Fast Parallel Construction of Correlation Similarity Matrices for Gene Co-Expression Networks on Multicore Clusters

Jorge González-Domínguez and María J. Martín

Grupo de Arquitectura de Computadores, Universidade da Coruña
Campus de Elviña s/n, 15071 A Coruña, Spain
jgonzalezd@udc.es, mariam@udc.es

Abstract
Gene co-expression networks are gaining attention in the present days as useful representations of biologically interesting interactions among genes. The most computationally demanding step to generate these networks is the construction of the correlation similarity matrix, as all pairwise combinations must be analyzed and complexity increases quadratically with the number of genes. In this paper we present `MPICorMat`, a hybrid MPI/OpenMP parallel approach to construct similarity matrices based on Pearson's correlation. It is based on a previous tool (`RMTGeneNet`) that has been used on several biological studies and proved accurate. Our tool obtains the same results as `RMTGeneNet` but significantly reduces runtime on multicore clusters. For instance, `MPICorMat` generates the correlation matrix of a dataset with 61,170 genes and 160 samples in less than one minute using 16 nodes with two Intel Xeon Sandy-Bridge processors each (256 total cores), while the original tool needed almost 4.5 hours. The tool is also compared to another available approach to construct correlation matrices on multicore clusters, showing better scalability and performance. `MPICorMat` is an open-source software and it is publicly available at https://sourceforge.net/projects/mpicormat/.

Keywords: Genetics, High Performance Computing, MPI, OpenMP, Similarity Matrix, Pearson's Correlation

1 Introduction

Gene regulatory networks (also known as co-expression networks) are graphical and mathematical models to illustrate the complex interactions that can be present among multiple genes [23]. The nodes and edges represent genes and interesting correlations, respectively. Connected groups of genes indicate biological relationships among them. The interactions that join the genes are inferred by analyzing experimental data from several samples obtained through technologies such as microarrays or short read sequencing [11, 22]. Different algorithms and

measures can be applied to infer the interaction among genes from a set of observations. The most flexible methods, like Bayesian networks [6], are not widely used on large datasets due to their high computational requirements. Instead, most researchers resort to correlation statistics, such as Mutual Information (MI) functions [1], or Pearson's and Spearman's correlations [2].

Construction of co-expression networks is highly time-consuming due to the large number of pairwise calculations to be performed; e.g., even for a moderately-sized dataset consisting of 50,000 genes there are about 1.25 billion pairwise tests to be performed. Since both the availability and size of datasets with gene expression values are increasing rapidly, finding fast and scalable solutions is of high importance to research in this area. High performance computing can help to reach this goal [15]. In this work we focus on accelerating on multicore clusters the approach followed by RMTGeneNet [7], which provides co-expression networks with high degree of robustness and has been employed in several biological experiments (see, for instance, [4, 5]). This tool consists of two steps implemented by different modules. On the one hand, the Construction of Correlation Matrix (`CCM`), which takes as input a file with the expression values for different genes observed from several samples by high throughput technologies and calculates the Pearson's correlation for all possible gene pairs. `CCM` provides as output a file with a matrix of correlation values called similarity matrix which is used as input by the second module: Random Matrix Modeling (`RMM`). `RMM` calculates a Random Matrix Threshold (RMT) [21], which allows to discard not biologically relevant correlations [14]. Values below the threshold are set to zero, and the result (i.e., the co-expression network) is an adjacency matrix where each non-zero value represents an edge.

Preliminary experimental evaluation determined that up to 70% of the `RMTGeneNet` runtime is spent in the first step. Therefore, we have implemented `MPICorMat`, a parallel version of the `CCM` module that exploits the computational capabilities of multicore clusters to construct similarity matrices employing Pearson's correlation. Users can benefit from `MPICorMat` to accelerate the most computationally demanding step, and then use its output as input for the `RMM` module of the original sequential tool.

