FPGA-Accelerated Causal Discovery with Conditional Independence Test Prioritization

Ce Guo*, Diego Cupello*, Wayne Luk*, Joshua Levine†, Alexander Warren† and Peter Brookes†
*Imperial College London, United Kingdom
{c.guo, diego.cupello20, w.luk}@imperial.ac.uk
†Intel, United Kingdom
{joshua.levine, alexander.warren, peter.brookes}@intel.com

***Abstract*—Causal discovery is a data mining approach that finds causal relations between variables from data. Causal discovery algorithms are computationally demanding when the data set has a high dimensionality or a large sample size. A promising way to expedite causal discovery is by utilizing FPGAs, but a significant drawback is that FPGA designs become inefficient when the on-chip memory cannot store the entire data set. This paper proposes Conditional Independence Test Prioritization (CITP), a novel approach that overcomes this limitation and enables fast FPGA-based causal discovery for large datasets with comparable speed and adequate accuracy to state-of-the-art methods. The main idea behind CITP is to design a workflow that allows a small subset of data to be stored in on-chip memory for prioritizing conditional independence tests. The paper provides experimental results that demonstrate the effectiveness of CITP in terms of both accuracy and speed. Our experiments show that for specific datasets, the proposed approach can respectively be 79 times, 2.6 times and 2.1 times faster than current CPU, GPU and FPGA designs.**

I. Introduction

For a long time, scientists and engineers have struggled with understanding how random variables are related to each other. To tackle this problem, a method called causal discovery has been developed, which uses data to identify causal relationships between variables. Causal discovery is a crucial means for data analysis and knowledge discovery, and it has been applied to a variety of fields, including biomedical science and economics. However, the process of causal discovery is computationally expensive. For instance, pcalg [1], a well-known software tool for causal discovery that operates on CPUs, takes up to three days to complete the causal discovery process for a gene expression dataset with 1643 dimensions on an 8-core Intel Xeon CPU running at 2.5GHz [2].

This paper focuses on constraint-based causal discovery methods where the most computationally expensive calculation is the conditional independence (CI) test. To accelerate this calculation, existing approaches utilize multi-core CPUs and GPUs to enumerate and execute CI tests in parallel. While this approach has been successful with CPUs and GPUs, it is challenging to apply a similar approach to FPGAs due to difficulties in efficient on-chip CI test execution. An FPGA-accelerated solution for causal discovery is Conditioning Set Filtering (CSF) [3]. Instead of speeding up CI tests, CSF runs a computationally expensive procedure on the FPGA to reduce the number of CI tests so that the remaining tests no longer create a speed bottleneck. However, CSF cannot function if the on-chip memory capacity of the FPGA is insufficient to hold the entire data matrix.

This paper proposes Conditional Independence Test Prioritization (CITP), a novel FPGA-based causal discovery approach. Unlike CSF, CITP has a critical feature that the data size is not limited by the on-chip memory capacity. Our main contributions include the following:

1) An acceleration strategy for CITP including a subgraph selection method, a priority score and an optimized CPU–FPGA collaboration workflow (Section III).
2) Design and implementation of CITP including a hardware block for priority score evaluation and software for subgraph selection (Section IV).
3) A comparative evaluation of CITP against state-of-the art causal discovery methods on CPUs, GPUs and FPGAs regarding accuracy and speed (Section V).

II. Background

This section provides a brief overview of the problem settings, commonly used algorithms, and acceleration methods for causal discovery.

A. Causal Discovery

A causal discovery algorithm is used to extract causal relations between random variables using data. Given an $N \times D$ data matrix containing N data points sampled from D variables, the algorithm extracts causal relations between the D variables and encodes them into a directed acyclic graph with D nodes.

This paper focuses on a type of causal discovery methods, constraint-based methods, because they offer advantages in terms of quality, speed, and ease of modularization [4], [5], [6]. The Peter-Clark (PC) algorithm and its variants [7] are representative techniques for constrained-based causal discovery. These techniques work in two stages. The first stage is skeleton discovery, where the algorithm builds an undirected causal graph from the data. The second stage is direction discovery, where the algorithm derives the direction of edges.

The skeleton discovery stage takes up most of the execution time. In this stage, the algorithm iteratively removes edges from a graph. At the beginning of each iteration, a CI test

enumerator produces a CI test $\mathcal{I}(v_a, v_b|C)$, which is a test indicating whether random variables v_a and v_b are conditionally independent given a set of random variables C. The random variable set C is usually referred to as the conditioning set [8] or the conditional set [9]. Then, a CI test executor runs the test using data to obtain a Boolean conclusion as to whether v_a and v_b are conditionally independent given C. If so, the edge remover deletes the edge (v_a, v_b) from the graph.

