Order-Independent Constraint-Based Causal Structure Learning for Gaussian Distribution Models using GPUs

Christopher Schmidt
Hasso Plattner Institute
Potsdam, Germany
christopher.schmidt@hpi.de

Johannes Huegle
Hasso Plattner Institute
Potsdam, Germany
johannes.huegle@hpi.de

Matthias Uflacker
Hasso Plattner Institute
Potsdam, Germany
matthias.uflacker@hpi.de

ABSTRACT

Learning the causal structures in high-dimensional datasets allows deriving advanced insights from observational data, thus creating the potential for new applications. One crucial limitation of state-of-the-art methods for learning causal relationships, such as the PC algorithm, is their long execution time. While, in the worst case, the execution time is exponential to the dimension of a given dataset, it is polynomial if the underlying causal structures are sparse. To address the long execution time, parallelized extensions of the algorithm have been developed addressing the Central Processing Unit (CPU) as the primary execution device. While modern multicore CPUs expose a decent level of parallelism, coprocessors, such as Graphics Processing Units (GPUs), are specifically designed to process thousands of data points in parallel, providing superior parallel processing capabilities compared to CPUs.
In our work, we leverage the parallel processing power of GPUs to address the drawback of the long execution time of the PC algorithm and develop an efficient GPU-accelerated implementation for Gaussian distribution models. Based on an experimental evaluation of various high-dimensional real-world gene expression datasets, we show that our GPU-accelerated implementation outperforms existing CPU-based versions, by factors up to 700.

CCS CONCEPTS

• **Mathematics of computing → Causal networks**; • **Computing methodologies → Machine learning approaches**; **Learning in probabilistic graphical models**; **Parallel algorithms**;

KEYWORDS

GPU-Acceleration, Parallelization, PC algorithm, Causal Inference, Conditional Independence Testing, Gaussian Distribution Model, Probabilistic Statistics

ACM Reference Format:
Christopher Schmidt, Johannes Huegle, and Matthias Uflacker. 2018. Order-Independent Constraint-Based Causal Structure Learning for Gaussian Distribution Models using GPUs. In *SSDBM '18: 30th International Conference on Scientific and Statistical Database Management, July 9–11, 2018, Bozen-Bolzano, Italy.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3221269.3221292

1 INTRODUCTION

Deriving knowledge from observational data is an active field of research in statistics and data mining. Understanding the relationships between observed variables in complex systems enables new insights and is of particular interest in the context of high-dimensional settings such as in personalized medicine or Internet of Things (IoT). For example, in genetic research the construction of gene regulatory networks inferred from gene expression data is of major importance. As gene regulatory networks can be seen as a practical embodiment of systems biology, they allow for solving a number of different biological and biomedical problems, e.g., for drug design or diagnostics [42]. In the context of IoT, the application of sensors results in an increased size and complexity of data sets collected from manufacturing processes. The derivation of causal structures enables the root cause analysis of errors in manufacturing processes [30].

Due to the work of Judea Pearl [38] and Spirtes et al. [47], the notion of causality has grown from a nebulous concept into a mathematical theory based on the probabilistic graphical modeling. Constraint-based algorithms for learning the underlying causal structure, e.g., the PC algorithm [48], use conditional independence (CI) tests on observational data to derive an undirected skeleton of the causal relationships. Building on this skeleton, the algorithms determine the orientation of the detected relationships in order to construct a causal graphical model. In this graphical model directed edges represent the causal relationships between the observed variables, depicted as vertices.

The distribution of the observed variables directly determines the selection of the appropriate CI tests [12]. In the context of IoT and genetic research the multivariate normal distribution is an often considered distribution model of the observed variables, e.g., see [25, 30, 42]. The corresponding CI tests are based on the partial correlations of the involved variables. In the worst case, the computational complexity of the algorithm is exponential with regard to the number of variables, which hinders its application in practice. However, even the polynomial complexity for sparse graphs introduces performance issues for the application in the context of high-dimensional datasets [20].

To address the long execution time, we investigate the use of GPUs for constraint-based causal structure learning in the context of Gaussian distribution models. GPUs have been proven to be suitable execution devices for computationally intensive machine learning algorithms, e.g., deep learning [18] or enterprise simulations [45]. Current devices, such as the NVIDIA Tesla V100 GPU, reach a peak performance of up to 14 TFLOPS, are equipped with 16 GB of on-chip High Bandwidth Memory (HBM) and outperform CPUs for certain machine learning tasks [36]. Achieving the peak performance of the device requires the adaption of the algorithm to match the Single Instruction Multiple Threads (SIMT) [24] execution model of a GPU. Our goal is to harness the parallel processing capabilities of the GPU to address the PC algorithm's long execution time for high-dimensional datasets. In particular, we target the skeleton discovery of the constraint-based causal structure learning algorithms. Our work builds upon the order-independent version of the PC algorithm that was introduced by Colombo et al. [9], the so-called PC-stable algorithm. On the basis of an underlying Gaussian distribution model we develop a GPU-accelerated implementation to process the first two levels of the constraint-based discovery algorithm.

Our contributions are the following:

- We examine real-world gene expression datasets with regard to the number of tests per level of the skeleton discovery within the PC-stable algorithm to determine performance critical parts.
- We develop a GPU-accelerated version for level 0 and level 1 of the skeleton discovery.
- We compare our GPU-accelerated implementation with the standard PC-stable algorithm's implementation executed on a CPU to demonstrate the performance benefits of a GPU.

The remainder of the paper is organized as follows. In Section 2 we provide preliminary information of causal graphical models and the Gaussian distribution model. This allows to introduce algorithms for constraint-based causal structure learning in Section 3. We investigate real-world gene expression datasets to show challenges in the application of the PC-stable algorithm in Section 4. Using these findings, we present our GPU-accelerated version of the skeleton discovery based on the Gaussian distribution model in Section 5 and provide an experimental evaluation in Section 6. We discuss related work in Section 7, summarize our work, and give an outlook in Section 8.

2 PRELIMINARIES

In this section, we introduce some necessary terminology in the context of causal graphical models and background information about the Gaussian distribution assumption.

2.1 Causal Graphical Model

Let $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ be a graph consisting of a finite set of N vertices $\mathbf{V} = (V_1, \ldots, V_N)$, each representing the observed variables V_i, $i = 1, \ldots, N$, and a set of edges $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$. An edge $(V_i, V_j) \in \mathbf{E}$ is called directed, i.e., $V_i \rightarrow V_j$, if $(V_i, V_j) \in \mathbf{E}$ but $(V_j, V_i) \notin \mathbf{E}$. If both $(V_i, V_j) \in \mathbf{E}$ and $(V_j, V_i) \in \mathbf{E}$ the edge is called undirected, i.e., $V_i - V_j$.

In this context, two vertices V_i and V_j, $V_i, V_j \in \mathbf{E}$, are called adjacent if there is an undirected edge $V_i - V_j$. The adjacency set $adj(\mathcal{G}, V_i)$ of the vertex $V_i \in \mathbf{V}$ in $\mathcal{G}$ are all vertices $V_j \in \mathbf{V}$ that are directly connected to V_i by an edge.