The rest of the paper is organized as follows. Section 2 presents previous works that address the parallel construction of similarity matrices as part of the co-expression network generation. Section 3 describes the parallel implementation of the correlation matrix construction. Section 4 summarizes the experimental environment, provides the scalability results of `MPICorMat` and compares it to two counterparts in terms of performance. Finally, concluding remarks are presented in Section 5.

2 Related Work

`TINGe` [24] is the closest work to our approach. It already provides a Message Passing Interface (MPI) implementation for the parallel construction of similarity matrices on multicore clusters, but it uses MI instead of Pearson's correlation for inferring the interactions between genes. Although, in general, MI is able to detect non-linear connections better than Pearson's correlation, some experiments have shown that it is not relevant in the case of gene co-expression networks [20]. Furthermore, it is not clear how robust MI-generated networks are. Thus, in order to maintain the same high degree of robustness as the `CMM` module of `RMTGenet` [7], our work is based on interactions found with Pearson's correlation.

Instead of adapting `TINGe` to work with Pearson's correlation, we have developed a novel parallel implementation that, as will be shown later, scales better with the number of nodes due to two reasons. On the one hand, `MPICorMat` follows a hybrid MPI/OpenMP approach that adapts better to the memory hierarchy of modern multicore clusters than the only-MPI

version available in `TINGe`. Hybrid implementations have been successfully applied to solve other bioinformatics problems such as the multiple sequence mapping [8] and the removal of duplicate DNA sequences [9]. On the other hand, we replicate input data among different MPI processes in order to avoid communications. Although memory consumption is higher, it is affordable on modern clusters. A similar approach has already been successfully applied to the parallelization of numerical algorithms [3, 19].

There also exist alternatives to construct similarity matrices on other parallel architectures such as manycore platforms. For instance, Xeon Phi coprocessors are gaining attention in the last years thanks to their high programmability and good performance, and previous works have already focused on these coprocessors to construct similarity matrices with both Pearson's correlation [13] and MI functions [16] (this last work is based on TINGe). Finally, `FastGCN` [12] and `CUDA-MI` [18] are similar implementations that exploit GPUs using the CUDA language.

3 Parallel Implementation

`MPICorMat` 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. It returns a file that contains a $n \times n$ similarity matrix with the Pearson's correlation values for each gene pair. Let x and y be two genes of a pair, and x_i, y_i the expression values of those genes for sample i. Pearson's correlation is calculated as follows:

$$P_{x,y} = \frac{m \sum_{i=1}^{m} x_i \cdot y_i - \sum_{i=1}^{m} x_i \cdot \sum_{i=1}^{m} y_i}{\sqrt{m \sum_{i=1}^{m} x_i^2 - (\sum_{i=1}^{m} x_i)^2} \cdot \sqrt{m \sum_{i=1}^{m} y_i^2 - (\sum_{i=1}^{m} y_i)^2}}$$

All the pairwise correlations are calculated similarly to the original `RMTGeneNet` tool: calling the available `gsl_stats_correlation` routine within the GNU Scientific Library (GSL) [10]. This library is publicly available under open-source license and installed by default in most of UNIX OS (common on clusters). `MPICorMat` asymptotic complexity is $O(n^2 m)$. In order to be exchangeable by the `CCM` module of `RMTGeneNet`, the syntax of the input and output files of `MPICorMat` are exactly the same as for the original module.

Before addressing the parallelization, we analyzed the code of this original sequential module and found inefficient memory accesses. Thus, our work started by optimizing these accesses (minimizing the allocation and deallocation of auxiliary arrays) so, as will be shown in Section 4, `MPICorMat` is significantly faster than the `CCM` module of `RMTGeneNet` even running on one single core.

