ACCELERATEDLiNGAM: LEARNING CAUSAL DAGS AT THE SPEED OF GPUS

Victor Akinwande
Carnegie Mellon University

J. Zico Kolter
Carnegie Mellon University, Bosch Center for AI

ABSTRACT

Existing causal discovery methods based on combinatorial optimization or search are slow, prohibiting their application on large-scale datasets. In response, more recent methods attempt to address this limitation by formulating causal discovery as structure learning with continuous optimization but such approaches thus far provide no statistical guarantees. In this paper, we show that by efficiently parallelizing existing causal discovery methods, we can in fact scale them to thousands of dimensions, making them practical for substantially larger-scale problems. In particular, we parallelize the LiNGAM method, which is quadratic in the number of variables, obtaining up to a 32-fold speed-up on benchmark datasets when compared with existing sequential implementations. Specifically, we focus on the causal ordering sub-procedure in DirectLiNGAM and implement GPU kernels to accelerate it. This allows us to apply DirectLiNGAM to causal inference on large-scale gene expression data with genetic interventions yielding competitive results compared with specialized continuous optimization methods.

1 INTRODUCTION

Causal discovery or inference aims to learn causal interactions in a data-driven way (Pearl et al., 2000; Imbens & Rubin, 2015). However, most causal discovery methods need to establish the causal relationships between every pair of variables in the data, which results in a complexity that is at least quadratic in the number of variables (Glymour et al., 2019). For instance, DirectLiNGAM Shimizu et al. (2011); Hyvärinen & Smith (2013) works by recursively performing regression and conditional independence tests between pairs of variables to establish the causal ordering and then iteratively regressing each variable on other variables higher in the established causal order. Although DirectLiNGAM allows for full determination of the causal structure under minimal assumptions of linearity, non-Gaussian errors, acyclicity, and the absence of hidden confounders, it has a complexity of O(d^3) where d is the number of variables.

To address this scalability limitation, a recent trend in the literature has been to formulate heuristics combined with continuous optimization or deep learning methods to learn the underlying structure of the data (Lee et al., 2019; Lopez et al., 2022; Montagna et al., 2023). However, care must be taken when ascribing any causal interpretation to such methods. While these methods enhance computational efficiency, and allow us to apply causal discovery to large datasets, they rely on restrictive assumptions, are sensitive to hyper-parameters and more importantly, their identifiability guarantees are often not established. In-fact, as we show in the sections that follow, NOTEARS Zheng et al. (2018) - a widely used method cannot recover the underlying structure in a simple causal DAG. As such, there is a compelling need to explore other avenues to achieve scalability of causal discovery methods.

In this paper, we describe an implementation of the LiNGAM analysis, which is accelerated by parallelization on consumer GPUs. We focus on the DirectLiNGAM and VarLiNGAM methods, but the ideas presented are easily applicable to other LiNGAM variants. Parallelization leads to a **32-fold** speed-up over the sequential version, enabling us to apply these methods to large-scale datasets. We apply DirectLiNGAM to gene expression data with genetic interventions ($d \approx 1000$). We observe that there is still a need to explore improved ways to speed up LiNGAM on GPUs, such as better I/O awareness and increasing the proportion of the algorithm that is parallelized. Our implementation is open-sourced on GitHub[1].

2 Background

We provide a brief overview of Functional Causal Models based methods for learning causal DAGs, we discuss the GPU execution model and how it is amenable to such methods, and we present the basic LiNGAM analysis and discuss recent methods in the literature that seek to learn DAGs using heuristics and continuous optimization. See Appendix section A.1 for introduction on causal DAGs.

2.1 Causal discovery based on Functional Causal Models (FCM)

FCMs represent a causal effect Y as a function of direct causes X, noise ϵ, and parameter sets θ, assuming that the transformation from causes to effects is invertible e.g Eqn. 1. For instance, in the linear, non-Gaussian acyclic model (LiNGAM) Shimizu et al. (2006), the causal direction can be determined when at most one of the noise term or cause is Gaussian. The post-nonlinear (PNL) causal model extends this by considering nonlinear effects such as sensor or measurement errors, and is identifiable in most cases Zhang & Hyvarinen (2012). FCMs provide a flexible framework for causal discovery by capturing the causal asymmetry in the data-generating process and accommodates various data distributions. When the exact functional form of the data-generating process is unknown or in cases with discrete variables and small cardinality, properties of the causal process may be obscured and precise identification of causal directions remains challenging.

