Analyzing Large Biological Datasets in Bioinformatics with Maximal Information Coefficient

Chao Wang, *Member IEEE,* Xi Li, Dong Dai, Aili Wang and Xuehai Zhou

Abstract— The maximal information coefficient (MIC) has been proposed to discover relationships and associations between pairs of variables. It poses significant challenges for bioinformatics scientists to accelerate the MIC calculation, especially in genome sequencing and biological annotations. In this paper we explore a parallel approach which uses MapReduce framework to improve the computing efficiency and throughput of the MIC computation. The acceleration system includes biological data storage on HDFS, preprocessing algorithms, distributed memory cache mechanism, and the partition of MapReduce jobs. Based on the acceleration approach, we extend the traditional two-variable algorithm to multiple variables algorithm. The experimental results show that our parallel solution provides a linear speedup comparing with original algorithm without affecting the correctness and sensitivity.

Index Terms—Bioinformatics; Large Biological Datasets; MIC; MapReduce;

1 Introduction

DNA microarray is able to simultaneously measure the expression levels of thousands of genes, generating huge amounts of data [1]. The analysis of the large scale data presents a tremendous challenge to biologists to gain biological deep insights from raw experiments. Along with the massive data generated from individual genes, examining dataset on a gene-by-gene basis is quite time consuming and difficult to carry out across an entire dataset. A typical example is measuring the relationships between genes by determining the associations between their expression profiles [2]. Imagine a huge data set with tens of thousands of records, each of which may contain tens of thousands of variable pairs—far too many to examine manually [3]. However, many undiscovered relationships are still hidden in the explosive large scale data, and how to identify important factors for a chosen objectivity efficiently has become an important scientific issue to be solved.

Determination of sequence similarity is one of the major steps in computational phylogenetic studies. During evolutionary history, not only DNA mutations for individual nucleotide but also subsequent rearrangements have occurred. It has been one of major tasks for computational biologists to develop novel mathematical descriptors for similarity analysis such that various mutation phenomena information would be involved simultaneously [4]. Figure 1 illustrates the basic step of the genome sequencing mapping. The top part in the figure indicates the sequence alignment, where the reference and the reads are all strings consisted of *'A', 'C', 'T', 'G',* as illustrated by the bottom of the figure [5]. Underlined characters in reads represent the gnome variations. The motivation is to identify the similarity of the referenced genome and the short reads. However, this process could be quite timing consuming and has posed significant challenges to the researchers.

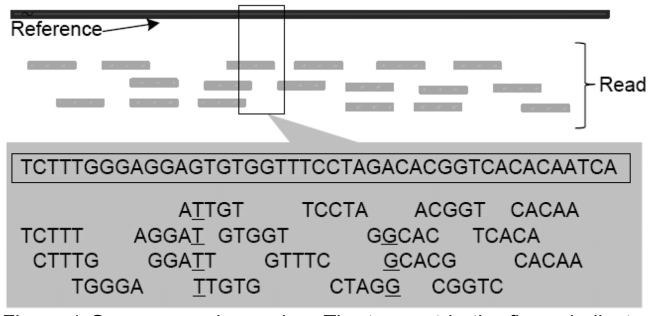

Reference
Read
TCTTTGGGAGGAGTGTGGTTTCCTAGACACGGTCACACAATCA
ATTGT TCCTA ACGGT CACAA
TCTTT AGGAT GTGGT GGCAC TCACA
CTTTG GGATT GTTTC GCACG CACAA
TGGGA TTGTG CTAGG CGGTC

Figure 1 Genome read mapping. The top part in the figure indicates the sequence alignment, where the reference and the reads are all strings consisted of 'A', 'C', 'T', 'G', as illustrated by the bottom of the figure. Underlined characters in reads represent the gnome variations.

To tackle this problem, Reshef et al. [6] proposed a novel correlation measurement "maximal information coefficient" (MIC), and presented a 1-D dynamic programming algorithm to calculate the MIC. MIC does not rely on the distributional assumptions of measured data and could identify a broad class of associations compared

- *C. Wang is with the University of Science and Technology of China, Hefei, 230027, Anhui, China. E-mail: cswang@ustc.edu.cn*
- *D. Dai is with School of Computer Science, Texas Tech University, Lubbock. Email: daidong@mail.ustc.edu.cn*
- *A. Wang is with the School of Software Engineering, University of Science and Technology of China, Suzhou, 215123, Jiangsu, China. E-mail: wangal@ustc.edu.cn.*
- *X.Li and X.Zhou are with the Suzhou Institute of University of Science and Technology of China, Suzhou, 215123, Jiangsu, China. E-mail: llxx,xhzhou@ustc.edu.cn.*
- *Manuscript received (insert date of submission if desired). Please note that all acknowledgments should be placed at the end of the paper, before the bibliography.*

with previous studies. Given a discrete random variable x and all its possible values, we can calculate the information content of any event $x=k$. Furthermore, given the distribution of x, we can calculate the average information content of its distribution. It has been named as Shannon Entropy $H(x)$. Based on the Shannon entropy, we can define the mutual information $I(x, y)$, which refers to the uncertainty about y, consequently $I(x,y)=H(y)-H(y|x)$. For example, if x and y are independent, we can easily calculate $I(x,y) = 0$. On the contrary, if x and y have strong relationship $y= f(x)$, then $H(y|x)=0$, and $I(x, y)$ reaches the maximum value.

Consider two given variables: the objective *(obj)* and the factor *(fac)*, with millions of sample pairs (fac_i, obj_i). To compute the mutual information of these two variables, we need to try all the different ways to draw a grid on the scatter plot of these two variables. After exploring all grids up to a maximal grid resolution, which are dependent on the sample size, we can compute for each (fac_i, obj_i) the largest possible mutual information (I_g) achievable by any $x*y$ grid. To process a fair comparison between grids of different dimensions, we can normalize these mutual information values into 0 to 1. According to the grids number, we can define a characteristic matrix $M = \{m_{x,y}\}$, where $m_{x,y}$ is the peak normalized mutual information from all the possible $x*y$ grids.

To propose a formal definition: for a grid G, I_g denotes the formalized mutual information of the probability distribution induced on the boxes of G, where the probability of a box is proportional to the number of data points falling inside the box. The $(x, y)^{th}$ entry $m_{x,y}$ of the characteristic matrix equals to max $\{I_g/log(m_{x,y})\}$. The maximum is taken over all $x*y$ grids G. MIC is the maximum of $m_{x,y}$ over ordered pairs (x, y) such that $x* y \leqslant B$, where B is a function of sample size, which is usually set to $n^{0.6}$. Every entry of M and MIC fall between 0 and 1, while MIC is also symmetric due to the symmetric mutual information. Due to that I_g depends only on the rank order of the data, MIC is invariant under order preserving transformations of the axes.

MIC has been regarded an effective statistic measure [6] to evaluate the association between two variables due to its two important heuristic properties: **generality** and **equitability**. **Generality** indicates that with sufficient sample size, the statistic should capture a wide range of interesting associations, not limited to specific function types (such as linear, exponential, or periodic), or even to all the functional relationships. The latter condition is desirable because not only do relationships take many functional forms, but also many important relationships (like superposition functions) are not well modeled by a specific function. **Equitability** means that the statistic should give similar scores to equally noisy relationships of different types. Furthermore, equitability is difficult to formalize for associations in general but has a clear interpretation in the basic case of functional relationships: an equitable statistic should give similar scores to functional relationship with similar R^2 values (given sufficient sample size). Besides these two properties, MIC gives rise to a large family of statistics, which was referred to as MINE, or maximal information-based nonparametric exploration. MINE statistics can be used not only to identify interesting associations, but also to characterize them according to properties such as nonlinearity and monotonicity.

However, the MIC computation algorithm is quite complex so that it is not feasible to be applied on large data sets for now. In this paper, we propose a solution based on MapReduce [7] and other infrastructures to solve this problem. Our solution extends current software algorithm into parallel manner and achieves linear speedup without the affecting the correctness and sensitivity. We use the open-source implementation of the distributed programming framework including Hadoop [8] and memcached [9] to implement this parallel algorithm.

In this paper, we propose a uniform accelerating cluster based system to solve this problem. We claim following contributions:

1) Our major contribution is the MapReduce acceleration implementation based on the state-of-the-art serial algorithm. The acceleration can achieve higher performance with correctness and satisfying scalability.

2) In this work, we intend to analyze and demonstrate important properties of the large scale association [10]. Based on the understanding of the problem as well as heterogeneous accelerators based cluster architecture, we show that accelerator's parallel processing power can be fully mobilized to achieve high speed data association detecting.

3) In order to improve the applicability of the algorithm, we extend the traditional algorithm to a multi-variable algorithm. Experimental results verify the correctness and the efficiency of the new algorithm.

The structure of this paper is organized as below. In Section 2, we summarize the related work, and then in Section 3 we first introduce the original algorithm, and then have some discussions on the MapReduce framework to accelerate the algorithm in parallel. In Section 4, we describe our MapReduce algorithm in detail, and propose a phase detection method to locate the execution stages. Then we demonstrate an approximate way to extend current solution to multi variables. Thereafter, the experimental results are described in Section 5. Finally, Section 6 concludes the paper and introduces future works.