3.1 MPI Parallelization

The most common programming model for distributed-memory systems is message-passing. We use MPI as it is established as a *de-facto* standard and provides a portable, efficient, and flexible approach for message-passing. MPI follows the Single Program Multiple Data (SPMD) paradigm, i.e., it splits the workload into different tasks that are executed on multiple processors. A parallel MPI program consists of several processes with associated local memory. In the traditional point of view each process is associated to one core, but now the association of each process to several threads that operate on different cores is becoming popular. Communication among processes is carried out through the interconnection network by using send and receive routines, and it is often the main cause of performance overhead.

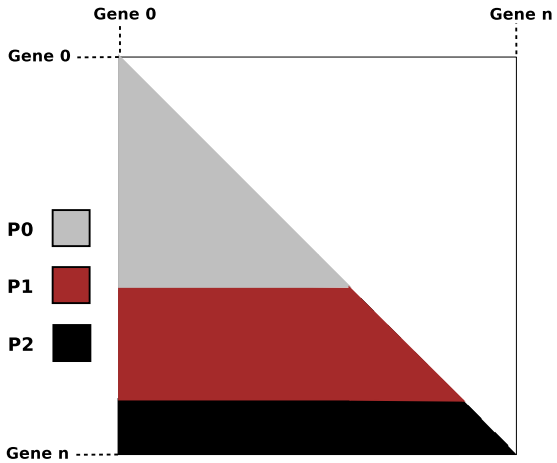

Figure 1: Distribution of gene pairs among processes.

As the Pearson's correlation must be performed for all gene pairs, the `MPICorMat` workload 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 (upper or lower triangular) must be calculated. Concretely $\frac{n \cdot (n-1)}{2}$ pairs. Our MPI implementation divides the workload (gene pairs) among the processes and, as computation is completely independent among pairs, processes can work over different pairs at the same time. Pairs are assigned by blocks of rows (the whole row to the same process). However, due to the triangular nature of the problem, creating blocks with the same number of rows would lead to unbalanced workload (some rows have more pairs than other). Instead, blocks with different number of rows are assigned to each process, trying to balance the number of pairs per process. Figure 1 illustrates an example for three processes. The ideal number of pairs per process would be $idealBlock = \frac{n \cdot (n-1)}{2 \cdot numP}$, being $numP$ the total number of processes. Process 0 has the first x rows assigned, with x large enough to guarantee that the number of pairs associated to this process is at least $idealBlock$. The same procedure is applied to the next rows and the remaining processes except for the last one, which computes all final rows that have not been assigned to previous process. As the rows are not divided among different processes, it is common that the first $numP - 1$ processes have more than $idealBlock$ gene pairs, whereas the last block is smaller. For instance, in a dataset with eight genes, `MPICorMat` performs 28 calculations. In this case, for three processes, the $idealBlock$ is equal to $\frac{28}{3} = 9.3$. Our tool assigns the initial four rows to the first process, the next two rows to the second one and only the last row to the last process. It means ten, eleven and seven pairs to each process, respectively. Remark nevertheless that, for realistic scenarios with $n >> numP$, the load balance is always very good, as the difference is almost insignificant compared to the total number of pairs.

In `MPICorMat` all processes start reading in parallel the input file and saving their own copy of the initial expression matrix, instead of distributing this matrix among processes as in `TINGe`. The main advantage is that the replication of the expression matrix reduces the number of communications. During `TINGe` computation the expression values of genes allocated in remote memory must be received with a two-sided communication before calculating the correlation of the associated pairs. In contrast, in `MPICorMat` all processes have all necessary gene information during the whole procedure. Therefore, `MPICorMat` gains at performance by avoiding internal