A graph $\mathcal{G}$ where all edges $\mathbf{E}$ are directed and $\mathcal{G}$ does not contain any cycle is a Directed Acyclic Graph (DAG). In the framework of causal inference, a directed edge $V_i \rightarrow V_j$ of a DAG represents a direct causal relationship of V_i to V_j [38, 48]. Moreover, a DAG entails information about the conditional independencies of the vertices via the d-separation criterion [38]. In this context, two variables $V_i, V_j \in \mathbf{V}$ are conditionally independent given a set $\mathbf{S} \subset \mathbf{V} \setminus \{V_i, V_j\}$ if and only if the vertices V_i and V_i are d-separated by the set $\mathbf{S}$. A distribution P of the variable set $V_1, \ldots, V_N$ that satisfies the above condition is called faithful. It is well known, that several DAGs can describe exactly the same CI information and form a Markov equivalent class [3]. Two Markov equivalent DAGs have the same skeleton $\mathcal{C}$, i.e., the undirected version of the DAGs, and the same v-structures. V-structures are triples (V_i, V_j, V_k) with directed edges $V_i \rightarrow V_j$ and $V_k \rightarrow V_j$ for not adjacent vertices V_i and V_k. Moreover, the corresponding Markov equivalent class can be described uniquely by a Complete Partially Directed Acyclic Graph (CPDAG) [8]. A CPDAG is a partially directed acyclic graph where all DAGs in the Markov equivalence class incorporate the same directed edges, and there exist two DAGs that incorporate the two directed versions of every undirected edge $V_i - V_j$ in the Markov equivalence class.

Hence, the focus lies on the estimation of the equivalence class of the DAG $\mathcal{G}$ based on the corresponding probability distribution P. In particular, if P is faithful with respect to $\mathcal{G}$, we have the case that

$$\begin{aligned} &\text{there is an edge } V_i - V_j \text{ in the skeleton of } \mathcal{G} \\ &\quad \Leftrightarrow V_i,\ V_j \text{ are dependent given all } \mathbf{S} \subset \mathbf{V} \setminus \{V_i, V_j\}, \end{aligned} \tag{1}$$

see [48]. Hence, the examination of the CI information of the observed variables $V_1, \ldots, V_N$ allow for the estimation of the skeleton $\mathcal{C}$ of the corresponding DAG $\mathcal{G}$. All constraint-based algorithms for causal structure learning share the estimation of the skeleton by an adjacency search of the involved variables as a common first step, see Section 3.1. The extension of the skeleton to the equivalence class of the DAG $\mathcal{G}$ can be done by the repeated application of deterministic orientation rules on the result of the first part, e.g., see [9, 16, 47].

2.2 Gaussian Distribution Model

In this work, we limit ourselves to the Gaussian case, where the vertex set $\mathbf{V} = \{V_1, \ldots, V_N\}$ is multivariate normal distributed. Note, that non-faithful multivariate normal distributions form a Lebesgue null set in the space of distributions associated with the underlying DAG $\mathcal{G}$ [32] such that we can assume faithfulness of our Gaussian distribution model. Moreover, from standard statistical theory, e.g. see [2], it follows that two variables V_i and V_j are independent if and only if the correlation coefficient $\rho(V_i, V_j)$ is equal to zero. Moreover, two variables V_i and V_j are conditionally independent given the set $\mathbf{S} \subset \mathbf{V} \setminus \{V_i, V_j\}$ if and only if the partial correlation $\rho(V_i, V_j | \mathbf{S})$ between V_i and V_j given $\mathbf{S}$ is equal to zero [19, 51].

Thus, we estimate correlations and partial correlations of the involved variables to obtain information about the independencies. Given n observations the sample correlation coefficient $\hat{\rho}(V_i, V_j)$ of the variables $V_i, V_j \in \mathbf{V}$ is given by

$$\hat{\rho}(V_i, V_j) = \frac{\sum_{s=1}^{n}\left(V_i^{(s)} - \overline{V_i}\right)\left(V_j^{(s)} - \overline{V_j}\right)}{\sqrt{\sum_{s=1}^{n}\left(V_i^{(s)} - \overline{V_i}\right)^2}\sqrt{\sum_{s=1}^{n}\left(V_j^{(s)} - \overline{V_j}\right)^2}} \tag{2}$$

with the arithmetic means $\overline{V_i}$, and $\overline{V_j}$ of the corresponding variables V_i, and V_j, i.e.,

$$\overline{V_i} = \sum_{s=1}^{n} V_i^{(s)}, \quad \overline{V_j} = \sum_{s=1}^{n} V_j^{(s)},$$

where $V_i^{(s)}$, and $V_j^{(s)}$ denotes the s-th entry of the variable V_i, and V_j, respectively, e.g., see [22]. The estimation of the partial correlation coefficient $\rho(V_i, V_j | \mathrm{S})$ can be efficiently derived via the inverse of the corresponding correlation matrix $\mathrm{Cor}(V_i, V_j, \mathrm{S})$, where $\mathrm{Cor}(V_i, V_j, \mathrm{S})_{k,l}$ is the sample correlation coefficient $\hat{\rho}(V_k, V_l)$ for all $V_k, V_l \in \mathrm{S} \cup \{V_i, V_j\}$ [51]. Given the corresponding correlation matrix $\mathrm{Cor}(V_i, V_j, \mathrm{S})$, the sample partial correlation coefficient can then be expressed as

$$\hat{\rho}(V_i, V_j | \mathrm{S}) = \frac{-r_{i,j}}{\sqrt{r_{i,i} r_{j,j}}}, \tag{3}$$

where $r_{i,j} = \mathrm{Cor}(V_i, V_j, \mathrm{S})^{-1}_{i,j}$. Note, that when the determinant of the correlation matrix is zero, we compute the pseudo-inverse using the Moore-Penrose generalized matrix inverse [40].

For testing whether the (partial) correlation is zero or not, we apply standard statistical hypothesis testing theory, e.g., see [23]. With $\hat{\rho}(V_i, V_j | \mathrm{S}) = \hat{\rho}(V_i, V_j)$ for $\mathrm{S} = \emptyset$, i.e., the correlation case, we apply Fisher's z-transform on the sample (partial) correlation coefficient

$$Z(V_i, V_j | \mathrm{S}) = \frac{1}{2} \log \left(\frac{1 + \hat{\rho}(V_i, V_j | \mathrm{S})}{1 - \hat{\rho}(V_i, V_j | \mathrm{S})} \right). \tag{4}$$

On this basis, we calculate the corresponding p-value

$$\mathrm{p}(V_i, V_j | \mathrm{S}) = 2\left(1 - \Phi\left(\sqrt{n - |\mathrm{S}| - 3}\left|Z(V_i, V_j | \mathrm{S})\right|\right)\right), \tag{5}$$

where $\Phi(\cdot)$ denotes the cumulative distribution function of a standard normal distribution. Hence, given the significance level α, we reject the null-hypothesis $\hat{\rho}(V_i, V_j | \mathrm{S}) = 0$ against the two sided alternative $\hat{\rho}(V_i, V_j | \mathrm{S}) \neq 0$ if for the corresponding p-value it holds that $\mathrm{p}(V_i, V_j | \mathrm{S}) \leq \alpha$. Since we need to perform many CI tests, α should not be interpreted as an overall significance level but as a tuning parameter, where smaller values of α tend to derive sparser graphs. Beside the possibility of optimizing a Bayesian Information Criterion [26], and stability selection [33] for choosing α given the true DAG, we are not aware of a fully satisfactory method in real-world applications. Oftentimes, $\alpha = 0.01$ is used as tuning parameter for application, see e.g. [9].

3 CAUSAL STRUCTURE LEARNING

In this section, we introduce the concept of constraint-based causal structure learning for estimating the CPDAG.

3.1 Constraint-Based Methods

As described in Section 2, one can use conditional independence tests to obtain the skeleton C of the CPDAG. Following (1), a naive strategy for learning the skeleton C would be to check the conditional independencies of all vertices $V_i, V_j \in \mathbf{V}$ given all possible subsets $\mathrm{S} \subseteq \mathbf{V} \setminus \{V_i, V_j\}$ [39]. Obviously, this would become computationally infeasible for a larger number of variables $V_1, \ldots, V_N$.

