Out-of-Core GPU-Accelerated Causal Structure Learning

Christopher Schmidt[1], Johannes Huegle[1], Siegfried Horschig[2], and Matthias Uflacker[1]

[1] Enterprise Platform and Integration Concepts,Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
{firstname.lastname}@hpi.de
[2] siegfried.horschig@student.hpi.de

Abstract. Learning the causal structures in high-dimensional datasets enables deriving advanced insights from observational data. For example, the construction of gene regulatory networks inferred from gene expression data supports solving biological and biomedical problems, such as, in drug design or diagnostics. With the adoption of Graphics Processing Units (GPUs) the runtime of constraint-based causal structure learning algorithms on multivariate normal distributed data is significantly reduced. For extremely high-dimensional datasets, e.g., provided by The Cancer Genome Atlas (TCGA), state-of-the-art GPU-accelerated algorithms hit the device memory limit of single GPUs and consequently, execution fails. In order to overcome this limitation, we propose an out-of-core algorithm for GPU-accelerated constraint-based causal structure learning on multivariate normal distributed data. We experimentally validate the scalability of our algorithm, beyond GPU device memory capacities and compare our implementation to a baseline using Unified Memory (UM). In recent GPU generations, UM overcomes the device memory limit, by utilizing the GPU page migration engine. On a real-world gene expression dataset from the TCGA, our approach outperforms the baseline by a factor of 95 and is faster than a parallel Central Processing Unit (CPU)-based version by a factor of 236.

Keywords: GPU-Acceleration · Out-of-core · Causal Structure Learning · PC Algorithm.

1 Introduction

Learning causal structures from observational data is an active field of research in statistics and data mining. Discovering the causal relationships between observed variables in complex systems fosters new insights and is of particular interest in the context of high-dimensional settings such as in personalized medicine. For example, in genetic research, the construction of gene regulatory networks inferred from gene expression data supports drug design or diagnostics [20].

Causal graphical modeling is a well-known concept for the formalization of causal structures [18, 7, 26], where directed edges between nodes represent causal

**This is a pre-print of an article published in Algorithms and Architectures for Parallel Processing ICA3PP 2019.
The final authenticated version is available online at: https://doi.org/10.1007/978-3-030-38991-8_7**

relationships between the observed variables. Algorithms for learning the structure of causal graphical models from observational data build upon two main approaches: score-based and constraint-based methods. Score-based approaches treat structure learning as an optimization problem over a score function on all possible graphical models which raises an NP-hard problem [3]. Constraint-based approaches apply a series of statistical tests to identify the conditional independence constraints of the causal structure as a first step and an orientation step that incorporates deterministic rules as a second step. While this approach may be exponential to the number of observed variables in the worst case, constraint-based algorithms such as the PC algorithm introduced by Spirtes et al. [26] run in polynomial time in sparse settings. Nevertheless, the algorithm's long runtime hinders its application in practice, in particular for high-dimensional data [13].

With recent advances in hardware technology, i.e., multi-core, many-core, and Graphics Processing Unit (GPU)-accelerated systems parallel versions of the algorithm have been developed [15, 16, 24, 13, 23, 28]. GPU-accelerated causal discovery implementations report a significant speedup of factors between 700 to $1,300$ compared to Central Processing Unit (CPU)-based implementations under the assumption of multivariate normal distributed data [23, 28]. These approaches support the application in practice reporting fast learning of causal structures for datasets with up to $5,361$ observed variables. While both implementations focus on the PC-algorithm, the GPU-accelerated adjacency search is also applicable to other constraint-based causal structure learning algorithms, such as FCI, RFCI, and CCD [26, 5, 21]. Yet, both implementations are limited by the GPU device memory and execution fails once data structures exceed available memory. Given, that memory requirement is quadratic to the number of observed variables in the multivariate normal distributed case, the limitation of current GPU-accelerated implementations is reached for extremely high-dimensional data. In particular, in genetic research, high-dimensional datasets are being collected and made available for research, e.g., see The Cancer Genome Atlas (TCGA) [1]. Resulting gene expression datasets from TCGA contain information on more than $55,000$ observed genes [19], exceeding the device memory of recent GPU generations.

In order to benefit from GPU-acceleration beyond device memory limitations, we propose an out-of-core causal structure learning algorithm, which enables a GPU-accelerated adjacency search for extremely high-dimensional datasets, i.e., as available in TCGA. In our approach, we split data into smaller-sized blocks that fit into device memory and yield enough statistical tests to occupy the compute cores of the GPU. In the worst case, the adjacency search of the PC Algorithm requires to conduct statistical tests between all observed variables, thereby requiring data from multiple blocks. Therefore, we incorporate knowledge of data dependencies across blocks for higher-order statistical tests and manually manage data transfer. By overlapping computation and data transfer using different CUDA Streams[3] we hide any transfer overhead. We compare our block-based implementation with manual memory management to a version

[3] http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#streams-cdp

This is a pre-print of an article published in Algorithms and Architectures for Parallel Processing ICA3PP 2019.
The final authenticated version is available online at: https://doi.org/10.1007/978-3-030-38991-8_7

based on automatic memory management utilizing the default page migration engine available in current NVIDIA V100 GPUs. Thereby, we show the scalability of our block-based implementation beyond device memory limitations for higher-order statistical tests within the PC algorithm and a speedup of the runtime on a gene expression dataset from TCGA by a factor of 95.

The remainder of this paper is structured as follows. Section 2 provides the necessary background on constraint-based causal structure learning algorithms, in particular, the PC algorithm. In Section 3, we discuss related work on parallel constraint-based causal structure learning and general out-of-core GPU-accelerated algorithms. Afterward, in Section 4, we present our GPU-accelerated block-based implementation of the adjacency search, which allows execution on datasets exceeding device memory limits. The evaluation of the implementation is provided in Section 5 before we conclude our work in Section 6.