$$Y = f(X, \epsilon; \theta_1) \tag{1}$$

2.2 Standard LiNGAM implementation

LiNGAM Shimizu et al. (2006) is characterized by properties that enable identification of causal relationships between variables. The observed variables x_i, $i \in \{i, \dots m\}$ in the dataset can be arranged in a causal order denoted by $k(i)$. This order implies that no later variable in the sequence causes any earlier variable, forming a recursive structure that can be represented as a DAG. This property also ensures there are no cyclic relationships between the variables. In addition, the value of x_i is determined linearly by the values of the variables before it in the causal order i.e. $x_i = \sum_{k(j)<k(i)} \theta_{ij} x_j + \epsilon_i + c_i$ where θ_{ij} represent the strength of the causal effect of x_j on x_i, ϵ_i is the noise term and c_i is an optional constant term. The final property is that the noise terms ϵ_i are mutually independent, continuous random variables with non-Gaussian distributions.

Constrained by these assumptions on the data-generating process, the independence and non-Gaussianity of ϵ provides valuable information needed to establish the causal direction say given a pair of variables. This is because we can easily identify independence between the cause and the error terms. As such, any causal model that is inconsistent with this identification is discarded. We illustrate this principle in Figure 3 (Appendix). Given data generated according a LiNGAM functional causal model as in Eqn. 1, the regression residual can only be independent of the independent variable in the correct causal direction.

In practice, datasets consist of more than two variables, so we need a general method for estimating the causal order for datasets of arbitrary dimensions. In Shimizu et al. (2011), DirectLiNGAM is presented as a method to achieve this. DirectLiNGAM begins by identifying a variable not caused by any other variable (termed exogenous) in the data through its independence from the residuals of multiple pairwise regressions. The effects of this variable are then removed from the other variables using least squares regression. Shimizu et al. (2011) showed that a LiNGAM structure also holds for the residuals of the regression, meaning that the residuals themselves can be treated as variables in a LiNGAM model. In addition, the causal ordering of the residuals corresponds to the causal ordering of the original observed variables. DirectLiNGAM therefore iteratively applies the same independence and removal steps to the residuals, finding the "exogenous" residual at each iteration. By repeatedly identifying and removing the effects of exogenous variables (or residuals), the model

[1] `https://github.com/Viktour19/culingam`

Algorithm 1 Causal ordering pseudo-implementation.

```
# X      : dataset [m, dim]
# U      : indices of variables [dim]
# _residual: computes the regression residual
# _diff_mutual_info: computes the difference of the mutual information

def search_causal_order(X, U):
k_list = np.zeros(len(U))
for i in U:
    k = 0
    for j in U:
        if i != j:
            xi = X[:, i]
            xj = X[:, j]
            xi_std = (xi - np.mean(xi)) / np.std(xi)
            xj_std = (xj - np.mean(xj)) / np.std(xj)

            ri_j = _residual(xi_std, xj_std)
            rj_i = _residual(xj_std, xi_std)

            mi_diff = _diff_mutual_info(xi_std, xj_std, ri_j, rj_i)
            k += np.minimum(0, mi_diff) ** 2

    k_list[i] = -k
return U[np.argmax(k_list)]
```

estimates the causal order of the original variables. We show the implementation to identify the variable at position $k(i)$ in Algorithm 1.

2.3 GPU EXECUTION MODEL

The causal ordering procedure (Algorithm 1) is the main computational bottleneck of the DirectLiNGAM algorithm, accounting for up to 96% of the overall wall-clock time (see Figure 1). The pseudo-implementation of the procedure also makes the limitation of DirectLiNGAM, in practice, glaring. The algorithm needs to compute statistical measures in an inner loop for each pair of variables in the data. Such computational structure results in a complexity quadratic in the number of variables making it difficult to apply on large-scale datasets. DirectLiNGAM takes 7 hours to process a dataset with 1 million observations and 100 variables on a high-performance AMD EPYC server CPU (see Figure 1).

