REVIEW ARTICLE

Parallel Algorithms for Inferring Gene Regulatory Networks: A Review

Omid Abbaszadeh, Ali Reza Khanteymoori* and Ali Azarpeyvand

Department of Electrical and Computer Engineering, University of Zanjan, Zanjan, Iran

ARTICLE HISTORY

Received: December 18, 2017
Revised: February 20, 2018
Accepted: May 22, 2018

DOI:
10.2174/1389202919666180601081718

Abstract: System biology problems such as whole-genome network construction from large-scale gene expression data are sophisticated and time-consuming. Therefore, using sequential algorithms are not feasible to obtain a solution in an acceptable amount of time. Today, by using massively parallel computing, it is possible to infer large-scale gene regulatory networks. Recently, establishing gene regulatory networks from large-scale datasets have drawn the noticeable attention of researchers in the field of parallel computing and system biology. In this paper, we attempt to provide a more detailed overview of the recent parallel algorithms for constructing gene regulatory networks. Firstly, fundamentals of gene regulatory networks inference and large-scale datasets challenges are given. Secondly, a detailed description of the four parallel frameworks and libraries including CUDA, OpenMP, MPI, and Hadoop is discussed. Thirdly, parallel algorithms are reviewed. Finally, some conclusions and guidelines for parallel reverse engineering are described.

Keywords: Gene regulatory network, Parallel algorithms, Parallel processing, Reverse engineering, CUDA, OpenMP, MPI, Hadoop.

1. INTRODUCTION

Each cell consists of thousands of genes. In each cell, the only small percentage of genes is expressed. Genes that are expressed interact with each other through mRNAs (messenger RNAs), proteins, or other types of molecules and managed cellular phenotypes and functions. Differences in gene expression are responsible for both morphological and phenotypic differences which indicate cellular reactions to environmental disturbances or hormonal stimuli [1]. There are several methods available for measuring gene expression level. Sequential Analysis of Gene Expression (SAGE) [2], DNA microarrays [3], Tiling arrays [4] and RNA-Seq [5] are the most used and important methods. The output of these methods is the expression profiles of genes that can be used in bioinformatics applications.

One of the main objectives of bioinformatics researchers is deciphering the gene-gene interactions which are known as constructing a Gene Regulatory Network (GRN) or reverse engineering from gene expression profiles. A GRN is a graphical representation that demonstrates associations between a set of genes. In this model, edges represent regulatory influence or co-expression relationships in the regulatory network or co-expression network, respectively, and nodes represent molecular entities like genes [6]. The knowledge regarding the gene network not only shed light on the biological processes such as cellular differentiation, division, and signaling, but also can provide valuable information for drug discovery, molecular biology, cancer-related, and medical-related research [7, 8]. For example, Imoto *et al.* [9] and di Bernardo *et al.* [10] studies are prominent works that used gene regulatory networks in drug discovery.

According to conventional wisdom, reverse engineering is a difficult problem, particularly in dealing with large-scale data. Considerable sequential algorithms have been developed to derive GRN model and meaningful information from experimental data. These algorithms can be categorized into seven main groups, namely; Boolean networks [11], Statistical methods like Partial-least-squares [12, 13], Differential equation systems [14], Bayesian networks [15, 16], Graphical Gaussian Models [17], Evolutionary approaches [18], and Information theory-based approaches [19]. Though there are various sequential algorithms for reverse engineering, they will not construct high dimensional gene networks and demonstrate valuable information like hub nodes, master-regulators, and some important regulated genes [20]. Furthermore, by increasing the size of data, the quality of the constructed network based on sequential algorithms is reduced (*e.g.* large number of false-positive edges in huge network).

Even with recent progress in reverse engineering, in order to construct an appropriate network from large-scale data, the use of new machine learning methods and high-performance computing are important and challenging at the same time [7, 21]. Recently, parallel and distributed reverse engineering algorithms have received significant attention. Therefore, most of the proposed algorithms are scalable and reasonably accurate for reconstruction of GRNs from large-scale datasets. In practice, parallel and distributed algorithms can considerably

*Address correspondence to this author at the Department of Electrical and Computer Engineering, University of Zanjan, Zanjan, Iran; Tel: (+98) 24 33052604; Fax: (+98) 24 33052604; E-mail: khanteymoori@znu.ac.ir

reduce the execution time and provide scalability without losing quality. In the past, one of the main difficulties in the implementation of parallel and distributed algorithms was lack of an efficient framework for developing parallel algorithms. Furthermore, parallel programming required a level of expertise that few researchers and biologists have. Today, the use parallel frameworks such as CUDA (Compute Unified Device Architecture), MPI (Message Passing Interface), OpenMP, and Hadoop can considerably ease the work of researchers when they need to implement the efficient parallel algorithms.

There are outstanding review papers covering the field of GRN inference. Some of the well-structured overviews of the general idea behind GRN inference and common mathematical modeling can be found in [19, 22-28]. Bansal *et al.* [22], Chai *et al.* [23], Lee and Tzou [24], and Schlitt *et al.* [25] prepared review papers on the computational approaches and a brief mathematical formulation for GRNs reconstruction. Sima *et al.* [26] reviewed dynamic methods that inferred GRN from time-series experiment data. Biswas *et al.* [27] reviewed evolutionary approach for GRN inference and Sirbu *et al.* [28] analyzed several evolutionary algorithms.

Based on our knowledge, no work has been carried out to review the parallel algorithms for GRN inference. Therefore, we want to highlight two major issues: algorithms and tools which have been implemented in parallel frameworks and different parallel frameworks that can be used in learning gene networks.

The paper is organized as follows: Section 2 introduces a brief overview of parallelism and parallel frameworks such as CUDA, MPI, OpenMP, and Hadoop. In Section 3, we review recent parallel algorithms in reverse engineering. Section 4 draws conclusion and prepares some directions for future research on parallel GRN inference.

2. PARALLEL FRAMEWORKS

In order to achieve the promise of GRN inference on large-scale datasets, it is necessary for existing GRN algorithms to be executed in parallel. Parallel programming is concerned with the distribution of a program among a set of processors and defines how they interact in order to make the results. One of the most important aspects of parallelism is its relation to the hardware and programming frameworks. There are several frameworks for parallel programming. CUDA, MPI, OpenMP, and Hadoop are the most popular frameworks for parallel and distributed programming. CUDA proposed by Nvidia is a parallel programming framework for Nvidia GPUs (Graphical Processing Unit). It is an extension to the C and C++ that provide a set of libraries for exploiting GPUs as general purpose processors. MPI (Message Passing Interface) and OpenMP (Open Multiprocessing) are the set of standard libraries for parallel programming in distributed and shared memory space environments, respectively. MPI uses message passing among the processes in clustered systems where it generally shared nothing. OpenMP is used for parallel programming in multi-core fashion which is generally based on shared memory architecture. Also, MPI can be used to distribute the algorithm when using multiple GPUs. Hadoop is a software framework that enables us to run applications and store big datasets in the distributed environments. In this section, we briefly introduce the organization of CUDA, MPI, OpenMP, and Hadoop.

2.1. GPU and CUDA

In the last decade, the clock speed of processors has remained constant. Therefore, processor designers came to the conclusion that complex multi-core processor is not the most efficient for massively parallel computing. Currently, processor designing trend is going to many-core approach such as GPUs or co-processors (such as the Xeon Phi) instead of complex multi-core processors [29]. This kind of architecture provides heterogeneous computing and achievable performance for SIMD (Single Instruction Multiple Data) programs. It describes programs with one instruction that performs the same operations on multiple data points simultaneously. This change of designing paradigm has had (and will have) a significant impact on the designing parallel algorithm [30].

Each GPU consists of a series of streaming multiprocessor (SM, or SMX in the latest architectures) which within each SM, a number of streaming processors (SP), known as cores, are placed in arrays and execute arithmetic and logical operations in parallel. Furthermore, each SM has a number of registers and a private per-block shared memory to transfer data between concurrent threads. According to the programming model, threads and thread blocks are distributed along SPs and SMs. There is another memory called global memory that is used to share data between the grid of SMs. A grid is a set of SMs that work independently and thus may be executed asynchronously in parallel [31].

