MPIGeneNet: Parallel Calculation of Gene Co-Expression Networks on Multicore Clusters

Jorge González-Domínguez* and María J. Martín

Abstract—In this work we present *MPIGeneNet*, a parallel tool that applies Pearson's correlation and Random Matrix Theory to construct gene co-expression networks. It is based on the state-of-the-art sequential tool *RMTGeneNet*, which provides networks with high robustness and sensitivity at the expenses of relatively long runtimes for large scale input datasets. *MPIGeneNet* returns the same results as *RMTGeneNet* but improves the memory management, reduces the I/O cost, and accelerates the two most computationally demanding steps of co-expression network construction by exploiting the compute capabilities of common multicore CPU clusters. Our performance evaluation on two different systems using three typical input datasets shows that *MPIGeneNet* is significantly faster than *RMTGeneNet*. As an example, our tool is up to 175.41 times faster on a cluster with eight nodes, each one containing two 12-core Intel Haswell processors. Source code of *MPIGeneNet*, as well as a reference manual, are available at https://sourceforge.net/projects/mpigenenet/.

Index Terms—Genetics, Gene Co-Expression Networks, Random Matrix Theory, High Performance Computing, MPI, Multithreading.

1 Introduction

The use of expression data, with values for different genes and samples, to construct co-expression networks has been proved very useful in biological analyses [1]. These networks can be used to illustrate the complex interactions that can be present among multiple genes, where the nodes and edges represent genes and interesting correlations, respectively. There exist several methods to construct co-expression networks. Most of them start by creating a so-called correlation or similarity matrix, i.e., a two-dimensional triangular matrix where each value is the similarity coefficient of one gene pair. Some examples of measures used in this step are Pearson's [2], Spearman [3], Theil-Sen [4] or Kendall [5] correlations. Those correlation values higher than a certain threshold represent an interaction between the two genes of the pair. Several techniques can be used to identify the correct threshold according to the values of the similarity matrix, e.g., permutation testing [6], linear regression [7], spectral graph theory [8], Fisher's tests [9], machine learning [10] or null models [2].

RMTGeneNet [11] is a cutting-edge tool to construct gene co-expression networks, which uses Pearson's correlation to generate a similarity matrix and calculates a Random Matrix Theory (RMT) threshold to discard non biologically relevant interactions. The main strength of this tool is the high sensitivity and robustness of its resulting networks, which makes it a good choice for interesting biological studies [12], [13], [14], [15]. However, its main drawback is its relatively high runtime for input datasets consisting of thousands of genes, which prevents the widely adoption of this method by the scientific community.

In this paper we present *MPIGeneNet*, an application that is able to accelerate the construction of co-expression networks on modern multicore clusters by applying parallel computing to both the calculation of the Pearson's correlation matrix and the identification of the RMT threshold. It means that *MPIGeneNet* is able to obtain the same robust and sensitive networks as *RMTGeneNet* but in significantly shorter time. It uses a hybrid approach that combines Message Passing Interface (MPI) processes and threads. Each MPI process launches multiple threads to efficiently exploit the cores available on each node and to reduce the memory requirements. Moreover, the one-sided asynchronous communications included in MPI 3 are used to overlap communication and computation and increase the scalability on several nodes.

The rest of the paper is organized as follows. Section 2 presents previous works related to the parallelization of the procedure to construct co-expression networks. Section 3 describes the parallel implementation of the different parts of *MPIGeneNet*. Section 4 provides the experimental evaluation in terms of runtime and scalability. Finally, concluding remarks are presented in Section 5.

• *J. González-Domínguez and M J. Martín are with the Grupo de Arquitectura de Computadores, Universidade da Coruña, Spain. E-mail: {jgonzalezd,mariam}@udc.es*

2 Related Work

As previously mentioned, *MPIGeneNet* follows the same approach as *RMTGeneNet* [11] to construct the co-expression matrices, i.e., calculation of the Pearson's correlation matrix and posterior filtering with an RMT threshold. Besides genetics, RMT has also been satisfactorily applied to other fields such as finance [16], neuroscience [17], sensor networks [18] or superconductors [19]. Up to our knowledge, although there exist previous efforts to accelerate the calculation of the RMT threshold on multicore clusters [20] and GPUs [21], these works only focus on the parallelization of the eigenvalues calculation, obtain low speedups and are not publicly available.

Acceleration of the co-expression networks construction on high performance computing systems has already been addressed, but all available tools apply approaches to discard non relevant gene interactions that are more simple than the RMT filter and they do not obtain networks with the same level of robustness and sensitivity. Concretely, both *TINGe* [22] and its Xeon Phi-based counterpart [23] use Mutual Information as score measure for the similarity matrix, while permutation testing for thresholding.