A much better approach is the PC algorithm by Spirtes et al. [48] that starts from a complete, undirected graph, and is estimating the skeleton C by recursively deleting edges based on the conditional independencies given increasing separation sets based on the adjacency structure. If the true DAG $\mathcal{G}$ is sparse, which is often a reasonable assumption, this method decreases the exponential complexity to a polynomial with respect to the number of vertices [16]. The PC algorithm was designed under the assumption of causal sufficiency, that is that there are no unmeasured common causes and no selection variables In order to allow for latent and selection variables, the FCI and RFCI algorithms were introduced [10, 48]. These algorithms extend the adjacency search of the PC algorithm through subsequently applied conditional independence test because of the latent variables. In order to allow for directed cyclic graphs under the assumption of causal sufficiency, another extension of the original adjacency search of the PC algorithm is the so called CCD-algorithm [43]. Since all these extensions share the adjacency search of the PC algorithm as a common first step, our improvements can be carried out directly.

The original version of the PC algorithm has shown to be depending on the order of the variable set $V_1, \ldots, V_N$. Therefore, we build our work upon the order-independent version of the PC algorithm that was introduced by Colombo et al. [9], the so-called PC-stable algorithm.

3.2 The PC-stable Algorithm

A sketch of the adjacency search, i.e. the first step of the PC-stable algorithm, is given in Algorithm 1. Starting with a complete undirected skeleton C the PC-stable algorithm uses CI-tests given an increasing separation set S of adjacent vertices in order to subsequently thin out the skeleton C. Hence, we only need to query CI tests of vertices V_i and V_j given separation sets S with size $l = 0$ up to the maximum size of the adjacency sets of the vertices in the underlying DAG $\mathcal{G}$, i.e., up to $\max_{V_i \in \mathbf{V}} |adj(\mathcal{G}, V_i) \setminus \{V_j\}|$ (see lines 8-16 in Algorithm 1). This makes the algorithm computationally feasible even for a large number of variables, and allows for its application even in high-dimensional settings [16].

For every level $l = 0, \ldots, \max_{V_i \in \mathbf{V}} |adj(\mathcal{G}, V_i) \setminus \{V_j\}|$ we compute and store the adjacency sets $a(V_i) = adj(C, V_i)$ of variables V_i with respect to the current skeleton C (see lines 4-6). Hence, at each level l the algorithm records the edges that need to be removed, but deletes these edges only when entering the next level $l + 1$. Note that this allows for an order-independent implementation of the original PC algorithm as proven by Colombo et al. [9].

First, for $l = 0$ all pairs of vertices $V_i, V_j \in \mathbf{V}$ are tested for marginal independence given an empty separation set $\mathrm{S} = \emptyset$. Therefore, given an overall tuning parameter α we apply the independence test derived in Subsection 2.2 and examine whether $\mathrm{p}(V_i, V_j | \emptyset) = \mathrm{p}(V_i, V_j | \mathrm{S}) \leq \alpha$, or not. If the two variables V_i, V_j are independent,

Algorithm 1 Adjacency search of PC-stable algorithm [9]
Input: Vertex set V, correlation matrix Cor
Output: Estimated skeleton C, separation sets **Sepset**

```
1:  Start with fully connected skeleton $C$ and $l = -1$
2:  repeat
3:      $l = l + 1$
4:      for all Variables $V_i$ in $C$ do
5:          Let $a(V_i) = adj(C, V_i)$;
6:      end for
7:      repeat
8:          Select pairs $(V_i, V_j)$ adjacent in $C$ with $|a(V_i) \setminus \{V_j\}| \geq l$
9:          repeat
10:             Choose separation set $S \subseteq a(V_i) \setminus \{V_j\}$ with $|S| = l$.
11:             if $p(V_i, V_j | S) \leq \alpha$ then
12:                 Delete edge $V_i - V_j$ from $C$;
13:                 Set $\mathbf{Sepset}(i, j) = \mathbf{Sepset}(j, i) = S$;
14:             end if
15:         until edge $V_i - V_j$ is deleted in $C$
16:         or all $S \subseteq a(V_i) \setminus \{V_j\}$ with $|S| = l$ have been chosen
17:     until all adjacent vertices $V_i$, and $V_j$ in $C$ such that
18:     $|a(V_i) \setminus \{V_j\}| \geq l$ have been considered
19: until each adjacent pair $V_i, V_j$ in $C$ satisfy $|a(V_i) \setminus \{V_j\}| \geq l$
20: return $C$, Sepset
```

the edge $V_i - V_j$ is deleted from C and the empty set $\emptyset$ is saved as separation set in $\mathbf{Sepset}(V_i, V_j)$ and $\mathbf{Sepset}(V_j, V_i)$ (see lines 11-13). After all pairs of vertices have been considered, the algorithm proceeds to the next step with $l = 1$.

When $l = 1$, the algorithm applies the CI-tests for vertices V_i and V_j that are still adjacent in the skeleton C. Therefore, we examine whether $p(V_i, V_j | S) \leq \alpha$ given the separation set $S \subset adj(C, V_i) \setminus V_j$ for subsets of size $l = 1$. Hence, if the variables V_i and V_j are found to be conditionally independent given the corresponding separation set S, the edge $V_i - V_j$ is removed, and S is saved in $\mathbf{Sepset}(V_I, V_j)$ and $\mathbf{Sepset}(V_j, V_i)$. If all pairs of adjacent vertices V_i and V_j of the current version of the skeleton C are considered, the algorithm again increases l by one.

This process continues until l reaches the maximum size of the adjacency sets of the vertices in the underlying DAG $\mathcal{G}$. At this point, the resulting skeleton C is then used as basis for the application of deterministic orientation rules in order to extend C to the corresponding CPDAG, e.g., see [9, 16, 47]

Note that Kalisch et al. [16] have proven the uniform consistency of the PC algorithm in our Gaussian distribution model. This implies that the algorithm consistently estimates the equivalence class of the underlying DAG $\mathcal{G}$ as the sample size increases.

4 RUNTIME ANALYSIS OF THE PC ALGORITHM

One limiting factor to the application of the PC algorithm in a real-world scenario is its computational complexity, which is, in the worst case, exponential to the number of variables $V_1, \ldots, V_N$ within the data, see Section 3. Hence, the number of CI tests conducted during the skeleton discovery has the strongest impact on the execution performance. While most CI tests come with a certain complexity with respect to both the number of variables and the number of observations, e.g., based on categorical data, CI tests for multivariate normal data benefit from the use of the correlations. In this particular case the observational data is processed prior to the skeleton algorithm and the CI tests itself are based on the pre-computed correlation matrix $\text{Cor}(V_1, \ldots, V_N)$ as described in Section 2. As the CI test for evaluating the conditional independence of the variables $V_i, V_j \in \mathbf{V}$ given a set $S \subseteq \mathbf{V} \setminus \{V_i, V_j\}$ is based on the corresponding partial correlation $\hat{\rho}(V_i, V_j | S)$, it has a computational complexity of $O\left(\left|\{V_i, V_j\} \cup S\right|^3\right)$.

In order to develop a parallel GPU-accelerated implementation of the adjacency search of the PC algorithm (see Algorithm 1), we conducted an investigation of several high-dimensional datasets and counted the number of conducted tests per level l. We base our investigation on gene expression datasets [14, 21, 31], which have been the foundation for the evaluation of the `Parallel-PC` algorithm by Le et al. [20].

Table 1: Characteristics of the gene expression datasets

Dataset	$|\mathbf{V}|$	Observations	$|\mathbf{E}|$
`NCI-60`	1,190	47	530
`MCC`	1,380	88	643
`BR51`	1,592	50	478
`S.aureus`	2,810	160	2,058
`S.cerevisiae`	5,361	63	2,086

Characteristics of the high-dimensional datasets are described in Table 1. They range from 1,190 to 5,361 variables. The number of observations is noted, as it influences the acceptance of a CI test. Hence, it has an impact on the number of detected edges for the skeleton discovery, also shown in Table 1. As described in Section 2, another criterion for the acceptance of a CI test and the corresponding deletion of an edge in the graphical model is given by the tuning parameter α. In our work, we follow Kalisch et al. [16] and chose $\alpha = 0.01$.