In 2007, Nvidia released CUDA framework which is an extension to the C and C++ and makes available using the GPU as general purpose GPUs [30]. CUDA provides three features for programmers: 1) Threads management, 2) Memories management, and 3) Synchronization features. These fine-grained features help us to divide the program into subprograms that can be executed in parallel and then integrate them. The written codes in CUDA contain one or more functions that are called kernels which are loaded to the GPUs and replicated in many threads. The programmer determines the number of threads for each kernel and manages the available memory spaces visible to the kernel functions [31, 32]. One of the main tasks in CUDA based parallel algorithms is to determine the threads, blocks, grids, and managing memory allocation, which is the source of differences in the performance of algorithms. In spite of the remarkable advantages in programming, GPU-based programming is different from CPU-based programming. Nevertheless, several packages were released whose users without any knowledge of GPU programming can also access the high-performance computing power of GPUs. OpenCL is another framework for cross-platform GPU programming maintained by the Khronos group, which can be run on different hardware platforms. Recently, Nobile *et al.* [33] studied some computational tools in bioinformatics that exploit GPUs as a processing engine.

2.2. MPI

MPI is the most used *de-facto* standard libraries for parallel programming based on message passing paradigm. It is a collection of libraries to send messages between computers or processes on the distributed memory environment. In MPI programming model, nodes have their own memory space, own processors, and communicate with each other to access memory space [34]. In addition, programmers must divide the tasks among the nodes with separate memory spaces and define the nodes communications and synchronize them. To facilitate parallel programming, MPI provides various libraries and functions to communicate, coordinate, and synchronize between distributed nodes. Most of the current bioinformatics code could be parallelized under the MPI models such as mpiBLAST [35] (Basic Local Alignment Search Tool), parallel version of the BLAST sequence alignment, and MPI-CMS [36], parallel implementation of the Cross Motif Search algorithm.

2.3. OpenMP

OpenMP is a set of high-level APIs (Application Programming Interfaces) which provides shared-memory based parallelism and multi-threading paradigm in multi-core environments. It consists of a set of compiler directives, libraries, and predefined interfaces that can be used in programming languages such as C/C++, Java, Python, and many other languages. After compiling OpenMP programs, threads negotiate with each other through shared memory space and hence increase the performance of the program. Similar to CUDA, OpenMP provides three feature for programmers: 1) controlling features that alter the flow in a program, 2) synchronization features for coordinating the execution of threads, and 3) data environment features for communicating between threads [37]. OpenMP provides a high-level abstraction that makes it well suited for high-performance computing programmers in shared memory environment. Therefore, one of the main advantages of OpenMP is that it does not require major changes for converting a sequential code to parallel one.

2.4. Hadoop

Before introducing Hadoop, Map-Reduce paradigm should be introduced. In this paradigm, data is divided into subsets, and then these subsets are assigned to the different machines for parallel processing. Finally, it brings together separate processes and returns the end result. The stage of division and allocation of data to machines is called Map, and bringing together and presenting the result is called reduce stage. Map-Reduce paradigm is suitable for big data analyzing due to its ability to execute the program in parallel over the cluster of computers data without loading the whole data into memory.

Hadoop is a Java-based framework that allows parallel and distributed programming across the distributed environment using Map-Reduce paradigm. It has two main components: YARN (Yet Another Resource Negotiator) and HDFS (Hadoop Distributed File System). YARN manages computational resources needed for distributed executions. HDFS prepares scalable and robust distributed file system for big data. Apart from the Hadoop, there are numerous software frameworks (such as Pig, Spark, Mahout *etc.*) that provide specific features whose users with no knowledge of distributed programming can also process large amounts of data on the specific domain [38]. Additionally, there are many bioinformatics tools which have been developed based on Hadoop such as CloudBLAST [39], distributed version of the BLAST2 algorithm using Hadoop framework, Eoulsan [40], a framework for RNA sequence data analysis, and Seqpig [41] and BioPig [42], for analyzing large-scale sequencing data.

Unfortunately, there is no “silver bullet” for parallel programming. Indeed, based on framework selection, parallel programming is more complex and different than sequential programming. Efficient distribution of tasks on the processing units, avoiding inefficient data replication, and unnecessary communication among the processing units are the vital factors that affect parallel programming performance. Each of the mentioned frameworks provides a different paradigm of parallel programming and have their own strong and weak points. The use of OpenMP for parallel programming is easier than other frameworks, but it runs on the shared-memory environment. MPI runs on shared and distributed memory but requires more changes in the sequential algorithm. Hadoop provides highly scalable and faults tolerant environment, but it is not always straightforward to implement sequential algorithms as a Map-Reduce program. Although exploiting CUDA leads to higher performance compared to using CPU in data-level parallel programming, but CUDA programming is more difficult than CPU programming and programmers need an in-depth understanding of the GPU architecture. Table **1** summarizes the characteristics of the frameworks based on usability, complexity, and scalability.

There are many reasons to integrate the two or more parallel programming frameworks. For example, CUDA-aware MPI programs, accelerate an existing single-GPU application to scale across multi-GPU application by using MPI. Apart from the above frameworks, several other projects which provide specific features have been developed.

Table **2** summarizes some important libraries and tools for programmers to efficiently exploit and integrate parallel frameworks. These libraries aim to develop more efficient parallel programs and provide high level abstraction for researchers with low experience in parallel and distributed programming.

3. PARALLEL ALGORITHMS

The crucial step in GRN inference is selecting the model. In this review, we focus only on the approaches that the modeling algorithms are parallel. Based on their mathematical models, in the next subsections, we will review parallel algorithms.

3.1. Bayesian Network Based Models

Modeling gene regulatory networks based on Probabilistic Bayesian Networks (PBN) have become popular in the bioinformatics community. The main advantages of PBN are the ability to represent the uncertainty in models, exibility, and integrating prior knowledge (*e.g.* biological knowledge) with experimental data. In 1999, for the first time, Murphy *et al.* [16] used the Bayesian network for GRN inference and

Table 1. Parallel framework comparison.

Framework	Programming Model	Framework Complexity	Programming Language	Ease of Use	Code Conversion Effort	Scalability
CUDA	SIMD	Fair	C/C++	Moderately	More	Low
OpenMP	Multi-thread	Low	Most Languages	Easy	Few	Low
MPI	SIMD/MIMD	Fair	Most Languages	Poor	More	Medium
Hadoop	Distributed	High	Java	Poor	More	High

Note: Framework complexity refers to the difficulty in using different frameworks.

Ease of use refers to the effort required to programming.

Code conversion refers to the effort required to changing the sequential code to parallel code.

thereafter, significant efforts focused on reverse engineering by PBNs. According to Pearl and Russell [43], PBN is a Directed Acyclic Graph (DAG) $G = (V, E)$, where $V = \{X_1, X_2, \cdots, X_n\}$, the set of nodes, represents random variables, and E is the set of directed edges, which represents cause-and-effect relationships such as regulation influence among the genes. A directional edge $X_i \rightarrow X_j$ indicates that X_i is parent of X_j or gene X_i regulates the gene X_j in GRN context. Mathematically, PBN encodes the Markov assumption that given its parents, each variable X_i is conditionally independent of its non-descendants. Based on this assumption, PBN compacts the joint probability distribution as follow:

$$P(X_1, \cdots, X_n) = \prod_{i=1}^{n} P(X_i | Pa(X_i)) \quad (1)$$

where $Pa(X_i)$ is the set of parents of X_i in the DAG.

For Bayesian network learning, many outstanding algorithms have been developed. Well-structured review of the Bayesian network learning is presented in [44]. In essence, learning PBNs from data consists of both parameter and structure learning (or model selection). Estimating the local conditional probabilities for each node is parameter learning and establishing the network as a candidate DAG is structure learning. Structure learning is more important than parameter learning in GRN inference because cause-and-effect interactions among the genes are determined at this step. Finding an exact network that fits on data, is NP-hard problem because the number of DAGs grows super-exponentially with the number of variables. This implies that exact algorithms can become a computationally intractable task and currently there is no polynomial time algorithm that can solve an NP-hard problem of large or even moderate input size. One way to tackle NP-hard problems is to design heuristic or parallel algorithms that reduce the computational time.