B. Accelerated Causal Discovery

Constraint-based techniques for discovering causal relationships are typically computationally expensive because of the large number of conditional independence (CI) tests involved. For example, the Munin causal graph [10] has 1041 nodes. The maximum number of CI tests executed during causal discovery is 1.012×10^{14} assuming that the maximum size of the conditioning set is 4. To speed up this process, a common and straightforward approach is to perform CI tests in parallel. For instance, ParallelPC [11], [12] is the first method to achieve parallel causal discovery, where CI tests with the same conditioning set size are run simultaneously on multiple CPU cores. Similarly, cuPC [2] utilizes multiple GPU cores to map the CI test generator and executors, optimizing memory access. gpuPC [13] also follows a similar approach to cuPC but is capable of handling discrete data. The main optimization of gpuPC is focused on the storage and updating of contingency tables, rather than the parallelization strategy.

To the best of our knowledge, Conditioning Set Filtering (CSF) [3] is the sole method that utilizes FPGAs to accelerate causal discovery. Unlike approaches that employ CPUs and GPUs for acceleration, CSF uses FPGAs to compute scores for conditioning sets. By setting a threshold to the scores, the algorithm can limit the number of CI tests instead of accelerating them. Consequently, the computation of the score becomes the new speed bottleneck. A major drawback of CSF is that the method fails to work for high-dimensional data sets that are too big to fit into the on-chip memory of the FPGA. In contrast, some modern problems require causal discovery from very high dimensional data. For instance, a popular way to validate agent-based models is the the structural comparison method [14]. The key procedure is to run causal discovery on a dataset on transformed time series whose dimensionality can go beyond 10^4.

III. Conditional Independence Test Prioritization

This section presents a new approach to speed up causal discovery using FPGAs called Conditional Independence Test Prioritization (CITP) in this section. We first examine the issues that arise when trying to accelerate causal discovery using FPGAs. Then, we outline the acceleration strategy used in CITP.

A. Challenges in FPGA-Accelerated Causal Discovery

In FPGA acceleration of causal discovery, a major problem is the restricted capacity of on-chip memory. A typical constraint-based causal discovery algorithm needs to access two key pieces of information: the data matrix and the draft graph. To ensure adequate data bandwidth, it is essential to employ fast on-chip memory resources on the FPGA device, like block RAM (BRAM). Nevertheless, both the data matrix and the draft graph demand significant memory space. If a dataset has high dimensionality or a large number of data points, it is improbable that it can fit into the on-chip memory of a conventional FPGA chip.

If we use FPGAs to decrease the quantity of CI tests instead of attempting to speed up the tests, the issue at hand remains quite difficult. Specifically, while we can circumvent FPGA-unfriendly operations in CI testing, we are confronted with the following three challenges:

Challenge of local CI test prioritization. To reduce the number of CI tests, it is necessary to pick up promising CI tests that are most likely to remove edges. The only approach we know, CSF, needs to use the on-chip memory to store global data related to all nodes in the graph to identify promising CI tests efficiently. However, if the data is too big to fit into the on-chip memory, we need to design an alternative CI test quality control method to use local data on a subset of nodes.

Challenge of data partitioning. Assuming that we have a CI test selection mechanism to address the previous challenge, we can transfer partial data to the FPGA. Nonetheless, transferring different subsets of data to the FPGA may result in varying degrees of statistical accuracy and computational efficiency. As a result, identifying a methodology to partition the data such that the most pertinent subsets are sent to the FPGA becomes a challenging task.

Challenge of serial dependency. A constraint-based causal discovery method requires updating the draft causal graph continually. It is challenging to start the computation for the next iteration until the current iteration finishes updating the draft graph since the next iteration necessitates an up-to-date draft graph. Also, the calculations in each iteration have distinct natures. For example, the enumeration of CI tests primarily involves memory and logical operations, while the execution of CI tests primarily involves numerical operations. Therefore, poor resource management or load balancing can result in significant resource underutilization.

B. Conditional Independence Test Prioritization

We propose a new technique called Conditional Independence Test Prioritization (CITP) to address the challenges noted in Section III-A. This approach requires close collaboration between a CPU and an FPGA platform. The CPU platform maintains a draft causal graph in its memory and iteratively removes edges. Initially, the draft graph is fully-connected. Each CITP iteration attempts to remove a set of edges from the causal graph. The algorithm stops when it cannot remove any edge in a CITP iteration.

A CITP iteration works as follows. First, the CPU platform partitions the draft graph into subgraphs and sends the most promising subgraph to the FPGA. Second, the FPGA platform evaluates a score for each CI test using only the subgraph data. The score indicates the likelihood that the CI test can

remove an edge when executed. Third, the FPGA platform sends the score to a priority queue on the CPU platform, which prioritizes tests with the highest scores. Finally, the CPU platform removes edges by executing CI tests in the priority queue. These steps can occur out of order to reduce resource idleness.

