Co-Design of Algorithm and FPGA Accelerator for Conditional Independence Test

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

***Abstract*—Conditional independence (CI) testing is a critical statistical method that determines conditional independence between variables using data. It is useful for various data mining applications, such as causal discovery, Bayesian inference, and agent-based model validation. However, the high volume of CI test queries and the large data sizes make CI testing computationally intensive. This paper proposes a hardware-oriented residual-based CI testing algorithm, co-designed with an FPGA accelerator, to address this issue. Our system accelerates CI tests by skipping least-squares computations algorithmically, enabling fixed-point operations in correlation evaluation and parallelization of permutation tests. Our experimental evaluation demonstrates that our method is as accurate as state-of-the-art CI testing approaches. Furthermore, our experimental implementation on an Intel Arria 10 FPGA delivers up to 32 times higher performance compared to state-of-the-art CI test tools running on eight Intel Xeon Silver 4110 CPU cores.**

***Index Terms*—conditional independence test, causal discovery, causal inference, graphical model, structural equation model**

I. Introduction

Conditional independence (CI) testing is an important task in many data mining and information retrieval applications, such as causal inference, feature selection, and model selection. However, the computational complexity of these tests can be prohibitive, particularly for high-dimensional data. A promising way to accelerate CI testing is to use specialized hardware on Field-Programmable Gate Arrays (FPGAs) which can be programmed to perform complex computations in parallel. Unlike GPUs, FPGAs offer great flexibility in terms of the types of computations, making them well-suited for tasks that require custom algorithms. This paper explores the benefits of using FPGAs for conditional independence testing by co-designing an algorithm and an FPGA-based hardware accelerator.

CI tests are known to be computationally demanding. For example, the CI tests for a causal inference task on the DREAM5-INSILICO expression data set can take up to 168 seconds using 40 Intel Xeon Gold 6148 CPU cores [1]. However, there are currently no FPGA designs available to accelerate CI testing.

Our study aims to address this issue by developing an FPGA-based system that accelerates the execution of residual-based CI tests. The main challenge is that the linear regression procedure used in these CI tests has hardware-unfriendly operations.

We propose using a k-nearest neighbors (kNN) method, specifically tailored for hardware implementation, to substitute the linear regression operation used in residual-based CI testing to remove information from the controlling set. Although kNN-based approaches are highly parallelizable on FPGAs, there has been no research on designing a kNN-based method for residual extraction on algorithm or hardware. Therefore, we co-design a kNN-based residual extraction algorithm, permutation testing facilities and the corresponding hardware, resulting in a highly efficient CI testing system. The contributions of this paper include the following:

- Co-design of algorithm and hardware for a kNN-based residual extractor that is optimized for fast execution on FPGA. (Section III)
- FPGA design for on-the-fly permutation generation and permutation testing that collaborate with the residual extractor to conduct CI testing. (Section IV)
- Comparative evaluations of our approach against other parallel CI testing tools, including experiments to assess speed and accuracy. (Section V)

To our knowledge, this paper presents the first FPGA-based CI testing accelerator. The organization of the paper is as follows. Section II discusses the background of CI tests, permutation testing and residual-based CI tests. Sections III and IV present the proposed co-design approach for residual-based CI testing, including a novel residual extractor and permutation testing facilities. Sections V and VI cover evaluation and conclusion.

II. Background

A. Conditional Independence Test

The conditional independence (CI) test determines whether two variables are conditionally independent of each other given the values of one or more additional variables. In other words, it tests whether two variables are correlated when additional variables are taken into consideration.

A CI test is in the form $I(\mathbf{u}, \mathbf{v}|Z)$ where $\mathbf{u}$ and $\mathbf{v}$ are two random variables; Z is a set of other random variables known as the controlling set of the test. This function tests if $\mathbf{u}$ and $\mathbf{v}$ are conditionally independent given the information in Z and gives a Boolean output. Note that the size of the controlling set in a CI test cannot be too large, because including too many variables in the controlling set can increase the risk of

false positives, where a significant relationship is detected even though there is no true relationship between the variables of interest. A typical setting for the maximum controlling set size is 5 [2].