Table 2: Percentage of conditional independence (CI) tests in skeleton discovery per level with tuning parameter $\alpha = 0.01$

Dataset	Total number of CI tests	Level 0	Level 1	Levels remaining
`NCI-60`	4,017,475	17.61%	82.16%	0.23%
`MCC`	21,950,296	4.34%	93.00%	2.66%
`BR51`	29,696,242	4.26%	95.61%	0.13%
`S.aureus`	170,430,911	2.32%	95.48%	2.20%
`S.cerevisiae`	69,321,855	20.73%	78.76%	0.51%

In Table 2, we depict the percentage of required CI tests per level l and state the total number of CI tests for each dataset. It becomes visible that most CI tests are conducted within level $l = 1$ and that, for all datasets, the second largest number of CI tests is conducted within level $l = 0$. The number of CI tests within the remaining levels reach at the most 2.66% of all conducted CI tests, e.g., for the MCC dataset. For all datasets the skeleton discovery reaches a maximum depth of level 4. Looking at the dataset with the highest number of dimensions, S.cerevisiae, 14, 367, 480 CI tests are conducted in level 0, 54, 598, 391 CI tests in level 1 and the remaining levels account for 355, 984 CI tests.

Since the computational complexity of the CI test for variables $V_i, V_j \in \mathbf{E}$ given S is $O\left(\left|\{V_i, V_j\} \cup \mathrm{S}\right|^3\right)$, CI tests become more expensive for separation sets in higher levels l. Therefore, taking only the number of conducted CI tests into consideration is not sufficient in order to determine the most time consuming part. Identifying this spot allows us to address the most critical part in order to improve the overall execution time of the PC algorithm.

Hence, we measure the execution time of an existing implementation of the PC algorithm. We choose the implementation provided by the R-package pcalg [17], which runs on a CPU. For the measurements we use the system and setup described in Section 6.

Table 3: Percentage of execution time in skeleton discovery per level with tuning parameter $\alpha = 0.01$

Dataset	Execution time	Level 0	Level 1	Levels remaining
NCI-60	8.06 s	0.27%	98.96%	0.77%
MCC	59.39 s	0.05%	94.57%	5.38%
BR51	67.57 s	0.05%	99.57%	0.38%
S.aureus	1037.23 s	0.01%	98.03%	1.96%
S.cerevisiae	479.59 s	0.09%	99.28%	0.63%

The results of our experimental investigation are shown in Table 3. We depict the percentages of the execution time for level $l = 0$, level $l = 1$ and all remaining levels $l \geq 2$. The impact of the tests within level $l = 0$ on the overall execution time of the skeleton discovery is almost non-existent on all datasets. Looking at the S.cerevisiae dataset it accounts for only 0.09% of the execution time, although 20.73% of all tests were conducted in this level. For the remaining levels $2 \leq l \leq 4$ we see that these also only have a small influence on the overall execution time. Compared to the percentages of CI tests conducted within levels $2 \leq l \leq 4$ an increase in their impact on the overall execution time becomes visible. This is due to the increased cost for the computations within higher levels l of the skeleton discovery. Level $l = 1$ dominates the overall runtime for all investigated gene expression datasets. For our GPU-accelerated version of the skeleton discovery, we will therefore focus on the first two levels with $l = 0, 1$. Extending our implementation to work for the remaining levels $l \geq 2$ is part of our future work.

5 GPU-ACCELERATED LEVEL 0 & 1 SKELETON DISCOVERY KERNELS

For the GPU-accelerated implementation of the first two levels $l = 0, 1$ of the adjacency search in Algorithm 1 we use CUDA as the parallel computing platform and programming model [34]. CUDA allows us to reuse existing mathematical functions required for the CI tests of variables V_i, V_j given S. In particular, for the calculation of the corresponding p-value $\mathrm{p}(V_i, V_j | \mathrm{S})$, as shown in Equation 5, we utilize the log1p() and normcdf() functions, from the CUDA Math API [35]. We implement a separate kernel for Level 0 and Level 1, as we require synchronization between the levels $l = 0$ and $l = 1$. The calculations are based on the corresponding correlation matrix $\mathrm{Cor}(V_i, V_j, \mathrm{S})$, the adjacency set $a(V_i) = adj(C, V_i)$, the number of n, the tuning parameter α, and the number of variables N and produce output containing the current version of the skeleton C, the calculated p-values $\mathrm{p}(V_i, V_j | \mathrm{S})$ and the vector of separation sets **Sepset**, for all V_i with adjacent V_j and $\mathrm{S} \subseteq a(V_i) \setminus \{V_j\}$.

For an efficient CUDA-based implementation GPU hardware specific characteristics need to be considered. The processing of a kernel follows the SIMT [24] execution model. Threads are organized in thread blocks, which are all executed on the same Streaming Multiprocessor (SM). Within a thread block 32 threads are grouped together into a warp executing the same instruction at the same time. In case of conditional branches, different instructions may be processed by threads within one warp, called warp divergence. This leads to inefficiencies, as all threads process all conditional branches. Note that only the correct result for a given thread and its conditional branch is stored.

Besides the specific execution model, the GPU provides several gigabytes of global memory, as well as some kilobytes of shared memory. While global memory is shared between all thread blocks, data within shared memory is only accessible within a thread block. Shared memory is also used for caching, as it provides lower latency and higher bandwidth than global memory. It is important to access global memory of the GPU efficiently. Threads within a warp should access global memory in a coalesced fashion to allow for optimal usage of the available bandwidth. Therefore, for our GPU-accelerated implementation of the two levels $l = 0, 1$ we need to avoid warp divergence, use shared memory and aim for coalesced memory accesses to global memory.

The kernel for Level 0 calculates the p-values $\mathrm{p}(V_i, V_j | \mathrm{S})$ for all variables $V_i, V_j \in \mathbf{V}$, given a separation set S of size 0. Hence, only the correlation between the two variables tested is required, see Section 2.2. To process the calculations for level $l = 0$ in parallel, a separate thread is started on the GPU for each pair of variables V_i, V_j that has to be tested for conditional independence. Internally each thread calculates the corresponding **p**-value using the equation shown in Equation 5.

For level $l = 1$ the p-values $\mathrm{p}(V_i, V_j | \mathrm{S})$ for the variables V_i, V_j are calculated for all adjacent separation sets $\mathrm{S} \subseteq a(V_i) \setminus \{V_j\}$ with cardinality $|\mathrm{S}| = 1$ based on the remaining edges in C.

Depending on the result from level $l = 0$, the number of tests to be calculated within level $l = 1$ varies, as variables have already been tested as conditional independent. For our CUDA-based kernel implementation we considered the following strategies to compute the tests for level $l = 1$ in parallel.

In a first version, the kernel is started with a thread for each test that has to be calculated, not considering any knowledge about removed edges from level $l = 0$. Prior to the calculation of the **p**-value $\mathrm{p}(V_i, V_j | \mathrm{S})$ the thread checks, whether it tests an existing edge or not using the stored adjacency set $a(V_i) = adj(C, V_i)$, and stops in case the calculation is not necessary.

This approach requires to either store multiple candidates for the separation set or to randomly chose an accepted candidate. The first option leads to a larger memory footprint in the GPU's global memory, while the second breaks with any ordering. Additionally, checking if the calculations are necessary on the GPU may lead to branching of the threads executed together within a warp, introducing a performance bottleneck.

A simple strategy to avoid these checks starts a separate kernel for each edge remaining after the `Level 0` kernel was executed. Thus the calculation for each separation set candidate for the edge is executed in parallel. The overhead of starting multiple kernels, potentially one for each variable, outweighs any performance gain.