Finally, some works only provide a parallel implementation of the first step (calculation of the correlation matrices), assuming that it will be the most computationally demanding phase. However, as will be shown in Section 4, calculation of a good filtering threshold might also require long runtime. Some examples are *FastGCN* [24] and *CUDA-MI* [25] for GPUs, *LightPCC* [26] for several Intel Xeon Phi coprocessors and *MPICorMat* [27] for multicore clusters. This last tool can be seen as a basis for *MPIGeneNet* as its functionality is included our tool. However, it has been significantly optimized using MPI 3 one-sided communication routines and extended with a parallel calculation of the RMT threshold.

3 Implementation

MPIGeneNet receives as input a file with the expression values for each gene and each sample. These expression data are stored in a $n \times m$ matrix (expression matrix), being n the number of genes and m the number of samples. The input file, as well as other configuration parameters, are specified in the command line. An explanation of all the arguments, as well as installation instructions, are included in the reference manual available with the tool. *MPIGeneNet* consists of three stages:

1) Construction of a two-dimensional matrix that includes the Pearson's correlation value for all gene pairs (similarity matrix).
2) Calculation of the RMT threshold.
3) Generation of the co-expression network by discarding those edges of the similarity matrix with correlation values lower than the threshold.

Instead of following the same approach as *RMTGeneNet*, which provides one different module for each step and requires three systems calls from the user, *MPIGeneNet* integrates the whole functionality in one program. This makes the tool easier to work with (the users only have to launch the application once) and avoids writing/reading from intermediate files among the modules. The first two stages are accelerated with a two-level parallelization. On the one hand, the workload is distributed among several MPI processes, which can work on different nodes of a compute cluster connected trough a network. MPI is established as a de-facto standard to develop programs for distributed-memory systems, and provides a portable, efficient, and flexible approach for message-passing. On the other hand, each process can launch several threads to exploit the computational capabilities of all the cores within each node. This hybrid parallel approach has been satisfactorily applied to other bioinformatics problems such as the removal of duplicate DNA sequences [28] or to multiple sequence alignment [29]. Remark that our implementation is flexible enough to allow the users to specify the desired number of MPI processes and threads. We discarded the parallelization of the last step as its impact in the runtime is almost negligible (less than 0.3% of the total runtime for all our sequential experiments).

Moreover, before addressing the parallelization, our work started by optimizing memory accesses on the three steps so that *MPIGeneNet* is significantly faster than the *RMTGeneNet* even running on one single core.

3.1 Parallel Construction of Pearson's Correlation Matrix

MPIGeneNet starts with all MPI processes reading in parallel the input file and saving their own copy of the initial expression matrix. Although this data replication leads to memory overhead, it is sometimes present in parallel computing to optimize performance by avoiding communication [30], [31]. In order to limit the memory overhead, *MPIGeneNet* does not create one MPI process per core (each one with its own copy of the expression matrices). Instead, each process is related to a group of cores and launches several OpenMP threads that are able to access shared memory, use the same copy of the matrix and collaborate to calculate the correlation of the gene pairs assigned to their parent process.

Regarding the workload distribution, as the Pearson's correlation must be calculated for all gene pairs, the workload of this step can be represented with a 2D matrix, where both axis x and y include all genes. Each point in the 2D subspace represents one gene pair. As correlation is a symmetric measure, only half of the matrix must be calculated. Concretely $\frac{n \times (n-1)}{2}$

Algorithm 1: Pseudo-code of the hybrid parallel algorithm on each process to construct the partial correlation matrix.

```
1  Read input expression matrix M
2  Calculate myIniRow and myLastRow
3  Create MPI_Window S for the whole correlation
     matrix and accessible to all processes
4  Initialize matrix of private scores myS := 1
5  Initialize iterator iter := 0
6  #pragma omp parallel for schedule(dynamic)
7  for each row i from myIniRow to myLastRow do
8  |   for each column j from 0 to i − 1 do
9  |   |   myS[iter] := CalcPearson(i, j) # GSL routine
10 |   |   iter++
   |   end
11 |   iter++ # Score for diagonal elements is 1.0
   end
12 for each process p do
13 |   Put myS in the correct position in the S that
   |     belongs to p
   end
```

Algorithm 2: Pseudo-code of the hybrid parallel algorithm on each process to calculate the RMT threshold.

```
1  Input: Correlation scores matrix S; First threshold to
     check iniThres; Threshold stride st; Process id myId;
     Number of processes NP
2  Create MPI_Window end with one integer accessible
     to all processes
3  Initialize end := 0
4  myThres := iniThres − myId × st
5  if myId == 0 then
6  |   Create MPI_Window RMT with one float
   |     accessible to all processes
   end
7  repeat
8  |   Build cut matrix C with the points of S higher
   |     than myThres
9  |   E := CalcEigenv(C) # Multithreaded MKL routine
10 |   if (E fulfills Chi-square test) & (end == 0) then
11 |   |   Put myThres in RMT
12 |   |   for each process p do
13 |   |   |   # Inform to p that the threshold was found
14 |   |   |   Put 1 in window end that belongs to p
   |   |   end
   |   end
   |   else
15 |   |   myThres := myThres − st × NP
   |   end
   until end == 1
```

pairs. *MPIGeneNet* divides the gene pairs among the processes so that they simultaneously work over different pairs. Pairs are assigned by blocks of rows (the whole row to the same process) with variable number of rows per block to balance the number of pairs assigned to each process. We refer to the *MPICorMat* [27] publication for further information about the workload distribution and the hybrid MPI/OpenMP implementation of this step.