There are three generic approaches to structure learning: score-based, constraint-based, and hybrid learning methods [45]. The first approach assigns a score to the candidate DAG by scoring functions and tries to optimize scoring criteria with a heuristic algorithm such as greedy search. Selecting an appropriate scoring function is very important since it is the key ingredient to reconstruct high-quality GRN by PBNs. These methods work well on small datasets with not too many variables. Constraint-based methods efficiently restrict the search space. Therefore, they can often work well on large datasets. Sparse Candidate Algorithm (SCA) [46] is one of the prominent constraint-based algorithms where each variable constrained to have at most k parents. Finally, hybrid methods are combinations of the score-based and constraint-based approaches.

Nikolov and Aluru [47] developed a parallel hybrid Bayesian structure learning for reverse engineering. They demonstrated that the main cause of error in SCA based approaches is misselected optimal parents (OP) from candidate parent (CP) set. To address this issue, inspired by parallel pairwise mutual information [48], authors created a mutual information based network to identify CP set for each node. They then developed a parallel exact algorithm for selecting OPs from CPs set. In order to do this, they checked all subsets of CP and elicited the OPs for each node by scoring function and eventually, OP sets were used to create an initial network. Note that obtained graph may contain cycles, which are detected and eliminated by exponentiation of adjacency matrix based on cycle length (shorter cycles before longer ones). The authors implement the proposed method in the Cray system with AMD many-core processors by using C++ and MPI library. In their evaluation of performance on data of size 500 genes and 100 observations, in the best case, the method inferred GRN in less than 2 minutes in the Cray AMD cluster with 1024 cores.

In the algorithm developed by Misra *et al.* [49], a massively parallel heuristic PBN structure learning was established to whole-genome network reverse engineering by exploiting Tianhe-2 and Stampede high-end heterogeneous supercomputers. The proposed method is similar to [47] based on differences in the scoring function for network evaluation, limited size of the CP set to reduce the computational complexity, and implementation techniques to achieve performance, scalability, and efficient load balancing. In order to efficiently distribute the work between the processing units, they performed hierarchical dynamic work distribution that first divides tasks across the cluster nodes, and then subdivides this task within a node.

One of the conventional approaches to parallel structure learning is dividing the whole network learning problem into several subnetworks learning, where each of them contains randomly sampled variables. Evidently, the main issue here is how to select an appropriate sampling approach. Tamada *et al.* [20] developed a parallel PBN structure learning algorithm for reverse engineering, based on the subnetwork strategy and random walking technique, called Neighbor Node Sampling and Repeat (NNSR). The authors demonstrated

Table 2. Some related libraries and projects on CUDA, MPI, OpenMP, and Hadoop.

Project	Description	URL
Spark	An open-source cluster-computing framework on Hadoop	http://spark.apache.org/
Pig	A query language based on Hadoop for basic calculations over large datasets	http://pig.apache.org/
Mahout	A distributed machine learning and data mining library on Hadoop	http://mahout.apache.org/
OpenMPI	Most used implementation of the MPI model. Open MPI 1.7 and later is CUDA-aware	https://www.open-mpi.org/
MVAPICH	CUDA-aware MPI implementation. It helps to run CUDA+MPI	http://mvapich.cse.ohio-state.edu/
Mars	A Map-Reduce framework on graphics processors	https://github.com/arianepaola/Mars
CuBLAS	An implementation of basic linear algebra subprograms on CUDA framework	https://developer.nvidia.com/cublas
JCUDA	Java bindings for CUDA libraries. It helps to run Hadoop Map task on GPUs	http://www.jcuda.org/
omp4j	An OpenMP like library for Java programming language	http://www.omp4j.org/
mpi4py	A library for MPI programming in python	http://pythonhosted.org/mpi4py/
PyCUDA	A library for integrating CUDA in python	https://github.com/inducer/pycuda

that the small sample size and appropriate sampling of the variables (or genes) lead to subnetworks that can efficiently demonstrate cause-and-effect relationships. Therefore, they propose a two-phase heuristic algorithm which first, at each iteration, using random sampling (all variables being equally likely), learns a new subnetwork of the set of sampled variables, and then creates the whole network by using neighbor node sampling based on the random walking on the subnetworks. In order to do this, they create a weighted graph by introducing edge frequency.

Edge frequency indicates the ratio of the number of occurrences of directed edge $i \to j$ in different subnetworks divided by the number of different subnetworks in which two variable i, j are selected together (greater number indicates a stronger cause-and-effect relationship). Next, random walk procedure selects a specific proportion of the nodes from a weighted graph and creates a large number of smaller subnetworks. The authors implemented the proposed method in C programming language and OpenMPI library on the 724 computation nodes with dual Intel quad core Xeon 3.0 GHz, in total 5792 cores. They applied the proposed method on Human Umbilical Vein Endothelial Cells (HUVECs) with 13731 transcripts and extracted GRN in less than 3 hours. Furthermore, the proposed method also extracts valuable information such as hub nodes and putative master regulators that are not achievable from the small network. Based on this model, Tamada *et al.* [50] have developed a software collection called SiGN. SiGN consists of two other parallel programs based on the graphical Gaussian model, SiGN-L1, and state space model [51], SiGNSSM.

One of the main sources of error in the statistical inference is overconfidence to model, which is generated by ignoring model uncertainty [52]. Model uncertainty refers to the situations in which there is no unique and agreed model for a specific problem. In most situations, the main cause of uncertainty is irrelevant variables in constructing the model. Inspired by ensemble learning, one way to tackle model uncertainty is Bayesian Model Averaging (BMA). BMA refers to the procedure of selecting variables by averaging posterior probability of the models in which each of them consists of a set of candidate variables or regulators in the GRN. The main challenge in the BMA is selecting an efficient model. Young *et al.* [53] proposed a Bayesian inference method for regression variables selection from time-series data based on the BMA, called ScanBMA. They have developed a greedy mechanism for picking appropriate models based on Occam's window principle. Parallel implementation of ScanBMA named as fastBMA [54] is available from https://github.com/lhhunghimself/fastBMA.

3.2. Information Theory Based Models

Due to easy implementation, simplicity, low computational cost, and ability to detect complex interactions, parallel Information Theory Based Models (ITBM) is somewhat superior in reverse engineering. In the last two decades, some attractive algorithms based on the information theory have been developed. The ITBMs such as correlation-based [55, 56], Mutual Information (MI) [57-60], and Gaussian Graphical Models [17] (GGM) are the main state-of-the-art approaches to extract dependency on biological networks inference. In following, along with the review, we will introduce mathematical details of some similarity measures which are the cornerstone of ITBMs. Pearson correlation (PC), Mutual Information (MI), and Partial Correlation are the main similarity measures that have been extensively used in the literature. Each of them has their own limitations and benefits. There is no proof that one is superior to others [61].

MI is often used as a similarity measure, which enables the detection of non-linear relationships among the variables. It is defined based on the individual and joint entropies in the following way:

$$I(X_i;X_j) = H(X_i) + H(X_j) - H(X_i,X_j) \tag{2}$$

where $H(X)$ is the differential entropy of a random variable X and is a measure of its uncertainty. In particular, for a continuous variable X, it is defined by:

$$H(X) = -\int P_X(x)\log P_X(x)dx \tag{3}$$

In 3, $P_X(x)$ is probability density function for continuous variable X. It can be estimated by different methods such as histogram plotting, kernel estimators, k-nearest neighbor estimators [62], and B-spline estimators [63]. Note that estimating probability density function is one of the challenging problems in the MI based approaches. Binning the continuous variables into quantile intervals is another way of estimating probability distribution. Within this approach, each continuous expression value is replaced by an integer value corresponding to the bin if fall into. This is defined as follows:

$$I(X_i;X_j) = \sum_{k=1}^{B}\sum_{l=1}^{B} x_{kl}\log\frac{x_{kl}}{x_{k+}x_{+l}} \tag{4}$$

where B is the number of bins, x_{kl} represents the joint probability $P(X_i = k, X_j = l)$, and $x_{k+} = \sum_l x_{kl}$ and $x_{+l} = \sum_k x_{kl}$ are marginal probabilities $P(X_i = k)$ and $P(X_j = l)$. This method is very simple and fast but is sensitive to the number of bins used. Based on this approach, Belcastro *et al.* [64] developed a parallel MI-based algorithm.