Algorithm 1: Pseudo-code of the hybrid parallel algorithm on each process.

```
1  Read input matrix M with the expression values of the genes
2  Calculate myIniRow and myLastRow
3  Initialize matrix of private scores myS := 1
4  Initialize iterator iter = 0
5  #pragma omp parallel for schedule(dynamic)
6  for each row i from myIniRow to myLastRow do
7  |   for each column j from 0 to i − 1 do
8  |   |   myS[iter] := CalculatePearson(i, j)
   |   |   iter++
   |   end
9  |   iter++ # Score for diagonal elements is 1.0
   end
10 Write partial result with MPI_File_Write(myS)
```

communications, at expense of memory overhead. Nevertheless, the memory requirements are reasonable for modern clusters. For instance, the largest input expression matrix used in Section 4 for experimental evaluation has 10,935,000 float elements (54,675 genes and 200 samples), i.e., 41 MB. Assuming that current clusters usually provide tens of gigabytes per node, there should not be memory constraints even for big data expression matrices.

The computation finalizes once the correlation of the assigned gene pairs has been calculated. Each process writes its partial result into the corresponding position of the output file with the efficient MPI I/O routines `MPI_File_Write()` and `MPI_File_Seek()`.

3.2 Hybrid MPI/OpenMP Parallelization

As mentioned in Section 2, there is a current trend to improve the performance of parallel codes using a hybrid MPI and multithreaded approach where each process is associated to a group of cores (usually all the cores that share memory within one node of the cluster) and it launches several threads to distribute its tasks among the cores of the group. `MPICorMat` uses OpenMP directives to create several threads per process. OpenMP is a parallel programming interface based on a set of compiler directives. It follows a fork-join model, where the master thread creates a number of slave threads that can perform different tasks in parallel and access to the same shared memory. Task assignment to threads can be done statically (known at the beginning of the execution) or dynamically (the tasks are assigned once threads finish the previous ones).

In a multicore cluster that contains N memory modules with C associated cores each, a pure MPI implementation like `TINGe` would use $N \cdot C$ processes and distribute data among all of them. Instead, the hybrid implementation provided by `MPICorMat` creates P MPI processes, each one with T OpenMP threads that collaborate to calculate the Pearson's correlation of the gene pairs assigned to their parent process, being $N \cdot C = P \cdot T$. Algorithm 1 shows the pseudocode of this hybrid approach. The odd-block distribution of rows is applied to only the P MPI processes, but each row is calculated by a different thread in parallel (T rows at the same time). A dynamic distribution of rows is performed to balance the workload among threads. This is achieved using the `omp parallel for` directive with dynamic scheduling over the external loop (Line 5).

The benefit of the hybrid implementation is that all the cores of the system are exploited but

Table 1: Datasets used for performance evaluation.

Name	Number of Genes	Number of Samples
GDS5037	41,000	108
GDS3242	61,170	128
GDS3244	61,170	160
GDS3795	54,675	200

reducing the number of MPI processes. This limits the memory overhead due to replicating the input matrix, as the threads can work over the same memory and thus only one copy per MPI process is needed. `MPICorMat` implementation is flexible enough to allow the users to specify the desired number of MPI processes and threads. A description of the configuration parameters, as well as installation and execution instructions, are available at the tool download website: https://sourceforge.net/projects/mpicormat/

4 Experimental Evaluation

Four datasets, with different number of genes and samples (see Table 1) were used in our experiments. They were downloaded from the Geo Expression Omnibus (GEO) Dataset Browser available at the National Center for Biotechnology Information (NCBI) website [17]. We have checked that the outputs of `MPICorMat` and the original `CCM RMTGeneNet` module are identical for the four datasets and different configurations of MPI processes and OpenMP threads. Thus, the evaluation shown in this paper is focused on performance in terms of execution time, as the robustness and accuracy of the method was already satisfactorily tested in [7].