An additional feature included in *MPIGeneNet* over *MPICorMat* is an efficient gather of the resulting similarity matrix in all processes using the support for Remote Memory Access (RMA) one-sided communications included in MPI since its 3.0 version. These kind of routines have been proved more efficient than traditional two-sided communication on several scenarios, thanks to avoiding synchronizations between source and destination processes [32], [33]. Concretely, we employ a `push` approach where:

1) Each process creates a MPI window with enough size to store the whole correlation matrix previously to the computation.
2) Once one process finishes its partial computation, it copies its fragment of correlation matrix in the correct position in each window.

This method generates a total of p^2 calls to the MPI function `Put`, does not require any synchronization and allows to overlap communication with computation (the destination processes can continue with the calculation of their partial similarity matrices while receiving other fragments).

Algorithm 1 summarizes the *MPIGeneNet* parallel approach to calculate the correlation matrix.

3.2 Parallel Calculation of the RMT Threshold

In order to filter non-relevant gene interactions from the correlation matrix, *MPIGeneNet* calculates a threshold using RMT and parallel computing as illustrated in Algorithm 2. According to RMT, the Nearest Neighbour Spacing Distribution (NNSD) of eigenvalues in real symmetric matrices follows Gaussian Orthogonal Ensemble (GOE) statistics if there exists correlation between nearest-neighbour eigenvalues, while it follows Poisson statistics if there is no correlation. Therefore, *MPIGeneNet* selects as threshold the point of the NNSD transition from Poisson to Gaussian, as it represents the point where real correlation exists for our genes. We refer to [34] for more information about RMT and its adequacy to select biologically relevant interactions.

To determine this point of transition *MPIGeneNet* iterates through successively smaller correlation thresholds. In each iteration it creates the so-called cut matrix (Line 8), with only those values of the correlation matrix higher than the analyzed threshold, and calculates the eigenvalues of such cut matrix (Line 9). Once these eigenvalues are calculated, they are provided as input to a Chi-square test (Line 10) that indicates whether the transition point has been reached and, thus, the threshold has been found. The eigenvalues calculation is the main performance bottleneck of each iteration and it is performed using the `ssyev` routine available in the MKL library [35]. The initial threshold and the stride employed for each iteration ($iniThres$

and st in Algorithm 2) are specified by the user through command line.

The algorithm is parallelized using a hybrid approach that combines MPI and the multithreaded MKL library. As the computation within different iterations is independent, the workload of this second stage is distributed by assigning different iterations to each MPI process in a round-robin way. This cyclic distribution is suitable for two reasons:

- We do not know in advance the number of iterations that will be necessary to find the RMT threshold.
- The work necessary to calculate the eigenvalues increases with lower thresholds (i.e., it becomes more expensive after each iteration).

Each MPI process works over its own copy of the similarity matrix (gathered at the end of the first step, as explained in the previous subsection), which is a good approach in order to avoid communications during the RMT iterations. An alternative to reduce memory requirements would be to use a distributed correlation matrix (each process has the same fragment as at the end of the first step). However, this version has been discarded as it would be very inefficient in terms of performance. Communications would be required in each iteration of the loop presented in Algorithm 2, as processes need the whole similarity matrix to calculate the eigenvalues for each threshold. This would lead to a non-affordable communication runtime overhead. Instead, the correlation matrix is replicated on each MPI process and the memory overhead is alleviated in *MPIGeneNet* by making use of the multithreading support of the MKL library.

Once one process finds the threshold that passes the Chi-square tests, it must send its value to Process 0, the only one in charge of the next step: discarding those edges of the similarity matrix with correlation values lower than the threshold. It must also indicate to all other processes that the goal has been achieved and they do not need to continue searching. This finalization procedure is also optimized using the RMA support provided by MPI. At the beginning of this step every process initializes with value 0 its own variable end, accessible to all process as a MPI_Window (Lines 2-3). Similarly, Process 0 creates a shared variable RMT to save the threshold (Line 6). As soon as one process finds the threshold, it assigns the proper value to RMT (Line 11), and updates the end variables of all processes with value 1 (Line 14). Thus, each process only has to check its own variable end at the beginning of each iteration to know whether it has to finalize.