Kernel-based estimators are computationally expensive when a large number of variables are available. To tackle this problem, Daub *et al.* [63] proposed a B-spline based method for binning continuous data. Within this approach, each continuous value is assigned to k bins with weights given by the B-spline function of order k defined over b knot points. For a continuous value, this function returns a vector of size b with k continuous non-negative weights that indicate to which bins the value should be assigned. Based on this idea, four parallel reverse engineering [48, 65-67] have been developed which will be discussed in detail below.

Zola *et al.* [48] proposed a parallel algorithm named TINGe (Tool Inferring Networks of Genes). TINGe is the first parallel software for reverse engineering which constructs the largest whole genome plant network. It uses B-spline based MI and provides efficient permutation testing for assessing statistical significance by rank transformation, Data Processing Inequality (DPI) to remove indirect relationships, and parallel processing for reverse engineering. DPI states that if three random variables X_i, X_j, X_k from a Markov Chain in that order *i.e.,* $X_i \rightarrow X_j \rightarrow X_k$, then $I(X_i;X_k) \le I(X_i;X_j)$ and $I(X_i;X_k) \le I(X_j;X_k)$. Indeed, if three genes g_i, g_j, g_k from a triangle in the network, DPI can be applied to remove the indirect edge among the three edges by removing the weakest MI value. This can significantly decrease false positive rate. In their performance evaluations on *Arabidopsis thaliana* of size 15222 genes and 3137 observations, the method inferred GRN in 30 minutes on a 2048-CPU Blue Gene/L and 2 hours and 25 minutes on a 8 node Cell blade cluster. Since, TINGe was successful, Misra *et al.* [65] implemented it on the Intel Xeon Phi single-chip coprocessor and Chockalingam *et al.* [67] developed a distributed version of TINGe on the Amazon EC2 cloud computing platform by using Hadoop framework.

Shi *et al.* [66] proposed a parallel MI-based algorithm by using B-spline function and CUDA framework, called CUDA-MI. By defining the weighting matrix ($N \times R$, N: number of genes; R: number of bins) in which each row of it indicates the weight coefficients of gene value in the set of the bin, CUDA-MI calculates pairwise MI in parallel among genes. Thereafter, the authors implemented their approach on the Nvidia Tesla C2050 GPU with 448 cores 1.15 GHz and compared it with Quad-Core i7 2.66 GHz CPU. By using single GPU version, their best acceleration was 82x, compared to the execution on multi-threaded CPU. Additionally, they combined CUDA-MI with ARACNE [57] method and the results of specifiity, sensitivity, and precision analysis revealed that the combined method is more efficient than simple ARACNE and TINGe software [48].

PC is another widely used correlation measure that detects linear relationships among the variables and is defined as follow:

$$\rho(X_i, X_j) = \frac{cov(X_i,X_j)}{\sigma(X_i)\sigma(X_j)} \tag{5}$$

where $cov(X_i, X_j)$ is the covariance of X_i, X_j, and $\sigma(X_i)$ is the standard deviation of X_i. Liang *et al.* [56] used PC based method for gene co-expression network reconstruction, called FASTGCN. They proposed a parallel algorithm that integrates genetic information entropy to preprocessing, PC for analyzing dependency, and z-score for coefficient normalization, and efficiently exploits GPU memory by using the zero-copy technique. The authors compared CUDA version of FASTGCN (implemented on Nvidia Tesla K20c with 2496 cores 760 MHz) against three versions of FASTGCN: Multi-core (Intel Xeon 16 cores 2.90GHz) CPU with 16 OpenMP thread, Single-thread CPU with C/C++ programming language, and Single-thread CPU with R programming and achieved 2x, 10x, 80x speedups respectively on the dataset containing 16000 genes of 590 individuals.

Zheng *et al.* [68] developed a new software based on their previous PCA-CMI (Path Consistency Algorithm based on Conditional Mutual Information) algorithm [69], known as CMIP. PCA-CMI is a well-known iterative algorithm for reverse engineering. At first, it creates a complete graph of size n (n is the number of genes) and at each iteration i ($i = 1, 2, \cdots, n$), by using i-order Conditional Mutual Information (CMI), quantifies relationships among two genes given their common i-neighbors. The CMI of variables X_i and X_j given X_k is defined as:

$$I(X_i;X_j|X_k) = H(X_i,X_k) + H(X_j,X_k) - H(X_k) - H(X_i,X_j,X_k) \tag{6}$$

where $H(X_i,X_j)$, $H(X_j,X_k)$, and $H(X_i,X_j,X_k)$ are joint entropies. High CMI value indicates that there may be a close relationship between the variables X_i and X_j given variable(s) X_k. After that, it deletes the edges with zero or low CMI value at each iteration. The time consumed for large-scale data and how to determine an appropriate edge deletion threshold are the main drawbacks of PCA-CMI. To overcome these drawbacks, the authors developed two parallel software by using CUDA and OpenMP frameworks and defined a mechanism for automatic threshold setting. In CUDA version of CMIP, pre-processed data is delivered to GPU cores for correlation calculation using a parallel model and in OpenMP version, loop calculation is accelerated with the multi-threading approach. CMIP attained acceptable performance compared to conventional methods.

Borelli *et al.* [70] proposed a new exhaustive search algorithm, which expresses the reverse engineering as a feature selection problem. In this way, feature selection can be viewed as an iterative searching method for selecting an optimal subset of genes which regulate target gene based on mean conditional entropy function as selecting criteria. The mean conditional entropy of variables X_i given X_j defined as:

$$H(X_j|X_i) = \sum_{x_i \in X_i} P(x_i) H(X_j|X_i = x_i) \tag{7}$$

Conditional entropy of X_j conditional on X_i refers to the average entropy of X_j conditional on the value of X_i, averaged over all possible values of X_i. Small value of conditional entropy indicates that X_i can well predict X_j or gene X_i associates the gene X_j in GRN context. Exhaustive search algorithm which is a time-consuming step has been implemented on the GPU and Multi-GPU in parallel. Furthermore, search algorithm has been implemented in global and local versions. Regulated genes of each target gene have been limited but not limited in the local and global search, respectively. Finally, the authors generated data by AGN simulator with 1024, 2048, 4096, and 8192 genes to evaluate their approach. They have compared the proposed method when it is implemented by one, two, and four GPUs with 240 core per GPU against CPU version which utilized six 3.2GHz core and OpenMP library. By using four GPU, their acceleration compared to the execution on CPU is 55, 110, 260, when there are 32, 64, and 128 target genes per block, respectively.

LegumeGRN [71] is a reverse engineering web tool, which has been implemented on multiple well-known reverse engineering algorithms. LegumeGRN developers have implemented a parallel version of TIGRESS [72] and GENIE3 [73], two popular algorithms for reverse engineering, which uses feature selection like methods as a mechanism for reverse engineering. GENIE3 uses tree-based ensemble feature selection method for reverse engineering on multifactorial expression data and TINGe uses LARS feature selection.

When dealing with high dimensional data and non-uniform distribution of variables, bias of MI estimator is one of the main sources of error. To overcome this problem, Kiraskov *et al.* [62] proposed an unbiased MI estimator based on K nearest neighbor (KNN) classifier. The main idea is estimating the probability densities from the distribution of its K nearest neighbor which implies minimally biased estimator. Sales and Romualdi [74] developed a parallel R package for reverse engineering based on KNN and MI, called PARMIGENE (PARallel Mutual Information estimation for GEne NEtwork reconstruction). The authors combined PARMIGENE with CLR, ARACNE, and MRNET, three state-of-the-art ITBMs which use MI for reverse engineering. Experimental results on *in-silico* datasets show that PARMIGENE estimator not only gives unbiased and more precise results, but is also faster than the other estimators.