In our implementation for the `Level 1` kernel, we keep the order of separation set candidates intact, assuming ascending order according to the position within the correlation matrix `cor`. The architecture of the NVIDIA GPU guarantees the execution of a thread block on a single SM and allows to synchronize these threads during the kernel execution. We leverage this feature to retain the order of separation set candidates, by processing all tests for conditional independence of a single edge within the same thread block. Furthermore, we avoid a larger memory footprint for storing multiple candidates for the separation set in the GPU's global memory, by utilizing the shared memory of the thread block.

At the beginning of the `Level 1` kernel shared memory is acquired, storing the calculated p-value $p(V_i, V_j | S)$ of each thread within the thread block. By calling the `__syncthreads()` function, we ensure that all p-values are calculated before a master thread reduces the results from shared memory. During the reduction, the list of calculated p-values and in case of an accepted test the skeleton C and the vector of separation sets **Sepset** are updated. This coincides with the order-independent version of the PC algorithm that was introduced by Colombo et al. [9]. Since, a single thread block is responsible for calculating all tests of level $l = 1$ for a given edge, the kernel is launched with a number of thread blocks equal to the number of edges within the graph.

Both kernels work on the same data structures, hence data is moved once from DRAM to the GPU internal HBM prior to the launch of the kernel for `Level 0`. Results are transferred back into DRAM after the execution of the kernel for `Level 1`. Currently, we utilize two separate data structures for the adjacency set $a(V_i) = adj(C, V_i)$ and the skeleton C. In order to minimize the memory footprint of these data structures, we consider to combine them in the future,e.g., by utilizing an upper triangular matrix for the adjacency set and a lower triangular matrix for the skeleton.

6 EVALUATION

For our evaluation, we use a system with an Intel i7-6700K CPU with 4 cores, hyper-threading and 64 GB of DRAM. The system is also equipped with an NVIDIA Titan X graphics card, with CUDA 5.2 compute capability, 24 streaming multiprocessors with 128 CUDA cores each and a warp size of 32. It reaches a peak performance of 6.14 TFLOPS, provides 12 GB of global memory and a memory bandwidth of 336 GB/s. We use the CUDA driver version 9.0.

6.1 Scalability of GPU-Accelerated Level 0 & 1 Skeleton Discovery Kernels

In a first experiment, we examine the scalability of the CUDA-based kernel implementations with a growing number of CI tests. We aim to investigate if our approach and the parallelization strategy are valid for both low- and high-dimensional datasets. To scale the number of CI tests, we generate correlation matrices with differing sizes, ranging from 10 variables, resulting in a 10 by 10 matrix, to a correlation matrix with 25, 000 variables. We chose 25, 000 variables deliberately, as higher-dimensional datasets would require more than the available 12 GB of global memory on the GPU. This hardware limitation is not considered in this paper but left open to be addressed in future work.

For the `Level 0` kernel the number of variables results in 45 CI tests for 10 variables and up to 312, 487, 500 CI tests for 25, 000 variables. The execution time is measured using the C++11 `chrono` library [1], starting prior to the kernel launch and stopping after a call of `cudaDeviceSynchronize()`. The same setup is used for all measurements of kernel execution times on the GPU. The measurements of the data transfer include memory allocation with `cudaMalloc()` and the freeing of memory using `cudaFree()`. In

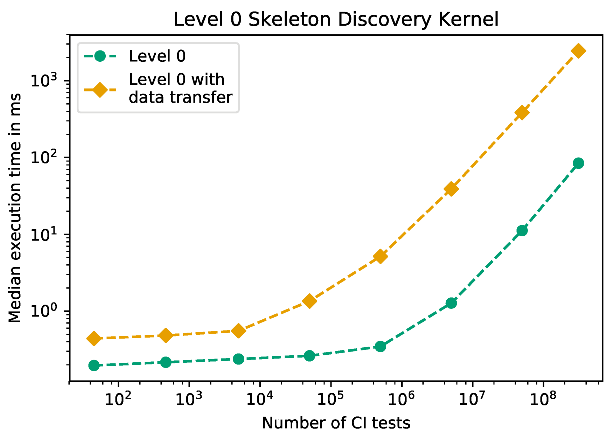

Figure 1: Execution time of the CUDA-based Level 0 skeleton discovery kernel with an increasing number of CI tests

Figure 1 we present the results for the `Level 0` kernel, excluding and including the data transfer to and from the GPU. The data transfer has a strong impact on the performance of the kernel, as it increases the execution time of the kernel by up to an order of magnitude.

In a standalone implementation, this poses a significant drawback. As we can reuse the data structures within the `Level 1` kernel this potential bottleneck becomes neglectable. In addition, it becomes visible that our parallel CUDA-based implementation scales linearly with respect to the number of CI tests for high-dimensional datasets. For low-dimensional datasets with up to 316 variables, represented by data point 4 in Figure 1, there is no significant difference in the execution time. An in-depth investigation of the kernel execution using the profiler `nvprof`[37], shows that this is due to an under utilization of the SMs on the GPU. The measured `sm_efficiency` ranges from 15% for 10 variables to 70% for 316 variables and reaches over 90% for all higher-dimensional datasets.

[1] http://en.cppreference.com/w/cpp/header/chrono

For all datasets, the achieved_occupancy indicates room for improvement of the implementation, as we did not exceed a value of 0.5. Hence, in future work, we should optimize the amount of work per warp, as well as the number of threads per thread block to fully utilize the GPU.

The number of CI tests executed for the Level 1 kernel is significantly higher. We use the same datasets, but considered a worst case, in which no edge is removed. We achieved this by setting the significance level α to 1.0. Hence, for 10 variables the kernel calculates 360 CI tests and for the generated dataset with 25, 000 variables the kernel executes approximately $7.8 \cdot 10^{12}$ CI tests.

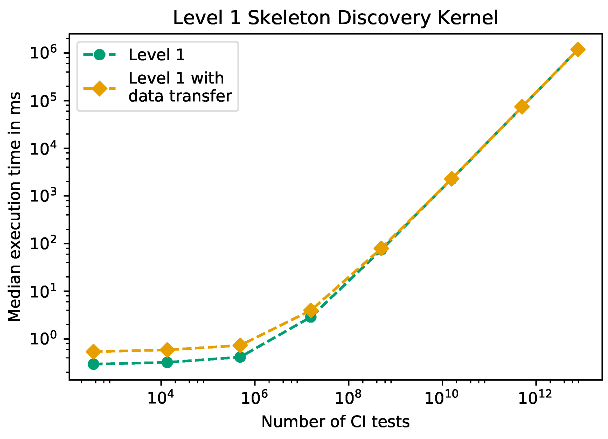

Figure 2: Execution time of the CUDA-based Level 1 skeleton discovery kernel with an increasing number of CI tests

The execution time for the Level 1 kernel, as shown in Figure 2, also scales linearly with the number of executed CI tests. Compared to our results for the Level 0 kernel, the overhead for the data transfer is comparably small and for higher dimensional datasets neglectable. The sm_efficiency shows low percentages for the first two measured data points and reaches almost 100% starting from 100 variables, as the kernel has to process 485, 100 CI tests and fully utilizes the available SMs. For the Level 1 kernel the achieved_occupancy also does not exceed a value of 0.5.

Considering both experiments, we conclude that processing up to approximately 500, 000 CI tests in parallel on a GPU, regardless of level $l = 0, 1$, does not change the execution time significantly. Once the available SMs on the device are fully utilized, for datasets requiring more tests, the execution time scales linearly with the number of CI tests.