2 Background

In this section, we provide necessary background on constraint-based causal structure learning, in particular, the PC-stable algorithm. Further, we introduce an existing GPU-accelerated version of the adjacency search in the context of the PC-stable algorithm's skeleton estimation.

2.1 Constraint-Based Causal Structure Learning

In the framework of causality according to Pearl [18] and Spirtes et al. [25], causal relationships between N observed variables V_i, $i = 1, \dots, N$, can be represented in a Directed Acyclic Graph (DAG) $\mathcal{G}$. In this DAG $\mathcal{G} = (\mathbf{V}, \mathbf{E})$, the vertices $\mathbf{V} = \{V_1, \dots, V_N\}$ represent the observed variables and the directed edges $\mathbf{E} \in \mathbf{V} \times \mathbf{V}$ between the vertices denote direct causal relationships, i.e., $V_i \to V_j$ for $i, j = 1, \dots, N$. Constraint-based methods for causal structure learning exploit the factorization properties of the joint distribution P of $\mathbf{V}$ and apply conditional independence (CI) tests to determine the Markov equivalence class of the DAG $\mathcal{G}$ that is uniquely described by a Complete Partially Directed Acyclic Graph (CPDAG) [2]. In particular, the undirected skeleton $\mathcal{C}$ of $\mathcal{G}$ together with all unshielded colliders $V_i \to V_j \leftarrow V_k$ for non-adjacent V_i and V_j with $i, j, k = 1, \dots, N$ in $\mathcal{G}$ entail the CI information of the joint distribution P of $\mathbf{V}$. Accordingly, based on the d-separation criterion [18], two variables V_i and V_j are conditionally independent given a set of variables $\mathbf{S}$ if and only if the vertices V_i and V_j are d-separated by the set $\mathbf{S} \subset \mathbf{V} \setminus \{V_i, V_j\}$ for $i, j = 1 \dots, N$. Hence, under the assumption of causal sufficiency and causal faithfulness of the joint distribution P of $\mathbf{V}$ the application of consistent CI tests enables to derive the Markov equivalence class of the true underlying DAG $\mathcal{G}$, e.g., see [4, 25, 9].

Constraint-based algorithms, such as the well-known PC algorithm introduced by Spirtes et al. [26], follow this theoretical foundation. The algorithm first determines the undirected skeleton $\mathcal{C}$ of $\mathcal{G}$ in an adjacency search through the repeated application of CI tests between variables given increasing separation sets $\mathbf{S}$ chosen from the set of adjacent vertices in $\mathbf{V}$. In a second step, the repeated application of deterministic orientation rules on the skeleton $\mathcal{C}$ orients edges, by orienting the detected unshielded colliders in $\mathcal{C}$ based on the examined separation set $\mathbf{S}$. Moreover, edges can be oriented through the application of further orientation rules such that neither additional unshielded colliders nor cycles are present in the resulting graph which results in the CPDAG.

For multivariate normal distributed variables $V_1, \dots, V_N$, two variables V_i and V_j are conditionally independent given a set of variables $\mathbf{S} \subseteq \{V_1, \dots, V_n\} \setminus \{V_i, V_j\}$ if and only if the corresponding partial correlation coefficient is equal to zero [12]. Hence, a consistent conditional independence test can be derived upon the corresponding sample partial correlation coefficient $\hat{\rho}(V_i, V_j | \, \mathbf{S})$ following standard statistical decision theory [14]. Moreover, in high-dimensional multivariate normal distributed settings, the PC algorithm is established as a computationally feasible and provable correct estimation procedure of the equivalence class of a sparse DAGs $\mathcal{G}$ [9].

While the original version of the PC algorithm depends on the order of the variables set $V_1, \dots, V_N$ the PC-stable algorithm [4] is an order-independent extension that is the basis for efficient parallel adaptions [13, 24], also in the context of GPU-accelerated implementations [23, 28]. This order-independent version of the adjacency search of the PC-stable algorithm is outlined in Algorithm 1.

Algorithm 1 Adjacency search of PC-stable algorithm [4]
Input: Vertex set V, tuning parameter α
Output: Estimated skeleton $\mathcal{C}$, separation sets **Sepset**

```
Start with fully connected skeleton $\mathcal{C}$ and $l = -1$
repeat
    $l = l + 1$
    for all Vertices $V_i$ in $\mathcal{C}$ do
        Let $a(V_i) = adj(\mathcal{C}, V_i)$;
    end for
    repeat
        Select a pair of variables $V_i$ and $V_j$ adjacent in $\mathcal{C}$ with $|a(V_i) \setminus \{V_j\}| \geq l$
        repeat
            Choose separation set $\mathbf{S} \subseteq a(V_i) \setminus \{V_j\}$ with $|\,\mathbf{S}\,| = l$.
            if $\mathrm{p}(V_i, V_j | \, \mathbf{S}) \geq \alpha$ then
                Delete edge $V_i - V_j$ from $\mathcal{C}$;
                Save $\mathbf{S}$ in Sepset;
            end if
        until edge $V_i - V_j$ is deleted in $\mathcal{C}$
        or all $\mathbf{S} \subseteq a(V_i) \setminus \{V_j\}$ with $|\,\mathbf{S}\,| = l$ have been chosen
    until all adjacent vertices $V_i$, and $V_j$ in $\mathcal{C}$ such that
    $|a(V_i) \setminus \{V_j\}| \geq l$ have been considered
until each adjacent pair $V_i, V_j$ in $\mathcal{C}$ satisfy $|a(V_i) \setminus \{V_j\}| \leq l$
return $\mathcal{C}$, Sepset
```

Following the previously introduced concepts the adjacency search of the PC-stable algorithm depicted in Algorithm 1 starts with a complete undirected skeleton $\mathcal{C}$ and uses CI tests with an increasing size of separation set $\mathbf{S}$ of adjacent vertices in the corresponding level l to subsequently remove edges between vertices that are determined as being independent (see lines 8-15).