The computational procedures in each CITP iteration include priority scoring, subgraph selection, and CPU–FPGA collaboration. Each procedure addresses a challenge noted in Section III-A.

1) Priority score evaluation on FPGA: To avoid the on-chip memory limit, the FPGA platform should be able to evaluate priority scores using the data for a small subgraph. For any given CI test $\mathcal{I}(v_a, v_b, C)$, we propose to compute the following priority score on the FPGA:

$$\phi(v_a, v_b, C) = 1 - |\mathrm{pearson}\big(\epsilon(v_a, C), \epsilon(v_b, C)\big)| \quad (1)$$

where $\mathrm{pearson}(u_\dagger, u_\ddagger)$ is the Pearson correlation coefficient between two vectors $u_\dagger$ and $u_\ddagger$; $\epsilon(u, C)$ is the error vector defined by

$$\epsilon(u, C) = X_C \times w^* - x_u \quad (2)$$
$$w^* = \mathrm{argmin}_w ||X_C \times w - x_u|| \quad (3)$$

In other words, we eliminate the information contained in the conditioning set like [3], [15], [16] and use the correlation between the two error vectors as a heuristic to determine the possibility of the two variables being conditionally independent. high score implies that v_a and v_b are likely to be independent given the information in C and the execution of the corresponding CI test $\mathcal{I}(v_a, v_b, C)$ should take place relatively early. It is important to note that our priority score in Equation 2 exclusively uses the data of a subgraph to calculate the scores, which **addresses the challenge of local CI test prioritization**.

2) Subgraph selection on CPU: At the start of each CITP iteration, the CPU platform selects a subgraph from the draft graph so that all scores can be computed using the data in the subgraph. When a subgraph is chosen, the data vectors corresponding to the selected nodes are sent to the FPGA platform. In this case, we aim to use the data vectors to compute scores for as many edges as possible. Therefore, we aim to choose a subgraph with the maximum number of edges interconnecting the nodes in the subgraph. In other words, we aim to select a subgraph $G' = (V', E')$ as follows:

$$V' = \mathrm{argmax}_{V_+} |E \cup V_+ \times V_+| \quad (4)$$
$$E' = E \cup V' \times V' \quad (5)$$

The problem formulation in Equation 4 and 5 provides a way to **address the challenge of data partitioning** using combinatorial optimization algorithms. We have proved that this optimization problem cannot be solved in polynomial time, but it is unnecessary to compute the optimal solution in practice. In Section IV, we design a greedy algorithm for this optimization problem.

3) Efficient collaboration between CPU and FPGA: The collaboration workflow depicted in Figure 1(a) is a straightforward collaboration plan resulting from the two aforementioned procedures. However, this workflow has a serial dependency in each iteration. To improve the efficiency of the collaboration workflow, we propose to make the following arrangements: First, the FPGA platform employs a ping-pong buffer in the on-chip memory. This buffer consists of two parts, each of which stores a subgraph. At any given time, only one part is active, and the FPGA uses this active part to compute CITP priority scores. After evaluating all scores of the active part, the active and inactive parts swap, and the CPU begins computing the subgraph for the next batch. Once the CPU finishes the computation, it sends the subgraph data to the inactive part of the ping-pong buffer. Second, the CPU platform performs CI tests without the goal of emptying the priority queue. As a result, the execution of the tests can be suspended when the FPGA updates the priority queue.

By putting these settings into practice, we obtain the workflow shown in Figure 1(b). This workflow eliminates the issue of resource idleness caused by the serial data dependency. The time required to transfer data from the CPU to the FPGA is amortized for by the computation time, and the priority queue update only briefly pauses the CPU platform. Consequently, the updated workflow **addresses the challenge of serial dependency**.

IV. Design and Implementation

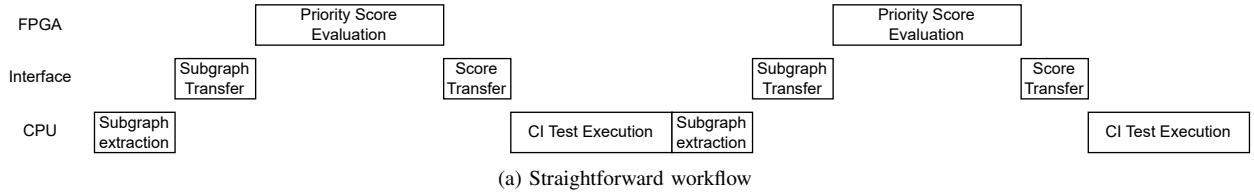
This section delves into CITP by presenting the hardware and software designs for CITP iterations and an experimental implementation.

