Constraint-Based Causal Structure Learning in Multi-GPU Environments

Christopher Hagedorn[1], Johannes Huegle[1]

[1]{firstname.lastname}@hpi.de, Hasso Plattner Institute, University of Potsdam, Potsdam, Germany

Abstract

Learning causal structures from real-world high-dimensional data remains challenging due to algorithmic complexity and resulting long runtimes. We propose two multi-GPU approaches to discover causal relationships in multivariate normal distributed data, often assumed in genetic datasets, to address the long runtimes. In fact, on a high-dimensional dataset from The Cancer Genome Atlas (TCGA), we achieved a reduction in runtimes from over 100 hours on a multi-core CPU system to around 5 minutes on a system with multiple Graphics Processing Units (GPUs). Our two proposed multi-GPU approaches differ in their memory management. One version relies on the concept of Unified Memory (UM), while the other uses explicit memory management. Experiments on synthetic data show that explicit memory management is better suited for causal structure learning. On the one hand, it is faster than the version relying on UM by factors of up to 75. On the other hand, the explicit memory-managed version is less prone to the system's GPU interconnect topology effects.

Keywords

Multi-GPU, Causal Structure Learning, Causal Graphical Model, Conditional Independence Testing

1. Introduction

Causal Structure Learning (CSL) is an omnipresent challenge in many domains and an active field of research in statistics and data mining [1, 2, 3, 4]. In genetic research, for example, deriving the causal structures of gene regulatory networks from gene expression data enables drug design or diagnostics [3]. In these settings, data is high-dimensional [5, 6, 7, 8] as the systems under observation, e.g., gene expressions, are very complex. To address high computational demands resulting from high-dimensional data, efficient methods for CSL have been proposed using a GPU as an accelerator [9, 8, 10, 11]. Execution of CSL on a GPU introduces hardware specific challenges, and new requirements arise when multi-GPU systems are considered. Therefore, we propose a GPU-accelerated CSL method for a multi-GPU setting, which extends previous work.

GPU-Accelerated Causal Structure Learning Based on the theoretical framework for causal reasoning [12], causal relationships between observed variables are modeled in a Directed Acyclic Graph (DAG) [4, 12, 13]. Hence, methods for CSL aim to estimate the DAG from observational data and are commonly based upon score-based or constraint-based approaches [14]. GPU-acceleration for CSL focuses on constraint-based methods, in particular the well-known PC algorithm proposed by Spirtes et al. [4], and its order-independent version PC-

LWDA'21: Lernen, Wissen, Daten, Analysen September 01–03, 2021, Munich, Germany

© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

stable [15]. As typical for constraint-based approaches, the algorithm determines an undirected skeleton of the DAG by conducting conditional independence (CI) tests in a first step. Note that the distribution of the observed variables directly determines the appropriate CI test [16]. In a second step, the undirected edges are oriented through the repeated application of deterministic orientation rules. The first and most time-consuming step, also known as adjacency search, is executed in parallel on the GPU [9, 8, 10, 11]. The adjacency search constitutes the majority to the complexity of the PC-stable, which is exponential to the number of variables in the worst case [4]. The existing GPU-accelerated approaches target execution on a single GPU and focus on multivariate normal distributed data [8, 10, 11], or on discrete data [9].

Challenges in GPU-Accelerated CSL and Requirements on Multi-GPU CSL In the context of GPU-accelerated CSL, the existing CPU-based algorithms for CSL have to be adapted for the Single Instruction Multiple Threads (SIMT) execution model of a GPU [9, 10, 11]. Kernel execution on a GPU requires that data has to be transferred between the device and the system's DRAM through interconnects such as PCI-E or NVLink. Depending on the available bandwidth, data transfer becomes a limiting factor if it outweighs computation. Yet, in modern systems with fast interconnect, data transfer between DRAM and GPU memory no longer remains a bottleneck [17]. Further, the on-chip memory of a GPU is limited in size, requiring the implementation of out-of-core approaches, e.g., through block processing [8]. While these three challenges have partly been addressed for GPU-accelerated CSL [8, 10, 11], additional requirements arise in multi-GPU systems. In multi-GPU systems, the GPUs are connected through interconnects, and the memory bandwidth may vary depending on hardware and topology. Thus, different access costs have to be considered when designing efficient implementations (Requirement I). Further, execution in a multi-GPU system requires careful task distribution between the GPUs to avoid performance throttling due to load imbalance (Requirement II).

Contribution We propose two GPU-accelerated versions of the adjacency search within PC-stable, which allow execution in multi-GPU systems. Specifically, we propose a naive version relying on the concept of UM with a page migration engine on modern GPUs, which is tuned according to common best practices [18]. This version is fast to implement and, thus, serves as a baseline for further optimization. Further, we propose a second version, which incorporates knowledge on the execution order with explicit memory management. We compare the performance of both versions in a set of experiments synthetic data, as well as real-world genetic data [7], from The Cancer Genome Atlas (TCGA) [5]. In the experiments, we focus on scalability with the problem size and the number of GPUs. Further, our experimental setup allows us to investigate the performance impact of the interconnect topology.