Conditional independence tests are commonly used in many areas of statistics, including causal inference, machine learning, and exploratory data analysis. The most commonly used CI test for continuous variables is the Fisher's Z-test [3]. Other methods for CI testing include residual-based methods, kernel-based methods [4], model-driven methods [5], binning methods [6] and feature ordering methods [7].

Despite the fact that CI tests can be time-consuming for various applications, there is currently no viable solution to speed up these tests using FPGAs. In contrast, CI tests are circumvented rather than accelerated on FPGA. For example, the FPGA-based Bayesian network structural learning system [8] employs a learning algorithm based on goodness-of-fit scores, whereas algorithms based on CI tests can be much faster and more accurate [9]. Similarly, [10] uses the FPGA to reduce the number of CI tests, but the tests themselves are computed on the CPU platform, making this approach unsuitable for CI test applications other than causal discovery.

B. Residual-based CI Test

Residual-based CI testing works as follows. A regression function takes the residuals from the two test variables $\mathbf{x}$ and $\mathbf{y}$ and returns two predicted vectors $\mathbf{x}'$ and $\mathbf{y}'$. Then, the residual vectors for the test variables $r(\mathbf{x}) = \mathbf{x} - \mathbf{x}'$ and $r(\mathbf{y}) = \mathbf{y} - \mathbf{y}'$ are calculated. In other words, a residual vector is difference between the actual target value and the predicted target value. Then, a correlation score s_0 is calculated based on $r(\mathbf{x})$ and $r(\mathbf{y})$. This procedure is repeated to compute the correlation scores, $s_1, s_2, s_3, \ldots, s_{M-1}$, for M different permutations of $\mathbf{y}$. The final decision is made based on the relative rank of s_0 among the M correlation scores. Major residual-based CI testing methods include the Kernel Conditional Independence Permutation Test (KCIPT), the Residual-based Conditional Independence Test (ReCIT) and the Fast residual-based Conditional Independence Test (FRCIT).

The Kernel Conditional Independence Permutation Test (KCIPT) [11] tests conditional independence by simplifying the problem into a two-sample test using permutation testing. The permutation preserves the joint distribution only when the null hypothesis of CI is valid. This approach allows for the incorporation of prior knowledge during the permutation step and shows competitive accuracy power even if the dimensionality of the controlling set grows to $|Z| = 5$.

The Residual-based Conditional Independence Test (ReCIT) [12] is the first method that directly uses the residuals to determine conditional independence. A critical theoretical foundation is a proof that if $\mathbf{x}$, $\mathbf{y}$, and Z are generated by a linear structural equation model with all external influences following Gaussian distributions, then $I(\mathbf{x}, \mathbf{y}|Z)$ if and only if $r(\mathbf{x})$ and $r(\mathbf{y})$ are independent. In other words, the CI test can be relaxed to an unconditional independence test between $r(\mathbf{x})$ and $r(\mathbf{y})$. The experimental results show that ReCIT outperforms the kernel-based method in [4] with linear non-Gaussian data.

The Fast residual-based Conditional Independence Test (FRCIT) [13] explores the independence between two linear combinations. This method is based on the conjecture that the first to fourth moments of the two linear combinations contain sufficient information to determine conditional independence. In the experiments, the FRCIT method outperforms kernel-based methods in terms of both speed and accuracy.

Residual-based conditional tests have been found to offer greater accuracy than conventional methods, such as Fisher's Z-test. However, the computational demands of residual-based CI tests, which involve linear regression and permutation testing, have limited their practical utility. For example, although residual-based CI tests have demonstrated significant advantages in terms of result quality in causal discovery, major causal discovery tools, such as `pcalg` [14], have not incorporated them due to their prolonged execution time, even for small datasets. Therefore, the development of efficient computational tools for residual-based CI tests is crucial for their application to large datasets in practical settings.

III. Co-design of Algorithm and FPGA for Residual Extraction

The first phase of residual-based CI testing involves residual extraction, wherein a linear regression algorithm removes the information provided by the controlling set from a target variable. Since the residual extraction process is computationally intensive in CI tests, it is essential to design an appropriate residual extractor for an accelerated CI testing system. This section presents our co-design approach for the residual extractor, outlining both the rationale and the specifics of creating a k-nearest neighbors residual extraction algorithm. This algorithm is amenable to FPGA implementation and can replace the linear regression algorithm in conventional residual-based CI tests.