2 Related Work

For years, the MIC has been applied in various bioinformatics fields successfully. For example, the MIC has been used as a measure to convert records of biological annotations into networks of the associated annotations [11]. Billions of samples and thousands of variables make the computing impossible and time consuming. MAPPFinder [2] presents an identification method for important biological processes, which is a software tool set for viewing and analyzing microarray data representing biological pathways or any other functional grouping of genes.

Meanwhile, accelerators like Cloud[12], GPU [13], and FPGA [14] have been major dominant engines in big data research, such as genome sequencing in bioinformatics

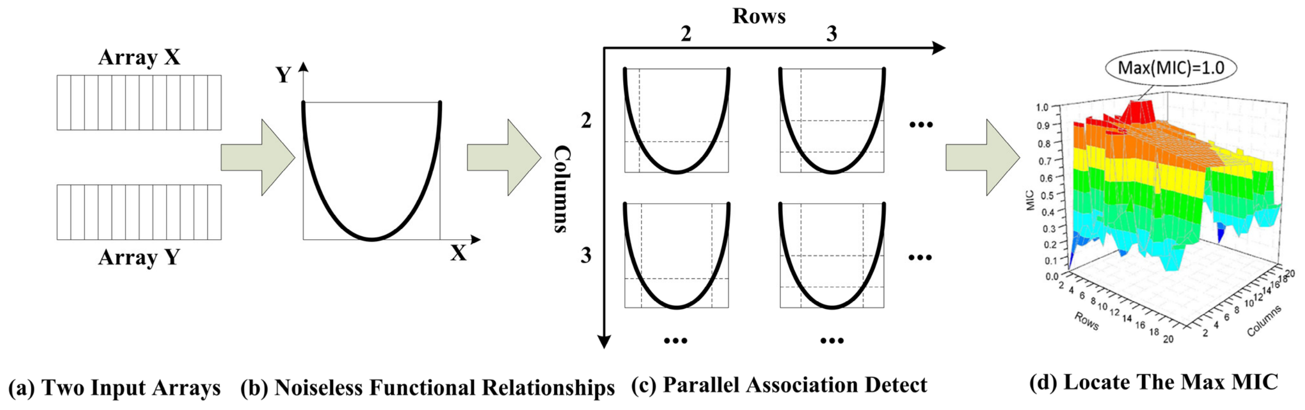


Figure 2. Detailed process of MINE algorithm description

[15-18]. Of these literatures, how to identify the data correlation has posed significant challenges to the academic and industrial researchers. There have been a wide range of methods for identifying interesting relationships between pairs of variables in large data sets in bioinformatics [19], including methods formulated around the axiomatic framework for measures of dependence [20], other state-of-the-art measures of dependence, and several nonparametric estimation techniques that can be used to score pairs of variables based on the relationship of the estimated curve. Methods such as splines and regression estimators [21] tend to be equitable across functional relationships but they fail to find many simple and important types of relationships that are not functional. Although these methods are not intended to provide generality, most of them are unsuitable for identifying all potentially interesting relationships in a large scale data set. Similar methods such as mutual information estimators, maximal correlation, principal curve, distance correlation, and the spearman rank correlation coefficient methods are able to detect broader classes of relationships. However, these methods are not equitable even in the basic case of functional relationships: They show a strong preference for some types of functions, even at identical noise levels. For example, Reshef [6] has established the generality of maximal information coefficient (MIC) through proofs, showing its equitability on functional relationships through simulations, and observe the intuitively equitable behavior on more general associations.

Although researchers have proven MIC is a creditable approach to detect the relationships between variable pairs, it still consumes significant time for analyzing large scale data set due to the complex calculation process. With respect to the optimization approaches, GPU and FPGA based approaches are two dominant methodologies in heterogeneous architecture design paradigms. For example, RapidMic [22] is a cross-platform tool for the rapid computation of the maximal information coefficient based on parallel computing methods. Through parallel processing, it can effectively analyze the large-scale biological datasets with a remarkable reduced computing time. Similarly a simulated annealing and genetic algorithm was developed [23] to facilitate the optimal calculation of MIC, and the convergence of SG was proved based on Markov theory. Lopez-Paz et.al [24] introduce the randomized dependence coefficient, which is a measure of nonlinear dependence between random variables of arbitrary dimension. Kinney et. al [25] identify artifacts in the reported simulation, in particular for the estimates of mutual information when these artifacts are removed. Recently Nature Biotechnology [26] solicits comments from several practitioners versed in data-intensive biological research. Their responses not only highlight the appeal of methods like MIC for biological research, but also raise some important reservations as to its widespread use and statistical power. Paninski [27] presents some results on the nonparametric estimation of entropy and mutual information. Kraskov et. al [28] present two classes of improved estimators for mutual information, from samples of random points distributed according to some joint probability density. To show the effectiveness of MIC in medical imaging field, Pluim et.al [29] summarize the MIC based registration of medical images. Reshef et.al [30] present the MIC calculation with more comprehensive understanding to show the effectiveness and efficiency. Similarly, many scientific applications have been optimized by GPU and FPGA accelerators, such as [31], [5] and [16]. In particular, these approaches consist of a series of nodes, each of which has both a CPU controller and a heterogeneous accelerator. All nodes are under the controlling of the scheduler that is responsible for issuing tasks and balancing workloads, which increase the design complexity and burden of the software programmers.

3 MIC Calculation Algorithm

To reduce the computation complexity in the original algorithm, Reshef [6] has introduced a dynamic programming improvement. In this section, we will demonstrate the algorithm first, and then analyze the strategy of parallelization using MapReduce framework on multiple computing machines.

3.1. Original Algorithm Description

In particular, the MINE algorithm is designed for heuristically generating the characteristic matrix of two-variable data sets. To calculate MIC matrix, the algorithm would ideally optimize over all possible grids.

For computational efficiency, the algorithm instead uses a dynamic programming algorithm which optimizes over a subset of the possible grids and performs the approximate computation well to the true value of MIC in practice. The MINE algorithm description can be divided into four steps as shown in Fig. 2:

(1) Input arrays. In this step the algorithm receives the two-variable input data.

(2) Noiseless functional relationships. In this step the received data is aligned to the two-dimensional coordinate quadrant. Actually the curve refers to the functional relationships with noises. In order to explain the following steps clearly, we use noiseless relationships for further discussion. Noiseless relationships mean that all the received data is part of a specific function.

(3) Parallel association detection. To calculate the MIC of a two-variable data set, we explore all the grids up to a maximal grid resolution, depending on the sample size. Then the largest possible mutual information achievable by any x-by-y grid is calculated for every pair of integers *(x,y)*.

(4) Locate the maximum MIC. These mutual information values are normalized to ensure a fair comparison between grids of different dimensions and to obtain modified values between 0 and 1. Finally the peak normalized mutual information achieved by any x-by-y grid refers to the MIC.

3.2. MapReduce Acceleration

Accelerate the original algorithm is necessary because it is time-consuming especially when facing large scale input data size. The huge size of input dataset usually cannot be stored in one server, and it is typically stored in distributed storage systems or distributed databases. Storing data across thousands of machines means each server only stores a small part of the dataset. In such condition, the original algorithm would become inefficient as it needs too many communications when processing data that is not stored locally. What's worse, the limitation that only one single sever can execute the algorithm makes it impossible for large input data.

To accelerate the original algorithm, we provide a system-level distributed framework including the storage pattern, the way to parallel original algorithm, and the components.

1) Storage Pattern: Before discussing the MapReduce solution, we first describe the storage pattern. The large dataset can be stored in all kinds of distributed storage systems. However, in our specific MapReduce based scenario, input data collected by recorders or loggers is stored as text files in HDFS cluster, as illustrated in Fig. 3. Each row of the logs.data contains thousands of variables. The first column of Fig. 3 indicates that the objective is to find out whether other variables have relationships with. From the second column, we store all the collected variables: fac_1, fac_2… fac_n. Please note that all the rows are stored unordered. In HDFS, this log file (*logs.data*) will be divided into chunks (typically 64 MB) and stored into different storage nodes. Usually HDFS will replicate each chunk into three servers, consequently when a MapReduce job is started, the Hadoop manager will try to schedule tasks running on the node with the tasks input data set. In our solution, we take advantage of this locality to achieve a better performance.