On a second look, we observe that each variables pair computation is independent of the others. This independence is a key characteristic that makes an algorithm suitable for parallel processing. In-fact, based on this, successful attempts to parallelize the algorithm on a super-computer have been made (Matsuda et al., 2022). However, GPUs present a more accessible approach and have not be explored. GPUs can perform the same operation on multiple data-points simultaneously (vectorization), and also handle high arithmetic intensity computations. Operations involving accumulations can also be efficiently parallelized on GPUs by using techniques like parallel reduction, where the work is divided among multiple threads and then combined to get the final result. This implies vectorized computations like the mean and standard deviation can be done much faster and computation of residuals and mutual information differences for different pairs can be done in parallel batches. Finally, recent work Shahbazinia et al. (2023) has demonstrated that LiNGAM is amenable to GPU acceleration, although the authors propose using a heuristic to prune the search procedure. This effectively modifies the algorithm, and the implementation is not available for us to benchmark.

2.4 CONTINUOUS OPTIMIZATION BASED STRUCTURE LEARNING

Learning DAGs with GPUs is certainly not a new idea. Continuous optimization methods for learning DAGs are amenable to acceleration on GPUs using packages like *PyTorch* or *Tensorflow*. Algorithms such as NOTEARS Zheng et al. (2018) and GOLEM Ng et al. (2020) simultaneously optimize over the DAGs structure and its parameters by defining a differentiable acyclicity constraint and enabling end-to-end optimization of a score function over graph adjacency matrices. NOTEARS minimizes the MSE between the observations and the model predictions:

$$\text{MSE}_X(\theta) = \frac{1}{m}\|X - X\theta\|_F^2$$

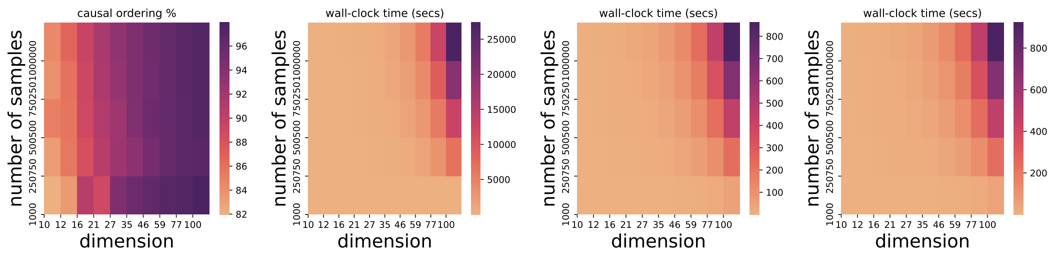

Figure 1: Benchmark of CPU (sequential) implementation of DirectLiNGAM. Given data with specified number of samples and dimensions, the causal ordering sub-procedure accounts for up to 96% of overall runtime **(left)**. It takes 7 hours on a CPU to process a dataset of 1 million samples with 100 variables **(second-left)**. Benchmark of GPU (parallel) implementation of DirectLiNGAM **(second-right)** and VarLiNGAM **(right)**. Given data with specified number of samples and dimensions, the parallel implementation achieves up to 32 times speed-up when compared to the sequential implementation. The benchmark is obtained using an NVIDIA RTX 6000 Ada with 18 176 cores.

where $\|\cdot\|_F$ denotes the Frobenius norm. NOTEARS includes a constraint defined with a trace exponential function that equals zero if and only if θ represents an acyclic graph, and a penalty term $\lambda\|\theta\|_1$ where $\lambda\|.\|_1$ is defined element-wise and λ is a hyper-parameter.

GOLEM performs Maximum Likelihood Estimation (MLE) under the assumption of Gaussian noise terms in the data, and includes both soft acyclicity and sparsity constraints.

Unfortunately, both methods make restrictive assumptions about the data-generating process, such as equal noise variance across observations. Furthermore, the assumption that the marginal variance of each variable is strictly larger than that of its ancestors in the DAG—a condition termed *varsortability*—has been shown to be a crucial property of the assumed data-generating process (Reisach et al., 2021; Ng et al., 2023). The non-convex nature of the optimization problem in GOLEM often necessitates careful initialization and sophisticated optimization strategies to ensure convergence to a meaningful solution. Moreover, neither NOTEARS nor GOLEM provides identifiability guarantees; they may not perform reliably on simulated datasets where the true underlying structure is known (see Section 3.1), and other such methods may not converge (Lopez et al., 2022). Furthermore, the effectiveness of these algorithms is highly sensitive to the choice of hyper-parameters, such as the sparsity threshold or the specific loss function employed. Selecting these hyper-parameters is non-trivial and may require extensive cross-validation or domain expertise, potentially limiting the practicality of these methods.