Experimental evaluation has been performed on a 16-node cluster, where each node contains two 8-core Intel Xeon E5-2660 Sandy-Bridge processors (16 cores at 2.20 GHz per node). Each processor has its own memory module, with 32 GB. Although the two memory modules available within one node can be seen as a whole 64 GB shared memory, this is really a Non-Uniform Memory Access (NUMA) system, as accesses to the closest 32 GB module are significantly faster than to the module available in the other processor. The memory hierarchy is completed with one 20 MB shared L3 cache per processor, and 32 and 256 KB private L1 and L2 caches per core. The system provides in total 256 cores and 1 TB of memory. The nodes are connected through InfiniBand FDR network, with bandwidth up to 56 Gbps. As for software, we use OpenMPI compiler v.1.7.2 with support for OpenMP v.3.0 and resort to the GSL library v.1.13 to perform the Pearson's correlation of the expression arrays.

All experimental runtimes shown along this section are the average of five identical executions, using the queue system of the cluster and running in exclusive mode (no works of other users are executed on the same nodes at the same time). Furthermore, runtime corresponds to the total time of the application, i.e., it includes the time needed to read/write input/output data, besides the time to construct the similarity matrix.

Before analyzing the scalability of `MPICorMat`, a preliminary experimental evaluation to determine the best configuration of MPI processes and OpenMP threads was performed. Consequently, two processes per node (one per NUMA region) and eight threads per process will be used from now on as this configuration obtains the best performance. Runtimes for all the datasets using this configuration and different number of nodes are presented in Table 2. Thanks to our hybrid approach, we obtain average speedup of 13.87 when using one whole

Table 2: Runtime (seconds) spent by `MPICorMat` with the four datasets using two processes per node and eight threads per process (best configuration). Times include reading/writing the input/output (less than 2% of the total time in all cases).

Nodes	Cores	GDS5037	GDS3242	GDS3244	GDS3795
1	1	2,539.96	6,652.75	8,365.14	8,139.81
1	16	186.81	488.93	595.44	572.06
2	32	95.47	251.55	301.45	291.40
4	64	49.73	125.98	151.10	145.76
8	128	25.83	66.05	80.79	77.99
16	256	13.84	33.46	41.02	39.26

node (parallel efficiency of 86.70%). When exploiting the 16 nodes of the cluster, less than 42 seconds are needed by `MPICorMat` to construct the similarity matrices of any of the four datasets (average parallel efficiency of 91.00% compared to the execution on only one node). Thanks to the efficient usage of MPI I/O routines, less than 2% of the execution time is spent reading and writing from files. These results show that parallel computing can be significantly beneficial for biologists, as `MPICorMat` running on a single core needs more than 2 hours and 15 minutes in the worst case.

Table 3 summarizes a performance comparison among `MPICorMat` and the state of the art tools `RMTGeneNet` [7] and `TINGe` [24]. This table provides the runtime of the three tools when constructing the similarity matrices for the four datasets. As `RMTGeneNet` is intrinsically sequential, it could only be executed for one single core. Results for both `MPICorMat` and `TINGe` include experiments with only one core, one whole node (16 cores) and the whole cluster (16 nodes, 256 cores). As `TINGe` does not provide multithreading support, it is executed with one MPI process per core. In order to provide a fair comparison, the configuration of the experiments is identical for all tools: same compiler (OpenMPI v.1.7.2), average time of five executions, employment of queue system, and nodes in exclusive mode even when only one core of the node is used. Furthermore, `RMTGeneNet` uses the same GSL routine (v.1.13) as `MPICorMat` to calculate Pearson's correlation. `TINGe` is executed with the default parameters: 10 bins and 4 spline levels to calculate the MI. We tried to include a performance comparison against `FastGCN` [12], a GPU implementation also based on Pearson's correlation. However, its high memory requirements made its execution abort due to memory constraints on a NVIDIA K20 GPU with 5 GB of memory.