A. Residual extraction on FPGA

A primary challenge in residual-based CI testing on FPGAs is the lack of suitable FPGA implementations for the linear regression operation in residual extraction. For instance, existing methods proposed in [15], [16], and [17] require storage and inversion of an $n \times n$ sketch matrix for each permutation of the dataset using on-chip memory. The high on-chip memory usage makes it challenging to leverage parallelism in permutation testing. Also, the architecture presented in [18], [19] offers a low-latency least squares solver for regression models with kernel functions. However, this architecture does not meet our requirements because residual extraction does not involve kernel transformation.

We find that, although linear regression is used in all the residual-based CI tests that we know, it is not the only option. Therefore, instead of trying to design hardware for linear regression, we seek an alternative residual extraction approach that can replace linear regression with considerations on algorithmic soundness and hardware-friendliness. Specifically,

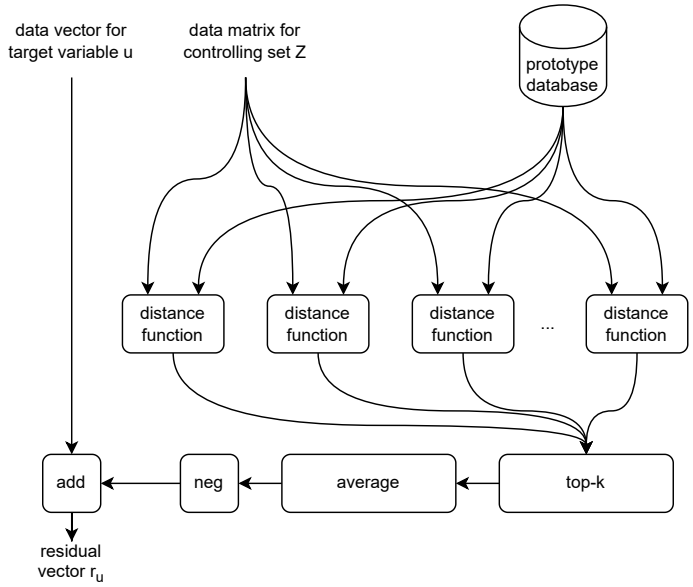


Fig. 1. Hardware design for low-precision kNN-based residual extraction

we find that a carefully designed k-nearest neighbors (kNN) method can meet both algorithm and hardware requirements. In the remainder of this subsection, we present our kNN algorithm and its hardware mapping for residual extraction.

A top-level view of the proposed residual extractor is shown in Fig. 1. The residual extractor takes the data vector for the target variable $\mathbf{u}$ and the data matrix for the controlling set Z. It produces the residual vector $r(\mathbf{u})$ by removing information related to the controlling set Z. This residual extractor calculates the residual for $u_i \in \mathbf{u}$ with four steps:

1) When an element of the data vector u_i and its corresponding assignment of the controlling variables Z_i are available, distance evaluators, shown as 'distance function' blocks in the figure, compute the distance between Z_i and all other assignments in a prototype database in parallel according to a distance metric.
2) A top-k element finder, denoted by the 'top-k' block in the figure, collects the target variable values of the k nearest assignments in the prototype database, namely $u^\circ_{t_0}, u^\circ_{t_1}, u^\circ_{t_2}, \ldots, u^\circ_{t_{k-1}}$.
3) An average calculator, shown as the 'average' block in the figure, calculates the average value $\hat{u}_i$ of the top-k values $u^\circ_{t_0}, u^\circ_{t_1}, u^\circ_{t_2}, \ldots, u^\circ_{t_{k-1}}$.
4) A subtractor, shown as a combination of a 'neg' block and an 'add' block in the figure, delivers the residual $r(u_i)$ by calculating the difference between the data value u_i and its estimate $\hat{u}_i$. In other words, the i-th element of $r(\mathbf{u})$ is calculated as follows:

$$r(u_i) = u_i - \hat{u}_i \tag{1}$$

The framework of the kNN design is similar to a conventional kNN method for regression, which is not the main focus of this study. Instead, our main focus is on the following two aspects that decide the algorithmic and hardware properties of the kNN based residual extractor:

- Prototype database. The prototype database is not a part of conventional kNN designs for regression, because the data points in the training set are naturally prototypes for conventional regression. In contrast, there is no training set in residual extraction. We design the following equation for data labels to collect a sub-sample from the data matrix:

$$j_q = \arg\min_j \left| u_j - \min \mathbf{u} - \frac{q(\max \mathbf{u} - \min \mathbf{u})}{|Q| - 1} \right| \tag{2}$$

  where j_q is a data index and $q \in [0..Q-1]$.
- Distance metric. The distance metric for our design is as follows:

$$d^2(\mathbf{z}^\dagger, \mathbf{z}^\ddagger) = \sum_{l=0}^{L-1} \left(\phi(z^\dagger_l, m_{data}) - \phi(z^\ddagger_l, m_{data}) \right)^2 \tag{3}$$

  where L is the common length of the two low-precision vectors $\mathbf{z}^\dagger$ and $\mathbf{z}^\ddagger$, which also equals the number of variables in the controlling set. The function $\phi(x, m)$ maps a data element x in any representation to an m-bit fixed-point number.

In the remaining two subsections, we discuss our considerations behind the design of prototype database and the distance metric.

B. Pre-computation of prototype database

This subsection discusses how the sub-sampling method in Equation 2 is designed. Our kNN-based residual extraction works by finding the nearest neighbors of a given data point, and then taking the average of the target values of those neighbors as the predicted value for the data point. This method is different from linear regression, which uses a parametric formula to predict the target value.

It is important to recognize that the goal of regression analysis in this scenario is not to maximize regression accuracy, but to strip off information from the controlling set. Therefore, accurate modern regression techniques including Gaussian processes [20], deep regression networks [21], and boosted regression trees [22] are not appropriate for our objective, as they may underestimate the regression residual [23] and compromise the evaluation of correlation coefficients. In the extreme case, where the regression algorithm perfectly predicts a data vector, the residual vector becomes a zero vector, which can completely confuse the correlation estimator.

We propose to pre-compute the prototype database to track behavior of linear regression. Linear regression has a strong theoretical foundation and consistently produces reliable results in residual-based CI tests. Therefore, if a method for estimating residuals can mimic the behavior of linear models, it may inherit their statistical reliability [24].

Our sub-sampling method, as presented in Equation 2, aims to promote the tracking of linear behaviors by the model. In particular, to control the error when interpolating a linear function, it is common to take interpolation points at regular

intervals from the domain of the function. We adopt a similar principle, but note that the domain of our problem may be multi-dimensional. Fortunately, the linearity we wish to track suggests that we can sample points at regular intervals on the **range** of the function, instead of the **domain**. Since the range of our kNN problem is one-dimensional, we can set $|Q|$ observation points $u_0^*, u_1^*, \ldots, u_{|Q|-1}^*$ across $[\min \mathbf{u}, \max \mathbf{u}]$ by:

$$u_q^* = \min \mathbf{u} + \frac{q(\max \mathbf{u} - \min \mathbf{u})}{|Q| - 1} \tag{4}$$

where $q \in [0..|Q-1|]$. However, the assignments in the controlling set may not cover these points. To resolve this issue, we find the nearest point in $\mathbf{u}$ for each point, resulting in the method based on Equation 2.

Even though the quality of the residuals is difficult to measure and reliant on the data, the proposed pre-computation approach can provide useful residuals. In general, regression algorithms inherently make mistakes because the true residual vector cannot be obtained due to the unknown or difficult-to-model nature of the causal relation. Fortunately, the error in residual vectors may not negatively impact the final decision because the statistical correlation is robust to outliers.

C. Optimization of distance metric

This subsection explains how the distance metric in Equation 3 is obtained. Unlike conventional hardware design for kNN, we have an upper bound for the dimensionality in kNN search. Specifically, the dimensionality of the kNN problem in residual extraction is equal to the size of the controlling set. An important property in CI testing is that the size of the controlling set must be small. Otherwise, the result of the CI test can be statistically invalid.