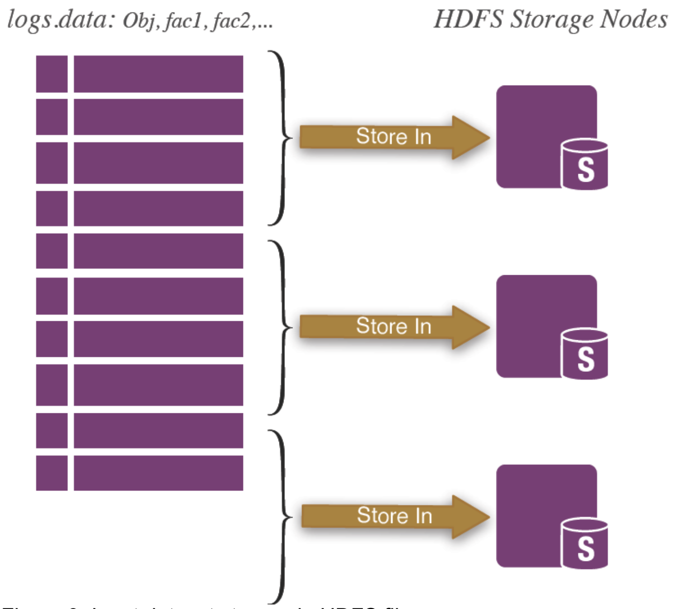


Figure 3. Input dataset storage in HDFS files.

2) Parallel Analysis and Design: According to the algorithm that has been presented in Reshef's work [6], it can be concluded that different *y* will generate completely different MIC results. This means we may be able to divide the computing process into different irrelevant parts according their *y* partition number, and execute them on different servers in parallel. However, due to that each server still needs the overall input dataset during this computation, it is difficult to run this computation in parallel. This means the algorithm is still infeasible facing the huge data set which cannot be accommodated on one machine.

Although we cannot parallelize the MIC computing procedure in such a straightforward way, we can try other non-trivial ways. As shown in Fig. 4. (a), the dependence graph of the original algorithm can be discovered inside each loop during the execution: there is $B/2$ loops (B denotes the maximum grids we want to draw) and each loop contains three dependent procedures: equally partition y-axis, optimally partition x-axis, and calculate $x * y$ grids' mutual information. Fig. 4. (b) demonstrates that we can change the inner-loop dependence to a nested-loop dependence. After getting rid of the inner-loop dependence, we will get three dependent loops, and accelerate the three tasks in sequence. The detailed loop unrolling techniques will be presented in Section 4.2 Loop Unrolling Process.

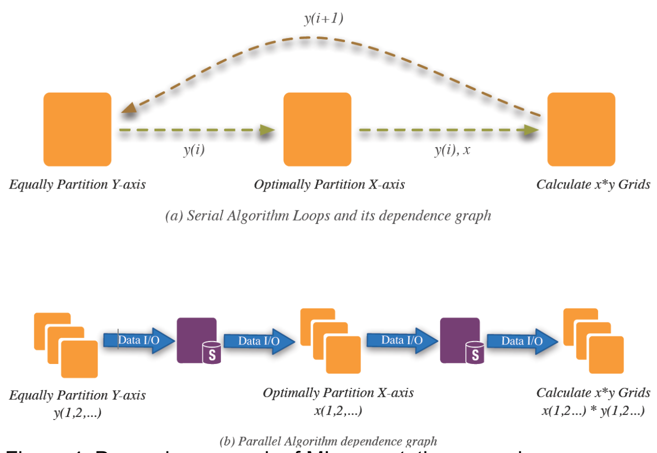


Figure 4. Dependence graph of MI computation procedure

Regarding the task partitioning, the first task is partitioning y-axis equally in a distributed manner. EquiPartitionY Axis should make sure that each part contains the same number of points. In HDFS, each physical server stores parts of dataset out-of-order, therefore we divide the entire dataset into different servers, each of which contains an equal range according to the points' y value. Thereafter, EquippartitionY Axis can be executed locally. So far, we have got the equal partitions for each equal y ranges in each server, and then we combine these equal partitions together to form an overall equal partition. Due to the imbalance of ranges, it is not possible to form an equal partition just by one combination. Anyway, a positive perspective is that we can start a multi-round task to make the partition as equal as possible.

The second task is the x-axis partitioning based on Theorem 1. Fig. 5 shows the overall procedure of optimal x-axis partition. All the nodes can be distributed into different servers using the same strategy as we did in equally partitioning y-axis. Given a server number of k, we divide the input dataset in HDFS into k ordered parts with equal range according to the points' x value. After this data distribution, we explore all the possible x-axis partitions locally in each server.

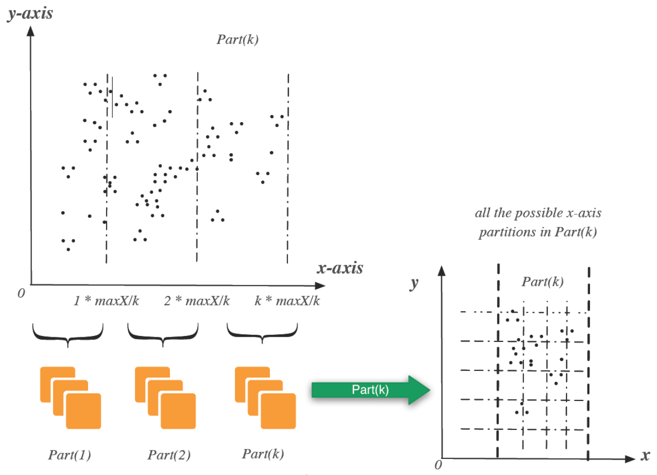


Figure 5. Partition x-axis with k fix partition lines.

After this step, we have calculated the entire possible x * y grids in the k parts (Part (k) in Fig. 5), subsequently we need to combine the intermediate results into the final result. As shown in Fig. 5, the x-axis partition at least contains k division due to that we have to divide all the data pairs into k parts, and run each part in a separate server. However, this fixed k parts would not affect the correctness at all because it is easily to choose k wisely: making it small enough without affecting the final result or we can even run the x-axis partition procedure again for a slice different k division. Two different k divisions can guarantee that we explorer all the possible x * y grids and provide the exactly correct results.

4. MapReduce Solution and Optimization

4.1 MapReduce Solution

The MapReduce based parallel MIC algorithm includes three dependent stages: 1) equally divide the y-axis in parallel, 2) explore all the possible x-axis partitions with every y-axis partition, and 3) calculate the maximum mutual information for each x * y grid.

Algorithm 4 DriverMain(D)

Require:

```
    Input data file $f$ from HDFS
 1: $d(y, x_1, x_{2,...,} x_n) \leftarrow$ fs.load($f$)
 2: $p(y, x_i) \leftarrow d(y, x_i), 1 < i < n$
 3: $p^\tau(x_i, y) \leftarrow d(x_i, y), 1 < i < n$
 4: for all $y \in \{2, \ldots, \lfloor B/2 \rfloor\}, x \leftarrow \lfloor B/y \rfloor$ do
 5:     $(Q_1, \ldots, Q_y) \leftarrow$ MREquiPartitionYAxis($p$, $y$)
 6:     $(I_{2,y}, \ldots, I_{x,y}) \leftarrow$ MROptimizePartitionXAxis($p$, $Q$)
 7:     $(Q_1^\tau, \ldots, Q_y^\tau) \leftarrow$ MREquiPartitionYAxis($p^\tau$, $y$)
 8:     $(I_{2,y}^\tau, \ldots, I_{x,y}^\tau) \leftarrow$ MROptimizePartitionXAxis($p^\tau$, $Q^\tau$)
 9: end for
10: for $(x, y)$ such that xy $\leqslant B$ do
11:     $I_{x,y} \leftarrow \max\{I_{x,y}, I_{x,y}^\tau\}$
12:     $M_{x,y} \leftarrow I_{x,y} / \min\{\log x, \log y\}$
13: end for
14: return $\{M_{x,y} : xy \leqslant B\}$
```

The Alg. 4 DirverMain describes the main function to drive these three stages to finish the MIC calculation. The y-axis partition results *(Q)* generated by MREquiPartitionYAxis will be used as the input set of MROptimizePartitionXAxis. Then MROptimizePartitionXAxis function will explore the entire possible x*y grids and produce the mutual information for different x*y grids. After normalizing the $I_{x,y}$ array, we can get the maximum mutual information array $M_{x,y}$. Meanwhile, the same algorithm will be applied to the same data pairs with reversed x and y to make sure the correctness of the approximation.

1) MREquiPartitionYAxis: Before partition dataset in y-axis based on MapReduce, we need some global variables: the sample size, denoted as n_y, the maximum y-value of all the samples, denoted as y_n, and the maximum x-value of all the samples, denoted as x_n. All these variables should be maintained while the input dataset was collected.