A. Hardware and Software Design

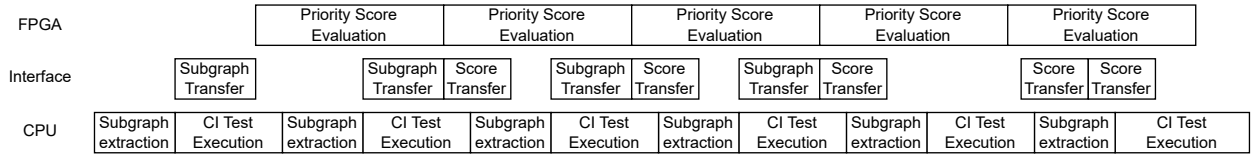
Our hardware design for FPGA-based CITP priority score evaluation is illustrated in Figure 2. This design is based on the CITP priority score from Equation 1. The design works as follows. First, the subgraph data containing the data vectors and candidate CI tests are received from the CPU platform and stored in on-chip memory. Then, for each candidate CI test, two least-squares solvers remove the information provided by the conditioning set from the two test variables and calculate the two error vectors using Equation 2. Finally, a Pearson correlation coefficient evaluator computes the score following Equation 1, which is then sent back to the CPU platform.

On the FPGA side, the acceleration strategy discussed in Section III reduces the complexity of the hardware and simplifies the design space, allowing major operations to be supported by well-optimized hardware implementations in existing research. In particular, our acceleration strategy allows the hardware to focus on only two calculations: (i) least-squares and (ii) evaluation of the Pearson's correlation coefficient. Both calculations have efficient FPGA designs, which we will discus in Section IV-B.

On the CPU side, the only major problem to solve is the optimization problem defined by Equation 4. To solve the optimization problem, we adapt a greedy algorithm originally



(a) Straightforward workflow



(b) Proposed efficient workflow

Fig. 1. Gantt charts for CPU–FPGA collaboration

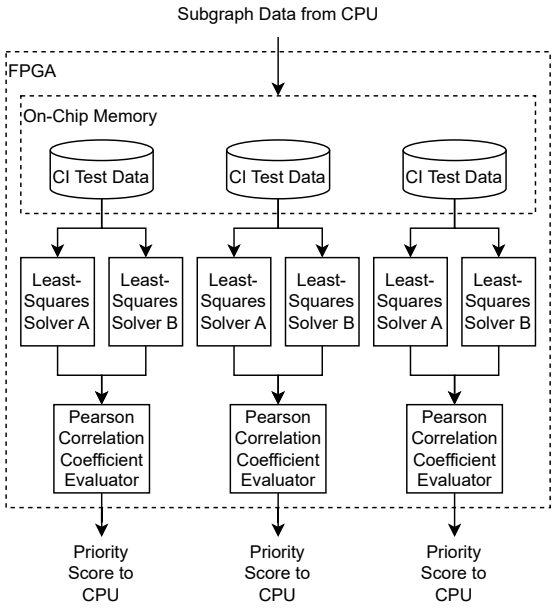


Fig. 2. FPGA-based hardware design for CITP priorty score evaluation

designed for the maximum clique problem [17]. The algorithm starts with a subgraph that consists of only one edge and then adds nodes to the subgraph iteratively. At each iteration, the algorithm expands the subgraph by adding a node that maximizes the number of edges to the subgraph. To account for the fact that different starting edges can lead to different numbers of edges in the final subgraph, we repeat the procedure with every edge in the draft graph and choose the subgraph with the most edges.

B. Implementation

We use 16-bit fixed-point numbers for data and 32-bit floating-point numbers for intermediate variables on the FPGA platform. We conduct experiments and find that 16-bit fixed-point numbers are reliable enough for computing priority scores. However, we use floating-point numbers for intermediate variables because we observed that fixed-point numbers can cause numerical instability issues in the least-squares solver.

TABLE I
RESOURCE USAGE ON INTEL ARRIA 10 FPGA

Resource	ALUT	RAM	FF	DSP
Total	854400	2713	1708800	1518
Used	635946	2468	957620	1282
Percentage Used	74%	91%	56%	84%

On the FPGA platform, we implement the hardware for score evaluation using a least-squares solver and a Pearson's correlation coefficient evaluator. To evaluate Equation 1, we implement the FPGA-based least-squares solvers proposed in [18]. The calculations in the Pearson correlation coefficient are mainly the calculation of the sample covariance. We implement the FPGA-based sample variance calculator in [19] for this calculation.