3 AcceleratedLiNGAM: Analysis, and Extensions

In this section, we discuss the considerations that make acceleration of Algorithm 1 efficient on GPUs. First, note that the outer loop over i and the inner loop over j are independent and thus can be parallelized. The dependency on the results of _residual and _diff_mutual_info for each pair (i, j) however, necessitates synchronization and memory management. Therefore, we parallelize over i, each block handles a different i value, computing k_list$[i]$ and within each i, we parallelize over j. We also ensure that within the inner loop, operations are ordered using GPU abstractions.

This parallelization scheme requires up to $dim * (dim - 1)$ cores. Profiling the sequential implementation (Figure 1), Amdahl's law suggests a theoretical speed-up of 25. In the limit of infinite processors, $speedup = 1/(1 - p)$, where p is the parallelizable portion of the algorithm (0.96 in our case) but this does not account for the increase in workload size with the number of processors (Gustafson, 1988). Finally, we do not need synchronization for updating k_list$[i]$ since the update order does not matter.

3.1 Efficient DirectLiNGAM implementation

We implemented AcceleratedLiNGAM on an NVIDIA RTX 6000 Ada. The implementation optimizes for memory use by performing parallel reductions in shared memory. Shared memory refers to a fast on-chip memory space shared among the threads of a block in the GPU. The results show a 32-fold speed-up of DirectLiNGAM when compared with the sequential implementation (see Figure 1). The basic LiNGAM analysis can be extended to auto-regressive modeling such as VarLINGAM Hyvärinen et al. (2010). See Appendix A.3 for extension of AcceleratedLiNGAM to VarLINGAM.

To validate that there are no logical errors in our parallel implementation, we compare the results of applying the sequential implementation with those of the parallel implementation on simulated data. We report the F1 score, recall, and structural hamming distance (SHD) over 50 simulations (different random seed) in Figure 2.

Let $G = (V, E)$ be a DAG where V is a set of vertices representing variables and E is a set of directed edges representing causal relationships between variables. Vertices are connected such that each vertex v_i at level l may have parents from the set of nodes at level $l-1$: $\forall v_i \in V, \exists l_i \in \{0,1\}$ such that $(v_j, v_i) \in E \Rightarrow l_j = l_i - 1$. Given G, we generate data such that the strength of the causal effect $\theta \sim \mathcal{N}(0,1)$, and the noise terms $\epsilon_i \sim \text{Uniform}(0,1), \quad \text{for } i = 1, 2, ..., m$. Comparison of the sequential and parallel implementation of DirectLiNGAM on this simulated data show that they produce the exact same result, and recover the true causal graph accurately (see Figure 2).

We evaluate NOTEARS on similarly simulated data selecting the best performance across a grid: $\{0.001, 0.005, 0.01, 0.05, 0.1\}$ of λ values. We obtain an F1 score of 0.79 ± 0.2, Recall of 0.69 ± 0.2 and SHD of 2.52 ± 1.67. This shows that even on data where the causal influences are simple, NOTEARS does not perform well.

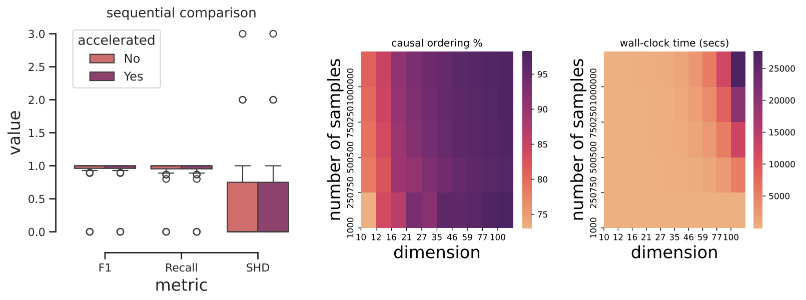

Figure 2: Comparison of parallel and sequential implementation of DirectLiNGAM. We simulate data according to a linear FCM with $10\,000$ samples, and 10 variables. Both implementations produce the exact same result **(left)**. Benchmark of CPU (sequential) implementation of VarLiNGAM. Given data with specified number of samples and dimensions, the causal ordering sub-procedure of DirectLiNGAM accounts for up to 96% of overall runtime **(middle)**. It takes 7 hours on a CPU to process a dataset of 1 million samples with 100 variables **(right)**.