For every level l, the adjacency sets $a(V_i) = adj(\mathcal{C}, V_i)$ of vertices V_i within the current skeleton $\mathcal{C}$ are computed and stored (see lines 4-6). Thus, at each level l, the algorithm marks edges for removal and deletes the edges only when entering the subsequent level $l+1$ which guarantees the order-independence.

For $l = 0$, all edges between adjacent vertices $V_i, V_j \in \mathbf{V}$ are deleted if the corresponding independence test, i.e., given an empty set $\mathbf{S} = \emptyset$, rejects the null-hypothesis of dependence against independence. In this sense, independence between V_i and V_j can be concluded if the p-value $\mathrm{p}(V_i, V_j|\emptyset)$ is greater than the significance level α and the empty set $\emptyset$ can be stored as separation set in **Sepset** (see lines 11-13 in Algorithm 1). Note, that the significance level α can be treated as a tuning parameter influencing the sparsity of the estimated skeleton. After testing all pairs of vertices the algorithm proceeds to the next level $l = 1$.

For $l = 1$, the algorithm applies the CI tests for variables V_i and V_j given a separation set of size 1 if they remained adjacent in the skeleton $\mathcal{C}$ after $l = 0$. Therefore, it is now examined whether it holds for the corresponding p-value that $\mathrm{p}(V_i, V_j|\,\mathbf{S}) \geq \alpha$ with subset $\mathbf{S} \subset adj(\mathcal{C}, V_i) \setminus V_j$ of size 1 until either all other subsets in $adj(\mathcal{C}, V_i) \setminus V_j$ have been considered or the variables V_i and V_j are found to be conditionally independent. If the variables V_i and V_j are found to be conditionally independent, the corresponding edge $V_i - V_j$ is removed, and the corresponding separation set $\mathbf{S}$ is stored in **Sepset**. Once all pairs of variables V_i and V_j that are adjacent in the current skeleton $\mathcal{C}$ are tested the algorithm proceeds to the next level $l+1$. The same procedure is repeated until l reaches the maximum size of the adjacency sets of the vertices $max_{V_i \in \mathbf{V}}|adj(\mathcal{G}, V_i) \setminus \{V_j\}|$ in the underlying DAG.

2.2 GPU-Accelerated Adjacency Search

For the case of multivariate normal distributed variables $V_1, \ldots, V_N$, approaches exist that allow the processing of the corresponding consistent CI tests on the basis of partial correlations in parallel on the GPU [23, 28]. In order to benefit from the parallel processing capabilities of GPUs, the CI tests are distributed to threads following the Single Instruction Multiple Threads (SIMT) model.

For level $l = 0$, a mapping of CI tests to threads is straightforward as, given an empty separation set $\mathbf{S} = \emptyset$ for each pair of vertices V_i and V_j, each thread is processing a single CI test which launches N^2 threads [23]. For subsequent level $l \geq 1$, a CI test given a subset $\mathbf{S}$ of size l is considered. Hence, testing a pair of vertices V_i and V_j for conditional independence within a single level l may require multiple CI tests given appropriate adjacent subsets $\mathbf{S}$ in the current skeleton $\mathcal{C}$. Therefore, current implementations join a fixed number of threads to blocks for each pair of vertices V_i and V_j, e.g., launching a single thread block with a number of threads. In this case, each thread within a thread block is responsible for processing several CI tests for the corresponding pair of vertices V_i and V_j given the appropriate subsets $\mathbf{S}$. This distribution of CI tests for the same pair of vertices V_i and V_j to threads within the same thread block allows sharing data structures using shared memory and improves performance [28]. Furthermore, early termination is possible through synchronization within the thread block such that unnecessary CI tests are avoided. This ensures that the separation sets $\mathbf{S}$ examined are consistent for GPU-based and CPU-based implementations [23].

3 Related Work

Constraint-based causal structure learning is an active field of research, in particular in the context of the PC algorithm [26]. Given the algorithm's long execution time due to its computational complexity several parallel adaptions and implementations have been proposed [15, 16, 13, 24, 23, 28]. Whereas the majority of work addresses CPUs as execution units, recently also two GPU-accelerated versions for multivariate normal distributed datasets have been proposed [23, 28]. Both GPU-accelerated versions state a significant speed-up of three to four orders of magnitude compared to existing CPU versions.

In the work of Schmidt et al. [23] a GPU-accelerated adjacency search for levels 0 and 1 is proposed. As most CI tests on publicly available gene expression datasets are executed on levels $l = 0, 1$ they introduce an implementation of GPU-accelerated kernels for these two cases only. Another limitation of their implementation is the applicability to datasets that fit in the device memory.

The work of Zare et al. introduces a GPU-accelerated implementation, called cuPC [28], which is capable of processing any level l on the GPU. An extension cuPC-S enables the sharing of intermediate results during the computation of CI tests, in particular, parts of matrix inversions for the calculation of the sample partial correlation coefficients are shared. They show that the sharing of intermediate results can further reduce the execution time in higher levels $l \geq 1$. Moreover, they add a compact step after each level, for better assignment of threads to edges which leads to a reduction of the required memory. Hence, it helps to lift the device memory limitation for extremely high-dimensional datasets, yet, in the worst case, i.e., a dense graphical model, it also fails to process.