To improve the collaboration efficiency between the FPGA-based hardware accelerator and CPU-based software, we implement two optimizations in our design. The first optimization is warm starting, where the CPU conducts an unconditional independence test on the initial graph before the first iteration to eliminate unconditionally independent edges. This optimization is meant to give the causal discovery algorithm a head start, allowing the edge subgraph selection routine to be effective from the beginning. The second optimization is early stopping. When the FPGA has not finished solving the least squares problem, the residuals may be inaccurate. However, since the accuracy of residuals does not directly determine the score, we terminate the least-squares solver early when the norm of the residual vector stops changing.

The FPGA design is programmed as an Intel DPC++ kernel and compiled using the oneAPI DPC++/C++ Compiler [20]. The source code is available at https://github.com/ceguo/citp. The compiled bitstream is intended for use with the Intel PAC card with the Intel Arria 10 GX FPGA, and the FPGA's clock speed is 240MHz. The resource usage is listed in Table I.

V. Evaluation

This section assesses the accuracy and speed of CITP by comparing the accuracy and speed of CITP with existing methods on CPUs and GPUs.

A. Experiment Setup

We compare CITP against the three other causal discovery techniques including pcalg [1] running on multi-core CPUs, cuPC [2] running on GPUs and CSF [3] running on FPGAs. We do not compare with bnlearn [21] or ParallelPC [12] because they do not demonstrate accuracy or speed advantages compared with pcalg in the experiments in [3] and [22].

TABLE II
Hardware Platforms

Platform	CPU	GPU	FPGA
Vendor	Intel	NVIDIA	Intel
Series	Core	GeForce	Arria
Model	i5-9500	RTX 3060	10 GX
Lithography	14nm	8nm	20nm
Cores	6	3584	N/A
Base Frequency	3.00 GHz	1.32 GHz	240 MHz
Max Frequency	4.40 GHz	1.78 GHz	240 MHz
Used in pcalg	✓	×	×
Used in cuPC [2]	×	✓	×
Used in CSF [3]	✓	×	✓
Used in CITP (Proposed)	✓	×	✓

Our hardware platforms are shown in Table II. Regarding compilation, the C++ software code for pcalg, is compiled with LLVM/Clang version 14.0.0 using the `-O2` optimization flag. We attempt to use the `-O3` optimization flag, but the code runs slower than the version with `-O2`. We use the NVIDIA CUDA compiler 11.1 to compile the GPU code for cuPC with the `-O3` optimization flag following the settings in [2].

TABLE III
Causal graphs used in experiments

Name	Nodes	Edges	Application
Hailfinder	56	66	Severe weather forecasting [23]
Hepar2	70	123	Liver disorder diagnosis [24]
Win95pts	76	112	Printer troubleshooting [25]
Pathfinder	109	195	Lymph node disease diagnosis [26]
Andes	223	338	Intelligent tutoring [27]
Diabetes	413	602	Diabetes diagnosis [28]
Pigs	441	592	Pedigree inference [29]
Link	724	1125	Human genetics [30]
Munin	1041	1397	Electromyography analysis [10]

Our focus is on testing both discovery speed and accuracy, we only use datasets with ground truths. However, the availability of these causal graphs in the public domain is limited. In the experiments, we use the largest real-world causal graphs we can find, as listed in Table III. These graphs have been considered very challenging in previous studies [1], [3], [7], [12], [22]. Based on the size classification scheme of causal graphs in [21], Hepar2, Hailfinder, and Win95pts are classified as large causal graphs with 50-100 nodes, while Pathfinder, Andes, Diabetes, Pigs, and Link are classified as very large causal graphs with 100-500 nodes; Munin is classified as a massive causal graph with 1041 nodes.

Regarding data sampling, we adopt the structural linear model to generate data samples from the tested graphs, as suggested in [3], [31]. Another sampling model for continuous data is the Gaussian model [11]. However, we do not use this model because it assumes that the data follows a Gaussian distribution, which is not applicable to various causal discovery scenarios [32]. We generate a dataset consisting of 10000 data points for each causal graph. This sample size is considered adequate for typical applications in genomic analysis [12] and validation of agent-based models [14]. As the size of the data increases, the accuracy of all algorithms is expected to improve, but the speed of causal discovery may decrease. Moreover, causal discovery tools like CSF, which have limitations on the data size may no longer function when the data is too large.

We evaluate the accuracy and speed of the skeleton discovery stage in causal discovery for each dataset, while ignoring the computation of directions of edges, as suggested in [33].

B. Accuracy

The CITP method has a different computational routine from existing methods, which may lead to different causal discovery accuracy. Fortunately, accuracy evaluation process of causal discovery algorithms is straightforward. Given the ground truth of the causal causal graph, the accuracy can be calculated as the number of edges correctly identified divided by the total number of edges.