6.2 Comparing GPU-based & CPU-based implementations on Gene Expression Data

In the following, we present measurements based on the real-world gene expression datasets described in Section 4 and compare the CUDA-based implementation with an existing implementation from the R-package pcalg [17] executed on the CPU. We choose the R-package for two reasons. First, R [41] is a widely accepted environment for statistical computations, providing a rich toolset and allowing integrating other languages to execute performance critical parts of the code, for example in C++. Second, the pcalg [17] R-package provides an implementation called stable.fast, which executes code written in C++ to efficiently calculate the skeleton for Gaussian distribution models. The implementation of the algorithm uses the package RcppArmadillo [13] to leverage the efficient linear algebra implementation of the Armadillo library [44]. In addition it supports openMP [11] for parallel execution of parts of the code.

We observed that setting the number of threads for openMP to the number of cores of our test machine led to an improvement in level 0. For the other levels of the skeleton discovery we experienced no improvements in execution time, on the contrary, we observed an increase in the overall runtime. We assume that this effect is observable due to a high cost for implicit synchronization. In our measurements, we used the best performing approach for each level. Hence we removed the openMP pragmas for any level other than level 0. To provide a fair comparison we considered the R-package parallel-PC [20], but experienced a slower execution time compared to the pcalg package, despite their parallel execution. We assume that this effect relates to the more efficient C++ based implementation used within the pcalg.

For the following measurements, we chose a significance level α of 0.01, include the data transfer between DRAM and GPU on-device memory in both directions, and present the minimum number of CI tests conducted for both level $l = 0$ and level $l = 1$. We obtained this lower bound from the CPU-based version, which stops iterating through the separation set candidates for an edge to be tested, as soon as a CI test is accepted and the edge is removed.

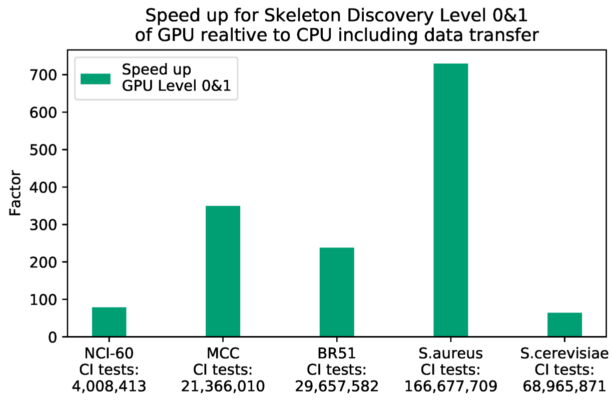

Figure 3: Speed up in Level 0 & 1 comparing GPU-based with CPU-based implementation

In Figure 3 we depict the speed up achievable by utilizing our proposed CUDA-based implementation. It ranges from a factor 64 for the S.cerevisiae dataset to a factor 729 for the S.aureus dataset. Interestingly there is no direct link between the speed up and the number of variables nor the number of CI tests. For example, the number of variables for the datasets MCC and BR51 are very similar, but more CI tests are required to process BR51 leading to less speed up. On the contrary, for the dataset S.aureus the largest speed up is achieved, processing the largest number of CI tests. To fully understand these results we take a closer look at the speed up of each kernel separately. In the graphs, in Figures 4 and 5 we do not take data transfer times into account.

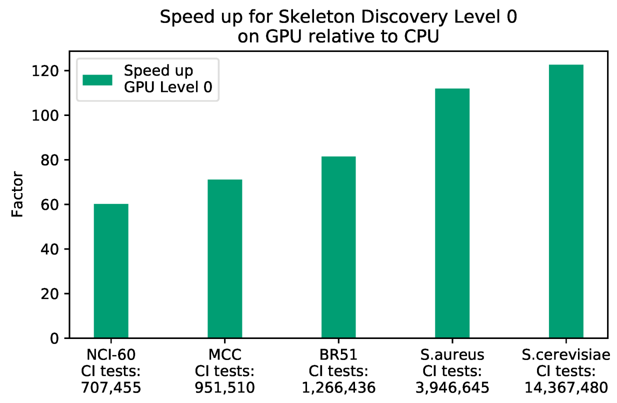

Figure 4: Speed up in Level 0 comparing GPU-based with CPU-based implementation

For `Level 0`, as shown in Figure 4, the number of conducted CI tests grows quadratic to the number of variables and is the same for the CPU and GPU version. We see an increasing speed up, when the number of conducted CI tests increases, which supports our idea to utilize the parallel processing capabilities of a GPU for the PC algorithm.

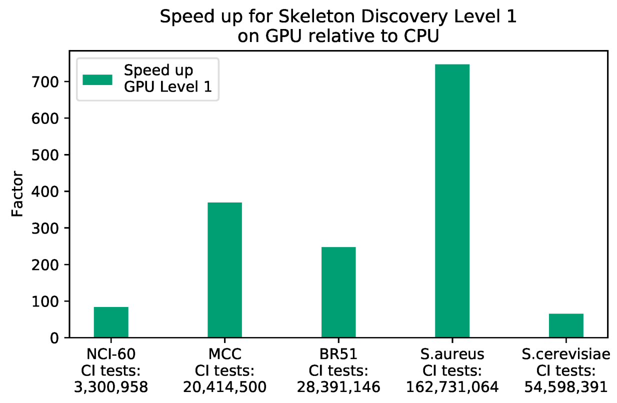

Figure 5: Speed up in Level 1 comparing GPU-based with CPU-based implementation

In Figure 5, the speed up for `Level 1` is shown. The irregularities, shown in Figure 3, clearly result from processing this level of the skeleton discovery. Profiling with `nvprof` did not show any significant differences between the kernel execution with different datasets. We assume the divergence is a result of our parallel processing approach and the underlying graphical model of the real-world gene expression datasets. Our GPU-accelerated implementation does not take any knowledge from the previous level into account, prior to the kernel launch. Hence, the kernel is launched for all possible edges and all separation set candidates. Using the information from the adjacency matrix `adj` each thread checks, if the edge to be tested has not been removed in the previous level. In addition, all separation set candidates are processed in parallel, even if the first one could have lead to an accepted CI test. This approach reduces the effect of branching but introduces new challenges, as the kernel potentially does unnecessary calculations. Such a step is not necessary for the implementation executed on the CPU. Therefore, for `BR51` CI tests are accepted earlier than for `MCC`, with regards to sequentially iterating through the separation set candidates, leading to additional overhead for the GPU-accelerated implementation. It remains future work, to do an in-depth investigation of this matter. Nevertheless, our CUDA-based implementation provides a significant speedup compared to a standard CPU-based implementation, for real-world gene expression datasets.

7 RELATED WORK

The PC algorithm [48] has been actively researched within the last years [5, 9, 15, 16, 20, 29, 30, 46, 49, 50]. The majority of the work focuses on extending the algorithm to address different domains of application [16, 30, 49, 50] or to reduce the complexity [5, 9, 15, 46]. Some work also investigates parallel execution strategies for the PC algorithm in order to cope with the potentially long execution times due to the computational complexity [20, 28, 29].

Le et al. [20] implement the `ParallelPC` R-package, for which they show a significant performance improvement, compared to the non-parallel version from the `pcalg` R-package, for real-world gene expression datasets. The implementation utilizes the CPU cores to process the CI tests within a level in parallel, thereby leveraging the order independence of the tests within a level [9]. They group all CI tests for a single edge together, processing them on the same core, because they observed that distributing CI tests for the same edge to different cores is not efficient. Depending on the data, this may lead to an unbalanced workload distribution among the processing units. Despite this potential shortcoming, we adopt the proposed strategy to process the CI tests of a single level in parallel and implement an adapted version targeting a GPU as an execution device.