The remainder of the paper is structured as follows. In Section 2, we provide background information on CSL. We discuss existing work on GPU-accelerated CSL and related work on execution in multi-GPU systems in Section 3. In Section 4, we explain our proposed multi-GPU adjacency searches for PC-stable and discuss their implementation. We evaluate the implemented algorithms experimentally and share our results in Section 5. Finally, we summarize our work, discuss the results, mention the current limitations of our approach and provide an outlook into future research directions in Section 6.

2. Preliminaries on Constraint-Based CSL

In the context of causal reasoning, let us define a Causal Graphical Model (CGM) $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ with a finite set of N vertices $\mathbf{V} = (V_1, \ldots, V_N)$, where each vertex represents an observed variable, and a set of edges $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$. An edge $(V_i, V_j) \in \mathbf{E}$, with $i, j = 1, \ldots, N$ and $j \neq i$, can either be directed $V_i \rightarrow V_j$ if $(V_i, V_j) \in \mathbf{E}$ and $(V_j, V_i) \notin \mathbf{E}$, or undirected $V_i - V_j$ if $(V_i, V_j) \in \mathbf{E}$ and $(V_j, V_i) \in \mathbf{E}$ [12]. Further, we assume that the CGM is *acyclic*, i.e., we do not allow for feedback loops, and that *causal sufficiency*, *faithfulness* and the *global Markov Property* hold [12]. Together with the graphical d-separation criterion, the global Markov Property allows determining the conditional independence relationships [12]. Two vertices $V_i, V_j \in \mathbf{V}$ are *d-separated* by a so-called separation set $\mathbf{S}$ with $\mathbf{S} \subset \mathbf{V} \setminus \{V_i, V_j\}$, if $\mathbf{S}$ blocks every path between V_i and V_j in $\mathcal{G}$. Thus, V_i, V_j are conditionally independent by $\mathbf{S}$, i.e., $V_i \perp\!\!\!\perp V_j | \mathbf{S}$. Consequently, if two vertices V_i and V_j are conditionally dependent given all possible subsets $\mathbf{S}$, the two vertices are connected through an edge in $\mathcal{G}$. In this work, we focus on constraint-based methods, particularly the PC-stable [15], and its GPU-accelerated extensions. The algorithm first learns an undirected skeleton $\mathcal{C}$ of the CGM by application of CI tests. Second, deterministic rules are applied to orient as many edges as possible [4, 15]. Note, given that several CGMs can describe exactly the same CI information and form a *Markov equivalence class* [19, 20] the focus on constraint-based algorithms lies on the estimation of the equivalence class of the CGM $\mathcal{G}$.

Within the first phase of the PC-stable, the adjacency search, the algorithm iteratively conducts CI tests with increasing size of the separation set $\mathbf{S}$, referred to as level l. In level $l = 0$, the algorithm starts with a fully connected CGM. Within each level, all remaining adjacent pairs of variables (V_i, V_j) are iterated and tested for CI given all remaining adjacent separation sets. In case of independence, the corresponding edge is removed in $\mathcal{C}$, the separation set $\mathbf{S}$ is stored, and no more CI tests are conducted for this pair of variables. A level l is completed once all adjacent pairs of variables have been considered. The adjacency search terminates once a level l is reached, in which no more separation sets $\mathbf{S}$ of size l can be constructed from adjacent vertices V. In this case, the estimated skeleton $\mathcal{C}$ and the determined separation sets are returned. For more detail and pseudocode, we refer to the work of Colombo and Maathuis [15].

In our work, we assume that the variables $\mathbf{V} = (V_1, \ldots, V_N)$ are multivariate normal distributed. Thus, we utilize an appropriate CI test for that particular distribution based upon the partial correlation coefficient [16]. A common choice for multivariate normal distributed data is Fisher's z-test [21], also used in GPU-accelerated CSL implementations [10, 8, 11]. The test builds upon the partial correlation coefficient $\rho(V_i, V_j | \mathbf{S})$ for a pair variables (V_i, V_j) given a separation set $\mathbf{S}$ [22, 23]. Using the partial correlation coefficient $\rho(V_i, V_j | \mathbf{S})$, the p-value is calculated and compared to a provided threshold α to determine independence for a pair of variables [10]. We denote Cor as the set of correlation coefficients $\rho(V_i, V_j | \mathbf{S})$ with $\mathbf{S} = \emptyset$ for all pairs of variables that is pre-calculated once for all pairs of variables.