TABLE IV
Discovery Accuracy

	Existing Approaches			Proposed Approach	
	pcalg [1]	cuPC [2]	CSF [3]	CITP	Ratio/Best
Hailfinder	0.7576	0.7576	0.7879	**0.8030**	100.00%
Hepar2	**0.2846**	**0.2846**	0.2683	0.2602	91.43%
Win95pts	**0.7143**	**0.7143**	0.6875	0.6964	97.49%
Pathfinder	TLE	TLE	0.8667	**0.8872**	100.00%
Andes	**0.7781**	**0.7781**	OCMLE	0.7751	99.61%
Diabetes	0.7907	0.7907	OCMLE	**0.8040**	100.00%
Pigs	**0.6571**	**0.6571**	OCMLE	0.6453	98.20%
Link	**0.4693**	**0.4693**	OCMLE	0.4489	95.65%
Munin	**0.5140**	**0.5140**	OCMLE	0.5125	99.71%

TLE: time limit exceeded (cannot finish in 7200 seconds).
OCMLE: on-chip memory limit exceeded (cannot start).

Table IV presents the accuracy results of various methods for causal discovery. The accuracy levels of all the implementations are comparable, and particularly, pcalg and cuPC demonstrate the same accuracy due to the equivalence of their algorithms. For each test case, the accuracy values for all methods are within a narrow range. Although CITP's accuracy is slightly lower than that of pcalg and cuPC, the variation is less than 0.03 across all test cases, which is regarded as negligible in the context of causal discovery according to [4].

Our observation that all implementation gives similar accuracy is different from the one in [3], where CSF has a small advantage. The reason for this difference is that the data sizes used in our study are larger than those used in the CSF paper, allowing for more accurate CI test results even if the test algorithms are statistically simple. In this case, simpler

Fisher's CI tests used in pcalg and cuPC can achieve similar accuracy to the kernel-based CI test in CSF. Our observation is another instance where more data outperforms smarter algorithms, which is a common occurrence in data mining [34], [35]. This observation further supports our opinion that the ability to handle large data sizes is a crucial feature in causal discovery tools.

C. Discovery Speed

We measure the causal discovery speed of different methods by taking the total execution time. Using the execution time record, we also calculate the speedup of the proposed approach over other methods.

TABLE V
DISCOVERY TIME IN SECONDS

	Existing Approaches			Proposed Approach	
	pcalg [1]	cuPC [2]	CSF [3]	CITP-CPU	CITP
Hailfinder	6.4	**3.0**	4.9	15.0	5.5
Hepar2	2.8	**1.3**	2.7	14.3	3.3
Win95pts	2.7	**1.7**	5.5	14.0	3.8
Pathfinder	TLE	TLE	20.1	87.2	**9.5**
Andes	12.5	**3.3**	OCMLE	56.3	4.5
Diabetes	3078.2	354.1	OCMLE	1489.8	**139.0**
Pigs	3424.1	219.6	OCMLE	662.4	**84.9**
Link	65.0	**12.7**	OCMLE	200.6	21.2
Munin	695.7	14.6	OCMLE	77.0	**8.8**

TABLE VI
SPEEDUP OF FPGA-BASED CITP OVER OTHER METHODS

	pcalg [1]	cuPC [2]	CSF [3]	CITP-CPU
Hailfinder	1.2	0.5	0.9	2.7
Hepar2	0.8	0.4	0.8	4.3
Win95pts	0.7	0.4	1.4	3.7
Pathfinder	N/A	N/A	2.1	9.2
Andes	2.8	0.7	N/A	12.5
Diabetes	22.1	2.5	N/A	10.7
Pigs	40.3	2.6	N/A	7.8
Link	3.1	0.6	N/A	9.5
Munin	79.1	1.7	N/A	8.8

Table V shows the speed of different approaches measured by the execution time in seconds. The CITP-CPU column corresponds to a multi-core CPU implementation of CITP. We do not show the speed of the single-core CPU implementation of CITP for brevity. The CPU version of CITP runs on six cores provides around 3 times speedup over the single-core serial version. This observation is similar to [22], where multiple causal discovery algorithms cannot provide more than 4 times speedup over the serial version, regardless of the number of CPU cores.

While a large causal graph with more nodes can result in longer discovery time, the discovery time is not solely determined by the graph size. The problem complexity also relies on the graph topology and data distribution. For instance, if the causal graph is small, but the topology and data make it challenging to eliminate edges with small conditioning sets, the causal discovery process can still be time-consuming.

CITP is the overall fastest for large causal graphs. It is respectively up to 79 times, 2.6 times and 2.1 times faster than pcalg, cuPC and CSF. The test cases where CITP provides the highest speed include Pathfinder, Diabetes, Pigs, and Munin. These causal graphs are among the largest causal graphs in the test cases with 109–1041 nodes. In contrast, the test cases where CITP is less than half speed of cuPC include Hailfinder, Hepar2 and Win95pts. These causal graphs are relatively small with only 56–76 nodes.