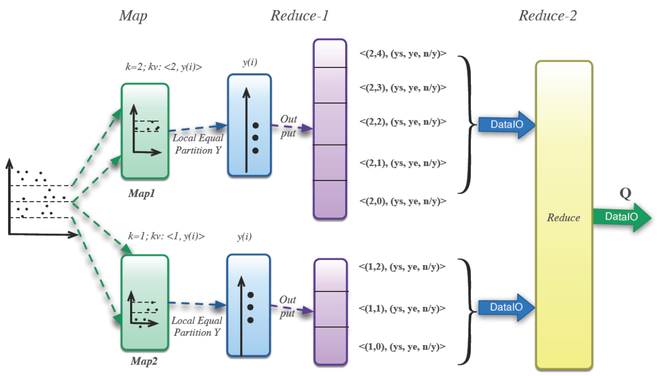


Figure 6. MR procedure of equally partition y-axis

Fig.6 illustrates the MapReduce procedure of MREquiPartitionYAxis. We firstly emit the $<k,y_i>$ to different reducers by judging whether $y_n/y\ (k) < y_i < y_n/y\ (k+1)$. Each reduce function should sort all the y_i that it receives, and generates equal partitions locally using the serial algorithm. In each reduce function, we emit $<(k, p_i), (y_s, y_e, n_{ki})>$ after equally partitioning the dataset locally. In these key-value pairs that the reduce function finally generates, k means this partition is generated in the k^{th} reducer, pi denotes this partition's index inside the k^{th} reducer, (y_s, y_e) shows the partition start point and the end point, n_{ki} means the node number in current partition. Considering the partition number is quite small (comparing with elements number), we process these $<(k, p_i),(y_s, y_e, n_{ki})>$ key-value pairs in one reduce function (Reduce-2 phase in Fig. 6). In this reduce function, we choose the partition that contains the least node numbers, and then combine them with their adjacent partitions, until we finally get y nearly equal partitions.

After phase Reduce-2, we get the y-axis partition Q, but it is not a perfect equal partition because the combination of two adjacent partitions may generate some unbalanced partition. To solve this problem, we should just run MROptEquipartitionYAxis iteratively to generate optimized equal partitions. The strategy is simple: In MROptEquipartitionYAxis, map all the sample pairs according partition Q, and send $<k, y_i>$ to reduce functions. The k^{th} reduce function will process nodes belong to the $\{q^{k-1}, q^k\}$. Each reduce function will sort all the received nodes in order, and check whether its nodes number is bigger than n_y/y, if exceed, it will emit the extra nodes with $<k+1, y>$ to next iteration. The reduce function in next iteration will repeat this operation. Users can set the iteration numbers with configurations, and after y rounds, an optimized hat Q is achieved.

2) MROptimizePartitionXAxis: MROptimizePartitionXAxis calculate the maximal mutual information of each possible x-axis partition for a given y-axis partition. After running MROptimizePartitionXAxis on a given y-axis partition, we will get the return vector represented as the set: $\{I_{2,y}, I_{3,y} \dots I_{\lfloor B/y \rfloor, y}\}$.

$$I\left(x,y\right) = \sum p\left(y\right) \log \frac{1}{p\left(y\right)} + \sum p\left(x,y\right) \log \frac{p\left(x,y\right)}{p\left(y\right)} \qquad (1)$$

According to the definition of mutual information illustrated in Equation 1, we can easily conclude that, for a specific Q (which means a given y-axis partition), the first part of mutual is a constant. So, we only need to calculate the maximum of the second part to get the largest $I(x,y)$. Calculating the maximum of the second part in Equation 1 means calculating the maximal value of following Equation:

$$\mathrm{W}\ (\mathrm{x},\mathrm{y}) = \sum p\left(x,y\right) \log \frac{p(x,y)}{p(y)}$$

This calculation can be easily paralleled in MapReduce way. Firstly, we divide the sample pairs into k parts using map function, which emits $<k, xi>$ to different reducers by judging whether $q_k < x_i < q_{k+1}$. Then, each part is processed in one node by a reduce function. Reduce function does the similar work as the original optimize PartitionXAxis except for one key difference: all the $W(x, y)$ calculation is based on the global information instead of the local information. For example, to calculate $p(x, y)$, we use overall sample nodes number n_y, not the node number belong to the k^{th} part. The reason we use global node number instead of the local number to calculate $W(x, y)$ can be easily explained by Equation 2. If we calculate $p(x, y)$ using global node number, then $W(x_i, y)$ can be used in calculating $W(x, y)$.

$$W\left(x,y\right) = \sum W\left(x_i,y\right) \qquad (2)$$

Based on this observation, we can distribute the complex $W(x, y)$ calculation to different reducers, and combine them together. The k^{th} reducer will use the modified serial version optimize PartitionXAxis to calculate $W(x, y)$, and it will return a vector as following:

$$\mathrm{V}\ (\mathrm{k}) = \left\{ W_{1,y}^{s(k)}, W_{2,y}^{s(k)}, \dots, W_{\lfloor B/y \rfloor, y}^{s(k)} \right\}$$

As a consequence, each reducer will emit $<k,W(k)>$ to the last phase reduce function. In the last reduce function, we combine $\{W(1), W(2),\dots\dots,W(k)\}$ into a maximal vector W as following equation shows. Although the final vector W is start from k not 2, it would not change the result.

$$\begin{pmatrix} W_{1,y}^{s(1)} \\ W_{2,y}^{s(1)} \\ \dots \\ \dots \\ \dots \\ W_{\lfloor B/y \rfloor,y}^{s(k)} \end{pmatrix} \oplus \dots \begin{pmatrix} W_{1,y}^{s(k)} \\ W_{2,y}^{s(k)} \\ \dots \\ \dots \\ \dots \\ W_{\lfloor B/y \rfloor,y}^{s(k)} \end{pmatrix} = \begin{pmatrix} W_{k,y} \\ W_{k+1,y} \\ \dots \\ \dots \\ \dots \\ W_{\lfloor B/y \rfloor,y} \end{pmatrix}$$

As we have described, we list the MROptimizePartitionXAxis algorithm bellow. The overall computation includes three phases, including one map phase and two reduce phases. At last, it will generate a vector W, which is the second part of the corresponding mutual information vector. We can easily calculate $I_{x,y}$ vector based on the vector V .

Algorithm 5 $\mathrm{MROptimizePartitionXAxis}(p, Q)$

```
Require:
    Input sample pairs $p$, y-axis partition Q;
1:  Map Phase
2:    $\mathrm{pair}_k \leftarrow local(p)$
3:    for each (x, y) in $\mathrm{pair}_k$
4:      if $q_k < x_i < q_{k+1}$
5:        send $\langle k, (x_i, y_i) \rangle$ to kth reducer
6:  Reduce Phase 1
7:    $\mathrm{pair}_k \leftarrow receive()$
8:    $\mathrm{SortPair}_k \leftarrow \mathrm{sort}(\mathrm{pair}_k)$
9:    $\left\{ W_{1,y}^{s(k)}, \ldots W_{\lfloor B/y \rfloor,y}^{s(k)} \right\} \leftarrow ApproxOptimizeXAxis(\mathrm{SortPair}_k, Q)$;
10:   send <k, $\left\{ W_{1,y}^{s(k)}, W_{2,y}^{s(k)}, \ldots, W_{\lfloor B/y \rfloor,y}^{s(k)} \right\}$ > to the final reduce
11: Reduce Phase 2
12:   $\{V(1), V(2), \ldots, V(k)\} \leftarrow receive()$
13:   for i from k to $\lfloor B/y \rfloor$
14:     $W_{i,y} = \sum W_{j,y}^{s(1)}$ for all $\sum j$ = k
15:   return $\left\{ W_{1,y}, W_{2,y}, \ldots, W_{\lfloor B/y \rfloor,y} \right\}$
```

3) Optimization for Combination: The last reduce phase of Alg. 5 contains a complex computation procedure. It may become impossible to calculate according to the data set's size. For example, we have a data set with *N* nodes, and then the maximal x-axis partition number would be $N^{0.6}/2$. If we have R reducers that execute the reduce phase 1 of Alg. 5, then it will generate R vectors that contains the partial best mutual information. Each vector W_i from these R vectors contains $N^{0.6}/2$ elements. To get the maximal W results from these vectors, it will need $(N^{0.6}/2)^R$ computations, which would be pretty huge as is shown in Table I.

TABLE I
COMPLEXITY OF SIX REDUCERS (1GHZ CPU)

Data Set Size	Computation Complexity	Time
1,000	$9.85e10^6$	10^{-3}s
10,000	$3.9e10^{12}$	10^3s
100,000	$1.56e10^{16}$	10^7s
1,000,000	$6.22e10^{19}$	10^{10}s

It is clear that the complexity will increase in an exponential order. This is obviously not acceptable for big data set. To solve this problem, we firstly sort the vectors inside each reduce function and add all the maximums from different reducers. If the summation is much less than 1.0, we can guarantee that the final combination result would be less than 1.0, and consequently those two variables do not have any strong relationship. Otherwise, if the summation is close to 1.0, we calculate the combination by choosing the nearby smaller elements from the sorted vectors each time, once we have fulfilled the MaxMI array or we find out that the normalized mutual information is much less than 1.0, we can stop and set the remained MaxMI array to be zero safely. In such way, the proposed methodology is able to alleviate the design complexity of the combination operations without sacrificing the correctness of the results.

4.2 Loop Unrolling Process

The dependence graph of the original algorithm is inside each loop during the execution. In this section we demonstrate the methodology to change the inner-loop dependence to nested-loop dependence. After getting rid of the inner-loop dependence, we will get three dependent loops, and accelerate the three tasks one by one.

In particular, we inherit the description of state-of-the art dataflow execution model proposed by [32] and [33]. Based on this model, we extend the definition to a general heterogeneous multicore computing scenario in this paper. Generally, dataflow execution model executes tasks with dependences using *tokens* to signal production and availability of parameters. Based on the token based technique, we make two crucial enhancements. First we associate tokens with objects instead of individual memory locations, to match the abstraction for functional source and destination parameters. Second, we assign each object multiple *read tokens* and a single *write token*, to manage both production and consumption of parameters.