The first conclusion that can be obtained is that the modifications to the memory accesses performed to the sequential code (already mentioned in Section 3) significantly improves the performance of `MPICorMat` even for sequential computation. Despite both `MPICorMat` and `RMTGeneNet` employ the same correlation measure and provide exactly the same results, our tool running on one single core is on average 1.97 times faster than `RMTGeneNet`. `MPICorMat` is also faster than `TINGe` for any number of nodes. One reason is the different similarity measure (`TINGe` employs MI instead of Pearson's correlation). In order to compare the parallel performance of `MPICorMat` and `TINGe`, Figure 2 shows the speedups of both tools. Speedups are calculated as $S = \frac{T_p}{T_s}$, where T_p is the parallel time and T_s the baseline sequential time. To avoid the influence of the choice of the similarity measure, T_s for each tool is the runtime of that tool when executed only on one core. The speedups shown in Figure 2 prove that `MPICorMat` is more scalable than `TINGe` for all datasets and any number of nodes (even using only the 16 cores within one node). For instance, speedups for `GDS3795` for `MPICorMat` and `TINGe` are 207.33 and

Table 3: Runtime (seconds) comparison among `MPICorMat` and other counterparts for the four datasets. `MPICorMat` uses two processes per node and eight threads per process (best configuration). All times include reading/writing the input/output.

Cores	Tool	GDS5037	GDS3242	GDS3244	GDS3795
1	`RMTGeneNet`	5,336.51	13,124.76	16,004.83	15,470.96
	`TINGe`	5,206.48	12,442.99	14,664.41	12,965.41
	`MPICorMat`	2,539.96	6,652.75	8,365.14	8,139.81
16	`TINGe`	398.78	1,041.91	1,400.93	1,016.63
	`MPICorMat`	186.81	488.93	595.44	572.06
256	`TINGe`	48.91	114.47	129.71	119.35
	`MPICorMat`	13.84	33.46	41.02	39.26

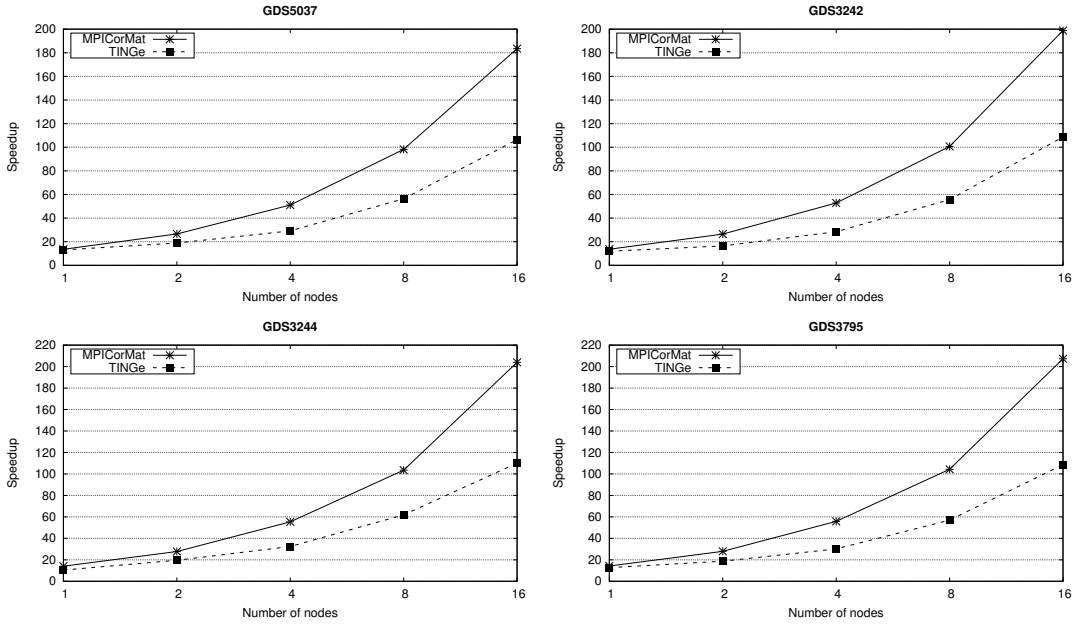

Figure 2: Scalability comparison between `MPICorMat` and `TINGe`. The speedup of each approach is obtained over the runtime necessary by the tool itself using one core. `MPICorMat` uses two processes per node and eight threads per process (best configuration). Experiments include reading/writing the input/output.