The higher performance of CITP over pcalg and cuPC for large causal graphs is likely due to the computational overhead of the priority score. In general, a graph with more nodes tends to have a larger number of CI tests. Proper prioritization can reduce the number of CI tests since early execution of critical CI tests can significantly simplify the draft graph. However, when the graph is small, the reduction in the number of CI tests is limited, and the computation for CI prioritization becomes a significant overhead. The time saved by the reduction in the number of CI tests can offset the time spent on priority score evaluation.

Our experiment results indicate that CITP has the potential to be applied to more challenging causal graphs, such as the one with 1643 nodes in [2] which took three days to run on a CPU. Unfortunately, we have no access to the data, and therefore have limited knowledge about the network topology and data distribution. However, we estimate that using our proposed technique, the causal discovery for the 1643-node graph in [2] can be completed within 10 minutes.

VI. CONCLUSIONS AND FUTURE WORK

Accelerating causal discovery using FPGAs is a challenging task. The only solution we know, conditioning set filtering (CSF), has a major drawback: it cannot work when the on-chip memory capacity is insufficient to hold the entire data matrix. This paper addresses the on-chip memory issue by proposing a new approach called conditioning independence test prioritization (CITP). CITP allows the data to remain in off-chip memory, hence removing the data size limit. In the experimental evaluation, CITP achieves similar accuracy with state-of-the-art causal discovery methods running on CPUs and GPUs. CITP is also faster than the CPU-based pcalg and GPU-based cuPC on large causal graphs.

Future work includes optimizing our approach to further improve its performance, power consumption, accuracy, and the scalability for larger data using cloud-based systems that combine CPUs, GPUs, and FPGAs, and applying the approach to demanding applications such as those in biomedical science and agent-based modeling.

ACKNOWLEDGMENTS

We thank the anonymous reviewers and the shepherd for their valuable comments and suggestions. The support of UK EPSRC (grant number EP/V028251/1, EP/L016796/1 and EP/N031768/1), SRC and Intel is gratefully acknowledged.

REFERENCES

[1] M. Kalisch, M. Mächler, D. Colombo, M. H. Maathuis, and P. Bühlmann, "Causal inference using graphical models with the R package pcalg," *Journal of statistical software*, vol. 47, no. 11, pp. 1–26, 2012.

[2] B. Zarebavani, F. Jafarinejad, M. Hashemi, and S. Salehkaleybar, "cuPC: CUDA-based parallel PC algorithm for causal structure learning on GPU," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 3, pp. 530–542, 2019.

[3] C. Guo and W. Luk, "Accelerating constraint-based causal discovery by shifting speed bottleneck," in *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 169–179, 2022.

[4] M. Scutari, C. E. Graafland, and J. M. Gutiérrez, "Who learns better Bayesian network structures: Accuracy and speed of structure learning algorithms," *International Journal of Approximate Reasoning*, vol. 115, pp. 235–253, 2019.

[5] M. Tsagris, G. Borboudakis, V. Lagani, and I. Tsamardinos, "Constraint-based causal discovery with mixed data," *International journal of data science and analytics*, vol. 6, no. 1, pp. 19–30, 2018.

[6] B. Huang, K. Zhang, J. Zhang, J. D. Ramsey, R. Sanchez-Romero, C. Glymour, and B. Schölkopf, "Causal discovery from heterogeneous/nonstationary data.," *Journal of Machine Learning Research*, vol. 21, no. 89, pp. 1–53, 2020.

[7] D. Colombo, M. H. Maathuis, *et al.*, "Order-independent constraint-based causal structure learning.," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3741–3782, 2014.

[8] J. Runge, "Discovering contemporaneous and lagged causal relations in autocorrelated nonlinear time series datasets," in *Conference on Uncertainty in Artificial Intelligence*, pp. 1388–1397, PMLR, 2020.

[9] P. Wang, J. Xu, M. Ma, W. Lin, D. Pan, Y. Wang, and P. Chen, "Cloudranger: Root cause identification for cloud native systems," in *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pp. 492–502, IEEE, 2018.

[10] S. Andreassen, F. V. Jensen, S. K. Andersen, B. Falck, U. Kjærulff, M. Woldbye, A. Sørensen, A. Rosenfalck, and F. Jensen, "MUNIN: an expert EMG assistant," in *Computer-aided electromyography and expert systems*, pp. 255–277, Pergamon Press, 1989.

[11] T. D. Le, T. Hoang, J. Li, L. Liu, H. Liu, and S. Hu, "A fast PC algorithm for high dimensional causal discovery with multi-core PCs," *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 16, no. 5, pp. 1483–1495, 2016.