4 Experimental Evaluation

The experimental evaluation of *MPIGeneNet* has been performed in terms of execution time, as our tool provides the same co-expression networks as *RMTGeneNet*, whose robustness and sensitivity has already been proved in [11]. This section provides a speed comparison of these two tools on two different multicore clusters. The first one consists of four nodes containing two eight-core Intel Xeon E5-2660 Sandy Bridge-EP processors each (i.e., 16 cores per node and 64 in total). The second system provides a total of 192 cores grouped into eight nodes. In this case each node consists of 24 cores (two Intel Haswell 2680 processors with 12 cores each). Both clusters are connected through an InfiniBand FDR network. More details about the hardware and software employed in each cluster can be seen in Table 1. A preliminary evaluation tested different configurations of processes and threads per node when executing *MPIGeneNet*. All the results shown in this section are obtained for the best ones:

- Two MPI processes per node (one per processor) on the Sandy-Bridge system, with eight threads per process.
- Four MPI processes per node, and six threads per process on the cluster based on Intel Haswell processors.

TABLE 1
Characteristics of the clusters used in the experimental evaluation.

	C1	C2
Nodes	4	8
CPU type	Intel Sandy Bridge	Intel Haswell
CPUs per node	2	2
Cores per CPU	8	12
Clock frequency	2.20GHz	2.50GHz
Memory per node	64GB	128GB
Network	InfiniBand FDR	
MPI Compiler	OpenMPI 1.7.2	
OpenMP support	3.0	
MKL version	11.3	
GSL version	1.13	1.16

Three datasets downloaded from the GEO Dataset Browser available at the NCBI website [36] were used for evaluation. The main characteristics that have impact on the execution time consumed by the tool are the number of genes, the number of samples and the value of the RMT threshold. On the one hand, the runtime to construct the correlation matrix increases quadratically with the number of genes and linearly with the number of samples (see Algorithm 1). On the other hand, the value of the RMT threshold is the main influence on the second step runtime as it determines the number of iterations to be performed (see Algorithm 2). These characteristics for the three employed datasets are shown in Table 2.

Tables 3 and 4 show the runtimes of the original *RMTGeneNet* and our hybrid MPI/Threads implementation on both systems. These runtimes include in all cases the time to read/write the input/output. The first conclusion that can be obtained is that the

TABLE 2
Characteristics of the datasets used in the experimental evaluation.

Name	Genes	Samples	RMT Threshold
GDS5037	41,000	108	0.846100
GDS3795	54,675	200	0.835100
GDS3244	61,170	160	0.959000

TABLE 3
Runtimes of *MPIGeneNet* using up to four nodes containing two Intel Xeon E5-2660 Sandy Bridge processors each (using two MPI processes per node and 8 threads per process). The original runtimes of the sequential *RMTGeneNet* tool are also included for comparison purposes.

Dataset	*RMTGeneNet*	*MPIGeneNet*		
		1 core	1 node	4 nodes
GDS5037	2h 29m 8s	1h 18m 50s	11m 24s	3m 42s
GDS3795	6h 21m 50s	3h 45m 17s	28m 17s	8m 36s
GDS3244	5h 7m 14s	2h 23m 43s	12m 43s	3m 47s

modifications to the memory accesses in the sequential code significantly improve the performance of *MPIGeneNet* even for sequential computation: our tool running on a single core is on average more that two times faster than *RMTGeneNet*. Furthermore, the use of our hybrid parallelization approach on a multicore cluster significantly reduces runtimes for the selected datasets. On average, *MPIGeneNet* is 55.31 times faster than *RMTGeneNet* using the four nodes of the first cluster. In fact, the speedup is up to 81.21 for the largest dataset (*GDS3244*). On the eight nodes of the Haswell-based cluster the average speedup increases to 117.02, while the runtime of this *GDS3244* dataset is significantly reduced from more than five hours using *RMTGeneNet* to around one minute (speedup of 175.41).

Remark that this article does not include a comparison to other available parallel tools such as TINGe [22] because it would not be fair, as the method to determine their threshold in the second step is more simple than RMT and it does not provide co-expression networks with the same level of sensitivity and robustness as *MPIGeneNet*.

4.1 Scalability analysis

It has been already shown that the speedup obtained by *MPIGeneNet* over *RMTGeneNet* is significant in all scenarios. However, the magnitude of the acceleration varies depending on the input dataset. This section provides a further analysis of the results obtained in the largest cluster (eight nodes and 192 cores) in order to provide an insight about the behavior of the different *MPIGeneNet* steps explained in Section 3. Concretely, Figure 1 shows the speedup for varying number of nodes of the whole application (left top graph) and the three steps separately. The baseline

TABLE 4
Runtimes of *MPIGeneNet* using up to eight nodes containing two Intel Haswell 2680 processors each (using four MPI processes per node and 6 threads per process). The original runtimes of the sequential *RMTGeneNet* tool are also included for comparison purposes.