4 Experiments

4.1 AcceleratedLiNGAM to gene expression data with genetic interventions

We experiment with the causal learning of gene regulatory networks from gene expression data, with genetic interventions, as detailed in Friedman et al. (2000); Pe er et al. (2001); Lopez et al. (2022). This approach is enabled by a single-cell RNA sequencing technique known as Perturb-Seq, as described by Dixit et al. (2016) . Perturb-Seq allows for targeted genetic interventions and the subsequent measurement of their effects on the complete gene expression profiles in hundreds of thousands of individual cells using single-cell RNA-seq.

Our experimental setup mirror that of Lopez et al. (2022), where the Perturb-CITE-seq dataset Frangieh et al. (2021) contains expression profiles from $218\,331$ melanoma (cancer) cells, after interventions targeting each of 249 genes. For each gene in the genome, measurement from a single-cell combines the identity of the intervention (the target gene) along with a count vector, that is the expression level of a particular gene. The dataset includes patient-derived melanoma cells with the same genetic interventions but exposed to three conditions: co-culture with T cells derived from the patient's tumor (73 114 cells) (these can recognize and kill melanoma cells), interferon (IFN)-γ treatment (87 590 cells) and control (57 627 cells). We retain cells from 20% of the interventions as a test set. The dimensions (samples, dim) of the train set for co-culture, IFN, and control datasets are (65 164, 964), (75 443, 964), and (50 539, 961) respectively.

We applied AcceleratedLiNGAM to each of the three datasets. Since LiNGAM analysis does not involve causal inference, after obtaining the weighted adjacency matrix, we apply standard Variational Inference (VI) methods to obtain both the interventional NLL (I-NLL) and the mean absolute error (I-MAE) across held-out interventions. Specifically, we use Stein VI Liu & Wang (2016) implemented in the *Pyro* package where we defined a model such that variables without direct causal influence on another variable are leaf nodes and otherwise are latent nodes with priors $\sim \mathcal{N}(0, 1)$. We generate 200 posterior samples, and optimize for 5000 iterations. DirectLiNGAM itself is not sensitive to the random seed.

On a high-level, Stein VI involves approximating the target distribution $p(x)$ by iteratively updating a set of particles. This process minimizes the KL divergence between the approximating distribution $q(x)$ and the target $p(x)$, which is achieved through the application of smooth transforms. The key insight is to apply a perturbation to the identity map, $T(x) = x + \epsilon\phi(x)$, where $\phi(x)$ is a smooth function that characterizes the perturbation direction, and ϵ is a small scalar representing the perturbation magnitude. This approach ensures T is an injective (one-to-one) map, maintaining the full rank of the Jacobian of T, and thereby guaranteeing the invertibility required for the change of variables formula to hold.

Table 1: Comparison of DirectLiNGAM with VI and DCD-FG method on the Perturb-CITE-seq datasets. We obtain the I-NLL (nll) and I-MAE (mae) on all three datasets. Lower values are better.

	Co-culture		**IFN**		**Control**	
	nll	mae	nll	mae	nll	mae
DirectLiNGAM	1.5	0.7	1.5	0.9	3	1.6
DCD-FG ($\approx$)	1.1	0.7	1.2	0.7	1.1	0.7

We compare the results from DirectLiNGAM, combined with VI, with those obtained using DCD-FG introduced in Lopez et al. (2022). DCD-FG works by combining a parameterized distribution over factor directed graphs with a hybrid likelihood model, optimized with an acyclicity constraint and was applied to the Perturb-CITE-seq datasets. We observe the I-MAE to be lower or about the same on the co-culture dataset (one leaf variable), and slightly higher on the IFN and control datasets (one and two leaf variables respectively) (see Table 1. We note DCD-FG is a continuous optimization based structure learning method and prone to many of the issues we have previously discussed but more pertinently, the results presented in Lopez et al. (2022) do have quite a bit of variance. We also observe the I-NLL of DirectLiNGAM to be slightly higher on all datasets. While this may seem like worse performance, it is interesting to note how the control dataset (i.e., no interventions) has the same I-NLL as the other two datasets with DCD-FG. However, with DirectLiNGAM, the I-NLL of the co-culture and IFN datasets are similar, but the control I-NLL is much higher. Since there is no ground truth data, it is difficult to determine if DCD-FG is overfitting but it seems likely.