108.63 on the 16 nodes (parallel efficiencies of 80.99% and 42.43%, respectively). These results show that our tool is faster than `TINGe` not only because calculation of Pearson's correlation is faster than MI, but also because the parallel implementation is more efficient, thanks to the hybrid MPI/OpenMP approach and the communication avoidance strategy.

5 Conclusions

In this paper we present the first step towards the parallelization of the robust `RMTGeneNet` tool for construction of regulatory networks from gene expression data. We developed `MPICorMat`, a tool whose goal is the parallelization of the first and most computationally demanding stage (construction of the correlation similarity matrices) on multicore clusters, with hybrid distributed/shared memory architecture. Our tool follows a two-level parallel approach. First, the workload is divided among MPI processes using an odd-block distribution by rows that balances the number of calculations per process. `MPICorMat` implementation also avoids communication among processes by replicating the input expression values. Nevertheless, the memory overhead due to data replication is minimized thanks to the second level of parallelism, where each MPI process launches several OpenMP threads.

Experiments were carried out on a cluster with 16 NUMA nodes, each one with two eight-core processors (256 total cores) and with four datasets, showing that parallel computing is a useful approach in order to accelerate the construction of similarity matrices. `MPICorMat` needs less than one minute to generate the matrix associated to any of the four input datasets, while `RMTGeneNet` needs between 1.5 and 4.4 hours. The input and output files follow the same syntax as the original tool so users can just substitute the first module of `RMTGeneNet` by `MPICorMat` and benefit from the accelerated runtime without further modifications. `MPICorMat` is significantly faster and more scalable than `TINGe`, to the best of our knowledge, the only available counterpart to construct similarity matrices on multicore clusters. `MPICorMat` source code, as well as its reference manual, are publicly available at `https://sourceforge.net/projects/mpicormat/`.

As future work we intend to parallelize the second module available in `RMTGeneNet`, which creates the RMT threshold and discards from the co-expression network those interactions that are biologically irrelevant. We also plan to include additional measures such as MI or Spearman's correlation for the calculation of the similarity matrices.

Acknowledgments

This research has been funded/supported by the Ministry of Economy and Competitiveness of Spain and FEDER funds of the EU, Project TIN2016-75845-P (AEI/FEDER, UE).

References

[1] Carsten O. Daub, Ralf Steuer, Joachim selbig, and Sebastian Kloskaw. Estimating Mutual Information Using B-Spline Functions - an Improved Similarity Measure for Analysing Gene Expression Data. *BMC Bioinformatics*, 5(118), 2004.

[2] A de la Fuente, Nan Bing, Ina Hoeschele, and Pedro Mendes. Discovery of Meaningful Associations in Genomic Data Using Partial Correlation Coefficients. *Bioinformatics*, 20(18):3565–3574, 2004.

[3] Michael Driscoll, Evangelos Georganas, Penporn Koanantakool, Edgar Solomonik, and Katherine 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.

[4] Dongliang Du, Nidhi Rawat, Zhanao Deng, and Fred G. Gmitter. Construction of Citrus Gene Co-expression Networks from Microarray Data Using Random Matrix Theory. *Horticulture Research*, 2(15026), 2015.

[5] Stephen P. Ficklin and Frank 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*, 8(7), 2013.

[6] N Friedman, M Linial, I Nachman, and D Peer. Using Bayesian Networks to Analyze Expression Data. *Journal of computational biology*, 7:601–620, 2000.

[7] Scott M. Gibson, Stephen P. Ficklin, Sven Isaacson, Feng Luo, Frank A. Feltus, and Melissa C. Smith. Massive-Scale Gene Co-Expression Network Construction and Robustness Testing Using Random Matrix Theory. *PLOS One*, 8(2), 2013.