Dataset	*RMTGeneNet*	*MPIGeneNet*		
		1 core	1 node	8 nodes
GDS5037	1h 39m 17s	44m 50s	5m 22s	1m 14s
GDS3795	4h 45m 26s	2h 24m 17s	13m 15s	2m 58s
GDS3244	3h 36m 36s	1h 27m 47s	5m 17s	1m 14s

is the *RMTGeneNet* runtime. Additionally, graphs of Figure 2 illustrate the impact of each *MPIGeneNet* step on the total runtime for one and eight nodes. The following conclusions can be obtained from these results:

- The highest speedups are obtained for the construction of the correlation matrices, and they are quite constant for the three different input datasets.
- The parallel implementation of the second step does not obtain so high speedups (although the acceleration is still significant), and the highest values are obtained for the *GDS3244* dataset, i.e., the one with less RMT iterations (see Table 2). The main reason for this lower speedup is the poor multithreaded support provided by the current version of the MKL `ssyev` routine, especially for lower thresholds that have a high number of non-zero elements on the cut matrix. This is of high importance because, as mentioned in Section 3.2, the calculation of the eigenvalues of the cut matrix requires most of the time for each iteration. For instance, the time due to calls to this MKL routine on the smallest dataset `GDS5037` on the Haswell based system is only reduced from around 1,300 seconds with one thread to 475 seconds using 6 threads (speedup of 2.74). Nevertheless, this widely used library is continuously updated, and *MPIGeneNet* could benefit from a better multithreaded support of future versions without any code modification.
- The performance improvement in the phase of extraction of *MPIGeneNet* over *RMTGeneNet* remains constant with the number of nodes. This step is not parallelized and the acceleration is due to more efficient memory accesses and I/O management.
- As expected from the previous conclusions, the whole *MPIGeneNet* speedups are higher for those datasets where the search of the RMT threshold has low influence on the runtime (i.e., where the threshold is high and not many iterations are necessary).
- The percentage of time spent creating the cor-

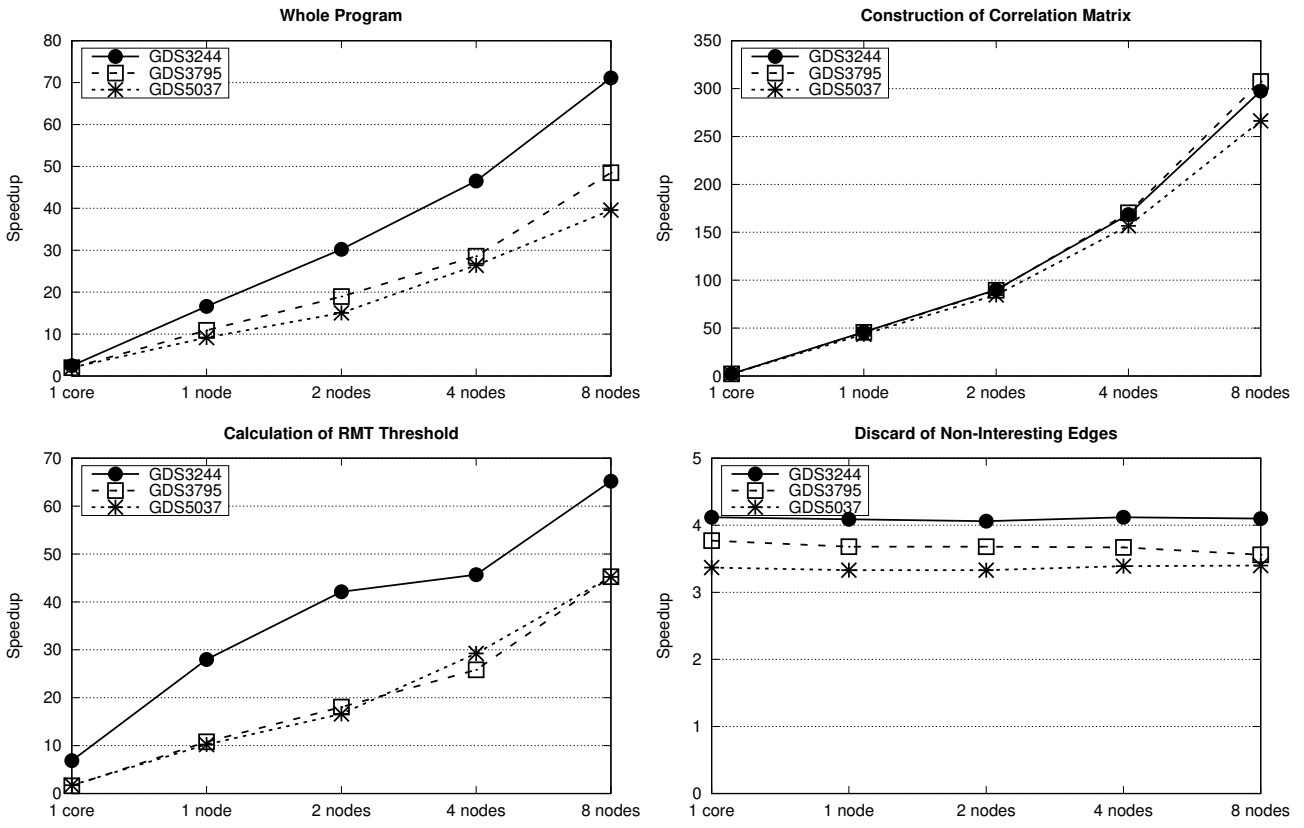

Fig. 1. Speedup of *MPIGeneNet* with varying number of cores and nodes using up to eight nodes containing two Intel Haswell 2680 processors each (when whole nodes are used, *MPIGeneNet* is configured with 4 MPI processes per node and 6 threads per process) over *RMTGeneNet*. The graphs represent the speedups of the whole algorithm (upper left) as well the three steps separately.

relation matrix decreases with the number of nodes thanks to the efficient parallelization of this step. Consequently, the percentage of time to find the RMT threshold increases with the number of nodes.