3. Related Work

In the context of CSL, GPU-acceleration focuses on constraint-based methods, in particular PC-stable [9, 8, 10, 11]. Most of the work focuses on multivariate normal distributed data [8, 10, 11],

and only one work considers discrete data [9]. All current work addresses the challenge to map PC-stable to the GPU specific execution model. In particular, Schmidt et al. [10] map the pairs of variables (V_i, V_j) within a level to thread blocks and threads within a thread block to conduct multiple CI tests based on different separation sets $\mathbf{S}$. The mapping enables synchronization within a thread block for early termination and provides ample opportunity for parallel execution. The implementation focuses on separation sets of size 0 or 1, as the authors argue that the majority of CI tests for gene expression data occur in these levels. In subsequent work [8], the authors extend their algorithm to scale beyond the memory limit of a single GPU. Therefore, they propose a block-based version of PC-stable that splits data into blocks fitting into memory. They overlap data transfer between DRAM and GPU and CPU operations with kernel execution on the GPU to reduce transfer overhead. Comparing to a simple version based on UM and the integrated page migration engine of a GPU, they show that their explicit memory-managed version is faster once the memory limit on a GPU is reached. Zarebavani et al. [11] implement cupc, a generalized implementation for multivariate normal distributed data, which allows processing separation sets of arbitrary size. Further, cupc shares intermediate results within CI tests, i.e., computation of pseudo-inverse matrices, to reduce computational overhead in higher levels. Lastly, cupc includes a compacting step to reduce memory footprint by removing deleted edges from the adjacency structure. While this compacting reduces the memory footprint, cupc does not scale beyond the available memory on a single GPU.

To the best of our knowledge, no work exists on multi-GPU CSL. Yet, generally, the efficient execution of applications designed for single GPUs in multi-GPU environments is of high interest [24]. Kernel execution is distributed to multiple GPUs based on memory access patterns to support development in multi-GPU systems [25]. Cabezas et al. [26] distribute work in a multi-GPU system exploiting remote memory access capabilities, which they can hide if they remain below 5%. Ganguly et al. [27] describe a framework that automatically switches between remote zero-copy access to host-pinned memory and page migration for efficiency.

In our work, we extend existing work on single GPU-accelerated CSL to a multi-GPU system. In particular, we consider an optimized block-based version with explicit memory management. We reduce work for splitting and merging results on the CPU and scale to multiple GPUs providing a separate CPU thread per GPU. Also, we extend a version relying on UM to target multiple GPUs. In detail, we include optimization hints to reduce overhead due to paging.

4. Multi-GPU Adjacency Search of PC-Stable

In this section, we describe two GPU-accelerated versions of PC-stable for execution in multi-GPU systems. First, we propose a naive version relying on the concept of UM provided with explicit page migration support in modern GPUs. Second, we propose a multi-GPU ready version, which relies on explicit memory management to leverage knowledge on the access pattern and execution order.

Unified Memory-Based (UM) Multi-GPU Adjacency Search of PC-Stable Modern GPUs include a dedicated page migration engine, e.g., see NVIDIA GPUs [28], to support the concept of UM. Based on the page migration engine, memory can be allocated on either CPU or GPU

and is accessible from any device. If the memory is unavailable on the requesting device, a page fault occurs, and the page migration engine transfers the requested page.
Approach: The UM multi-GPU adjacency search of PC stable follows the general idea of previous work [10]. Separate kernels are launched for each level l, and CI tests are conducted in parallel on the GPU. To enable execution on multiple GPUs, we make the following adaptions. First, the data structures are allocated using the concept of UM making all data available for each GPU. Next, the pairs of variables of the CGM $\mathcal{G}$ are split equally by the number of GPUs, and each GPU becomes responsible for processing an equal share (cf. Requirement II). Thus, within each level, l one kernel is launched on each GPU that processes the GPU's dedicated share of pairs of variables. Once each kernel in a level l is finished, the next level is executed with the same mapping for the pairs of variables to the GPUs. This approach does not explicitly address Requirement I but relies on internal page migration mechanisms to avoid performance dips.
Implementation: We utilize the CUDA framework [29] for the implementation of the UM approach and build upon an existing single GPU implementation of constraint-based CSL [10]. In contrast to the existing implementation, we make the following adaptions for execution in a multi GPU system. First, all data structures are allocated using *cudaMallocManaged()*, which enables access from CPU and GPUs using the page migration engine. The primary data structures are matrices of size $N \times N$ that store Cor, $\mathcal{C}$ and the determined separation sets $\mathcal{S}$ for independent pairs of variables. Further, auxiliary data structures, e.g., to store the calculated p-values or for locking, are also allocated with *cudaMallocManaged()*. Next, a separate CPU thread is spawned per GPU to launch the kernels on the corresponding GPU. The CPU threads are synchronized after each level l to ensure order-independence [15]. The GPU kernels are adapted so that each GPU thread can process multiple consecutively stored elements, i.e., all elements within a page. The size is determined through an additional parameter to allow for optimal settings in different systems. In our experiments, we set this parameter to correspond to the system's page size of $4KB$, which yields the best results. Furthermore, the implementation uses common best practices [18] to guide memory management based upon calls to *cudaMemAdvise()*. In detail we set data structures that are mostly read, e.g., Cor to *cudaMemAdviseSetReadMostly* for each GPU. Note, the set of correlation coefficients is pre-calculated and provided as input to the algorithm. This calculation can be executed on the GPU [30], and for selected datasets makes up a tenth of the runtime of GPU-accelerated CSL, (cf. runtimes in [11] and [30]). Additional data structures for the pairs of variables read from and written to exclusively by one GPU, are guided with the hint *cudaMemAdviseSetPreferredLocation* and prefetched to the GPU using *cudaMemPrefetchAsync()*. Lastly, *cudaMemAdviseSetAccessedBy* is used for any other data structure, which is occasionally read from another GPU's memory to avoid page faults. For detail on each advice, we refer to the CUDA Toolkit Documentation [31].