3.3. Differential Equation-Based Models

Ordinary differential equations that are based on the biochemical systems theory are popular approaches for reverse engineering. In this model, by using a non-linear function f, regulatory interactions between genes can be expressed as follow:

$$x_i(t) = f_i(x_1, \cdots, x_N, u, \theta_i) \tag{8}$$

where $x_i(t)$ describes the expression level of gene i at time t, θ_i and u are the interaction parameters among genes and the external perturbation of gene, respectively. To date, one of the most prominent methods is a type of systems of ordinary differential equations called S-Systems. The general form of an S-System for representing a gene regulatory network is the as follow:

$$\frac{dx_i}{dt} = \underbrace{\alpha_i \prod_{j=1}^{N} x_j^{g_{i,j}}}_{activation} - \underbrace{\beta_i \prod_{j=1}^{N} x_j^{h_{i,j}}}_{degradation}, \forall i \tag{9}$$

where x_i is the expression level of gene i, and N is the total number of genes in the network. The non-negative parameters α_i and β_i are rate constants; $g_{i,j}$ and $h_{i,j}$ are kinetic orders that reflect the interaction from gene j to gene i in the activation and degradation processes, respectively. The parameter estimation of an S-system model is a large-scale optimization problem that is computationally expensive.

Lee *et al.* [75] and Jostin and Jaeger [76] developed a GRN model based on S-system. They proposed two distributed evolutionary algorithms for solving large-scale S-system parameters estimation. Lee *et al.* [75] combined Genetic Algorithm (GA) and Particle Swarm Optimization (PSO). The authors used two fitness function based on the Minimum Square Error (MSE) and exploited island model parallelism. In this way, the entire population is divided into the number of subpopulations and each of them is independently executed on the one or more processor(s). The algorithm is implemented on top of the Hadoop platform. Jostin and Jaeger [76] developed parallel island evolutionary algorithm, which is faster and more accurate than the comparable simulated annealing algorithm.

Xiao *et al.* [77] recently developed an asynchronous parallel algorithm to improve the accuracy and lower time complexity of large-scale GRN inference by combining splitting technology and ODE. The authors demonstrated that the sparsity and modularity of large-scale GRNs are much higher than the small-scale GRNs. In this paper, the whole network decomposes into clusters based on the MI criteria and each cluster is modeled by ODE. They used Gaussian elimination process for parameter estimation.

Gardner *et al.* [78] developed an algorithm *via* a set of ODEs on the series of steady-state RNA expression, called NIR (Network Identification by multiple Regression). NIR

Table 3. Advantages and disadvantages of computational methods.

Model	Strength	Weakness
Bayesian network	• Facilitate the incorporation of prior knowledge and experimental data • Able to cope with incomplete and noisy data • Handle with uncertainty	• Feedback regulations not allowed • Learning structure of the Bayesian network is NP-hard, therefore, can only apply to small-scale networks • Cannot model time series data
Information theory	• Easy to parallelize • Low computational cost • Able to detect complex interactions	• Can have a high rate of false positives in high dimensional data • Poor asymptotic behaviour under high dimensional data
Differential equation	• Suitable for time series and steady-state data • Model positive and negative feedback interactions	• Difficult to find optimal parameter values • Applicable to small-scale networks

constructs a first-order model of regulatory interactions and uses multiple linear regression to estimate model parameters. Due to the high time complexity, like the other sequential algorithms, sequential NIR cannot be used with large-scale datasets with thousands of genes. Gregoretti *et al.* [79] developed a parallel version of NIR algorithm. They argued that parameter estimation of NIR can be done independently by decomposing data matrix into a set of sub-matrices. In addition to speedup, the results of tests on large datasets show that the parallel NIR produces many fewer errors.

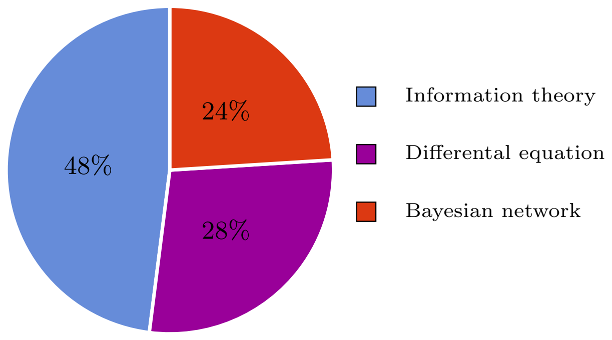

Fig. (1). Parallel framework usage in the reviewed papers.

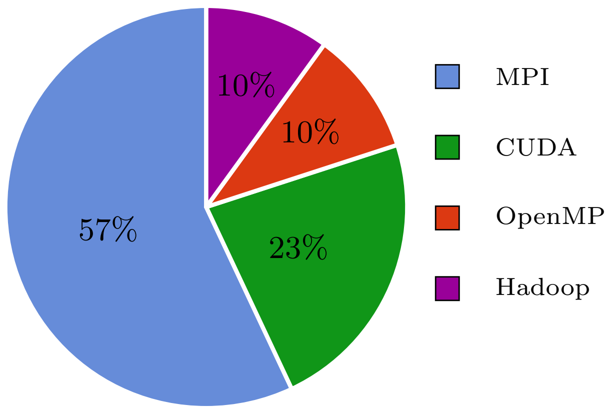

Fig. (2). Modeling methods usage in the reviewed papers.

Differential evolution is a population-based approach that holds promise for parameter estimation of ordinary differential equations and is appropriate to be parallelized [80] because the evaluation of the populations is independent of each other. In this approach, a problem is iteratively solved until no further improvement on the solution with regard to a given objective function. In each iteration, a new population is created *via* a migration technique in which the best individual from each population is selected and copied to another population [81]. Kozlov and Samsonov [82] and Ramirez *et al.* [83] proposed a parallel differential evolution algorithm for differential equations parameter estimation by using MPI library and CUDA framework, respectively.

As discussed in the introduction, there are many algorithms for GRN modeling from expression data. In this article, we reviewed only the approaches that its modeling algorithms were parallel. Table **3** shows some of the strengths and weaknesses of computational methods which provides useful insights on GRN reconstruction.

CONCLUSION AND DISCUSSION

According to reviewed papers, parallel approaches mostly use MPI library (Fig. **1**). This can have several reasons. One is that some frameworks, such as CUDA, are only supported on specific hardware and programming language. Another reason is that MPI can be used in a wider range of problems than other frameworks. In spite of the complexity of MPI programming, the last reason is that the researchers in MPI programming have a greater ability than CUDA and Hadoop frameworks. However, none of the frameworks are complete and have their own limitations. As discussed earlier, hybrid parallel programming such as MPI-OpenMP, MPI-CUDA, and OpenMP-CUDA is a good idea to achieve better performance and increase flexibility.

Mathematical modeling is an alternative categorization. Based on the reviewed papers, information theory based and differential equations based approaches are often used than PBN models (Fig. **2**). There are two important reasons for this: first, these approaches are more prevalent among bioinformatics researchers; second, their branch-less nature makes

Table 4. Parallel GRN inference algorithms.