5 Conclusion

The construction of regulatory networks from gene expression data is an important problem in bioinformatics and systems biology as these networks can help to understand the complex interactions that occur among genes. Nowadays, there exist several approaches to construct these networks, using different correlation measures and posterior filters. Each tool has advantages and drawbacks and is usually suitable for certain type of datasets. In this paper we focus on using Pearsons correlation and RMT filters, which has been proved to be especially efficient at keeping the robustness and sensitiveness of the generated networks at expenses of a long runtime (several hours for a moderately-size dataset with tens of thousands genes and hundreds of samples). To accelerate the execution time we present *MPIGeneNet*, the first parallel tool that can exploit the computationally capabilities of multicore clusters to accelerate the construction of co-expression networks based on RMT thresholds.

Our tool follows a hybrid two-level parallelization. First, it includes a MPI implementation that divides the workload among MPI processes. It is optimized by using efficient RMA one-sided communications available in MPI since its version 3.0, as well as communication avoiding techniques that replicate the expression and correlation matrices. The memory overhead due to data replication is minimized thanks to the second level of parallelism, where each MPI process launches several threads sharing memory. Moreover, *MPIGeneNet* relies on the widely employed GSL and MKL libraries to perform mathematical functions. These libraries are in continuous evolution, so *MPIGeneNet* will benefit from future library updates without requiring any modification in its code.

The experimental evaluation on several scenarios (two clusters and three input datasets) proved that *MPIGeneNet* obtains the same co-expression matrices as *RMTGeneNet* (a sequential counterpart), but it is faster. First, on average *MPIGeneNet* needs half of time to construct the network even using the same resources as *RMTGeneNet* (one core). Furthermore, the runtime can be significantly reduced when running on

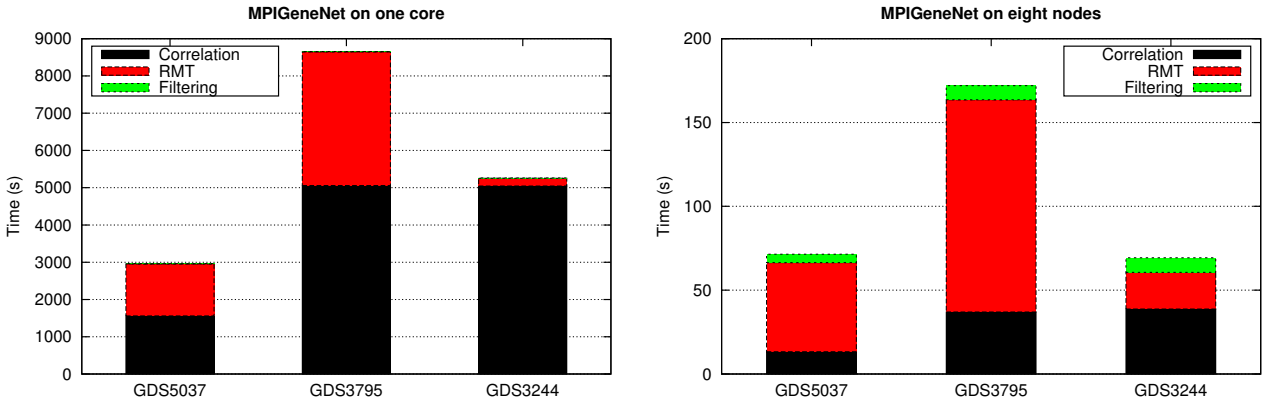

Fig. 2. Runtime breakdown of the different parts of *MPIGeneNet* running on one core and eight nodes containing two Intel Haswell 2680 processors each (4 MPI processes per node and 6 threads per process).

several nodes of the clusters. Although the magnitude of the acceleration depends on the characteristics of the dataset (concretely, on the number of iterations needed to find the RMT threshold), on average *MPIGeneNet* is 117.02 faster than *RMTGeneNet* on a cluster with eight nodes that contains 24 cores each (with a maximum speedup of 175.41).

Source code of *MPIGeneNet*, as well as a reference manual, are available at https://sourceforge.net/projects/mpigenenet/.

As future work we plan to include parallel implementations different than Pearson's and RMT into *MPIGeneNet*, and analyze the advantages and drawbacks of each approach in terms of both accuracy and performance.