Madsen et al. [29] discuss an implementation strategy for a parallel PC algorithm, which targets a shared memory computer and one implementation strategy for a computer cluster with distributed memory. They utilize the concept of Balanced Incomplete Block Designs [27] to distribute the computations of CI tests to threads in a shared memory system and to assign computations to worker nodes in a distributed system. Using these approaches, they experimentally show that a speed-up is achieved by parallel processing the CI tests on randomly generated samples from real-world networks. In contrast to our work, they focus on a discrete data distribution, while we consider a Gaussian distribution model. In addition, we explicitly target a GPU as an execution device to process the CI tests in parallel.

To the best of our knowledge, there does not exist any work on accelerating the PC algorithm by using coprocessors, specifically a GPU. The application of GPUs for computationally expensive tasks is an active field of research. A lot of the work in this domain focuses on accelerating machine learning algorithms [1, 6] or statistical computations, e.g., calculating the correlation coefficients [4]. Of particular interest for GPU-acceleration, are Deep Convolutional Neural Networks Deep Learning [18], which operate on matrices. Libraries, such as cuDNN [7] provide implementations to process these matrices efficiently on a GPU. We see these approaches as an inspiration for our work, to investigate the applicability of a GPU for the learning phase of the PC algorithm, the skeleton discovery, which is an example of a computationally expensive algorithm, since it has a runtime exponential to the number of variables.

8 SUMMARY AND OUTLOOK

In our work, we present an extension of the PC-stable algorithm, which utilizes the parallel processing capabilities of a GPU. Through an investigation of real-world gene expression datasets, we argue for the relevance of improving the performance of CI tests, considering Gaussian distribution model, within the levels $l = 0, 1$ of the adjacency search of the PC algorithm. We discuss our CUDA-based implementation of these two kernels and provide an evaluation regarding the scalability of the implementation with regards to the number of CI tests. For both cases, we include data transfer to the GPU. While it has a significant overhead for level $l = 0$, its overhead for level $l = 1$ is neglectable. Additionally, we compare our GPU-accelerated implementation with an existing implementation from the R-package `pcalg`, executed on the CPU. The measurements are based on the real-world gene expression datasets, which motivate our work, and show an improvement of factors up to 122 for level 0, up to 746 for level 1 and up to 729 for the combination of both, including data transfer. Therefore, we conclude that the parallel processing capabilities of a GPU are well suited to accelerate parts of the PC algorithm, in order to improve its application for real-world scenarios.

In the future, we aim to further investigate and improve the performance of our implementation of the kernel for `Level 0` and `Level 1`. We want to examine if processing multiple CI tests within a single thread can lead to performance improvements and increase the achieved occupancy of the SMs. This could be realized by using shared memory and reducing the number of accesses to global memory. Furthermore, a major goal in future work is an extension of our implementation to the remaining levels, and to provide a complete GPU-accelerated skeleton discovery implementation. Part of future work also needs to address the current limitation to datasets that fit in the on-device memory of the GPU. On the one hand it remains open to investigate the extension to multiple GPUs, thus having more memory. On the other hand one could also consider a strategy to split the graph into subgraphs, which fit in the on-device memory and to process these sequentially.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-scale Machine Learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'16)*. USENIX Association, Berkeley, CA, USA, 265–283. http://dl.acm.org/citation.cfm?id=3026877.3026899

[2] H. Ahrens. 1975. Dempster, A. P.: Elements of Continuous Multivariate Analysis. Addison-Wesley Publ. Co., Reading, Mass. 1969. XII, 388 S. *Biometrische Zeitschrift* 17, 7 (1975), 468–468. https://doi.org/10.1002/bimj.19750170714

[3] Steen A. Andersson, David Madigan, and Michael D. Perlman. 1997. A Characterization of Markov Equivalence Classes for Acyclic Digraphs. *The Annals of Statistics* 25, 2 (1997), 505–541. http://www.jstor.org/stable/2242556

[4] Joshua Buckner, Justin Wilson, Mark Seligman, Brian Athey, Stanley Watson, and Fan Meng. 2010. The gputools package enables GPU computing in R. *Bioinformatics* 26, 1 (Jan. 2010), 134–135. https://doi.org/10.1093/bioinformatics/btp608

[5] A. Cano, M. Gómez-Olmedo, and S. Moral. 2008. A Score Based Ranking of the Edges for the PC Algorithm. In *Proceedings of the Fourth European Workshop on Probabilistic Graphical Models*, Manfred Jaeger and Thomas D. Nielsen (Eds.). 41–48.

[6] Bryan Catanzaro, Narayanan Sundaram, and Kurt Keutzer. 2008. Fast Support Vector Machine Training and Classification on Graphics Processors. In *Proceedings of the 25th International Conference on Machine Learning (ICML '08)*. ACM, New York, NY, USA, 104–111. https://doi.org/10.1145/1390156.1390170

[7] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. cuDNN: Efficient Primitives for Deep Learning. *CoRR* abs/1410.0759 (2014). arXiv:1410.0759 http://arxiv.org/abs/1410.0759

[8] David Maxwell Chickering. 2002. Learning Equivalence Classes of Bayesian-network Structures. *J. Mach. Learn. Res.* 2 (March 2002), 445–498. https://doi.org/10.1162/153244302760200696

[9] Diego Colombo and Marloes H. Maathuis. 2014. Order-independent Constraint-based Causal Structure Learning. *J. Mach. Learn. Res.* 15, 1 (Jan. 2014), 3741–3782. http://dl.acm.org/citation.cfm?id=2627435.2750365

[10] Diego Colombo, Marloes H. Maathuis, Markus Kalisch, and Thomas S. Richardson. 2011. Learning High-dimensional DAGs with Latent and Selection Variables. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence (UAI'11)*. AUAI Press, Arlington, Virginia, United States, 850–850. http://dl.acm.org/citation.cfm?id=3020548.3020648

[11] Leonardo Dagum and Ramesh Menon. 1998. OpenMP: An Industry-Standard API for Shared-Memory Programming. *IEEE Comput. Sci. Eng.* 5, 1 (Jan. 1998), 46–55. https://doi.org/10.1109/99.660313

[12] A. P. Dawid. 1979. Conditional Independence in Statistical Theory. *Journal of the Royal Statistical Society. Series B (Methodological)* 41, 1 (1979), 1–31. http://www.jstor.org/stable/2984718

[13] Dirk Eddelbuettel and Conrad Sanderson. 2014. RcppArmadillo: Accelerating R with High-performance C++ Linear Algebra. *Comput. Stat. Data Anal.* 71 (March 2014), 1054–1063. https://doi.org/10.1016/j.csda.2013.02.005

[14] Marloes H Maathuis, Diego Colombo, Markus Kalisch, and Peter Bühlmann. 2010. Predicting causal effects in large-scale systems from observational data. 7 (04 2010), 247–8. https://doi.org/10.1038/nmeth0410-247

[15] J. Abellán and M. Gómez-Olmedo and S. Moral. 2006. Some Variations on the PC Algorithm. In *Proceedings of the Third European Workshop on Probabilistic Graphical Models (PGM' 06)*. 1–8.

[16] Markus Kalisch and Peter Bühlmann. 2007. Estimating High-Dimensional Directed Acyclic Graphs with the PC-Algorithm. *J. Mach. Learn. Res.* 8 (May 2007), 613–636. http://dl.acm.org/citation.cfm?id=1248659.1248681

[17] Markus Kalisch, Martin Mächler, Diego Colombo, Marloes H. Maathuis, and Peter Bühlmann. 2012. Causal Inference Using Graphical Models with the R Package pcalg. *Journal of Statistical Software, Articles* 47, 11 (2012), 1–26. https://doi.org/10.18637/jss.v047.i11

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'12)*. Curran Associates Inc., USA, 1097–1105. http://dl.acm.org/citation.cfm?id=2999134.2999257

[19] Steffen L Lauritzen. 1996. *Graphical models*. Vol. 17. Clarendon Press.