Explicit Memory-Managed (EM) Multi-GPU Adjacency Search of PC-Stable Explicitly managing data transfer requires designing the implementation to ensure that all data required within a kernel is available on the GPU before its launch. Techniques such as overlapping data transfer and kernel execution mitigate upfront transfer costs. Further, explicitly managing the memory avoids any overhead introduced by UM [32] and allows for execution on GPUs without page migration support.

Algorithm 1 Explicit memory-managed multi-GPU adjacency search of PC-stable
Input: Vertex set $\mathbf{V}$, set of correlation coefficients Cor, tuning parameter α, block size bs, GPU count g
Output: Estimated skeleton $\mathcal{C}$, separation sets $\mathbf{S}$

1: Start with fully connected skeleton $\mathcal{C}$ and $l = -1$
2: $dataBlocks = Split(V, Cor, \mathcal{C}, \mathbf{S}, bs)$
3: $threadList = createThreads(g)$
4: **repeat**
5: $\quad l = l + 1$
6: $\quad$ **for all** b in $dataBlocks$ **do**
7: $\quad\quad$ **for all** Vertices V_i in $\mathcal{C}_b$ **do**
8: $\quad\quad\quad$ Let $a(V_i)_b = adj(\mathcal{C}_b, V_i)$;
9: $\quad\quad$ **end for**
10: $\quad$ **end for**
11: $\quad queue = fillQueue(dataBlocks)$
12: $\quad$ **for all** t in $threadList$ **do**
13: $\quad\quad$ **repeat**
14: $\quad\quad\quad$ b = $queue.pop()$
15: $\quad\quad\quad$ Transfer data of b to GPU_t
16: $\quad\quad\quad$ **if** $l == 0$ **then**
17: $\quad\quad\quad\quad$ $BlockCITest(b, \alpha)$ on GPU_t
18: $\quad\quad\quad$ **else**
19: $\quad\quad\quad\quad$ $sepBlockList = SeparationSetComb(b, l, dataBlocks)$
20: $\quad\quad\quad\quad$ **for all** s in $sepBlockList$ **do**
21: $\quad\quad\quad\quad\quad$ Transfer data of s to GPU_t
22: $\quad\quad\quad\quad\quad$ $BlockedCITest(b, s, \alpha)$ on GPU_t
23: $\quad\quad\quad\quad$ **end for**
24: $\quad\quad\quad$ **end if**
25: $\quad\quad\quad$ Transfer data of b from GPU_t
26: $\quad\quad$ **until** $queue$ is empty
27: $\quad$ **end for**
28: **until** each adjacent pair V_i, V_j in $\mathcal{C}$ satisfy $|a(V_i) \setminus \{V_j\}| \leq l$
29: $Merge(dataBlocks)$
30: **return** $\mathcal{C}$, $\mathbf{S}$

Approach: Algorithm 1 outlines the EM multi-GPU adjacency search of PC-stable in which data structures are split into equal blocks, which are distributed to the GPUs (cf. Requirement II). Our approach builds upon existing work [8], which we further optimize and extend to a multi-GPU setting. The algorithm receives the vertex set $\mathbf{V}$, the set of pre-calculated correlation coefficients Cor, the tuning parameter α, the block size bs, and the number of GPUs g to use during execution. The block size bs is chosen to avoid memory over-allocation on a single GPU on the one hand, yet it is large enough to achieve high occupancy on each GPU. Note that g is limited to the number of available GPUs in the system. The algorithm starts with a fully