When the execution encounters a function (in this paper, denoted as a *task*) to be considered for dataflow execution, it requests read (write) tokens for objects in the function read (write) set. The pending task is ready for execution only after it has acquired all its requisite objects. Upon completion, it releases the tokens which are then spawned to the shelved function(s) if necessary. When a shelved function has acquired its requisite tokens, it can be *unshelved* and submitted for execution. In this section, we model the loop based applications on a general state-of-the-art cluster hardware model, and then present the phase based detection and pipeline techniques.

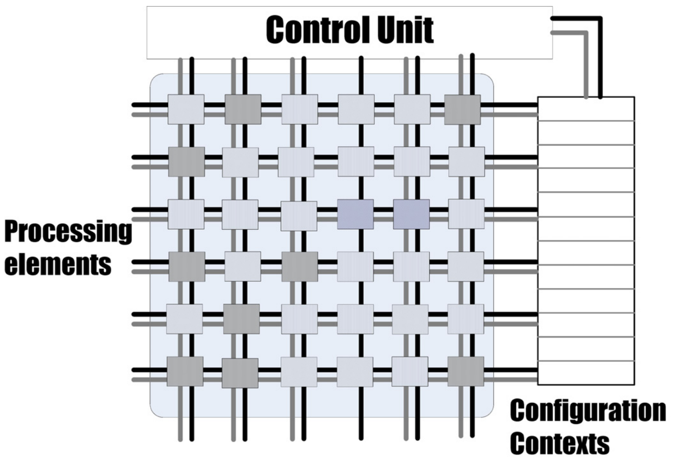


Figure 7. PE based Cluster Architecture.

A high level general architecture framework of the clusters is illustrated in Fig. 7. The proposed architecture is based on massive computing nodes, each of which can perform specific MapReduce operations.

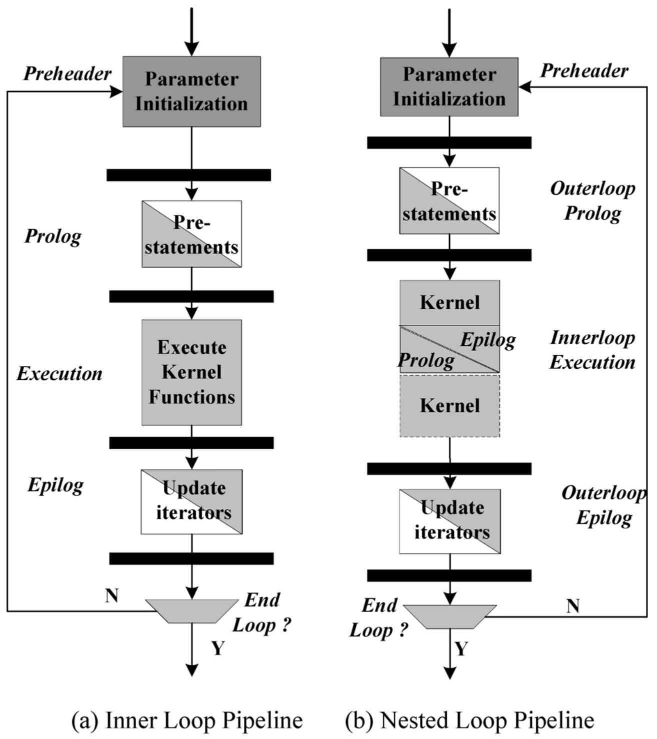


Figure 8. Inner-loop and Nested-Loop Pipelining.

4.2.1 Phase Definition

Before the DFG can be mapped to different nodes at run time, it is essential to analyze the loop based task behavior and detect different phases. Considering the program in Fig. 8 (a), inside each round inside the loop, different tasks (Task T…G) are executed in a pipeline. Every time the pipeline should be fulfilled and flushed, this can be regarded as ***prolog phase*** and ***epilog phase*** [34], while the execution at full pipeline can be treated as the ***kernel phase***. One complexity in pipelining is how to correctly handle prolog and epilog, which are the initial and the final iterations when the pipeline is not full. Prolog/epilog code could be generated separately from the kernel code during compilation, which however is very likely to increase the code size or the number of configurations, and therefore detrimental for clusters, as the cluster can keep only a very limited number of configurations locally. Furthermore, if instead prolog/epilog configurations can be generated from the kernel configurations at runtime by the control unit itself, no extra configuration will be necessary. The other parameters of which the control unit needs to be aware include LC (Loop Count), which is the trip count of the loop, and EC (Epilog Count), which is the schedule length in terms of stages.

4.2.2 Inner-Loop Pipelining

An inner-loop pipeline can be viewed as a four-step process consisting of preheader, prolog, kernel, and epilog, which is illustrated in Fig. 8 (a). The preheader, which includes loop-invariant code execution as well as issuing some pipeline-related tasks, is performed primarily by the main processor, which may enlist the help of PE arrays. However, the other steps, prolog, kernel, and epilog, are performed autonomously by the PE arrays in the cluster.

Single-loop pipelining can generate a pipeline for the innermost loop only. Therefore single-loop pipelining is also Inner-Loop Pipelining (ILP). The conventional approach to loop nests has been inner-loop pipelining with the outer loop implemented in software on a main processor. This approach not only suffers from significant performance overhead due to communication in hybrid architectures, but it is also unable to take advantage of outer-loop pipelining. We address those problems by overlapping outer-loop iterations through sophisticated rescheduling on cluster, called Outerloop Pipelining. Note that mapping the entire loop nest onto cluster can not only reduce communication delay, but also eliminate the need for communication at all.

4.2.3 Nested-Loop Pipelining (NLP).

Fig. 8 (b) illustrates the execution of nested-loop pipelining after merging, which can be divided into three parts. The first part is from the beginning up to the first prolog, which can be called outerloop prolog. The second part is from the first kernel up to the last PIC, right before the last kernel. This part is periodic, consisting of multiple repetitions of the kernel-EIC-kernel-PIC pattern, and may be called outerloop kernel. The rest can be called outerloop epilog. The PE can autonomously execute the outerloop kernel plus the first prolog and the last kernel and epilog. The rest can be done in software on the main processor automatically.

4.3 Multi-Variable Algorithm

After the process is divided into different phases, we have described the entire MapReduce solution for detecting associations between two variables. However, most relationships are between the objectivity and more than two variables. For example, $z = f(x, y)$ denotes that z has a functional relationship with x and y, but it is still possible that (z, x) and (z, y) do not contain strong associations. So, we need to extend the original algorithm to process multi-variable situations.

Generally, there are two ways to define the mutual information with multiple variables: the higher-order mutual information and the mutual information for multiple variables. Higher-order mutual information is defined like: $I(x; y; z) = H(x) + H(y) + H(z) - H(x, y, z)$. It only shows the entropy of multiple variables *(x, y, and z)* joint distribution. However, in our case, we need to know how much information a set of variables $X_1;X_2,\ldots\ldots,X_k$ have about a given outcome or target variable C, this should be defined as in equation 3:

$$\begin{aligned} &I(C; X_1, X_2, \ldots X_k) = H(C) - H(C|X_1, X_2, \ldots X_k) \\ &I(C; X_1, X_2, \ldots X_k) = H(C) + H(X_1, X_2 \ldots X_k) \\ &-H(C, X_1, X_2 \ldots X_k) \end{aligned} \quad (3)$$

As a consequence, for three variables, we know that $I(z{:}x,y) = H(z)+H(x, y) - H(z, x, y)$ and we can use the same strategy described in previous sections to calculate MIC:

we can equally divide the *z*-axis to make *H(z)* maximal, and for each z-axis partition, we need to explore all the possible *x, y* partitions to make *H(x, y)-H(z, x, y)* to reach the peak. However, this will be much more complex than in 2-dimension because this is an arbitrary x-axis and arbitrary y-axis exploration. As a result, the MapReduce solution is more important because when the dimension increases, the grids we need to explore would increase exponentially. In this situation, only the parallelized algorithm can finish this complex computation in a reasonable time. Besides, along with the dimension increase, the sampled nodes number also increases which calls for more computation ability.