We focus on saving FPGA resources and not on the mathematical strength of the distance metric, because the low-dimensional nature of CI tests avoids the curse of dimensionality for nearest neighbor methods. In general, nearest-neighbors methods rely on the distance between neighboring points to make predictions, which can become increasingly difficult as the number of dimensions increases [25]. As a result, nearest-neighbors method can become less accurate and less efficient in high-dimensional spaces. To cope with the high dimensionality, a general-purpose kNN design needs to consider algorithmic issues when dealing with high dimensionality. One common technique used in conventional kNN designs is to employ sophisticated distance measures. However, since we only need to deal with low-dimensional kNN problem, we tend to keep the complexity of the distance metric as simple as possible.

To develop a mathematically simple and FPGA-friendly distance metric, we start from the standard Euclidean distance:

$$d(\mathbf{v}^\dagger, \mathbf{v}^\ddagger) = \sqrt{\sum_{l=0}^{L-1} (v_l^\dagger - v_l^\ddagger)^2} \tag{5}$$

This distance metric has been demonstrated to work well on low-dimensional data, which is beneficial for residual extraction in CI tests. Additionally, the algorithm complexity of a CI test is low with this distance metric. Assuming that the permutation test involves the residual vectors for M permutations of N data points, the time complexity for a CI test is $O(M \cdot N \cdot |Z| \cdot |Q|)$, while the space complexity is $O(|Z| \cdot |Q|)$.

The square root operation in the Euclidean distance in Equation 5 can be resource-consuming on FPGAs, but we can eliminate it. The goal of kNN residual extraction is to select the k nearest neighbors. In this case, whether a controlling set assignment c_q° is a near neighbor only depends on is relative position among $c_0^\circ, c_1^\circ, \ldots, c_{Q-1}^\circ$. Therefore, if we consider the computation of the k nearest neighbors of c_i as a procedure that repetitively takes away the minimum element from c_q° without replacement for k times, we only need to ensure the correctness of the **index** of the controlling set assignment with the minimum distance rather than the **distance value**. On the other hand, the square root function $f(x) = \sqrt{x}$ is monotonically increasing on the domain of real numbers. As a result, instead of using the standard Euclidean distance in Equation 5, we can safely omit the square root calculation without losing any accuracy in kNN regression.

$$d^2(z^\dagger, z^\ddagger) = \sum_{l=0}^{L-1} (z_l^\dagger - z_l^\ddagger)^2 \tag{6}$$

We propose to further optimize the distance metric for low-precision computation for efficient implementation on FPGAs. In general, lower-precision arithmetic operations are faster and more resource-efficient than their higher-precision counterparts on FPGAs. In particular, since we are targeting large data sizes with limited on-chip memory resources, we hope to use low-precision data types to reduce the memory footprint of intermediate values to leave more space to store the prototype set. In addition to hardware considerations, low-precision algorithms can be more robust to noise and errors in data. Therefore, they may be advantageous with the applications where noise and errors are inevitable such as causal discovery.

We enable low-precision computation by rounding the data matrix into m_{data}-bit fixed-point numbers, we obtain the function in Equation 3. Let each element in the data matrix be an m_{data}-bit fixed-point number. The maximum number of bits for all intermediate variables, including distance values and residuals, can be predetermined. Regarding the distance, the difference vector from the subtraction is an L dimensional vector where each element can be represented with an $(m_{data}+1)$-bit fixed-point number. Since the square of each element can fit into $2(m_{data}+1)$ bits, the corresponding number of bits required by the distance is

$$m_{dist} = \lceil \log_2 L \rceil + 2(m_{data} + 1) \tag{7}$$

Accordingly, an element in the residual vector must be the difference between two data elements, which requires $(m_{data}+1)$ bits.

IV. Parallel Permutation Test on FPGA

Even with efficient residual extraction using FPGAs, performing CI testing using the extracted residuals on FPGAs is still challenging. The permutation test involves a strict sequence where the upstream permutation generation process is also time-consuming. Although there are hardware modules available that can speed up these two procedures, effective load-balancing strategies must be implemented to prevent data starvation. Unfortunately, load-balancing in this scenario is exceptionally challenging because the relative duration of the two procedures depends on the data distribution.