Approaches to overcome the device memory limitation of a GPU are subject to research [8, 6, 27, 17, 22, 29]. Application-specific out-of-core algorithms avoid hitting device memory limits by splitting the specific task or data and processing the parts independent from each other [8, 6]. These approaches either require a dividable-task [27] or a redesign of the algorithm [8]. Furthermore, the out-of-core algorithms require manual data management, i.e., data transfers, and orchestration, i.e., kernel launches. Generic frameworks to overcome the memory limitation provide the developer with API functions to allocate memory that is managed by the framework [17, 22]. This enables automatic data transfers, according to the memory regions requested by the GPU during kernel execution, using the GPU driver and the memory management unit of the GPU [17]. Thus, the available memory capacity is effectively extended to the capacity of DRAM

**This is a pre-print of an article published in Algorithms and Architectures for Parallel Processing ICA3PP 2019.
The final authenticated version is available online at: https://doi.org/10.1007/978-3-030-38991-8_7**

in the system [22] or going even beyond to nonvolatile memory (NVM) [17]. In the context of Unified Memory (UM), this capacity increase is achieved through the addition of 49-bit virtual addressing and on-demand page migration within recent GPU generations [22]. Yet, previous performance evaluations of UM on Kepler GPUs indicate that the utilization of UM introduces performance overhead with marginal improvement in code complexity [11].

4 An Out-Of-Core GPU-Accelerated Parallel Adjacency Search for the PC Algorithm

In the following section, we introduce two implementation strategies for an out-of-core GPU-accelerated adjacency search. The first implementation which can be treated as a baseline integrates the concept of UM into the GPU-accelerated skeleton discovery of Schmidt et al. [23]. Thus, it relies on a generic framework to overcome the device memory limitation. For the second implementation, we propose block-based algorithm of the adjacency search. In contrast to the first approach, the second requires manual data management and orchestration, i.e., data transfer and kernel launches.

4.1 Unified Memory Based Adjacency Search

With the introduction of a page migration engine on recent NVIDIA GPU generations [22], it is possible to use CUDA API calls to address larger memory than available on-chip. Specifically, a call to `cudaMallocManaged()` allocates memory, which is accessible from both the CPU and the GPU. In order to incorporate page migration into the kernels for levels $l = 0, 1$, for detail see [23], we adapt our implementation in the following way. Data structures required during kernel execution are allocated using `cudaMallocManaged()`, making them accessible from both CPU and GPU. Once, the GPU kernels are launched data is managed by the GPU driver and is transferred to the device transparently via the page migration engine. Thus, API calls for the explicit allocation of memory on the GPU and API calls to initiate data transfer to and from the device are removed. The source code of the kernels launched on the GPU remains unchanged.

4.2 Block-Based Adjacency Search

The block-based adjacency search follows the idea of splitting the data processed during the adjacency search into smaller blocks, that are independent of each other, to overcome the device memory limitation. The implementation aims to avoid any overhead introduced by UM [11] and aims to avoid dependency on novel GPU features. Following previous GPU-accelerated implementations [23, 28], our block-based adjacency search assumes multivariate normal distributed data that yields to consistent CI tests on the basis of the corresponding sample partial correlation coefficients $\hat{\rho}(V_i, V_j \,|\, \mathbf{S})$. As the sample partial correlation coefficients can be derived from the sample correlation coefficients between the

Algorithm 2 Block-based Adjacency Search of PC-stable algorithm
Input: Vertex set V, correlation matrix Cor, tuning parameter α, block size bs
Output: Estimated skeleton $\mathcal{C}$, separation sets **Sepset**

```
 1: Start with fully connected skeleton $\mathcal{C}$ and $l = -1$
 2: repeat
 3:     $l = l + 1$
 4:     for all Vertices $V_i$ in $\mathcal{C}$ do
 5:         Let $a(V_i) = adj(\mathcal{C}, V_i)$;
 6:     end for
 7:     $blocks = Split(V, \mathrm{Cor}, \mathcal{C}, \mathbf{Sepset}, bs)$
 8:     for all $b$ in $blocks$ do
 9:         Transfer $b$ to GPU
10:         if $l == 0$ then
11:             $BlockCITest(b, \alpha)$
12:         else
13:             $sepsetblocks = SepSetCombination(b, l, blocks)$
14:             for all $s$ in $sepsetblocks$ do
15:                 Transfer $s$ to GPU
16:                 $BlockCITest(b, s, \alpha)$
17:             end for
18:         end if
19:         Transfer $b$ from GPU
20:     end for
21:     $Merge(blocks)$
22: until each adjacent pair $V_i, V_j$ in $\mathcal{C}$ satisfy $|adj_{out}(V_i) \setminus \{V_j\}| \leq l$
23: return $\mathcal{C}$, Sepset
```

variables V_i, V_j and $\mathbf{S}$, our implementation operates on the correlation matrix Cor that contains the precomputed sample correlation coefficients $\hat{\rho}(V_i, V_j)$ for all $i, j = 1, \ldots, N$, for more information we refer to [23].