Reference	Data Type	Based on	Framework	(Co)processor	Source Available	Description
[66]	Discrete	Information Theory	CUDA	GPU	√3	Known as CUDA-MI
[56]	Continuous	Information Theory	CUDA	GPU	√4	Known as FastGCN
[68]	Discrete	Information Theory	CUDA	GPU	√5	Known as CMIP
[83]	Continuous	Differential Equation	CUDA	GPU	-	-
[70]	Discrete	Information Theory	CUDA-OpenMP	GPU	-	-
[74]	Continuous	Information Theory	OpenMP	-	√6	Known as PAR-MIGENE
[76]	Continuous	Differential Equation	MPI	-	-	-
[77]	Continuous	Differential Equation	MPI	-	-	Known as LSGPA
[53]	Continuous	Bayesian Network	MPI	-	√7,8	Known as fast-BMA
[50]	Continuous	B-S-L[1]	MPI	-	√9	Known as SiGN
[65]	Discrete	Information Theory	MPI	Intel Xeon Phi	-	Based on TINGe
[49]	Discrete	Bayesian Network	MPI	Intel Xeon/ Intel Xeon Phi	-	-
[82]	Continuous	Differential Equation	MPI	Intel Xeon	-	Known as DEEP
[48]	Continuous	Information Theory	MPI	-	√10	Known as TINGe
[64]	Discrete	Information Theory	MPI	Intel Xeon	-	-
[79]	Continuous	Differential Equation	MPI	-	-	Known as Parallel NIR
[20]	Discrete	Bayesian Network	MPI	Intel Xeon	√11	-
[47]	Discrete	Bayesian Network	MPI	Cray AMD	-	-
[75]	Continuous	Differential Equation	Hadoop	-	-	-
[67]	Continuous	Information Theory	Hadoop	-	-	-
[71]	Continuous	*[2]	-	-	√12	Known as LegumeGRN

[1]B-S-L: Bayesian Network, State Space Model, L1-regularization

[2]A software which have implemented multiple well-known reverse engineering algorithms

[3]https://sites.google.com/site/liuweiguohome/cuda-mi

[4]http://ibi.zju.edu.cn/software/FastGCN/

[5]http://www.picb.ac.cn/CMIP/

[6]https://cran.r-project.org/web/packages/parmigene/index.html

[7]https://github.com/lhhunghimself/fastBMA, fastBMA is a parallel implementation of ScanBMA

[8]https://www.bioconductor.org/

[9]http://sign.hgc.jp/

[10]http://aluru-sun.ece.iastate.edu/doku.php?id=tinge_gena

[11]http://bonsai.hgc.jp/~tamada/hgc/suppl/GWGN/index.html

[12]https://legumegrn.noble.org/cc.html

them attractive for parallelism. However, there are several parallel algorithms in literature developed for PBN structure learning, which can be used in the context of GRN problems with minor modifications. As discussed earlier, using PBNs in addition to prior knowledge (*e.g.* gene ontology or biological knowledge) can ultimately improve accuracy and have a reasonable biological justification.

In order to perform parallel inference, selecting modeling approach and parallel framework are essential steps. In this work, we reviewed parallel algorithms on GRN inference problem. We also briefly explained parallel frameworks for programming and development of algorithms. Table **4** summarizes the research works we have found within the literature's which use parallelism in the reverse engineering process.

As a result of our studies, we propose some guidelines to facilitate decision-making for parallel reverse engineering:

- GRNs often are modular [84]. Modularity is a suitable property for parallel reverse engineering and based on this, researchers can develop efficient parallel algorithms.
- Based on the reviewed papers, much less attention has been paid to the knowledge-based approach. Therefore, developing parallel knowledge-based algorithm is an interesting idea.
- In gene expression dataset, sample sizes are substantially smaller than the number of available genes. This is known as "large p small n" problem, so researchers must take this into account to design more efficient method.

Sequential inference algorithms are highly limited to the size of the dataset and often do not provide valuable information such as hub genes, master regulators, and many others. Parallel algorithms for large-scale GRN problems deliver fast and useful results. However, this field is interdisciplinary, involving parallel algorithms design, bioinformatics, and machine learning. Therefore, in this paper, parallel reverse engineering algorithms are reviewed from the perspective of parallel frameworks used, bioinformatics knowledge used for inference, and mathematical modeling methods.

CONSENT FOR PUBLICATION

Not applicable.

CONFLICT OF INTEREST

The authors declare no conflict of interest, financial or otherwise.

ACKNOWLEDGEMENTS

Declared none.

REFERENCES

[1] Lockhart, D.J.; Winzeler, E.A. Genomics, gene expression and DNA arrays. *Nature,* **2000**, *405*(6788), 827-836. Available from: https://www.nature.com/articles/35015701

[2] Velculescu, V.E.; Zhang, L.; Vogelstein, B.; Kinzler, K.W. Serial analysis of gene expression. *Science,* **1995**, *270*(5235), 484-487. Available from: http://science.sciencemag.org/content/270/5235/484

[3] Rays, M.; Chen, Y.; Su, Y.A. Use of a cDNA microarray to analyse gene expression patterns in human cancer. *Nat. Genetics,* **1996**, *14*(4), 367-370.

[4] Bertone, P.; Stolc, V.; Royce, T.E.; Rozowsky, J.S.; Urban, A.E.; Zhu, X.-A.; Rinn, J.L.; Tongprasit, W.; Samanta, M.; Weissman, S.; Gerstein, M.; Snyder, M. Global identification of human transcribed sequences with genome tiling arrays. *Science,* **2004**, *306*(5705), 2242-2246. Available from: http://science.sciencemag.org/content/306/5705/2242.full

[5] Wang, Z.; Gerstein, M.; Snyder, M. RNA-seq: A revolutionary tool for tran- scriptomics. *Nat. Rev. Genet.,* **2009,** *10*(1), 57-63.

[6] Zhang, B.; Horvath, S. A general framework for weighted gene co-expression network analysis. *Stat. Appl. Genet. Mol. Biol.,* **2005**, *4*, Article 17.

[7] Marbach, D.; Costello, J.C.; Kuffner, R.; Vega, N.M.; Prill, R.J.; Camacho, D.M.; Allison, K.R.; Kellis, M.; Collins, J.J.; Stolovitzky, G. Wisdom of crowds for robust gene network inference. *Nat. Methods,* **2012**, *9*(8), 796-804.

[8] Basso, K.; Margolin, A.A.; Stolovitzky, G.; Klein, U.; Dalla-Favera, R.; Califano, A. Reverse engineering of regulatory networks in human b cells. *Nat Genet.,* **2005**, *37*(4), 382-390.

[9] Imoto, S.; Savoie, C.J.; Aburatani, S.; Kim, S.; Tashiro, K.; Kuhara, S.; Miyano, S. Use of gene networks for identifying and validating drug targets. *J. Bioinform. Comput. Biol.,* **2003**, *1*(3), 459-474.

[10] di Bernardo, D.; Thompson, M.J.; Gardner, T.S.; Chobot, S.E.; Eastwood, E.L.; Wojtovich, A.P.; Elliott, S.J.; Schaus, S.E.; Collins, J.J. Chemogenomic profiling on a genome-wide scale using reverse-engineered gene networks. *Nat. Biotechnol.,* **2005**, *23*(3), 377-383.

[11] Liang, S.; Fuhrman, S.; Somogyi, R. Reveal, a general reverse engineering algorithm for inference of genetic network architectures. *Pac. Symp. Biocomput.,* **1998**, 18-29. Available from: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.8.5380

[12] Pihur, V.; Datta, S.; Datta, S. Reconstruction of genetic association networks from microarray data: A partial least squares approach. *Bioinformatics,* **2008**, *24*(4), 561-568.

[13] Boulesteix, A.L.; Strimmer, K. Partial least squares: A versatile tool for the analysis of high-dimensional genomic data. *Brief. Bioinform.,* **2007**, *8*(1), 32-44.

[14] Chen, T.; He, H.L.; Church, G.M. Modeling gene expression with differential equations In: *Pacific symposium on biocomputing*, vol. 4, **1999**, pp. 4.

[15] Friedman, N.; Linial, M.; Nachman, I.; Pe'er, D. Using bayesian networks to analyze expression data. *J. Comput. Biol.,* **2000**, *7*(3-4), 601-620.

[16] Murphy, K.; Mian, S. Modelling gene expression data using dynamic bayesian networks. **1999**, Technical Report: University of California.

[17] Wille, A.; Zimmermann, P.; Vranová, E.; Fürholz, A.; Laule, O.; Bleuler, S.; Hennig, L.; Prelic, A.; von Rohr, P.; Thiele, L.; Zitzler, E.; Gruissem, W.; Bühlmann, P. Sparse graphical gaussian modeling of the isoprenoid gene network in *Arabidopsis thaliana*. *Genome Biol.,* **2004**, *5*(11), R92.

[18] Chan, Z.S.H.; Havukkala, I., Jain, V.; Hu, Y.; Kasabov, K. Soft computing methods to predict gene regulatory networks: An integrative approach on time-series gene expression data. *Appl. Soft Computing,* **2008**, *8*(3), 1189-1199.