This section presents a permutation testing method that avoids the load-balancing problem. Section IV-A presents a method to generate permutations for the target variable y on the fly, eliminating the need for load-balancing between permutation generation and residual extraction. Section IV-B explains how the permutation generator and residual extractor work together for permutation testing. Finally, Section IV-C showcases our experimental implementation of the CI testing system on an Intel Arria 10 FPGA.

A. On-the-fly permutation generation on FPGA

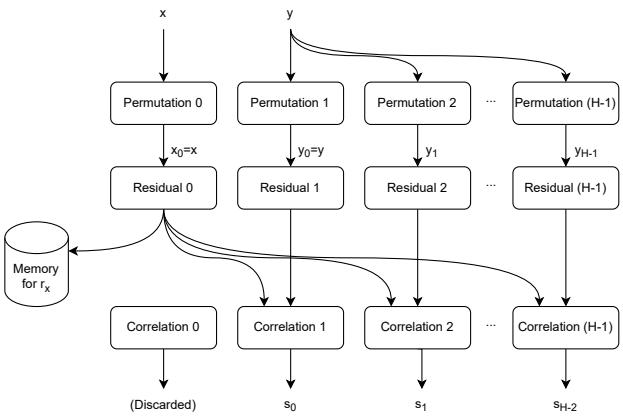
The generation of random permutations is straightforward. Given an array, it is as simple as applying a random shuffling algorithm to the array that contains the data vector. For example, the Fisher-Yates shuffling algorithm [26], [27] can finish shuffling an array with N elements in $\mathrm{O}(N)$ time with $\mathrm{O}(1)$ additional space.

We propose to permute a vector of indices instead of values. In other words, we use integers $0..(N-1)$ to produce a permutation $\iota_0, \iota_1, \iota_{N-1}$ so that the elements in u_{ι_k} constitute a permutation of $\mathbf{u}$. When a permutation index vector ι is precomputed, the proposed approach allows the permutation of $\mathbf{u}$ to be generated on the fly.

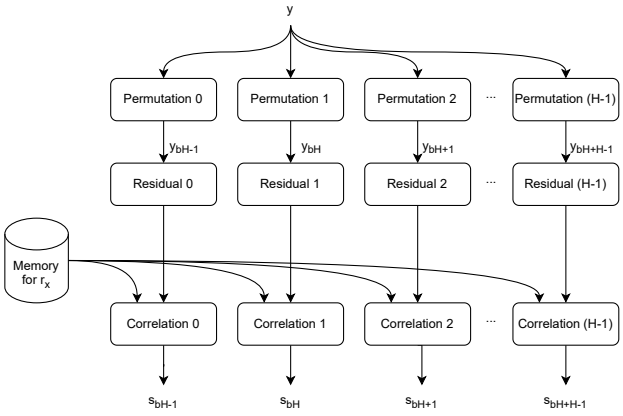
The proposed approach is hardware-oriented. Index permutation requires the storage of permutation indices, in addition to the values themselves, which takes up extra memory of $\mathrm{O}(n)$, while value permutation can be applied to the vector in an in-place manner, requiring only $\mathrm{O}(1)$ additional memory. Furthermore, index permutation requires random traversal compared to value permutation, where the values are already in the expected order after the main permutation operation, making it easier for the residual evaluator to visit them sequentially. However, with index permutation, only the index vector is in the expected order by the end of the permutation procedure, and every time the residual evaluation unit needs an element from the reordered vector, it requires random access to the memory for the value for the value in addition to sequential access for the index.

B. Permutation testing on FPGA

Permutation testing can be parallelized well in general. However, the calculation of the residual $r(\mathbf{x})$ requires special attention, particularly since the residual $r(\mathbf{x})$ is involved in the correlation evaluation for $\mathbf{y}$ and $\mathbf{y}'_0, \mathbf{y}'_1 \cdots \mathbf{y}'_{M-1}$. To avoid repetitive computation, an optimization is to compute the residual $r(\mathbf{x})$ before all correlation computation, and store $r(\mathbf{x})$ in memory. In this case the correlation algorithm is free to access any part of $r(\mathbf{x})$ to compute statistics on the whole residual vector.