Algorithm 2 sketches the block-based adjacency search executed on a GPU, which extends the PC-stable algorithm (see Algorithm 1). Within each level l, the relevant data structures are split into a set of blocks b in *blocks* (see line 7 in Algorithm 2). Note, the input parameter bs determines the sizes of data structures within a block b in *blocks*. In this sense, the operation `Split()` returns an iterable list *blocks* of blocks b, for which each block b contains disjunct $bs \times bs$-submatrices of the following data structures: the correlation matrix Cor, the skeleton $\mathcal{C}$. Moreover, it returns corresponding subsets with cardinality bs of the current the adjacency set $a(V_i)$, the separation sets **Sepset**, and auxiliary data structure containing the calculated p-values $\mathrm{p}(V_i, V_j | \mathbf{S})$ as well as a mapping $m(b)$ of the positions of the $bs \times bs$ submatrices of Cor and $\mathcal{C}$ in b to their original positions. Zero-padding is applied in the case that the dimension of the dataset is not a multiple of bs. Next, the list *blocks* of blocks b is iterated and each block b is processed. First, all data structures in b are transferred to the GPU (see line 8 and 9). If the algorithm operates on level $l = 0$, the kernel can be launched directly as depicted in lines 10 and 11 of Algorithm 2. In this case, the algorithm operates on the fully connected skeleton $\mathcal{C}$, such that all CI tests between V_i and all V_j from the subset with cardinality b of the adjacency set $a(Vi)$ can be executed on the basis of the $bs \times bs$-submatrix Cor that incorporates the corresponding sample correlation coefficients. Processing of the edges in the corresponding undirected bs-dimensional subgraph of $\mathcal{C}$ in `BlockCITest()` follows the standard procedure within the lines 8 and 18 of Algorithm 1. Afterwards, the corresponding result of block b, e.g., the separation sets and p-values as well as the updated part of $\mathcal{C}$, is transferred back to the CPU where it is merged together with all other block results within *blocks* (see line 19-21).

For all remaining levels $l \geq 1$, a CI test between a pair of variables V_i and V_j adjacent in the b-dimensional subgraph of $\mathcal{C}$ requires a separation set $\mathbf{S}$ of size l drawn from the subset with cardinality b from $a(V_i)$. Note that, in contrast to level $l = 0$, the derivation of the p-values $\mathrm{p}(V_i, V_j | \mathbf{S})$ corresponding to CI test may not be restricted to the sample partial correlation coefficient that are available in $bs \times bs$-submatrix of Cor within block b. Thus, additional $bs \times bs$-submatrices of Cor, which we call separation set blocks are required. The number of separation set blocks increases with l. For a given block b, the list of sets of required separation set blocks, *sepsetblocks*, is determined within the function `SepSetCombination()` (see line 13). The list contains all $bs \times bs$ disjunct submatrices of Cor that contain at least one sample correlation coefficient that is needed to derive the p-value $\mathrm{p}(V_i, V_j | \mathbf{S})$ of the corresponding CI-test between the variables V_i and V_j given a subset $\mathbf{S}$ with $|\,\mathbf{S}\,| = l$ of the subset of $a(V_i) \backslash \{V_j\}$ with cardinality bs. As depicted in the lines 14 till 17 in Algorithm 2, each element s in the list *sepsetblocks* is transferred to the GPU such that a GPU kernel can be launched, which conducts all necessary CI tests between the variables V_i and V_j given the corresponding set $\mathbf{S}$. Afterward, the block b contains all required conditional independence results, e.g., separation sets and p-values as well es the corresponding bs-dimensional subgraph of $Skel$, and is transferred from the GPU into the host memory (see line 19). Once, all blocks b in *blocks* have been processed the results are merged in line 21. This process is repeated for an increasing level l until the same termination criterion in line 19 of the PC-stable algorithm sketched in Algorithm 1 is satisfied (compare line 22 of Algorithm 2).

Following previous work [23], we provide an implementation of the block-based algorithm for levels $l = 0, 1$. This enables to investigate the performance improvements of the introduced concepts for CI tests with ($l = 1$) and without ($l = 0$) a separation set. Note, that according to cupc [28] the behaviour for levels $l \geq 2$ is similar to level $l = 1$. In the implementation of Algorithm 2, we incorporate the following optimizations. First, in level $l = 0$, we do not transfer the adjacencies $a(V_i)$ and the separation sets **Sepset** reducing the memory footprint during kernel execution. In the level $l = 0$, the deletion of an edge can be carried out on the skeleton $\mathcal{C}$ directly, without influencing any other CI test. Thus, the adjacencies $a(V_i)$ are not necessary. Furthermore, an empty separation set is used in level $l = 0$, which we assume as the initial value in the data structure **Sepset** for each pair of variables V_i, V_j. Hence, setting it within the

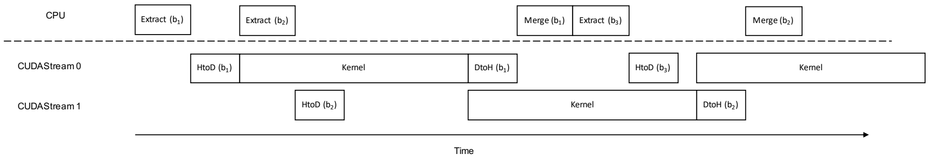

Fig. 1. Overlapping execution on CPU with data transfer and execution on GPU within the adjacency search of the block-based PC algorithm.

kernel for $l = 0$ is not required. Note, that the same optimization applies to the kernel for level $l = 0$ in the UM-based adjacency search. Second, we overlap data extraction and merging on the CPU, as well as, data transfer to the GPU with kernel execution, as shown in Figure 1, in order to reduce the overall runtime of the block-based algorithm. We realize the overlap by using two separate CUDA streams and adapt the proposed Algorithm 2, accordingly. In the implementation we first obtain the mapping for the blocks b, thus enabling to extract the appropriate data structures within the for loop in line 8 of Algorithm 2 for each block b independently. The same applies to the separation set blocks. Hence, all required data structures to launch a kernel on block b are extracted on the CPU and transferred to the GPU on one CUDA stream, while at the same time a GPU kernel is executed in a second, separate, CUDA stream, processing a second block from $blocks$. Once the kernel execution is finished, data is transferred back to the host and merged on the CPU, within the for loop.