[19] De Jong, H. Modeling and simulation of genetic regulatory systems: A literature review. *J. Comput. Biol.,* **2002,** *9*(1), 67-103.

[20] Tamada, Y.; Imoto, S.; Araki, H.; Nagasaki, M.; Print, C.; Charnock-Jones, D.S.; Miyano, S. Estimating genome-wide gene networks using nonparametric bayesian network models on massively parallel computers. *IEEE/ACM Trans. Comput. Biol. Bioinform.,* **2011**, *8*(3), 683-697.

[21] Hecker, M.; Lambeck, S.; Toepfer, S.; van Someren, E.; Guthke, R. Gene regulatory network inference: Data integration in dynamic models-a review. *Biosystems,* **2009**, *96*(1), 86-103.

[22] Bansal, M.; Belcastro, V.; Ambesi-Impiombato, A.; Bernardo, D.D. How to infer gene networks from expression profiles. *Mol. Syst. Biol.,* **2007**, *3*, 78.

[23] Chai, L.E.; Loh, S.K.; Low, S.T.; Mohamad, M.S.; Deris, S.; Zakaria, Z. A review on the computational approaches for gene regulatory network construction. *Comput. Biol. Med.,* **2014,** *48*, 55-65.

[24] Lee, W.P.; Tzou, W.S. Computational methods for discovering gene networks from expression data. *Brief Bioinform.,* **2009**,

10(4), 408-423.

[25] Schlitt, T.; Brazma, A. Current approaches to gene regulatory network modelling. *BMC Bioinformatics,* **2007**, *8*(Suppl 6), S9.

[26] Sima, C.; Hua, J.; Jung, S. Inference of gene regulatory networks using time-series data: A survey. *Curr. Genomics,* **2009**, *10*(6), 416-429.

[27] Biswas, S.; Acharyya, S. Neural model of gene regulatory network: A survey on supportive meta-heuristics. *Theory Biosci.,* **2016**, *135*(1-2), 1-19.

[28] Sîrbu, A.; Ruskin, H.J.; Crane, M. Comparison of evolutionary algorithms in gene regulatory network model inference. *BMC Bioinformatics,* **2010**, *11*, 59. Available from: https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-11-59

[29] Borkar, S. Thousand core chips: A technology perspective In: Proceedings of the 44th Annual Design Automation Conference, ACM, **2007**, pp. 746-749.

[30] Diaz, J.; Munoz-Caro, C.; Nino, A. A survey of parallel programming models and tools in the multi and many-core era. *IEEE Trans. Parallel Distr. Syst.,* **2012**, *23*(8), 1369-1386.

[31] Nickolls, J.; Buck, I.; Garland, M.; Skadron, K. Scalable parallel programming with cuda. *Queue,* **2008**, *6*(2), 40-53.

[32] CUDA Nvidia. Nvidia cuda c programming guide. Nvidia Corporation, **2011.**

[33] Nobile, M.S.; Cazzaniga, P.; Tangherloni, A.; Besozzi, D. Graphics processing units in bioinformatics, computational biology and systems biology. *Brief Bioinform,* **2017**, *18*(5), 870-885.

[34] Gropp, W.; Lusk, E.; Skjellum, A. Using MPI: Portable parallel programming with the message-passing interface, vol. 1; MIT Press, **1999**.

[35] Lin, H.; Ma, X.; Feng, W.; Samatova, N.F. Coordinating computation and i/o in massively parallel sequence search. *IEEE Trans. Parallel Dist. Syst.,* **2011**, *22*(4), 529-543.

[36] Ferretti, M.; Musci, M.; Santangelo, L. Mpi-cms: A hybrid parallel approach to geometrical motif search in proteins. *Concurr. Comp. Pract. Exp.,* **2015**, *27*(18), 5500-5516.

[37] Chandra, R.; Menon, R.; Dagum, L.; Kohr, D.; Maydan, D.; McDonald, J. *Parallel Programming in OpenMP,* 1st ed. Morgan Kaufmann, **2001**.

[38] Taylor, R.C. An overview of the hadoop/mapreduce/hbase framework and its current applications in bioinformatics. *BMC Bioinformatics,* **2010,** *11*(Suppl 12), S1.

[39] Matsunaga, A.; Tsugawa, M.; Fortes, J. Cloudblast: Combining mapreduce and virtualization on distributed resources for bioinformatics applications In: IEEE Fourth International Conference on eScience, **2008**, pp. 222-229.

[40] Jourdren, L.; Bernard, M.; Dillies, M.A.; Le Crom, S. Eoulsan: A cloud computing-based framework facilitating high throughput sequencing analyses. *Bioinformatics,* **2012**, *28*(11), 1542-1543.

[41] Schumacher, A.; Pireddu, L.; Niemenmaa, M.; Kallio, A.; Korpelainen, E.; Zanetti, G.; Heljanko, K. Seqpig: Simple and scalable scripting for large sequencing data sets in hadoop. *Bioinformatics,* **2014**, *30*(1), 119-120.

[42] Nordberg, H.; Bhatia, K.; Wang, K.; Wang, Z. Biopig: A hadoop-based analytic toolkit for large-scale sequence data. *Bioinformatics,* **2013**, *29*(23), 3014-3019.

[43] Pearl, J. Bayesian networks. Department of Statistics, UCLA, USA, **2011**.

[44] Needham, C.J.; Bradford, J.R.; Bulpitt, A.J.; Westhead, D.R. A primer on learning in bayesian networks for computational biology. *PLoS Comput. Biol.,* **2007**, *3*(8), e129. Available from: http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.0030129

[45] Chickering, D.; Geiger, D.; Heckerman, D. Learning bayesian networks: Search methods and experimental results In: Proceedings of Fifth Conference on Artificial Intelligence and Statistics, **1995**, pp. 112-128.

[46] Friedman, N.; Nachman, I.; Pe´er, D. Learning bayesian network structure from massive datasets: The sparse candidate algorithm In: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence. Morgan Kaufmann Publishers Inc., **1999**, pp. 206-215.

[47] Nikolova, O.; Aluru, S. Parallel bayesian network structure learning with application to gene networks. In: 2012 International Conference for High Performance Computing, Networking, Storage and Analysis (SC), IEEE, **2012**, pp. 1-9.

[48] Zola, J.; Aluru, M.; Sarje, A.; Aluru, S. Parallel information-theory-based construction of genome-wide gene regulatory networks. *IEEE Trans. Parallel Dist. Syst.,* **2010**, *21*(12), 1721-1733.

[49] Misra, S.; Vasimuddin, M.; Pamnany, K.; Chockalingam, S.P.; Dong, Y.; Xie, M. Aluru, M.R.; Aluru, S. Parallel bayesian network structure learning for genome-scale gene networks. In SC14: International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, **2014,** pp. 461-472.

[50] Tamada, Y.; Shimamura, T.; Yamaguchi, R.; Imoto, S.; Nagasaki, M.; Miyano, S. Sign: Large-scale gene network estimation environment for high performance computing. *Genome Informatics,* **2011**, *25*(1), 40-52.

[51] Tamada, Y.; Yamaguchi, R.; Imoto, S.; Hirose, O.; Yoshida, R.; Nagasaki, M.; Miyano, S. Sign-ssm: Open source parallel software for estimating gene networks with state space models. *Bioinformatics*, **2011**, *27*(8), 1172-1173.

[52] Hoeting, J.A.; Madigan, D.; Raftery, A.E.; Volinsky, C.T. Bayesian model averaging: A tutorial. *Statist. Sci.,* **1999**, *14*(4), 382-401.

[53] Young, W.C.; Raftery, A.E.; Yeung, K.Y. Fast bayesian inference for gene regulatory networks using scanbma. *BMC Syst. Biol.,* **2014**, *8*, 47.

[54] Hung, L.H.; Shi, K.; Wu, M.; Young, W.C.; Raftery, A.E.; Yeung, K.Y. fastBMA: Scalable network inference and transitive reduction. *Gigascience,* **2017**, *6*(10), 1-10.