(a) Batch 0



(b) Batch $b > 0$

Fig. 2. Hardware execution for permutation testing

To realize the optimization for $r(\mathbf{x})$, we design the hardware to work in batches as shown in Figure 2. The two parts of the figure refer to the same hardware architecture. The architecture contains H data pipelines. Each data pipeline contains a permutation generator discussed in Section IV-A, residual extractor discussed in Section III-A, and a correlation evaluator which computes the correlation coefficient. These pipelines compute the coefficient scores for permutations of $\mathbf{y}$ in batches. In Batch 0, one residual unit is in charge of the residual of the vector $\mathbf{x}$. The other residual units are in charge of residuals for the permutations of $\mathbf{y}$. Whenever the residual unit for $\mathbf{x}$ delivers one residual vector element, the element is broadcasted to the residual memory as well as the correlation estimators for other variables. By the end of this batch, a complete copy of the residual vector $r(\mathbf{x})$ is available in the residual memory r_x. In Batch b for $b > 0$, all residual units are in charge of computing the residuals for permutations of $\mathbf{y}$. At this time, the residual vector of $\mathbf{x}$ is taken from the residual memory and broadcasted to all the correlation

TABLE I
RESOURCE USAGE FOR LARGEST DESIGN ($|Z| = 6$)

Resource	ALUT	RAM	FF	DSP
Total	854400	2713	1708800	1518
Used	405110	1821	1284534	1392
Used(%)	47%	67%	75%	92%

estimators.

The computational efficiency of an implementation scales up with the number of fixed-point multipliers. For example, the number of multipliers required by a fully-pipelined implementation is

$$N^{\times} = |Z| \cdot |Q| + N^{\times}_{\text{coef}} \tag{8}$$

where $N^{\times}_{\text{coef}}$ is the number of multipliers used by the correlation coefficients. Thus, the number of multipliers for each pipeline grows linearly with the product of the size of the controlling $|Z|$ and the size of the prototype set $|Q|$. The residual extractor uses in each pipeline uses most of the multipliers due to the parallel computation of the distance metric following Equation 3. Additionally, the space complexity of CI testing is independent of the number of permutations M or the number of data points N, implying no theoretical limit for the data size if the data matrix is not stored in the on-chip memory.

Given an implementation with fixed $|Z|$ and $|Q|$, the time spent on each CI test grows linearly with the number of data points N. This is because the permutation generator produces the permutations on the fly and do not block the pipeline. The residual extractor delivers residual vectors at a consistent pace for each data element since the size of the prototype set keeps unchanged.

C. Experimental Implementation

We implemented an experimental setup using an Intel Arria 10 GX 10AX115S2F45I1SG FPGA with 1150K logic elements. The FPGA technology used is TSMC 20nm, and the design power consumption of the FPGA is 66W. The hardware design is described in OpenCL and compiled with the Intel FPGA SDK for OpenCL 19.4. The frequency of the FPGA is set to 240MHz.

The design parameters we use in the implementation are as follows. (i) The number of prototypes $|Q|$ is set to 64. We take this value to avoid the extreme case where all attributes in the low-precision representation of the data take a unique value. (ii) The number of neighbors k is set to $\sqrt{|Q|} = 8$ following [28]. (iii) The number of bits for data elements m_{data} is set to 16. We assume that data are stored in fixed-point format so that the mapping function in Equation 3 simply takes the most significant 16 bits from the data without numerical computation. (iv) The number of pipelines H is set to 32, 16, 8, 6, 5 and 5 respectively for $|Z| = 1..6$. The resource usage for the largest design when $|Z| = 6$ is shown in Table I.

The major concern of our implementation is to take advantage of the hardware resources by deploying as many permutation-residual pipelines as possible. Therefore, we tend to use a small prototype database in each pipeline. Note that this strategy is opposite to conventional kNN design in classification and regression. A conventional kNN design requires a search through a large dataset of prototypes. As a result, a common optimization to exploit available FPGA resources is to compute the distances with as many prototypes as possible in parallel. This approach is not ideal for our problem, as we are more focused on removing information rather than making predictions, which means that the high degree of parallelism of distance computation is not useful and can reduce the number of pipelines H.

We implement the one-pass Pearson coefficient design developed in [29] as the correlation estimator. This calculation can be fully pipelined, which can avoid a potential load-balancing problem. Specifically, in a straightforward hardware design, a load-balancing problem arises when one residual computation unit is busy while other residual units are idle because they do not have enough data.