5 Evaluation

For the evaluation of the proposed block-based adjacency search executed on a GPU, we conduct the following two experiments. First, we compare the performance of our block-based approach to the baseline implementation that applies the concept of UM with a page migration engine to overcome the device memory limit. Here, we focus on an examination of the approaches with regards to their scalability with respect to an increasing number of multivariate normal distributed variables, assuming a fully connected underlying causal graphical model. Since the assumption of a fully connected causal graphical model is not realistic demonstrating the worst case, we conduct a second experiment. Therefore, we examine a real-world gene expression dataset from the TCGA project that has been used in the context of integrative gene selection approaches [19]. Thereby, we investigate the performance benefits of the block-based approach compared to a CPU-based implementation and its applicability to real-world high-dimensional datasets.

The experiments are executed on an NVIDIA V100 card, with 32 GB of high bandwidth memory, inside an enterprise-grade server with 2 Intel ® Xeon ® Gold 6148 CPU with 20 cores each. The GPU card is connected via PCI-E version 4. Furthermore, the server is equipped with 1.5 TB of RAM, allowing to keep all data in memory during the execution of the experiments. The operating system is an Ubuntu 18.04 and the NVIDIA driver version 410.79 is installed with CUDA version 9.1. For measurements of the runtime we utilize the `system_clock` from the `C++` standard library `chrono`. This allows to include all CPU-based operations, such as splitting and merging of blocks, kernel launches and data transfer in our measurements. In particular, we measure the runtime for each level $l = 0, 1$ separately. If not stated differently, we report the median value of 10 measured executions for dataset sizes until $40,000$ variables, for higher dimensional datasets we report the median value of 3 measured executions, due to long runtimes. For the block-based approach, we choose a block size bs of $2,560$, which maps to the hardware specifications of the NVIDIA V100. For brevity we omit measurement results, which confirmed the chosen block size.

5.1 Experiments on Scalability Properties

In the first experiment, we examine the scalability with regards to the number of variables in the dataset. We consider datasets with $5,000$ to $55,000$ variables, increasing with a step size of $5,000$. Hence, we scale beyond the dimensions of previously used datasets for evaluation of GPU-accelerated causal structure learning algorithms [23, 28], which did not exceed the available device memory. Based on available gene expression datasets from TCGA [19], we choose $55,000$ variables as our upper limit for the experiment. Under the assumption of double-precision values, all data structures required during kernel execution result in a maximum memory footprint of 0.47 GB for $5,000$ variables to 56.35 GB for $55,000$ variables in level $l = 0$. Respectively, the maximum memory footprint in level $l = 1$ for $5,000$ variables is 0.65 GB and for $55,000$ variables is 78.88 GB. Note, the difference is due to two memory optimizations in level $l = 0$ applicable to approaches, as explained in Section 4. Furthermore, in order to eliminate any influence of the underlying graph structure on our measurements, we assume that all possible CI tests have to be conducted within each level l. This coincides with a fully connected graph and sketches a worst case for the number of CI tests to be conducted. Hence, in level $l = 0$, over 12 million CI tests are conducted for $5,000$ variables and over a billion CI tests are conducted for $55,000$ variables. For level $l = 1$ this results in over 60 billion CI tests for $5,000$ variables and around 83 trillion CI tests for $55,000$ variables.

The measurements presented in Figure 2 display the median execution times for the adjacency search in level $l = 0$ on the left and the adjacency search for level $l = 1$ on the right. The vertical blue line marks the device memory limit of the GPU used in the experiments. Note, previous implementations would fail on datasets whose number of variables is located to the right of the vertical line. Furthermore, the measurements of the block-based adjacency search is drawn in a yellow line denoted with GPU Block-Based and the UM-based adjacency search drawn in a green line denoted with GPU UM.

Considering the adjacency search for level $l = 0$ our measurements show that the block-based approach has a continuously higher execution time compared to the UM-based approach. In fact, the UM-based approach outperforms the block-based approach by a factor of up to 15 for datasets with few variables to a factor of 4.4 for datasets with $55,000$ variables. In-depth analysis with the

**This is a pre-print of an article published in Algorithms and Architectures for Parallel Processing ICA3PP 2019.
The final authenticated version is available online at: https://doi.org/10.1007/978-3-030-38991-8_7**

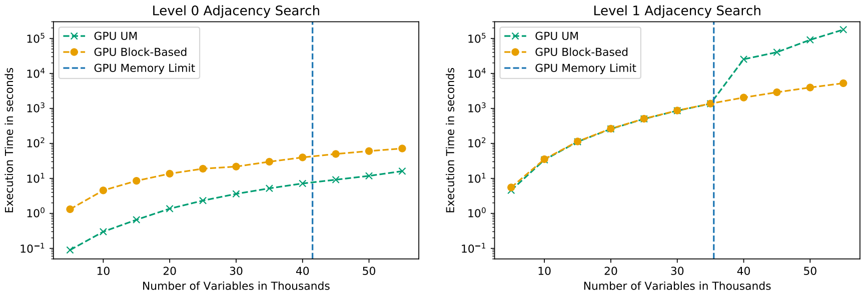

Fig. 2. Median execution times of the GPU UM-based (green) and the GPU Block-based (yellow) implementation for level 0 (left) and level 1 (right) of the skeleton discovery with an increasing number of variables; with all possible CI tests conducted

`nvprof` profiler has shown that the total data transfer time exceeds the kernel execution time. Additional time has to be added for the block extraction and merge on the CPU, which is not efficiently overlapped. Both approaches scale well, beyond the device memory limit. As no separation sets are required in level $l = 0$ and coalesced memory accesses is used such that the number of page faults scales linearly with the dataset size. Thus, the UM-based approach shows good performance beyond device memory limitations. Note, that the performance gap between both approaches decreases with an increasing number of variables.