ACKNOWLEDGMENTS

This research has been funded/supported by the Ministry of Economy and Competitiveness of Spain, Project TIN2016-75845-P (AEI/FEDER, UE) as well as supported by Xunta de Galicia (Centro Singular de Investigacion de Galicia accreditation 2016-2019, ref. EDG431G/01). We gratefully thank CESGA (Galicia Supercomputing Center) for providing access to the Finis Terrae II supercomputer.

REFERENCES

[1] P. Minguez and J. Dopazo, "Assessing the Biological Significance of Gene Expression Signatures and Co-Expression Modules by Studying Their Network Properties," *PLOS One*, vol. 6, no. 3, 2011.

[2] A. Gobbi and G. Jurman, "A Null Model for Pearson Coexpression Networks," *PLOS One*, vol. 10, no. 6, 2015.

[3] J. Nie, R. Stewart, H. Zhang, J. A. Thomson, F. Ruan, X. Cui, and H. Wei, "TF-Cluster: A Pipeline for Identifying Functionally Coordinated Transcription Factors via Network Decomposition of the Shared Coexpression Connectivity Matrix (SCCM)," *BMC Systems Biology*, vol. 5, no. 53, 2011.

[4] H. Peng, S. Wang, and X. Wang, "Consistency and Asymptotic Distribution of the TheilSen Estimator," *Journal of Statistical Planning and Inference*, vol. 138, no. 6, pp. 1836–1850, 2008.

[5] F. Gómez-Vela, C. D. Barranco, and N. Díaz-Díaz, "Incorporating Biological Knowledge for Construction of Fuzzy Networks of Gene Associations," *Applied Soft Computing*, vol. 42, pp. 144–155, 2016.

[6] S. L. Carter, C. M. Brechbuhler, M. Griffin, and A. T. Bond, "Gene Co-Expression Network Topology Provides a Framework for Molecular Characterization of Cellular State," *Bioinformatics*, vol. 20, no. 15, pp. 2242–2250, 2004.

[7] S. Persson, H. Wei, J. Milne, G. P. Page, and C. R. Somerville, "Identification of Genes Required for Cellulose Synthesis by Regression Analysis of Public Microarray Data Sets," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, no. 24, pp. 8633–8638, 2005.

[8] A. D. Perkins and M. A. Langston, "Threshold Selection in Gene Co-Expression Networks Using Spectral Graph Theory Techniques," *BMC Bioinformatics*, vol. 10, no. Suppl 11, 2009.

[9] R. R. Nayak, M. Kearns, R. S. Spielman, and V. G. Cheung, "Coexpression Network Based on Natural Variation in Human Gene Expression Reveals Gene Interactions and Functions," *Genome Research*, vol. 19, pp. 1953–1962, 2009.

[10] G. W. Bassela, E. Glaabc, J. Marqueza, M. J. Holdsworth, and J. Bacardit, "Functional Network Construction in Arabidopsis Using Rule-Based Machine Learning on Large-Scale Data Sets," *The Plant Cell*, vol. 23, no. 9, pp. 3101–3116, 2011.

[11] S. M. Gibson, S. P. Ficklin, S. Isaacson, F. Luo, F. A. Feltus, and M. C. Smith, "Massive-Scale Gene Co-Expression Network Construction and Robustness Testing Using Random Matrix Theory," *PLOS One*, vol. 8, no. 2, 2013.

[12] S. P. Ficklin, F. Luo, and F. A. Feltus, "The Association of Multiple Interacting Genes with Specific Phenotypes in Rice Using Gene Coexpression Networks," *Plant Physiology*, vol. 154, no. 1, pp. 13–24, 2010.

[13] S. P. Ficklin and F. A. Feltus, "Gene Coexpression Network Alignment and Conservation of Gene Modules between Two Grass Species: Maize and Rice," *Plant Physiology*, vol. 156, no. 3, pp. 1244–1256, 2011.

[14] D. Du, N. Rawat, Z. Deng, and F. G. Gmitter, "Construction of Citrus Gene Coexpression Networks from Microarray Data Using Random Matrix Theory," *Horticulture Research*, vol. 2, no. 15026, 2015.

[15] S. P. Ficklin and F. A. Feltus, "A Systems-Genetics Approach and Data Mining Tool to Assist in the Discovery of Genes Underlying Complex Traits in Oryza Sativa," *PLOS One*, vol. 8, no. 7, 2013.

[16] A. Onatski, "Asymptotics of the Principal Components Estimator of Large Factor Models with Weakly Influential Factors," *Journal of Econometrics*, vol. 168, no. 2, 2012.

[17] G. Wainrib and J. Touboul, "Topological and Dynamical Complexity of Random Neural Networks," *Physical Review Letters*, vol. 110, no. 11, 2013.