5 Conclusion

Machine learning practitioners often encounter use-cases where predictive performance alone does not suffice. For instance, in healthcare, accurately predicting patient outcomes is crucial, but understanding the causal factors behind diseases can lead to more effective treatments and health policies (Räisänen et al., 2006; Barros et al., 2022). Similarly, in genomics, identifying the causal relationships between genetic markers and diseases is vital for developing targeted therapies and personalized medicine approaches (Burgess et al., 2018; Lopez et al., 2022). In all these domains, the focus is not solely on prediction but also on uncovering the underlying causal mechanisms. By addressing the scalability limitations of causal discovery methods with statistical guarantees, we aim to enable the widespread application of causal inference in large-scale data analysis.

References

Vesna Barros, Itay Manes, Victor Akinwande, Celia Cintas, Osnat Bar-Shira, Michal Ozery-Flato, Yishai Shimoni, and Michal Rosen-Zvi. A causal inference approach for estimating effects of non-pharmaceutical interventions during covid-19 pandemic. *Plos one*, 17(9):e0265289, 2022.

Stephen Burgess, Christopher N Foley, and Verena Zuber. Inferring causal relationships between risk factors and outcomes from genome-wide association study data. *Annual review of genomics and human genetics*, 19:303–327, 2018.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Atray Dixit, Oren Parnas, Biyu Li, Jenny Chen, Charles P Fulco, Livnat Jerby-Arnon, Nemanja D Marjanovic, Danielle Dionne, Tyler Burks, Raktima Raychowdhury, et al. Perturb-seq: dissecting molecular circuits with scalable single-cell rna profiling of pooled genetic screens. *cell*, 167(7): 1853–1866, 2016.

Chris J Frangieh, Johannes C Melms, Pratiksha I Thakore, Kathryn R Geiger-Schuller, Patricia Ho, Adrienne M Luoma, Brian Cleary, Livnat Jerby-Arnon, Shruti Malu, Michael S Cuoco, et al. Multimodal pooled perturb-cite-seq screens in patient models define mechanisms of cancer immune evasion. *Nature genetics*, 53(3):332–341, 2021.

Nir Friedman, Michal Linial, Iftach Nachman, and Dana Pe'er. Using bayesian networks to analyze expression data. In *Proceedings of the fourth annual international conference on Computational molecular biology*, pp. 127–135, 2000.

Daniel Y Fu, Hermann Kumbong, Eric Nguyen, and Christopher Ré. Flashfftconv: Efficient convolutions for long sequences with tensor cores. *arXiv preprint arXiv:2311.05908*, 2023.

Clark Glymour, Kun Zhang, and Peter Spirtes. Review of causal discovery methods based on graphical models. *Frontiers in genetics*, 10:524, 2019.

John L Gustafson. Reevaluating amdahl's law. *Communications of the ACM*, 31(5):532–533, 1988.

Aapo Hyvärinen and Stephen M Smith. Pairwise likelihood ratios for estimation of non-gaussian structural equation models. *The Journal of Machine Learning Research*, 14(1):111–152, 2013.

Aapo Hyvärinen, Kun Zhang, Shohei Shimizu, and Patrik O Hoyer. Estimation of a structural vector autoregression model using non-gaussianity. *Journal of Machine Learning Research*, 11(5), 2010.

Guido W Imbens and Donald B Rubin. *Causal inference in statistics, social, and biomedical sciences*. Cambridge University Press, 2015.

Hao-Chih Lee, Matteo Danieletto, Riccardo Miotto, Sarah T Cherng, and Joel T Dudley. Scaling structural learning with no-bears to infer causal transcriptome networks. In *PACIFIC SYMPOSIUM ON BIOCOMPUTING 2020*, pp. 391–402. World Scientific, 2019.

Qiang Liu and Dilin Wang. Stein variational gradient descent: A general purpose bayesian inference algorithm. *Advances in neural information processing systems*, 29, 2016.

Romain Lopez, Jan-Christian Hütter, Jonathan K. Pritchard, and Aviv Regev. Large-scale differentiable causal discovery of factor graphs. In *Advances in Neural Information Processing Systems*, 2022.