connected skeleton $\mathcal{C}$. As a first step, the primary data structures, i.e., matrices for Cor, $\mathcal{C}$ and $\mathbf{S}$, are split into smaller blocks according to the block size bs (cf. line 2 in Algorithm 1). These data blocks are stored in a list $dataBlocks$. Note that a single split operation optimizes the existing block-based approach [8] which could extract the same block multiple times. Next, for each GPU used, a separate CPU thread is created. After this preparation, the adjacency search is executed until for all adjacent pairs of variables there exist no more possible separation sets in the respective skeleton $\mathcal{C}$ with increasing level l (cf. lines 4 - 28 in Algorithm 1). Within this search part, the order-independence [15] is ensured retrieving the adjacent vertices within the current level across all blocks (cf. lines 6 - 10 in Algorithm 1). Next, a central queue $queue$ is filled with all blocks from $dataBlocks$, and the threads start processing the elements within $queue$ until it is empty. We aim to fulfill Requirement II and provide a load balancing mechanism by utilizing a central queue for task processing. In detail, each thread t pops a single block b from the $queue$ and transfers the data of b to the corresponding GPU GPU_t. If level $l = 0$, the kernel conducting all CI tests for block b is launched on GPU GPU_t. For level $l > 0$, all possible candidate blocks to be considered for the separation sets are calculated and stored in $sepBlockList$ first (cf. line 19 in Algorithm 1). The elements s in $sepBlockList$ are iterated and data for the current element s is transferred to GPU GPU_t. Next, the kernel conducting the CI tests for the combination of block b and separation set blocks s within the current level l is launched. The resulting updated parts of the estimated skeleton $\mathcal{C}_b$ and the separation sets $\mathbf{S}_b$ of b are transferred back from the GPU GPU_t (cf. line 25 in Algorithm 1). Finally, the blocks are merged into single data structures again, and the estimated skeleton $\mathcal{C}$ and the separation sets $\mathbf{S}$ are returned. Note that during the algorithm, all data transfer occurs only between the system DRAM and each GPU memory separately. Thus, we avoid costs caused by the particular topology and interconnects between GPUs during execution (cf. Requirement I).
Implementation: For the implementation of Algorithm 1, we use the CUDA framework [29] and extend the block-based implementation of constraint-based CSL targeting a single GPU [8]. All data structures are allocated using *cudaHostRegister()* and data transfer is overlapped with kernel execution using *cudaMemcpyAsync()* using multiple CUDA streams per GPU. For the multi-GPU extension, we implement a centralized queue and consumer threads following existing concepts for load-balancing CSL on a multi-core CPU system [33].

5. Experiments

In this section, we experimentally evaluate our two multi-GPU adjacency searches of PC-stable (cf. Section 4) on an Intel-based multi-GPU system (cf. Figure 1). The system has two Intel ® Xeon ® Gold 6148 CPUs that are connected via Intel Ultra Path Interconnect (UPI). Each CPU is connected via a PCI-E Switch to two NVIDIA V100 GPUs. The system has four NVIDIA V100 GPUs in total. The GPUs are connected in a ring topology via NVLink 2.0, either using two lanes between GPUs $0-1$, $0-2$, $2-3$ or using a single lane between GPUs $1-3$. This leaves no direct connection between GPUs $0-3$, $1-2$. We refer to these three groups of GPUs in our study as, *direct two lanes*, *direct one lane*, and *indirect* respectively. The theoretical one-directional bandwidth is 25 GB/s for one NVLink 2.0 lane and 16 GB/s for PCI-E version 3. The system runs CUDA version 11.3 with driver version 465.19. Note that we set the tuning

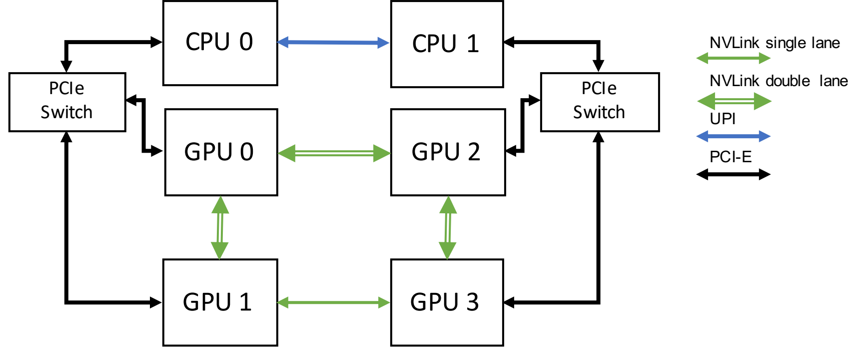

Figure 1: Interconnect topology of the Intel-based multi-GPU system considered in this work. Blue connections refer to UPI, black connections refer to PCI-E and green connections refer to NVLink 2.0 either single or double lanes.

parameter $\alpha = 0.01$ for all experiments and report median runtimes of multiple runs to avoid any hardware or operating system side effects.

In the experiments, we examine the performance of the kernels for different levels l, respectively the implication of the underlying memory accesses, sequential ($l = 0$) vs. random ($l \geq 1$). We focus on levels $l = 0$ and $l = 1$ according to existing work [10, 8]. The results for $l = 1$ are transferable to higher level [11]. We also compare the behavior of the two approaches with a varying number of GPUs and increasing size of variables, respectively, memory footprint. At last, we compare the runtimes on real-world gene expression data [7] extracted from TCGA [5].