[18] W. Hachem, P. Loubaton, X. Mestre, J. Najim, and P. Vallet, "A Subspace Estimator for Fixed Rank Perturbations of Large

Random Matrices," *Journal of Multivariate Analysis*, vol. 114, pp. 427–447, 2013.
[19] C. Beenakker, "Random-Matrix Theory of Majorana Fermions and Topological Superconductors," *Reviews of Modern Physics*, vol. 87, no. 1037, 2015.
[20] M. Zhu and Q. Wu, "Transcription Network Construction for Large-Scale Microarray Datasets Using a High-Performance Computing Approach," *BMC Genomics*, vol. 9, no. Suppl1, 2008.
[21] J. Ingram and M. Zhu, "GPU Accelerated Microarray Data Analysis Using Random Matrix Theory," in *13th IEEE International Conference on High Performance Computing and Communications (HPCC'11)*, Banff, Canada, 2011.
[22] J. Zola, M. Aluru, A. Sarje, and S. Aluru, "Parallel Information-Theory-Based Construction of Genome-Wide Gene Regulatory Networks," *IEEE Trans. on Parallel and Distributed Systems*, vol. 21, no. 10, pp. 1721–1733, 2010.
[23] S. Misra, K. Pamnany, and S. Aluru, "Parallel Mutual Information Based Construction of Genome-Scale Networks on the Intel Xeon Phi Coprocessor," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 12, no. 5, pp. 1008–1020, 2015.
[24] M. Liang, F. Zhang, G. Jin, and J. Zhu, "FastGCN: A GPU Accelerated Tool for Fast Gene Co-Expression Networks," *PLOS One*, vol. 10, no. 1, 2015.
[25] H. Shi, B. Schmidt, W. Liu, and W. Müller-Wittig, "Parallel Mutual Information Estimation for Inferring Gene Regulatory Networks on GPUs," *BMC Research Notes*, vol. 4, no. 189, 2011.
[26] Y. Liu, T. Pan, and S. Aluru, "Parallel Pairwise Correlation Computation on Intel Xeon Phi Clusters," in *28th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'16)*, Los Angeles, CA, USA, 2016.
[27] J. González-Domínguez and M. J. Martín, "Fast Parallel Construction of Correlation Similarity Matrices for Gene Co-Expression Networks on Multicore Clusters," in *17th International Conference on Computer Science (ICCS'17)*, Zurich, Switzerland, 2017.
[28] J. González-Domínguez and B. Schmidt, "ParDRe: Faster Parallel Duplicated Reads Removal Tool for Sequencing Studies," *Bioinformatics*, vol. 32, no. 10, pp. 1562–1564, 2016.
[29] J. González-Domínguez, Y. Liu, J. Touriño, and B. Schmidt, "MSAProbs-MPI: Parallel Multiple Sequence Aligner for Distributed-Memory Systems," *Bioinformatics*, vol. 32, no. 24, pp. 3826–3828, 2016.
[30] E. Georganas, J. González-Domínguez, E. Solomonik, Y. Zheng, J. Touriño, and K. Yelick, "Communication Avoiding and Overlapping for Numerical Linear Algebra," in *Proc. 24th ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'12)*, Salt Lake City, UT, USA, 2012.
[31] M. Driscoll, E. Georganas, P. Koanantakool, E. Solomonik, and K. Yelick, "A Communication-Optimal N-Body Algorithm for Direct Interactions," in *Proc. 27th International Parallel and Distributed Processing Symposium (IPDPS'13)*, Boston, MS, USA, 2013.
[32] T. Hoefler, J. Dinan, R. Thakur, B. Barrett, P. Balaji, W. Gropp, and K. Underwood, "Remote Memory Access Programming in MPI-3," *ACM Transactions on Parallel Computing*, vol. 2, no. 2, 2015.
[33] S. Kumar and M. Blocksome, "Scalable MPI-3.0 RMA on the Blue Gene/Q Supercomputer," in *Proc. 21st European MPI Users' Group Meeting (EuroMPI'14)*, Kyoto, Japan, 2014.
[34] F. Luo, Y. Yang, J. Zhong, H. Gao, L. Khan, D. K. Thompson, and J. Zhou, "Constructing Gene Co-Expression Networks and Predicting Functions of Unknown Genes by Random Matrix Theory," *BMC Bioinformatics*, vol. 8, no. 299, 2007.
[35] Intel, "Math Kernel Library," https://software.intel.com/en-us/intel-mkl, 2017.
[36] NCBI, "Geo Expression Omnibus (GEO) Dataset Browser," https://www.ncbi.nlm.nih.gov/sites/GDSbrowser, 2017.

Jorge González-Domínguez received the B.Sc., M.Sc. and PhD. degrees in Computer Science from the University of A Coruña, Spain, in 2008, 2010 and 2013, respectively. He is currently an Assistant Professor in the Grupo de Arquitectura de Computadores at the Universidade da Coruña, Spain. His main research interests are in the areas of high performance computing for bioinformatics and PGAS programming languages.

María J. Martín received the B.Sc. (1993), M.Sc. (1994) and PhD. (1999) degrees in physics from the University of Santiago de Compostela, Spain. Since 1997, she has been on the faculty of Computer Science at the Universidade da Coruña, where she is currently an Associate Professor of Computer Engineering. Her major research interests include parallel algorithms and applications, cluster and grid computing and fault tolerance for message-passing applications.