Kazuhito Matsuda, Kouji Kurihara, Kentaro Kawakami, Masafumi Yamazaki, Fuyuka Yamada, Tsuguchika Tabaru, and Ken Yokoyama. Accelerating lingam causal discovery with massive parallel execution on supercomputer fugaku. *IEICE TRANSACTIONS on Information and Systems*, 105(12):2032–2039, 2022.

Francesco Montagna, Nicoletta Noceti, Lorenzo Rosasco, Kun Zhang, and Francesco Locatello. Scalable causal discovery with score matching. *arXiv preprint arXiv:2304.03382*, 2023.

Ignavier Ng, AmirEmad Ghassami, and Kun Zhang. On the role of sparsity and dag constraints for learning linear dags. *Advances in Neural Information Processing Systems*, 33:17943–17954, 2020.

Ignavier Ng, Biwei Huang, and Kun Zhang. Structure learning with continuous optimization: A sober look and beyond. *arXiv preprint arXiv:2304.02146*, 2023.

Dana Pe er, Aviv Regev, Gal Elidan, and Nir Friedman. Inferring subnetworks from perturbed expression profiles. *BIOINFORMATICS-OXFORD-*, 17:S215–S224, 2001.

Judea Pearl et al. Models, reasoning and inference. *Cambridge, UK: CambridgeUniversityPress*, 19(2):3, 2000.

Ulla Räisänen, Marie-Jet Bekkers, Paula Boddington, Srikant Sarangi, and Angus Clarke. The causation of disease–the practical and ethical consequences of competing explanations. *Medicine, Health Care and Philosophy*, 9:293–306, 2006.

Alexander Reisach, Christof Seiler, and Sebastian Weichwald. Beware of the simulated dag! causal discovery benchmarks may be easy to game. *Advances in Neural Information Processing Systems*, 34:27772–27784, 2021.

Amirhossein Shahbazinia, Saber Salehkaleybar, and Matin Hashemi. Paralingam: Parallel causal structure learning for linear non-gaussian acyclic models. *Journal of Parallel and Distributed Computing*, 176:114–127, 2023.

Shohei Shimizu, Patrik O Hoyer, Aapo Hyvärinen, Antti Kerminen, and Michael Jordan. A linear non-gaussian acyclic model for causal discovery. *Journal of Machine Learning Research*, 7(10), 2006.

Shohei Shimizu, Takanori Inazumi, Yasuhiro Sogawa, Aapo Hyvarinen, Yoshinobu Kawahara, Takashi Washio, Patrik O Hoyer, Kenneth Bollen, and Patrik Hoyer. Directlingam: A direct method for learning a linear non-gaussian structural equation model. *Journal of Machine Learning Research-JMLR*, 12(Apr):1225–1248, 2011.

Kun Zhang and Aapo Hyvarinen. On the identifiability of the post-nonlinear causal model. *arXiv preprint arXiv:1205.2599*, 2012.

Xun Zheng, Bryon Aragam, Pradeep K Ravikumar, and Eric P Xing. Dags with no tears: Continuous optimization for structure learning. *Advances in neural information processing systems*, 31, 2018.

A Appendix

A.1 Causal Directed Acyclic Graphs

Causal Directed Acyclic Graphs (DAGs) are a fundamental structure in causal inference and graphical models. A causal DAG consists of nodes, each representing a random variable. These variables can be anything of interest, such as different attributes, events, or states. The edges in a causal DAG are acyclic, directed and represent causal relationships between the variables. An edge from variable X_i to X_j implies X_i has a direct causal influence on X_j. Similarly, the absence of a direct edge from X_i to X_j indicates that, according to the model, X_i does not have a direct causal effect on X_j given the other variables and edges in the model. Causal DAGs encode the assumptions about the conditional independence of the variables meaning the joint probability distribution over all possible values of the variables can be factorized into a product of conditional distributions given by:

$$P(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} P(X_i | \text{Parents}(X_i))$$