Level Measurements The kernel for level $l = 0$ and $l \geq 1$ have a distinct difference. Due to an empty separation set $\mathbf{S}$ in level $l = 0$, the calculation of the p-value requires only a single access to Cor. Therefore, conducting multiple CI tests in parallel results in sequential and thus predictable access into the data structures. In contrast, in levels $l \geq 1$, exemplary $l = 1$, the calculation of the p-value requires access to multiple random locations within Cor, according to the pair of variables (V_i, V_j) and respective separation set $\mathbf{S}$. Thus, processing multiple pairs of variables and respectively CI tests in parallel results in partially random accesses. To understand the resulting implications for the UM and EM multi-GPU adjacency searches of PC-stable, we measure the runtimes on different data sizes, i.e., number of vertices N, using a varying number of GPUs g. Accordingly, in Table 1, we report the speed-up factor of the EM over the UM approach for the level $l = 0$ and $l = 1$ separately. For level $l = 0$, the UM approach is faster than the EM approach in almost all cases. The following two effects explain this result. On the one hand, in combination with prefetching mechanisms, the page migration engine can handle the sequential memory access pattern well in the UM approach. However, on the other hand, splitting the data into blocks and introducing a central queue in the EM approach introduce a non-negligible overhead. For level $l = 1$, the UM approach is faster when using a single or two GPUs for datasets with up to $40,000$ vertices. The EM approach is faster by factors of up to 2.6 for datasets with up to $40,000$ vertices and three or four GPUs, and faster by factors of up to 75.7 for larger datasets. The results for level $l = 1$ are explained through two effects. First, for small datasets and only two GPUs the GPU topology does not impact runtimes,

Table 1
Speed-up factor of explicit memory-managed (EM) over unified memory-based (UM) multi-GPU adjacency search for level $l = 0$ (top) and level $l = 1$ (bottom).

GPUs g \ Vertices N		5k	10k	20k	30k	40k	50k	60k	70k	80k
Level $l = 0$	1	0.3	0.3	0.2	0.3	0.4	0.4	0.4	0.4	0.5
	2	0.6	0.4	0.3	0.3	0.3	0.2	0.3	0.4	0.4
	3	1.0	0.9	0.7	0.6	0.6	0.6	0.4	0.7	0.8
	4	1.0	1.1	0.8	0.7	0.7	0.6	0.6	0.6	0.9
Level $l = 1$	1	0.6	0.6	0.6	0.7	0.7	20.2	43.8	60.0	73.8
	2	0.8	0.6	0.9	0.7	0.7	1.4	25.0	53.2	75.7
	3	2.6	1.9	1.8	2.3	2.6	3.0	15.4	45.7	70.8
	4	1.8	2.0	1.6	1.4	1.6	2.3	9.9	40.0	66.7

which becomes an issue for three and four GPUs, e.g., causing many remote data accesses and data migrations. For larger dataset sizes, i.e., above $40,000$ vertices, the memory of a single GPU is exceeded, and the number of page faults increases drastically, e.g., when running on a single GPU. Yet, the same effect is also visible when running on multiple GPUs, but occurs at a larger dataset size. In contrast, the EM approach does not suffer from these performance drops, as it is independent of GPU topology introduced costs, fulfilling Requirement I.

Scalability Measurements In Figure 2, we report the runtime in seconds for the UM and the EM GPU-accelerated adjacency search with a varying number of GPUs and an increasing number of variables. In contrast to the previous measurements, the presented absolute runtimes include both levels $l = 0, 1$. The vertical lines mark the limits of the GPU memory concerning the memory footprint of the data structures for a certain number of variables. Note that the first line from the left marks the limit of a single GPU, the second of two GPUs and so forth. Our results show that for specific numbers of variables, the UM approaches drop in performance. This behavior is a result of page faults occurring once the GPU memory is exceeded by several GB; compare measures for the UM approach on a single GPU. For the UM approach on multiple GPUs, the performance drop occurs for a larger number of variables and is not as severe. Note that exceeding the size of a single GPU does not cause the performance drop when executing on more than one GPU. Still, once the memory of a second GPU is exceeded, the drop in performance is visible even when running on three or four GPUs. We assume that this behavior results from inter- GPU communication, page faults, and page migrations between GPUs. For the EM approach running on multiple GPUs we cannot see any similar performance degradation, but the runtime scales quadratic to the number of variables, as expected. For a smaller number of variables, both approaches can result in better performances, e.g., if only two GPUs are considered, the UM approach is superior. In comparison, for three and four GPUs the EM approach is faster. We assume that the inter- GPU communication reduces performance.

Note, for the measures with 2 and 3 GPUs, we utilize the first GPUs $0, 1$ or $0, 1, 2$, as these combinations have the most direct two lane connections (cf. Figure 1. In a separate experiment, we compare the impact of the three connection groups (cf. Section 2) on the performance of