[20] Thuc Le, Tao Hoang, Jiuyong Li, Lin Liu, Huawen Liu, and Shu Hu. 2015. A fast PC algorithm for high dimensional causal discovery with multi-core PCs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* (02 2015).

[21] Thuc Duy Le, Lin Liu, Junpeng Zhang, Bing Liu, and Jiuyong Li. 2015. From miRNA regulation to miRNA-TF co-regulation: computational approaches and challenges. *Briefings in Bioinformatics* 16, 3 (2015), 475–496. https://doi.org/10.1093/bib/bbu023

[22] Joseph Lee Rodgers and W Alan Nicewander. 1988. Thirteen Ways to Look at the Correlation Coefficient. *The American Statistician* 42, 1 (1988), 59–66. https://doi.org/10.1080/00031305.1988.10475524

[23] Erich L Lehmann and Joseph P Romano. 2006. *Testing statistical hypotheses*. Springer Science & Business Media.

[24] Erik Lindholm, John Nickolls, Stuart Oberman, and John Montrym. 2008. NVIDIA Tesla: A Unified Graphics and Computing Architecture. *IEEE Micro* 28, 2 (March 2008), 39–55. https://doi.org/10.1109/MM.2008.31

[25] Ren C Luo, M-H Lin, and Ralph S Scherp. 1988. Dynamic multi-sensor data fusion system for intelligent robots. *IEEE Journal on Robotics and Automation* 4, 4 (Aug 1988), 386–396. https://doi.org/10.1109/56.802

[26] Marloes H Maathuis, Markus Kalisch, and Peter Bühlmann. 2009. Estimating high-dimensional intervention effects from observational data. *The Annals of Statistics* 37, 6A (12 2009), 3133–3164. https://doi.org/10.1214/09-AOS685

[27] Anders L. Madsen, Frank Jensen, Antonio Salmerón, Martin Karlsen, Helge Langseth, and Thomas D. Nielsen. 2014. A New Method for Vertical Parallelisation of TAN Learning Based on Balanced Incomplete Block Designs. In *Probabilistic Graphical Models*, Linda C. van der Gaag and Ad J. Feelders (Eds.). Springer International Publishing, Cham, 302–317.

[28] Anders L. Madsen, Frank Jensen, Antonio Salmerón, Helge Langseth, and Thomas D. Nielsen. 2015. Parallelisation of the PC Algorithm. In *Proceedings of the 16th Conference of the Spanish Association for Artificial Intelligence on Advances in Artificial Intelligence - Volume 9422*. Springer-Verlag New York, Inc., New York, NY, USA, 14–24. https://doi.org/10.1007/978-3-319-24598-0_2

[29] Anders L. Madsen, Frank Jensen, Antonio Salmerón, Helge Langseth, and Thomas D. Nielsen. 2017. A Parallel Algorithm for Bayesian Network Structure Learning from Large Data Sets. *Know.-Based Syst.* 117, C (Feb. 2017), 46–55. https://doi.org/10.1016/j.knosys.2016.07.031

[30] Katerina Marazopoulou, Rumi Ghosh, Prasanth Lade, and David Jensen. 2016. Causal Discovery for Manufacturing Domains. *CoRR* abs/1605.04056 (2016). arXiv:1605.04056 http://arxiv.org/abs/1605.04056

[31] Daniel Marbach, James C. Costello, Robert Küffner, Nicole M. Vega, Robert J. Prill, Diogo M. Camacho, Kyle R. Allison, Manolis Kellis, James J. Collins, Andrej Aderhold, Gustavo Stolovitzky, and et al. 2012. Wisdom of crowds for robust gene network inference. *Nature Methods* 9, 8 (8 2012), 796–804.

[32] Christopher Meek. 1995. Strong Completeness and Faithfulness in Bayesian Networks. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence (UAI'95)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 411–418. http://dl.acm.org/citation.cfm?id=2074158.2074205

[33] Nicolai Meinshausen and Peter Bühlmann. 2010. Stability selection. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 72, 4 (2010), 417–473. http://dx.doi.org/10.1111/j.1467-9868.2010.00740.x

[34] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. 2008. Scalable Parallel Programming with CUDA. , Article 16 (2008), 14 pages. https://doi.org/10.1145/1401132.1401152

[35] NVIDIA Corporation. 2017. CUDA Math API. Retrieved March 10, 2018 from http://docs.nvidia.com/cuda/pdf/CUDA_Math_API.pdf

[36] NVIDIA Corporation. 2017. NVIDIA TESLA V100 GPU ACCELERATOR. Retrieved February 12, 2018 from http://www.nvidia.com/content/PDF/Volta-Datasheet.pdf

[37] NVIDIA Corporation. 2018. Profiler User's Guide. Retrieved March 10, 2018 from http://docs.nvidia.com/cuda/pdf/CUDA_Profiler_Users_Guide.pdf

[38] Judea Pearl. 2009. *Causality: Models, Reasoning and Inference* (2nd ed.). Cambridge University Press, New York, NY, USA.

[39] Judea Pearl and Thomas Verma. 1991. A Theory of Inferred Causation. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 441–452. http://dl.acm.org/citation.cfm?id=3087158.3087202

[40] Roger Penrose. 1955. A generalized inverse for matrices. In *Mathematical Proceedings of the Cambridge Philosophical Society*, Vol. 51. Cambridge University Press, Cambridge University Press, 406–413. https://doi.org/10.1017/S0305004100030401

[41] R Core Team. 2017. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. http://www.R-project.org/

[42] Andrea Rau, Florence Jaffrézic, and Grégory Nuel. 2013. Joint estimation of causal effects from observational and intervention gene expression data. *BMC systems biology* 7, 1 (Oct 2013), 111. https://doi.org/10.1186/1752-0509-7-111

[43] Thomas Richardson. 1996. A Discovery Algorithm for Directed Cyclic Graphs. In *Proceedings of the Twelfth International Conference on Uncertainty in Artificial Intelligence (UAI'96)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 454–461. http://dl.acm.org/citation.cfm?id=2074284.2074338

[44] Conrad Sanderson and Ryan Curtin. 2016. Armadillo: a template-based C library for linear algebra. *Journal of Open Source Software* 1, 2 (Oct 2016), 26. https://doi.org/doi:10.21105/joss.00026

[45] Christian Schwarz, Christopher Schmidt, Michael Hopstock, Werner Sinzig, and Hasso Plattner. 2016. Efficient Calculation and Simulation of Product Cost Leveraging In-Memory Technology and Coprocessors. In *The Sixth International Conference on Business Intelligence and Technology (BUSTECH 2016)*.

[46] Marco Scutari. 2010. Learning Bayesian Networks with the bnlearn R Package. *Journal of Statistical Software, Articles* 35, 3 (2010), 1–22. https://doi.org/10.18637/jss.v035.i03

[47] Peter Spirtes. 2010. Introduction to Causal Inference. *J. Mach. Learn. Res.* 11 (Aug. 2010), 1643–1662. http://dl.acm.org/citation.cfm?id=1756006.1859905

[48] Peter Spirtes, Clark N Glymour, and Richard Scheines. 2000. *Causation, Prediction, and Search*. MIT press.

[49] Peter Spirtes and Kun Zhang. 2016. Causal discovery and inference: concepts and recent methodological advances. *Applied Informatics* 3, 1 (18 Feb 2016), 3. https://doi.org/10.1186/s40535-016-0018-x

[50] Michail Tsagris, Giorgos Borboudakis, Vincenzo Lagani, and Ioannis Tsamardinos. 2018. Constraint-based causal discovery with mixed data. *International Journal of Data Science and Analytics* (02 Feb 2018). https://doi.org/10.1007/s41060-018-0097-y

[51] Joe Whittaker. 2009. *Graphical Models in Applied Multivariate Statistics*. Wiley Publishing.