A.2 Causal asymmetry principle

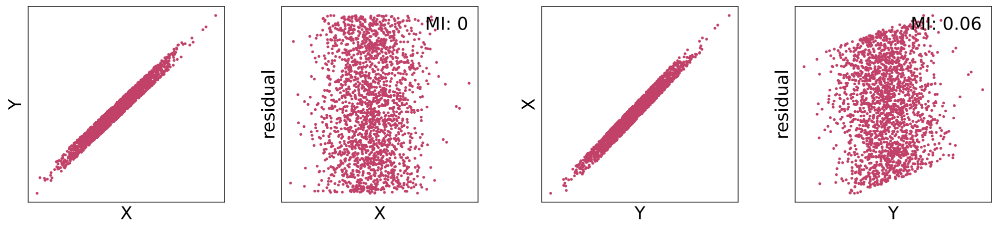

Figure 3: Illustration of the causal asymmetry principle underpinning LiNGAM. Given data generated according a LiNGAM functional causal model as in Eqn. 1, the regression residual can only be independent of the independent variable in the correct causal direction (top figure). This holds for any distribution of the noise except Gaussian. Independence is measured using the Mutual Information (MI).

A.3 Extension: Efficient VarLiNGAM implementation

The basic LiNGAM analysis can be extended to auto-regressive modeling. In Hyvärinen et al. (2010), VarLiNGAM is introduced. The key idea is to combine FCMs and vector autoregressive (VAR) models to capture both instantaneous and lagged influences among multiple time series. The model is expressed as: $x(t) = \sum_{\tau=0}^{k} \theta_\tau x(t-\tau) + \epsilon(t)$ where $x(t)$ is the observed time series at time t, θ_τ is the matrix of causal effects with time lag τ and $\epsilon(t)$ is the noise term (innovation). Similar to all LiNGAM analyses, the model assumes the noise terms are mutually independent, non-Gaussian, and the matrix θ_τ corresponding to instantaneous effects forms an acyclic graph.

On a high-level, VarLiNGAM works by first estimating coefficients of the VAR model using standard auto-regressive modeling techniques, and then transforms the estimated VAR coefficients based on the causal adjacency matrix estimated using DirectLiNGAM thereby accounting for the direct causal relationships while isolating the indirect effects captured by the VAR coefficients. We refer the reader to Hyvärinen et al. (2010) for more details. The key thing to note here is the same algorithm in Algorithm 1 dominates the runtime of VarLiNGAM (See Figure 2) and so we obtain a similar speed-up of 30 with the GPU implementation.

A.4 Low level CUDA implementation details

All the benchmark results are run on an NVIDIA A6000 Ada GPU with 48 GB of memory, 18 176 cores and 91.1 TFLOPS single-precision performance. The register file size is 64K 32-bit registers per Streaming Multiprocessor (SM), the maximum number of registers per thread is 255, the maximum number of thread blocks per SM is 16, the shared memory capacity per SM is 100 KB, and the maximum shared memory per thread block is 99 KB.

Our parallelization scheme requires up to $dim * (dim - 1)$ cores (see Algorithm 1). Profiling the sequential implementation (Figure 1), Amdahl's law suggests a theoretical speed-up of 25. In the limit of infinite processors, $speedup = 1/(1 - p)$, where p is the parallelizable portion of the algorithm (0.96 in our case) but this does not account for the increase in workload size with the number of processors (Gustafson, 1988). Finally, we do not need synchronization for updating k_list$[i]$ since the update order does not matter.

A key optimization is how the kernel uses shared memory for intermediate reduction results within each block. This pattern is sensitive to the number of threads because it assumes that the shared memory array is fully populated, and after storing in shared memory, we perform a reduction within the block. We set shared memory per thread block to 96 KB. We implement warp tiling (for optimal latency, and efficient synchronization) and notice a 20% speed-up but due to the non-associative nature of floating-point operations, naive implementation of parallel reduction may lead to rounding errors which we observe in the residual computation. As such, we leave this optimization for future work.

Finally, optimizing GPU implementations of machine learning methods for I/O awareness has achieved significant success recently, especially when combined with the use of cores optimized for fast matrix multiplication (Tensor cores) (Dao et al., 2022; Fu et al., 2023). A profile of our GPU kernels reveals that the two most time-consuming operations are *poll* and *pthread_cond_timedwait*, both accounting for about 50% of the execution time. These operations are associated with waiting for I/O operations or synchronization primitives, indicating substantial room for optimization to enhance I/O awareness, and efficient synchronization. The remainder of the DirectLiNGAM and VarLiNGAM implementations, which are not parallelized, involve several regression analysis. Although, we utilize heavily optimized libraries like *numpy* and *scikit-learn* for these regressions, there remains potential for speed-up through parallelism with Tensor cores.