To reduce the complexity of high dimension computation, we revise and extend the default 2-dimension algorithm. In particular, we equally partition all dimensions except the last one, and we change the last dimension d times sequentially from the d dimensions. Finally, the Alg. 4 should be revised to Alg. 6:

Algorithm 6 MultiVarsDriverMain(D)

```
Require:
    Input data file $f$ from HDFS
 1: $d(z, x_1, x_{2,...,}x_d) \leftarrow \mathrm{fs.load}(f)$
 2: for $k < d$
 3:   $p(z, x_k, x_{1,...,}x_d) \leftarrow d(z, x_i), 1 < i < n$
 4: end for
 5: forall possible $k, 1 \leqslant k \leqslant d$ do
 6:   for $z \in \{2, \ldots, \lfloor B/2^d \rfloor\}$ do
 7:     for $x_1 \in \{2, \ldots, \lfloor B/z \rfloor\}$ do
 8:       for $x_2 \in \{2, \ldots, \lfloor B/(x_1 * z) \rfloor\}$ do
 9:       ... ...
10:       $x_k = \lfloor B/(z * x_{1,...,}x_d) \rfloor$
11:       $(I_{x_1,...,x_k:2}, \ldots, I_{x_1,...,x_k:z})^w$
            $\leftarrow$ MapReduceMaxMIOptK($p, x^d, z, w$)
12:       end for
13:     ... ...
14: end for
15: for $(x_1, x_{2,...,}x_d, z)$ such that $x_1 * \ldots * x_k * z \leqslant B$ do
16:   $I_{z,x_1,x_2,...,x_d} \leftarrow \max\{I_{x_1,...,x_k:z}{}^w\}$
17: end for
18: return $\{I_{z,x_1,x_2,...,x_d} : x_1 * ... * x_k * z \leqslant B\}$
```

Alg. 6 illustrates how we modify our two variable versions into a multi-variable version. For an object z, a set of factors x_1; x_2,……,x_d, $xi \in R$, and all factors belong to R^d. We first choose a dimension: x_k to be partitioned optimally, while others are partitioned equally. From step 6, every extra dimension will result in one *for* loop. When it comes to x_k in step 10, we determine a partition scenario: $I_{z,x1,x2,\ldots,xd}$, and call MapReduce job to calculate the maximum mutual information.

5. EXPERIMENTS AND ANALYSIS

In this section, we will describe the experiments on our MapReduce solution, including the correctness, execution time, and speedup evaluation for two variables. Moreover, the accuracy and efficiency of the multi-variable algorithm is also presented. Due to that the complexity of original algorithm increases notably fast when data set grows, it would be impossible to compute a large enough data set in one server. So in our speed experiments, we mainly measure the speedup in line with the increase of the cluster size to show the speed advantage of our parallel algorithm.

5.1 Hardware and Software Configuration

We set up a small cluster of 7 nodes, each of which runs Fedora 17 on Xeon dual-core 2.53 GHz CPU and 6GB memory. All machines are connected with a single gigabit Ethernet link. All nodes are in the same hosting facility and therefore the round-trip time between any pair of machines was less than a millisecond. In our experiment, we use Hadoop 1.0.3 as basic storage and processing framework, which supports HDFS federation and new generation of MapReduce framework YARN [35]. HDFS uses multiple independent Namenodes/Namespaces to scale the name service horizontally. YARN aka MRv2 divides the two major functions of the Job-Tracker and leverages the system performance. Besides using HDFS as the main storage layer, we use Memcached [9] to help multiple MapReduce jobs with intermediate data sharing.

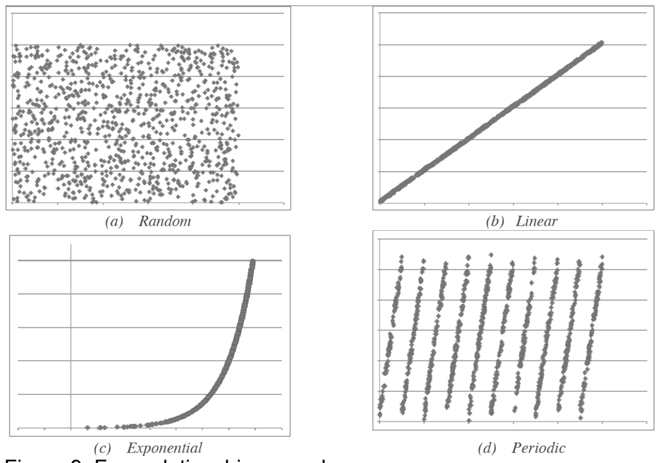


Figure 9. Four relationships sample

5.2 Correctness Evaluation

In the correctness experiment, we compare MIC scores computed by our MapReduce version with the serial version for some relationships and some non-functional relationships. Fig.9 shows four statistics we used, which are Random, Linear, Exponential, and Periodic relationship types.

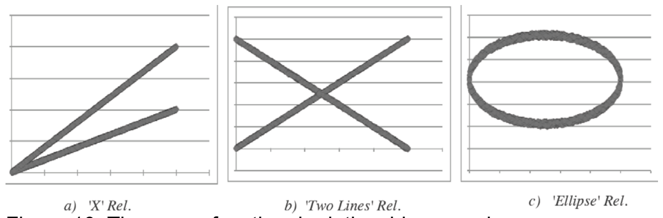


Figure 10. Three non-functional relationships sample

Fig. 10 illustrates three non-functional associations we used include ’X’, ’Two Lines’, and ’Eclipse’. Our MapRe-

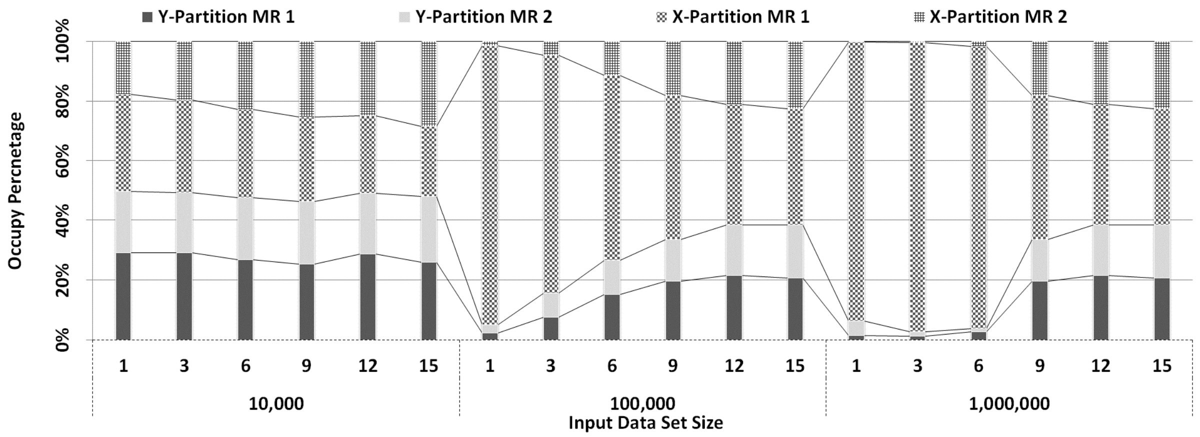


Figure 11. Time Occupation of 4 MR Jobs

duce implementation has a fixed *k* x-axis partitions and this may influence the correctness. So we choose different *k* values for these different kinds of relationships and check how *k* affects the correctness by comparing with the original algorithm.

Table II reports that the fixed *k* x-partition does not affect the correctness of the MIC calculation. In most situations, the differences of MIC score between MapReduce algorithm and the serial algorithm for these functional and nonfunctional relationships are insignificant enough to be treated as a random deviation. In fact, this correctness can be proven through our MapReduce algorithm design and implementation. If there are large enough sample pairs comparing with the server number *k*, the results are unlikely to be different from the original version.

TABLE II
MIC SCORE COMPARISON FOR 7 RELATIONSHIPS, SERVER NUMBER FROM 1 TO 12 (SERVER NUMBER EQUALS TO K)

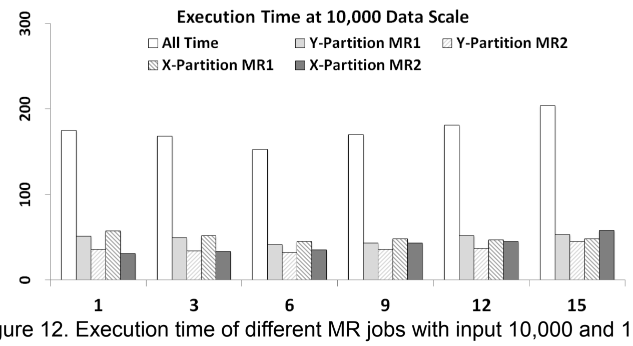
Relationship Type	Serial Alg.	N_4	N_8	N_{10}	N_{12}
Random	0.01	0.08	0.09	0.08	0.08
Linear	0.91	0.96	1.03	1.04	1.03
Exponential	0.96	0.92	0.96	0.92	0.94
Periodic	0.98	0.96	0.92	0.95	0.93
'X'	0.68	0.65	0.66	0.60	0.68
'Two Lines'	0.68	0.66	0.64	0.67	0.66
'Eclipse'	0.69	0.65	0.63	0.65	0.63

5.3 Execution Time and Speedup Evaluation

Hadoop's MapReduce framework will distribute the Map and Reduce tasks into multiple servers automatically according to the input data set. If a Hadoop cluster stores files into 64MB blocks and the input file is 2GB, then the file (in our condition is logs.data) will be divided into 27 blocks and each block will be stored in three servers (the default replication factor of HDFS). This input file (logs.data) will make Hadoop's MapReduce framework start 27 mapper tasks, with the limitation of the system parameter mapred.tasktracker.map.tasks.maximum. However, in most cases, we need to set the reduce tasks number manually to provide more computation power to our MapReduce jobs. In our speedup experiments, we manually set the Reduce tasks number and leave the map tasks number set by Hadoop.