[12] T. D. Le, T. Xu, L. Liu, H. Shu, T. Hoang, and J. Li, "ParallelPC: an R package for efficient causal exploration in genomic data," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, (New York, NY, USA), pp. 207–218, Springer, 2018.

[13] C. Hagedorn and J. Huegle, "GPU-accelerated constraint-based causal structure learning for discrete data," in *SIAM International Conference on Data Mining*, (Virtual Event), pp. 37–45, SIAM, 2021.

[14] M. Guerini and A. Moneta, "A method for agent-based models validation," *Journal of Economic Dynamics and Control*, vol. 82, pp. 125–141, 2017.

[15] H. Zhang, K. Zhang, S. Zhou, J. Guan, and J. Zhang, "Testing independence between linear combinations for causal discovery," in *AAAI Conference on Artificial Intelligence*, vol. 35, (Vancouver, Canada), pp. 6538–6546, AAAI, 2021.

[16] G. Doran, K. Muandet, K. Zhang, and B. Schölkopf, "A permutation-based kernel conditional independence test," in *UAI*, pp. 132–141, 2014.

[17] S. S. Skiena, *The algorithm design manual*, vol. 2. Springer, 1998.

[18] D. Yang, G. D. Peterson, H. Li, and J. Sun, "An FPGA implementation for solving least square problem," in *2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*, pp. 303–306, IEEE, 2009.

[19] D. B. Thomas and W. Luk, "Estimation of sample mean and variance for Monte-Carlo simulations," in *2008 International Conference on Field-Programmable Technology*, pp. 89–96, IEEE, 2008.

[20] J. Reinders, B. Ashbaugh, J. Brodman, M. Kinsner, J. Pennycook, and X. Tian, *Data parallel C++: mastering DPC++ for programming of heterogeneous systems using C++ and SYCL*. Springer Nature, 2021.

[21] M. Scutari, "Learning Bayesian networks with the bnlearn R package," *Journal of Statistical Software*, vol. 35, no. i03, pp. 1–22, 2010.

[22] J. Huegle, C. Hagedorn, M. Perscheid, and H. Plattner, "MPCSL-a modular pipeline for causal structure learning," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 3068–3076, 2021.

[23] B. Abramson, J. Brown, W. Edwards, A. Murphy, and R. L. Winkler, "Hailfinder: A Bayesian system for forecasting severe weather," *International Journal of Forecasting*, vol. 12, no. 1, pp. 57–71, 1996.

[24] H. Wasyluk, A. Onisko, and M. Druzdzel, "Support of diagnosis of liver disorders based on a causal Bayesian network model," *Medical Science Monitor*, vol. 7, no. 1; SUPP, pp. 327–332, 2001.

[25] D. Heckerman and M. P. Wellman, "Bayesian networks," *Communications of the ACM*, vol. 38, no. 3, pp. 27–31, 1995.

[26] D. Heckerman, "Probabilistic similarity networks (technical report stan-cs-1316)," *Palo Alto, CA: Stanford University, Departments of Computer Science and Medicine*, 1990.

[27] A. S. Gertner and K. VanLehn, "Andes: A coached problem solving environment for physics," in *Intelligent Tutoring Systems: 5th International Conference, ITS 2000 Montréal, Canada, June 19–23, 2000 Proceedings*, pp. 133–142, Springer, 2002.

[28] H. Temurtas, N. Yumusak, and F. Temurtas, "A comparative study on diabetes disease diagnosis using neural networks," *Expert Systems with applications*, vol. 36, no. 4, pp. 8610–8615, 2009.

[29] U. Kjærulff, "Inference in Bayesian networks using nested junction trees," *Learning in Graphical Models*, pp. 51–74, 1998.

[30] C. S. Jensen and A. Kong, "Blocking Gibbs sampling for linkage analysis in large pedigrees with many loops," *The American Journal of Human Genetics*, vol. 65, no. 3, pp. 885–901, 1999.

[31] H. Zhang, S. Zhou, and J. Guan, "Measuring conditional independence by independent residuals: Theoretical results and application in causal discovery," in *AAAI Conference on Artificial Intelligence*, vol. 32, (New Orleans, LA, USA), pp. 2029–2036, AAAI, 2018.

[32] S. Shimizu, "LiNGAM: Non-gaussian methods for estimating causal structures," *Behaviormetrika*, vol. 41, pp. 65–98, 2014.

[33] H. Zhang, S. Zhou, K. Zhang, and J. Guan, "Residual similarity based conditional independence test and its application in causal discovery," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, pp. 5942–5949, 2022.

[34] P. Domingos, "A few useful things to know about machine learning," *Communications of the ACM*, vol. 55, no. 10, pp. 78–87, 2012.

[35] G. Hulten and P. Domingos, "Mining complex models from arbitrarily large databases in constant time," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 525–531, 2002.