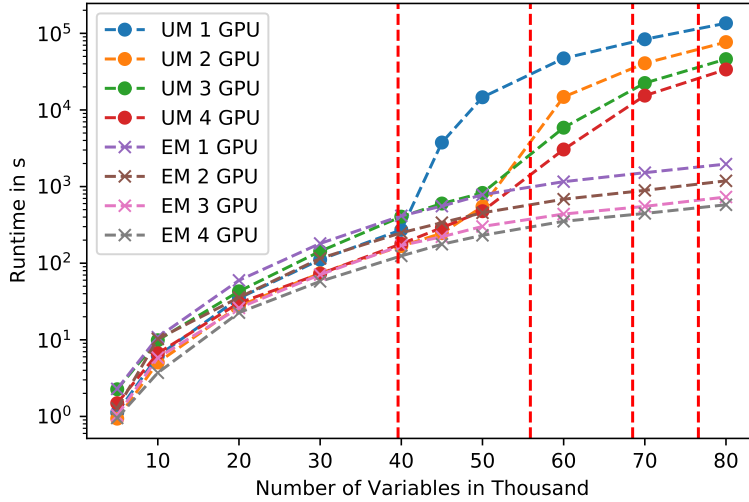

Figure 2: Comparing the unified memory-based (UM) to the explicit memory-managed (EM) GPU-accelerated adjacency search with a varying number of GPUs and increasing dataset size. Reporting total runtime for both levels $l = 0, 1$. The red dotted vertical lines mark the memory limit of one, two, three, and four GPUs (from left to right). Note, operations partially use triangular matrices.

the UM version to investigate shortcomings concerning Requirement I. We limit the execution to two GPUs and vary their combination. If the GPUs belong to the group *direct two lanes*, the highest bandwidth is available. Accordingly, the fastest runtime is achieved. The runtime is in average 12% faster than execution on GPUs from the *direct one lane* group, and 345% faster than execution on GPUs from the *indirect* group.

Measurements on Real-World Gene Expression Data In Table 2, we compare the runtimes of the UM and EM multi-GPU adjacency search on a real-world gene expression dataset from TCGA [5]. The dataset contains 55,572 variables with 3,189 observations [7]. For reference, we also report the runtime measurements from existing work [8] of the R-package pcalg [34] for the same dataset executed on a multi-core CPU. Execution of the adjacency search for the TCGA dataset on a multi-GPU system with four GPUs can result in speed-up of up to a factor of $1,253$ compared to execution on a multi-core CPU system. Furthermore, the EM version is the fastest, with a runtime of 5.15 minutes and a factor of 2.17 faster than the UM version.

Table 2
Runtimes of the unified memory-based (UM) and the explicit memory-managed (EM) GPU-accelerated adjacency search executed on four GPUs, and a parallel CPU implementation (measurement taken from [8]) for both levels $l = 0, 1$ on the TCGA dataset consisting of $55, 572$ variables.

CPU pcalg (on 32 cores)	GPU UM (GPU count $g = 4$)	GPU EM (GPU count $g = 4$, block size $bs = 2048$)
107.6 hours	11.17 minutes	5.15 minutes

6. Discussion and Future Work

In this work, we propose two multi-GPU approaches for the adjacency search of PC-stable to learn causal structures from high-dimensional data. The two approaches differ in their memory management, with one approach relying on the concept of UM and the other approach using explicit memory management. Through an experimental evaluation, we find that the explicit memory management version is better suited for the case of constraint-based CSL for larger datasets and a larger number of GPUs. The unified memory-based approach's runtime is negatively affected by the GPU interconnect topology and page faults due to irregular access patterns in the adjacency search of PC-stable.

Currently, our experiments are limited to separation sets of a maximum size of 1. We assume that more random accesses occur for larger separation sets and note that the explicitly memory-managed (EM) version should incorporate a pruning mechanism for blocks that are no longer required, i.e., lack enough adjacent vertices. Furthermore, we do not consider the impact of different adjacency structures, e.g., through reordering variables, in the context of load balancing between multiple GPUs in both multi-GPU strategies. Our study focuses on multivariate normal distributed data only and does not consider multi-GPU systems with hardware-supported cache coherence [18], e.g., IBM Power 9.

In future work, we plan to incorporate and evaluate our multi-GPU approaches for CI tests suited for other data, such as discrete case [9]. In this case, the execution strategy differs, i.e., mapping threads and threads blocks to processing tasks. Further, the CI tests require more global memory for auxiliary data structures and require significantly more accesses to global memory, as all observations are accessed. Hence, it remains to investigate if similar scaling behavior is achieved.

References

[1] J. Huegle, C. Hagedorn, M. Uflacker, How Causal Structural Knowledge Adds Decision-Support in Monitoring of Automotive Body Shop Assembly Lines, in: IJCAI, 2020. Demos.

[2] D. Malinsky, D. Danks, Causal Discovery Algorithms: A Practical Guide, Philosophy Compass (2018).

[3] A. Rau, F. Jaffrézic, G. Nuel, Joint estimation of causal effects from observational and intervention gene expression data, BMC systems biology (2013).

[4] P. Spirtes, C. Glymour, R. Scheines, Causation, Prediction, and Search, Second Edition, Adaptive Computation and Machine Learning, MIT Press, 2000.

[5] Cancer Genome Atlas Research Network, J. N. Weinstein, E. A. Collisson, G. B. Mills, K. R. Shaw, B. A. Ozenberger, K. Ellrott, I. Shmulevich, C. Sander, J. M. Stuart, The Cancer Genome Atlas Pan-Cancer analysis project, Nat Genet (2013).