For the adjacency search in level $l = 1$ the runtime of both approaches is similar up to a dataset with $35,000$ variables. For datasets with a number of variables that scales beyond this number, the device memory limit is exceeded. The block-based approach shows similar scalability on these higher-dimensional datasets compared to the lower-dimensional datasets. In contrast, the UM-based approach's performance drops significantly beyond the GPU memory limit. In fact, the block-based approach outperforms the UM-based approach by factors from 12.4, for $40,000$ variables up to 34, for $55,000$ variables. Profiling with `nvprof` revealed that for the UM-based approach the number of page faults increases drastically once the device memory limit is hit. For $35,000$ variables the number of page faults is around $140,000$ and increases to almost 12 million for $40,000$ variables. The performance difference is accounted for by memory stalls, for page accesses that result in page faults. When conducting a CI test with a separation set of size $l = 1$, the values for the separation set can be arbitrarily scattered across pages within the required data structures. Hence, the page migration engine reloads previously evicted pages, as caching is not trivial.

Comparing the measured execution times for both levels, it is evident that the runtime in level $l = 1$ is significantly higher, due to the larger amount of CI tests that are conducted. Yet, for the smallest dataset in level $l = 1$ approximately 40 times more CI tests are conducted compared to the largest dataset in level $l = 0$, in less time. We account that for overhead in data transfer in level $l = 0$, which is also shown in [23] and results in a poorer ratio of computation per memory access.

Table 1. Execution times of the CPU- and the two GPU-based implementations over both levels $l = 0, 1$ on TCGA dataset consisting of $55,572$ variables with α of 0.01.

CPU pcalg (on 32 cores)	GPU UM	GPU Block-Based (2560 block size)
107.6 hours	43.4 hours	0.46 hours

5.2 Experiments on Real-World Gene Expression Data

In our second experiment, we examine the performance of the approaches on a real-world gene expression dataset from TCGA. The dataset contains $55,572$ variables with $3,189$ observations [19]. We set the tuning parameter α to 0.01, which is a common option found in literature [4]. In this setting, one-third of all edges are removed within level $l = 0$ leading to fewer tests in level $l = 1$. In addition to both GPU-accelerated implementations of the PC-stable algorithm's adjacency search, we executed a CPU-based OpenMP-enabled version chosen from the well-known R-package `pcalg` [10] (version 2.6). We conducted the CPU-based measurements on a separate system equipped with Intel ® Xeon ® E7-4850 v4 CPUs with 16 cores and 2 TB of DRAM. For the execution, we allowed OpenMP to scale to 32 cores and limited the execution to levels $l = 0, 1$, by setting the parameter of the $m.max = 1$. Using the skeleton method `stable.fast` an underlying `C++` extension is used in the adjacency search, in which we integrate the time measurements. In Table 1, we state the measured execution times for the three approaches. The GPU block-based approach executes the adjacency search for level $l = 0, 1$ in 27.35 minutes. The GPU UM-based approach requires 43.4 hours to finish, suffering from a large number of page faults. In comparison, the CPU-based version runs for over 4 days. Thus, the GPU block-based approach outperforms, the GPU baseline using UM by a factor of 95 and the CPU-based implementation by a factor of 236.

6 Discussion and Conclusion

In this paper, we proposed an out-of-core GPU-accelerated adjacency search to overcome the GPU device memory limitation. The adjacency search is a substantial part of constraint-based causal structure learning algorithms, such as the PC-algorithm. It is used for the estimation of the Markov equivalence class of the underlying causal graphical model $\mathcal{G}$ from observational data. Our proposed algorithm splits the correlation matrix of multivariate normal distributed data and other relevant data structures, e.g., the skeleton, into small blocks such that datasets that usually exceed device memory can be processed efficiently on the GPU. We compare the approach to a baseline implementation using the concept of UM to overcome the memory limit. The baseline implementation relies on the page migration engine available in recent NVIDIA GPU-generations, which automatically transfers data between host and device memory. This baseline is easy to implement and shows good performance for level $l = 0$ of the adjacency search. In this case, it outperforms the block-based approach, which suffers from the overhead of block extraction and merging. In contrast, we show that the baseline suffers severely from page faults in case of arbitrarily scattered memory accesses, which occur in the adjacency search for level $l \geq 1$. For these cases, the performance gain of the block-based approach is significant, with factors up to 34. Furthermore, on real-world gene expression data, we show that the block-based approach outperforms the naive baseline by a factor of 95 and a parallel CPU-based version by a factor of 236. While our current implementation is limited to levels $l = 0, 1$, we assume similar behavior for higher levels $l \geq 2$. In future work, we aim to extend our implementation to these higher levels to evaluate limitations with regards to the number of required separation set blocks. Furthermore, adding a compact procedure, similar to the one described in cuPC [28], could be of interest since implementations may benefit from the condensed memory layout. Summarizing, we conclude that the block-based approach is well suited to extend GPU-accelerated causal structure learning algorithms to extremely high-dimensional datasets, which exceed the available device memory and could not benefit from the parallel processing capabilities of the GPU before.

