.: BWtrs: a Tool for Searching for Tandem Repeats in DNA Sequences Based on the Burrows–Wheeler Transform. Genomics **96**(5), 316–321 (2010)

[30] Samsi, S., Helfer, B., Kepner, J., Reuther, A., Ricke, D.O.: A Linear Algebra Approach to Fast DNA Mixture Analysis Using GPUs. In: Proceedings of the 2017 IEEE High Performance Extreme Computing Conference (HPEC '17), pp. 1–6 (2017)

[31] Savari, H., Hadiniya, N., Savadi, A., Naghibzadeh, M.: Microsatellite Finder Algorithm with High Memory Efficiency for Even Super Long Sequences. In: Proceedings of the 2020 10th International Conference on Computer and Knowledge Engineering (ICCKE), pp. 1–5 (2020)

[32] Song, J.H., Lowe, C.B., Kingsley, D.M.: Characterization of a Human-Specific Tandem Repeat Associated with Bipolar Disorder and Schizophrenia. The American Journal of Human Genetics **103**(3), 421–430 (2018)

[33] Trost, B., Engchuan, W., Nguyen, C.M., Thiruvahindrapuram, B., Dolzhenko, E., Backstrom, I., Mirceta, M., Mojarad, B.A., Yin, Y., Dov, A., et al.: Genome-Wide Detection of Tandem DNA Repeats that Are Expanded in Autism. Nature **586**(7827), 80–86 (2020)

[34] Usdin, K.: The Biological Effects of Simple Tandem Repeats: Lessons from the Repeat Expansion Diseases. Genome Research **18**(7), 1011–1019 (2008)

[35] Voet, A.R., Simoncini, D., Tame, J.R., Zhang, K.Y.: Evolution-Inspired Computational Design of Symmetric Proteins. In: Computational Protein Design, pp. 309–322. Springer (2017)