Besides building Hadoop as the storage layer and the MapReduce environment, we also introduce a Memcached instance to accelerate the reading of intermediate results and global variables. As illustrated in Fig. 11, the MapReduce algorithm includes at least two stages: Y-Partition stage and X-Partition stage. The y-axis partition generated by Y-Partition stage will be used in X-Partition stage. It is clear that the y-axis partition would not be large because it only includes *ys* (the y-axis partition number, belongs to *2,……, B/2*) elements. As a result, to accelerate the processing speed, we store this y-axis partition results into the memory cache (Memcached) in every server, then each X-Partition stage task can access these data locally.

In our implementation, both Y-Partition and XPartition



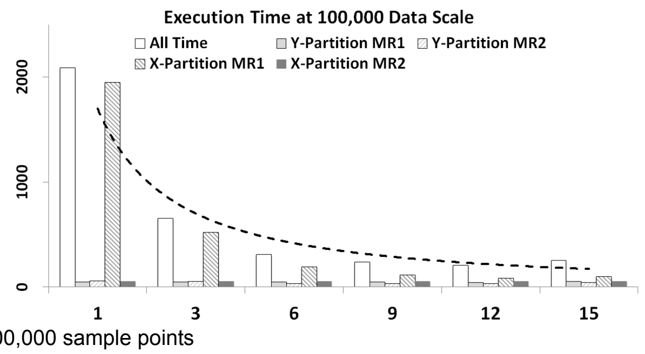


Figure 12. Execution time of different MR jobs with input 10,000 and 100,000 sample points

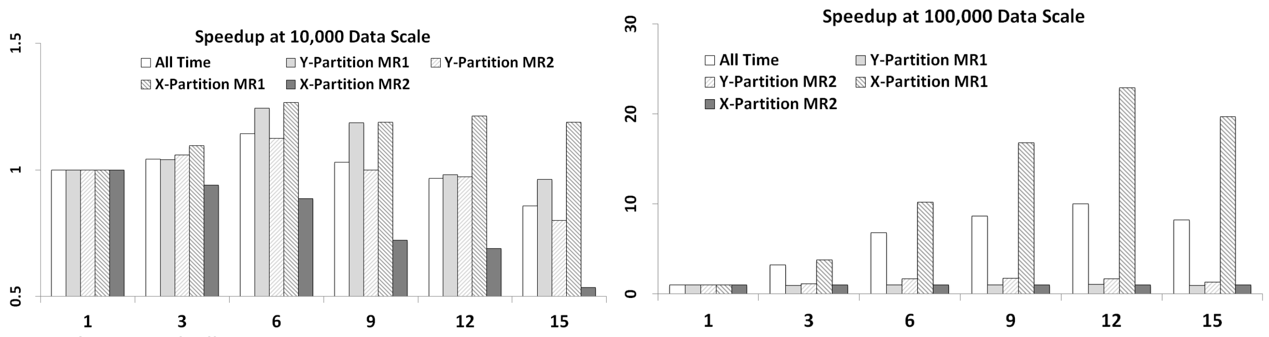


Figure 13. Speedup of different MR jobs with input 10,000 and 100,000 sample points

stages include two MapReduce jobs, therefore the total computation time is the summation of the four MapReduce jobs. We name these four MapReduce jobs as Y-Partition MR1, Y-Partition MR2, X-Partition MR1, and X-Partition MR2. Fig. 11 illustrates the percentage of these four MapReduce jobs occupied in the overall execution time for different input dataset sizes and number of reducer tasks. It presents that when the input data set increases, an increasing percent of execution time is consumed by the X-Partition MR1 job, this is due to that we need to explore all the possible x * y grids and calculate their mutual information accordingly. Also when the reducer task number increases (from 1 to 15), the percentage of the most time-consuming job (X-Partition MR1) is descendent. This is because our parallel solution decreases the execution time of X-Partition MR1 more obviously than other three MR jobs. In general, the X-Partition MR1 is the longest MR job and our solution can reduce its execution time with good scalability by adding more servers.

Fig.12 illustrates the execution time of the overall MapReduce computation procedure with different servers and different input dataset size. The x-axis of these two figures denotes the number of Reduce tasks we assigned, and the y-axis refers to the execution time in second. It is interesting that when there are only 10,000 sample points (Fig.12) the execution time does not reduce while the server number is increasing. The reason is that MapReduce jobs need lots of network communications and disk I/O, so if the data size is small, the communication delay and I/O cost will be more obvious. This situation will disappear when input data size increases, the execution time decreases dramatically when adding more servers. For example, it needs 2087 seconds to finish the computation in one server but it only cost nearly 653 seconds (31.3%) to finish the same computation using three servers. Our MapReduce solution can achieve linear speedup according to our experiments.

Furthermore, the speedup of the algorithm is illustrated in Fig. 13. We derived the speedup from the execution time with both 10,000 data scale and 100,000 data scale. For 10,000 data scale, the speedup is not scalable with the number of MapReduce jobs. The speedup of the total execution increases to 1.14x only when the number of reduce tasks increases to 6, but then decreases to 0.85x when the reduce tasks increase to 15. This is due to the communication overheads and scheduling overheads. By contrast, the scalability at the 100,000 data scale is much more satisfying. In particular, the speedup of the total execution increases to 9.99x when the the number of reduce tasks increase to 12, and then remains flat afterwards.

5.4 Accuracy of the Multi-Variable Algorithm

In order to evaluate the correctness of the new designed algorithm, we use six statistics, which are Exponential, Periodic, X, Two Lines and Eclipse relationship types.

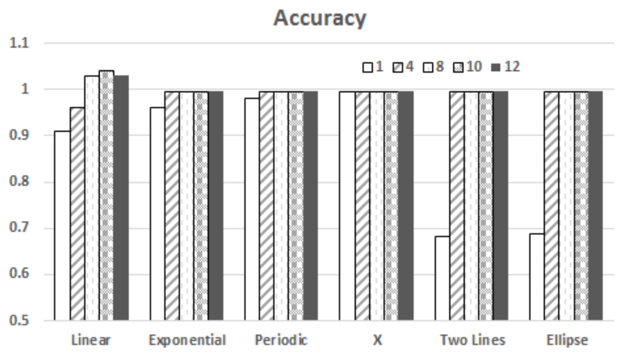


Figure 14. Accuracy of the of the Multi-Variable Algorithm

Experimental results of accuracy is illustrated in Figure 14, which is quite similar to the results in Table II. X-axis denotes the different number of reducer tasks, while y-axis refers to the accuracy. It depicts that the all the relationship types are very close to 100%. For Exponential statistics, the accuracy increases from 96.0% to 99.6%. For Two Lines statistics, the accuracy increases from 68.2% to 99.6%, and for Eclipse statistics, the accuracy increases from 68.8% to 99.6%. Experimental results demonstrate that the accuracy of the multi-variable algorithm can achieve high accuracy when the number of reducer tasks is larger than 4.

5.5 Efficiency of the Multi-Variable Algorithm

We evaluated the execution time and the speedup for the multi-variable algorithm. The experimental results are presented in Figure 15.

The left part of Figure 15 stands for the execution time, and it illustrates that the execution time of the algorithm decreases significantly with the increase of the reducer tasks. Of the four calculations, the X-Partition MR1 execu-

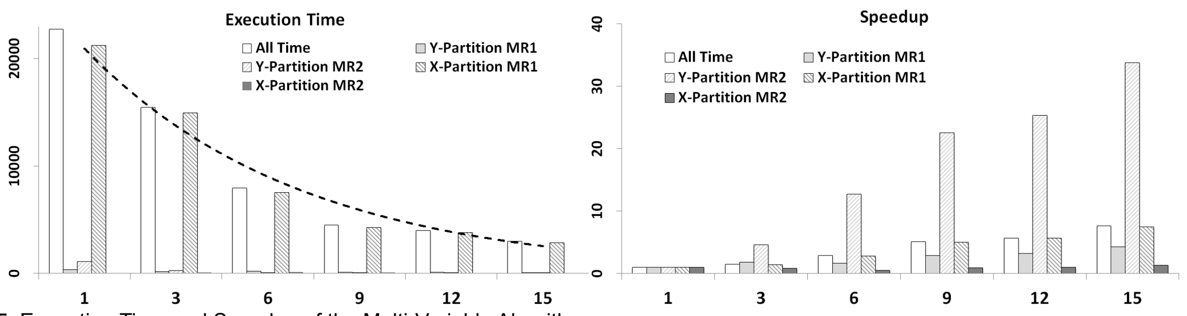


Figure 15. Execution Time and Speedup of the Multi-Variable Algorithm

tion time is dominant. Based on the execution time, the speedup chart is derived to show the scalability of the algorithm, as illustrated in the right part of Figure 15. The speedup reaches 7.58x when the reducer tasks grows to 15. Results show that the algorithm is able to achieve a scalable speedup with satisfying scalability.