[8] Jorge González-Domínguez, Yongchao Liu, Juan Touriño, and Bertil Schmidt. MSAProbs-MPI: Parallel Multiple Sequence Aligner for Distributed-Memory Systems. *Bioinformatics*, 2016. Online.

[9] Jorge González-Domínguez and Bertil Schmidt. ParDRe: Faster Parallel Duplicated Reads Removal Tool for Sequencing Studies. *Bioinformatics*, 32(10):1562–1564, 2016.

[10] Brian Gough. *GNU Scientific Library Reference Manual - Third Edition*. Network Theory Ltd., 2009.

[11] Ya Guo, Quanhu Sheng, Jiang Li, Fei Ye, David C. Samuels, and Yu Shyr. Large Scale Comparison of Gene Expression Levels by Microarrays and RNAseq Using TCGA Data. *PLOS One*, 8(8), 2013.

[12] Meimei Liang, Futao Zhang, Gulei Jin, and Jun Zhu. FastGCN: A GPU Accelerated Tool for Fast Gene Co-Expression Networks. *PLOS One*, 10(1), 2015.

[13] Yongchao Liu, Tony Pan, and Srinivas 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.

[14] Feng Luo, Yunfeng Yang, Jianxin Zhong, Haichun Gao, Latifur Khan, Dorothea K. Thompson, and Jizhong Zhou. Constructing Gene Co-Expression Networks and Predicting Functions of Unknown Genes by Random Matrix Theory. *BMC Bioinformatics*, 8(299), 2007.

[15] Victor E. Malyshkin. Peculiarities of Numerical Algorithms Parallel Implementation for Exa-Flops Multicomputers. *International Journal of Big Data Intelligence*, 1(1), 2014.

[16] Sanchit Misra, Kiran Pamnany, and Srinivas Aluru. Parallel Mutual Information Based Construction of Genome-Scale Networks on the Intel Xeon Phi Coprocessor. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 12(5):1008–1020, 2015.

[17] National Center for Biotechnology Information (NCBI). Geo Expression Omnibus (GEO) Dataset Browser. https://www.ncbi.nlm.nih.gov/sites/GDSbrowser, Last visited: November 2016.

[18] Haixiang Shi, Bertil Schmidt, Weiguo Liu, and Wolfgang Müller-Wittig. Parallel Mutual Information Estimation for Inferring Gene Regulatory Networks on GPUs. *BMC Research Notes*, 4(189), 2011.

[19] Edgar Solomonik and James Demmel. Communication-Optimal Parallel 2.5D Matrix Multiplication and LU Factorization Algorithms. In *Proc. 17th International European Conference on Parallel and Distributed Computing (Euro-Par'11)*, pages 90–109, Burdeaux, France, 2011.

[20] L Song, P Langfelder, and S Horvath. Comparison of Co-Expression Measures: Mutual Information, Correlation, and Model Based Indices. *BMC Bioinformatics*, 13(328), 2012.

[21] E P. Wagner. Random Matrices in Physics. *SIAM review*, 9:1–23, 1967.

[22] S R. Zhao, W P. Fung-Leung, A Bittner, K Ngo, and X J. Liu. Comparison of RNA-Seq and Microarray in Transcriptome Profiling of Activited T Cells. *PLOS One*, 8(2), 2013.

[23] X Zhu, M gerstein, and M Snyder. Getting Connected: Analysis and principles of Biological Networks. *Genes & Dvelopment*, 21(9):1010–1024, 2007.

[24] Jaroslaw Zola, Maneesha Aluru, Abhinav Sarje, and Srinivas Aluru. Parallel Information-Theory-Based Construction of Genome-Wide Gene Regulatory Networks. *IEEE Transactions on Parallel and Distributed Systems*, 21(10):1721–1733, 2010.