References

1. Cancer Genome Atlas Research Network, Weinstein, J.N., Collisson, E.A., Mills, G.B., Shaw, K.R., Ozenberger, B.A., Ellrott, K., Shmulevich, I., Sander, C., Stuart, J.M.: The Cancer Genome Atlas Pan-Cancer analysis project. Nat Genet **45**(10), 1113–1120 (Oct 2013)
2. Chickering, D.M.: Learning Equivalence Classes of Bayesian-network Structures. J. Mach. Learn. Res. **2**, 445–498 (Mar 2002)
3. Chickering, D.M., Heckerman, D., Meek, C.: Large-Sample Learning of Bayesian Networks is NP-Hard. J. Mach. Learn. Res. **5**, 1287–1330 (Dec 2004)
4. Colombo, D., Maathuis, M.H.: Order-independent Constraint-based Causal Structure Learning. J. Mach. Learn. Res. **15**(1), 3741–3782 (Jan 2014)
5. Colombo, D., Maathuis, M.H., Kalisch, M., Richardson, T.S.: Learning High-dimensional DAGs with Latent and Selection Variables. In: Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence. pp. 850–850. UAI'11, AUAI Press, Arlington, Virginia, United States (2011)
6. Endo, T.: Realizing Out-of-Core Stencil Computations Using Multi-tier Memory Hierarchy on GPGPU Clusters. In: 2016 IEEE International Conference on Cluster Computing (CLUSTER). pp. 21–29 (Sep 2016)
7. Heckerman, D., Geiger, D., Chickering, D.M.: Learning Bayesian Networks: The Combination of Knowledge and Statistical Data. Machine Learning **20**(3), 197–243 (Sep 1995)
8. Kabir, K., Haidar, A., Tomov, S., Bouteiller, A., Dongarra, J.: A Framework for Out of Memory SVD Algorithms. In: Kunkel, J.M., Yokota, R., Balaji, P., Keyes, D. (eds.) High Performance Computing. pp. 158–178. Springer International Publishing, Cham (2017)
9. Kalisch, M., Bühlmann, P.: Estimating High-Dimensional Directed Acyclic Graphs with the PC-Algorithm. J. Mach. Learn. Res. **8**, 613–636 (May 2007)
10. Kalisch, M., Mächler, M., Colombo, D., Maathuis, M.H., Bühlmann, P.: Causal Inference Using Graphical Models with the R package pcalg. Journal of Statistical Software, Articles **47**(11), 1–26 (2012)

11. Landaverde, R., Tiansheng Zhang, Coskun, A.K., Herbordt, M.: An investigation of Unified Memory Access performance in CUDA. In: 2014 IEEE High Performance Extreme Computing Conference (HPEC). pp. 1–6 (Sep 2014)
12. Lauritzen, S.L.: Graphical models, vol. 17. Clarendon Press (1996)
13. Le, T., Hoang, T., Li, J., Liu, L., Liu, H., Hu, S.: A fast PC algorithm for high dimensional causal discovery with multi-core PCs. IEEE/ACM Transactions on Computational Biology and Bioinformatics (02 2015)
14. Lehmann, E.L., Romano, J.P.: Testing statistical hypotheses. Springer Science & Business Media (2006)
15. Madsen, A.L., Jensen, F., Salmerón, A., Langseth, H., Nielsen, T.D.: 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. pp. 14–24. Springer-Verlag New York, Inc., New York, NY, USA (2015)
16. Madsen, A.L., Jensen, F., Salmerón, A., Langseth, H., Nielsen, T.D.: A Parallel Algorithm for Bayesian Network Structure Learning from Large Data Sets. Know.-Based Syst. **117**(C), 46–55 (Feb 2017)
17. Markthub, P., Belviranli, M.E., Lee, S., Vetter, J.S., Matsuoka, S.: DRAGON: Breaking GPU Memory Capacity Limits with Direct NVM Access. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis. pp. 32:1–32:13. IEEE Press, Piscataway, NJ, USA (2018)
18. Pearl, J.: Causality: Models, Reasoning and Inference. Cambridge University Press, New York, NY, USA, 2nd edn. (2009)
19. Perscheid, C., Grasnick, B., Uflacker, M.: Integrative Gene Selection on Gene Expression Data: Providing Biological Context to Traditional Approaches. Journal of Integrative Bioinformatics (2018)
20. Rau, A., Jaffrézic, F., Nuel, G.: Joint estimation of causal effects from observational and intervention gene expression data. BMC systems biology **7**(1), 111 (Oct 2013)
21. Richardson, T.: A discovery algorithm for directed cyclic graphs. In: Proceedings of the Twelfth International Conference on Uncertainty in Artificial Intelligence. pp. 454–461. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1996)
22. Sakharnykh, N.: Beyond GPU Memory Limits with Unified Memory on Pascal (Dec 2016), https://devblogs.nvidia.com/beyond-gpu-memory-limits-unified-memory-pascal/
23. Schmidt, C., Huegle, J., Uflacker, M.: Order-independent Constraint-based Causal Structure Learning for Gaussian Distribution Models Using GPUs. In: Proceedings of the 30th International Conference on Scientific and Statistical Database Management. pp. 19:1–19:10. SSDBM '18, ACM, New York, NY, USA (2018)
24. Scutari, M.: Bayesian Network Constraint-Based Structure Learning Algorithms: Parallel and Optimized Implementations in the bnlearn R Package. Journal of Statistical Software, Articles **77**(2), 1–20 (2017)
25. Spirtes, P.: Introduction to Causal Inference. J. Mach. Learn. Res. **11**, 1643–1662 (Aug 2010)
26. Spirtes, P., Glymour, C.N., Scheines, R.: Causation, Prediction, and Search. MIT press (2000)
27. Wu, J., JáJá, J.: Achieving Native GPU Performance for Out-of-Card Large Dense Matrix Multiplication. Parallel Processing Letters **26**(2), 1–17 (2016)
28. Zare, B., Jafarinejad, F., Hashemi, M., Salehkaleybar, S.: cupc: Cuda-based parallel PC algorithm for causal structure learning on GPU. CoRR (2018)
29. Zheng, T., Nellans, D., Zulfiqar, A., Stephenson, M., Keckler, S.W.: Towards high performance paged memory for gpus. In: 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA). pp. 345–357 (March 2016)