6 CONCLUSION AND FUTURE WORKS

In this paper, we have proposed an acceleration technique for large scale biological datasets using MapReduce framework. The framework is based on the analysis of the classic MIC computation for detecting associations between two variables. The task sets are partitioned into Map and Reduce jobs which can run in parallel on the HDFS storage system and memory cache to speedup the computation of MIC. Furthermore, in order to explore the relationship among multiple factors, we extended the conventional two-variable algorithm to a more general multi-variable solution. Experimental results demonstrate that the MapReduce accelerator can achieve a significant speedup with satisfying scalability and flexibility.

Although the results are promising there are numerous directions worth pursuing. As future work, the algorithm for multi-variable situation will be investigated with more state-of-the-art applications. Meanwhile, we will apply the algorithm to cutting edge applications in different circumstances.

ACKNOWLEDGMENT

This work was supported by the National Science Foundation of China under grants (No. 61379040, No. 61272131, No. 61202053, No. 61222204, No. 61221062), Jiangsu Provincial Natural Science Foundation (No. SBK201240198), Fundamental Research Funds for the Central Universities No. WK0110000034, and the Strategic Priority Research Program of CAS (No. XDA06010403). The authors deeply appreciate many reviewers for their insightful comments and suggestions.

REFERENCES

[1]. M, S., et al., *Quantitative monitoring of gene expression patterns with a complementary DNA microarray.* Science, 1995. **270**: p. 467-70.

[2]. Doniger, S.W., *MAPPFinder: using Gene Ontology and GenMAPP to create a global gene-expression profile from microarray data.* Genome Biology, 2003. **4**(R7).

[3]. Howe, D., et al., *Big data: The future of biocuration.* Nature, 2008. **455**(7209): p. 47-50.

[4]. Qi, X., et al., *A Novel Model for DNA Sequence Similarity Analysis Based on Graph Theory.* Evolutionary Bioinformatics, 2011. **7**: p. 149-158.

[5]. Olson, C.B., et al. *Hardware Acceleration of Short Read Mapping*. in *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*. 2012.

[6]. Reshef, D.N., et al., *Detecting Novel Associations in Large Data Sets.* Science, 2011. **334**: p. 1518-1524.

[7]. Jeffrey, D. and G. Sanjay, *MapReduce: simplified data processing on large clusters.* Communications of the ACM 2008. **51**(1): p. 107-113.

[8]. Apache. *Hadoop*. 2014; Available from: ttp://hadoop.apache.org/.

[9]. Memcached. *Memcached:Free & open source, high-performance, distributed memory object caching system*. 2014; Available from: http://memcached.org/.

[10]. Dai, D., et al. *Detecting Associations in Large Dataset on MapReduce*. in *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. 2013.

[11]. TV, K., P. BH, and U. EC., *Analyzing large biological datasets with association networks.* Nucleic Acids Res, 2012. **40**(17): p. e131.

[12]. Dai, D., et al. *Sedna: A Memory Based Key-Value Storage System for Realtime Processing in Cloud*. in *In Cluster Computing Workshops*. 2012.

[13]. Trapnell, C. and M.C. Schatz, *Optimizing data intensive GPGPU computations for DNA sequence alignment.* Parallel Computing 2009. **35**(8-9): p. 429-440.

[14]. Wang, C., et al. *Big data genome sequencing on Zynq based clusters* in *2014 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 2014.

[15]. Chen, P., et al., *Accelerating the Next Generation long read mapping with the FPGA-based system.* IEEE/ACM Transactions on Computational Biology and Bioinformatics, 2014(Preprint).

[16]. Tang, W., et al. *Accelerating Millions of Short Reads Mapping on a Heterogeneous Architecture with FPGA Accelerator*. in *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*. 2012.

[17]. Chen, Y., B. Schmidt, and D.L. Maskell. *Accelerating short read mapping on an FPGA (abstract only)*. in *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*. 2012. Monterey, California, USA:

ACM.

[18]. Chen, P., et al., *Accelerating the Next Generation long read mapping with the FPGA-based system.* IEEE/ACM Transactions on Computational Biology and Bioinformatics, 2014. **PP**(99): p. 1.

[19]. Wang, C., et al., *Heterogeneous Cloud Framework for Big Data Genome Sequencing.* IEEE/ACM Transactions on Computational Biology and Bioinformatics, 2014. **12**(1): p. 166-178.

[20]. Renyi, A., *On measures of dependence.* Acta Mathematica Hungarica, 1959(3): p. 441-451.

[21]. Hastie, T., R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2009: Springer-Verlag.

[22]. Tang, D., et al., *RapidMic: Rapid Computation of the Maximal Information Coefficient.* Evolutionary Bioinformatics, 2014(10): p. 11-16.

[23]. Zhang, Y., et al., *A Novel Algorithm for the Precise Calculation of the Maximal Information Coefficient.* Scientific Reports, 2014. **4**.

[24]. Lopez-Paz, D., P. Hennig, and B. Schölkopf. *The randomized dependence coefficient*. in *Advances in Neural Information Processing Systems (NIPS)*. 2013.

[25]. Kinney, J.B. and G.S. Atwal, *Equitability, mutual information, and the maximal information coefficient.* Proceedings of the National Academy of Sciences, 2014. **111**(9): p. 3354-3359.

[26]. Anonymous, *Finding correlations in big data.* Nature Biotechnology, 2012. **30**(4): p. 334-335.

[27]. Paninski, L., *Estimation of entropy and mutual information.* Neural Computation, 2003. **15**(6): p. 1191-1253.

[28]. Kraskov, A., H. Stögbauer, and P. Grassberger, *Estimating mutual information.* Phys. Rev. E, 2004. **69**: p. 066138

[29]. Pluim, J.P.W., J.B.A. Maintz, and M.A. Viergever, *Mutual-information-based registration of medical images: a survey.* IEEE Transactions on Medical Imaging, 2003. **22**(8): p. 986-1004.

[30]. Reshef, D.N., et al., *Cleaning up the record on the maximal information coefficient and equitability.* Proc. Natl. Acad. Sci, 2014. **111**(33): p. 3362-3363.

[31]. Wang, Z., et al. *Accelerating subsequence similarity search based on dynamic time warping distance with FPGA*. in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*. 2013. Monterey, California, USA: ACM.

[32]. Wang, C., et al. *Phase Detection for Loop-Based Programs on Multicore Architectures*. in *IEEE Cluster*. 2012.

[33]. Gupta, G. and G.S. Sohi. *Dataflow execution of sequential imperative programs on multicore architectures*. in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. 2011. Porto Alegre, Brazil: ACM.

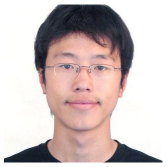
[34]. Kim, Y., et al., *Improving performance of nested loops on reconfigurable array processors.* ACM Transactions on Architecture and Code Optimization (TACO), 2012. **8**(4): p. 1-23.

[35]. *Apache Hadoop NextGen MapReduce (YARN)*. 2014; Available from: http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html.

BIOGRAPHY

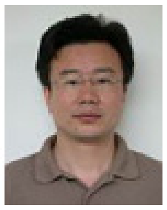
Chao Wang (M'11) received B.S. and Ph.D degree from University of Science and Technology of China, in 2006 and 2011 respectively, both in of computer science. He is an associate researcher with Embedded System Lab in Suzhou Institute of University of Science and Technology of China, Suzhou, China. His research interests focus on Multicore and reconfigurable computing. He has authored more than 60 publications and patents, including IEEE TC, ACM TCBB, TACO and FPGA conferences. He is now the editor board member of MICPRO, IET CDT, serves as Publicity Chair of HiPEAC 2015 and ISPA 2014, guest editor for TCBB and IJPP. He is a member of the IEEE, ACM and senior member of CCF.

Xi Li is a Professor and vice dean in the School of Software Engineering, University of Science and Technology of China. There he directs the research programs in Embedded System Lab, examining various aspects of embedded system with the focus on performance, availability, flexibility and energy efficiency. He has lead several national key projects of CHINA, several national 863 projects and NSFC projects. Prof. Li is a member of ACM and IEEE, a senior member of CCF.



Dong Dai is a research scientist in School of Computer Science, Texas Tech University. He received B.S. and Ph.D degree from University of Science and Technology of China, in 2006 and 2013 respectively, both in of computer science. His research interests include cloud computing and distributed computing.

Aili Wang is with School of Software Engineering, University of Science and Technology of China. She serves as the guest editor of Applied Soft Computing, and International Journal of Parallel Programming. Meanwhile she is a reviewer for International Journal of Electronics. She has published over 10 International journal and conference articles in the areas of software engineering, operating systems, and distributed computing systems.



Xuehai Zhou is the executive dean of School of Software Engineering, University of Science and Technology of China, and Professor in the School of Computer Science. He serves as general secretary of steering committee of computer College fundamental Lessons, and technical committee of Open Systems, China Computer Federation. He has lead many national 863 projects and NSFC projects. Prof. Zhou has published over 100 International journal and conference articles in the areas of software engineering, operating systems, and distributed computing systems. Prof. Zhou is a member of ACM and IEEE, a senior member of CCF.