[6] T. D. Le, T. Xu, L. Liu, H. Shu, T. Hoang, J. Li, ParallelPC: An R Package for Efficient Causal Exploration in Genomic Data, in: PAKDD, 2018.

[7] C. Perscheid, B. Grasnick, M. Uflacker, Integrative Gene Selection on Gene Expression Data: Providing Biological Context to Traditional Approaches, JIB (2018).

[8] C. Schmidt, J. Huegle, S. Horschig, M. Uflacker, Out-of-Core GPU-Accelerated Causal Structure Learning, in: ICA3PP, 2020.
[9] C. Hagedorn, J. Huegle, GPU-Accelerated Constraint-Based Causal Structure Learning for Discrete Data, in: SDM, 2021.
[10] C. Schmidt, J. Huegle, M. Uflacker, Order-independent Constraint-based Causal Structure Learning for Gaussian Distribution Models Using GPUs, in: SSDBM, 2018.
[11] B. Zarebavani, F. Jafarinejad, M. Hashemi, S. Salehkaleybar, cuPC: CUDA-Based parallel PC algorithm for causal structure learning on GPU, IEEE TPDS (2020).
[12] J. Pearl, Causality: Models, Reasoning, and Inference, Cambridge University Press, 2000.
[13] D. Heckerman, D. Geiger, D. M. Chickering, Learning Bayesian Networks: The Combination of Knowledge and Statistical Data, Machine Learning (1995).
[14] M. Maathuis, M. Drton, S. Lauritzen, M. Wainwright, Handbook of Graphical Models, 1st ed., CRC Press, Inc., 2018.
[15] D. Colombo, M. H. Maathuis, Order-Independent Constraint-Based Causal Structure Learning, JMLR (2014).
[16] A. P. Dawid, Conditional Independence in Statistical Theory, JRSSB (1979).
[17] C. Lutz, S. Breß, S. Zeuch, T. Rabl, V. Markl, Pump Up the Volume: Processing Large Data on GPUs with Fast Interconnects, in: SIGMOD, 2020.
[18] N. Sakharnykh, Memory Management on Modern GPU Architectures, 2019. URL: https://developer.nvidia.com/gtc/2019/video/s9727.
[19] S. A. Andersson, D. Madigan, M. D. Perlman, A Characterization of Markov Equivalence Classes for Acyclic Digraphs, Annals of Statistics (1997).
[20] D. M. Chickering, Learning Equivalence Classes of Bayesian-Network Structures, JMLR (2002).
[21] R. A. Fisher, Frequency Distribution of the Values of the Correlation Coefficient in Samples from an Indefinitely Large Population, Biometrika (1915).
[22] S. L. Lauritzen, Graphical Models, Clarendon Press Oxford, 1996.
[23] J. Whittaker, Graphical Models in Applied Multivariate Statistics, Wiley Publishing, 2009.
[24] A. Li, S. L. Song, J. Chen, J. Li, X. Liu, N. R. Tallent, K. J. Barker, Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect, IEEE TPDS (2020).
[25] T. Ben-Nun, E. Levy, A. Barak, E. Rubin, Memory Access Patterns: The Missing Piece of the Multi-GPU Puzzle, in: SC, 2015.
[26] J. Cabezas, L. Vilanova, I. Gelado, T. B. Jablin, N. Navarro, W. W. Hwu, Automatic Parallelization of Kernels in Shared-Memory Multi-GPU Nodes, in: ICS, 2015.
[27] D. Ganguly, Z. Zhang, J. Yang, R. Melhem, Adaptive Page Migration for Irregular Data-intensive Applications under GPU Memory Oversubscription, in: IEEE IPDPS, 2020.
[28] N. Sakharnykh, Beyond GPU Memory Limits with Unified Memory on Pascal, 2016. URL: https://devblogs.nvidia.com/beyond-gpu-memory-limits-unified-memory-pascal/.
[29] J. Nickolls, I. Buck, M. Garland, K. Skadron, Scalable Parallel Programming with CUDA, in: ACM SIGGRAPH 2008 Classes, 2008.
[30] C. Schmidt, J. Huegle, Towards a GPU-Accelerated Causal Inference, in: HPI Future SOC Lab - Proceedings 2017, 2020.
[31] NVIDIA Corporation, Cuda toolkit documentation, release: 11.3.1, 2021. URL: https://docs.nvidia.com/cuda/index.html.

[32] R. Landaverde, Tiansheng Zhang, A. K. Coskun, M. Herbordt, An investigation of Unified Memory Access performance in CUDA, in: IEEE HPEC, 2014.
[33] C. Schmidt, J. Huegle, P. Bode, M. Uflacker, Load-Balanced Parallel Constraint-Based Causal Structure Learning on Multi-Core Systems for High-Dimensional Data, in: PMLR, 2019.
[34] M. Kalisch, M. Mächler, D. Colombo, M. H. Maathuis, P. Bühlmann, Causal Inference Using Graphical Models with the R Package pcalg, JSS, Articles (2012).