[55] Wen, X.; Fuhrman, S.; Michaels, G.S.; Carr, D.B.; Smith, S.; Barker, J.L.; Somogyi, R. Large-scale temporal gene expression mapping of central nervous system development. *Proc. Natl. Acad. Sci. U.S.A.,* **1998**, *95*(1), 334-339.

[56] Liang, M.; Zhang, F.; Jin, G.; Zhu, J. FastGCN: A gpu accelerated tool for fast gene co-expression networks. *PLoS One*, **2015**, *10*(1), e0116776. Available from: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0116776

[57] Margolin, A.A.; Nemenman, I.; Basso, K.; Wiggins, C.; Stolovitzky, G.; Dalla Favera, R.; Califano, A. Aracne: An algorithm for the reconstruction of gene regulatory networks in a mammalian cellular context. *BMC Bioinformatics*, **2006**, *7*(Suppl 1), S7.

[58] Faith, J.J.; Hayete, B.; Thaden, J.T.; Mogno, I.; Wierzbowski, J.; Cottarel, G.; Kasif, S.; Collins, J.J.; Gardner, T.S.; Jeremiah, J. Large-scale mapping and validation of *Escherichia coli* transcriptional regulation from a compendium of expression profiles. *PLoS Biol.*, **2007**, *5*(1), e8. Available from: http://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.0050008

[59] Meyer, P.E.; Kontos, K.; Lafitte, F.; Bontempi, G. Information-theoretic inference of large transcriptional regulatory networks. *EURASIP J. Bioinform. Syst. Biol.,* **2007**, 79879.

[60] Butte, A.J.; Kohane, I.S. Mutual information relevance networks: Functional genomic clustering using pairwise entropy measurements. *Pac. Symp. Biocomput.,* **2000**, 418-429.

[61] Song, L.; Langfelder, P.; Horvath, S. Comparison of co-expression measures: Mutual information, correlation, and model based indices. *BMC Bioinformatics,* **2012**, *13*, 328. Available from: https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-13-328

[62] Kraskov, A.; Stogbauer, H.; Grassberger, P. Estimating mutual information. *Phys. Rev. E,* **2004**, *69*(6), 066138.

[63] Daub, C.O.; Steuer, R.; Selbig, J.; Kloska, S. Estimating mutual information using b-spline functions-an improved similarity measure for analysing gene expression data. *BMC Bioinformatics*, **2004**, *5*, 118. Available from: https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-5-118

[64] Belcastro, V.; Gregoretti, F.; Siciliano, V.; Santoro, M.; D'Angelo, G.; Oliva, G.; Bernardo, G. Reverse engineering and analysis of genome-wide gene regulatory networks from gene expression profiles using high-performance computing. *IEEE/ACM Trans. Comput. Biol. Bioinform,* **2012**, *9*(3), 668-678.

[65] Misra, S.; Pamnany, K.; Aluru, S. Parallel mutual information based construction of whole-genome networks on the intel (r) xeon phi (tm) coprocessor. *IEEE/ACM Trans. Comput. Biol. Bioinform,* **2015**, *12*(5), 1008-1020.

[66] Shi, H.; Schmidt, B.; Liu, W.; Muller-Wittig, W. Parallel mutual information estimation for inferring gene regulatory networks on gpus. *BMC Res. Notes*, **2011**, *4*(1), 1.

[67] Chockalingam, S.P.; Aluru, M.; Aluru, S. Information theory based genome-scale gene networks construction using mapreduce. In 2015 IEEE 22nd International Conference on High Performance Computing (HiPC); IEEE, **2015**, pp. 464-473.

[68] Zheng, G.; Xu, Y.; Zhang, X.; Liu, Z.P.; Wang, Z.; Chen, L.; Zhu, X.G. Cmip: A software package capable of reconstructing genome-wide regulatory networks using gene expression data. *BMC Bioinformatics*, **2016**, *17*(Suppl 17), 535.

[69] Zhang, X.; Zhao, X.M.; He, K.; Lu, L.; Cao, Y.; Liu, J.; Hao, J.K.; Liu, Z.P.; Chen, L. Inferring gene regulatory networks from gene expression data by path consistency algorithm based on conditional mutual information. *Bioinformatics*, **2012**, *28*(1), 98-104.

[70] Borelli, F.F.; de Camargo, R.Y.; Martins, D.C.; Rozante, L.C. Gene regulatory networks inference using a multi-GPU exhaustive search algorithm. *BMC Bioinformatics,* **2013**, *14*(Suppl 18), S5.

[71] Wang, M.; Verdier, J.; Benedito, V.A.; Tang, Y.; Murray, J.D.; Ge, Y.; Becker, J.D.; Carvalho, H.; Rogers, C.; Udvardi, M.; He, J. LegumeGRN: A gene regulatory network prediction server for functional and comparative studies. *PLoS One,* **2013,** *8*(7), e67434. Available from: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0067434

[72] Haury, A.C.; Mordelet, F.; Vera-Licona, P.; Vert, J.P. Tigress: Trustful inference of gene regulation using stability selection. *BMC Syst. Biol.*, **2012**, *6*(1), 145.

[73] Huynh-Thu, V.A.; Irrthum, A.; Wehenkel, L.; Geurts, P. Inferring regulatory networks from expression data using tree-based methods. *PLoS One,* **2010**, *5*(9), e12776. Available from: https://bmcsystbiol.biomedcentral.com/articles/10.1186/1752-0509-6-145

[74] Sales, G.; Romualdi, C. Parmigenea parallel r package for mutual information estimation and gene network reconstruction. *Bioinformatics,* **2011**, *27*(13), 1876-1877.

[75] Lee, W.P.; Hsiao, Y.T.; Hwang, W.C. Designing a parallel evolutionary algorithm for inferring gene networks on the cloud computing environment. *BMC Syst. Biol.,* **2014**, *8*, 5. Available from: https://bmcsystbiol.biomedcentral.com/articles/10.1186/1752-0509-8-5

[76] Jostins, L.; Jaeger, J. Reverse engineering a gene network using an asynchronous parallel evolution strategy. *BMC Syst. Biol.,* **2010**, *4*, 17. Available from: https://bmcsystbiol.biomedcentral.com/articles/10.1186/1752-0509-4-17

[77] Xiao, X.; Zhang, W.; Zou, X. A new asynchronous parallel algorithm for inferring large-scale gene regulatory networks. *PLoS One*, **2015,** *10*(3), e0119294. Available from: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0119294

[78] Gardner, T.S.; di Bernardo, D.; Lorenz, D.; Collins, J.J. Inferring genetic networks and identifying compound mode of action *via* expression profiling. *Science,* **2003**, *301*(5629), 102-105. Available from: http://science.sciencemag.org/content/301/5629/102

[79] Gregoretti, F.; Belcastro, V.; di Bernardo, D.; Oliva, G. A parallel implementation of the network identification by multiple regression (nir) algorithm to reverse-engineer regulatory gene networks. *PLoS One,* **2010**, *5*(4), e10179. Available from: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0010179

[80] Alba, E. Parallel evolutionary algorithms can achieve superlinear performance. *Inform. Process. Lett.,* **2002**, *82*(1), 7-13.

[81] Spirov, A.; Holloway, D. Using evolutionary computations to understand the design and evolution of gene and cell regulatory networks. *Methods*, **2013**, *62*(1), 39-55.

[82] Kozlov, K.; Samsonov, A. Deep-differential evolution entirely parallel method for gene regulatory networks. *J. Supercomputing,* **2011**, *57*(2), 172-178.

[83] Ramirez-Chavez, L.E.; Coello, C.; Rodriguez-Tello, E. A gpu-based implementation of differential evolution for solving the gene regulatory network model inference problem. In: Proceedings of Fourth International Workshop on Parallel Architectures and Bioinspired Algorithms, **2011**, pp. 21-30.

[84] Segal, E.; Shapira, M.; Regev, A.; Pe'er, D.; Botstein, D.; Koller, D.; Friedman, N. Module networks: Identifying regulatory modules and their condition-specific regulators from gene expression data. *Nat. Genet.,* **2003**, *34*(2), 166-176.