V. EVALUATION

This section presents a comparative evaluation of our approach against other parallel CI testing methods running on CPUs and GPUs, assessing both accuracy and computational efficiency.

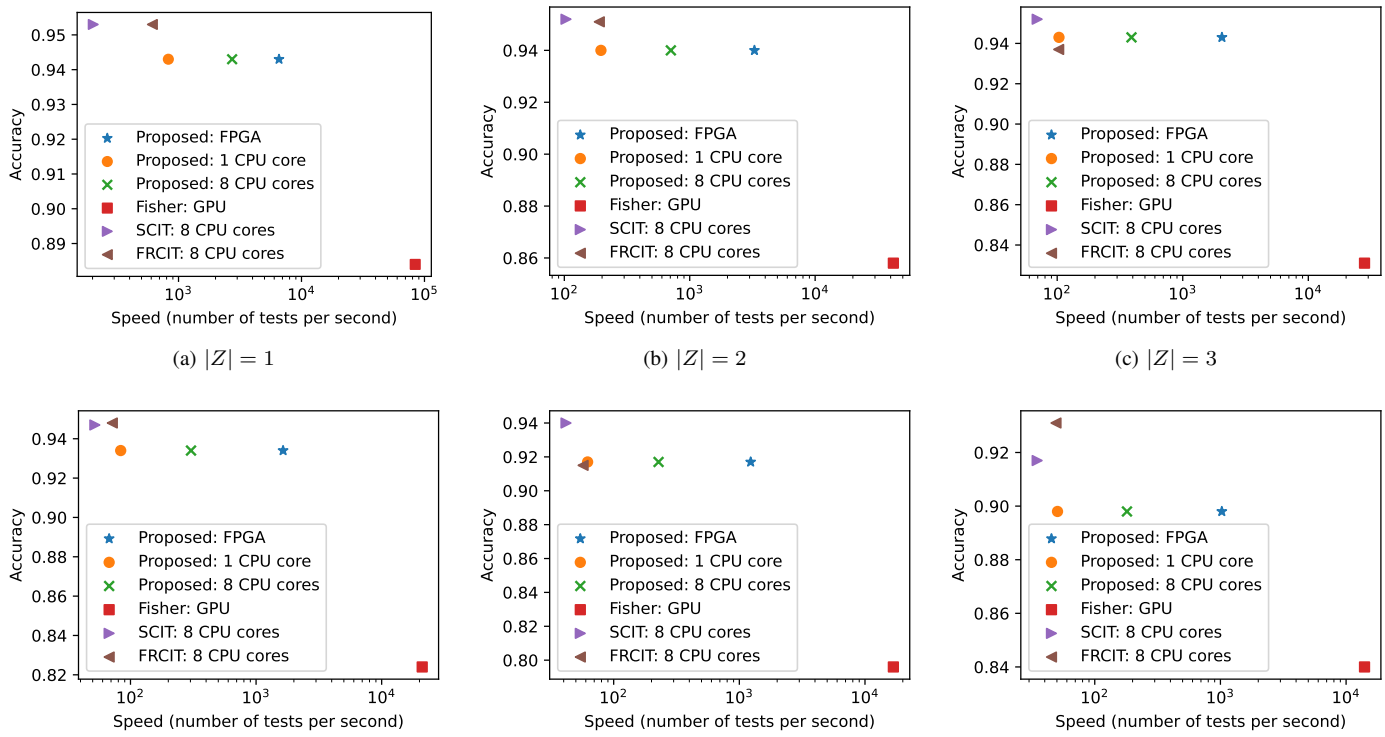
A. Experiment Setup

The CPU platform is equipped with an Intel Xeon Silver 4110 CPU, which operates at 2.1GHz and is based on 14nm technology. It features eight physical cores that support sixteen threads. The platform has 192GB of DDR4 memory. In contrast, the GPU platform has an NVIDIA GeForce RTX 3060 Ti, which runs at 1665MHz and is fabricated with 8nm technology. The GPU has 4864 CUDA cores and 8GB GDDR6x memory.

We conduct tests by generating 1000 datasets for each controlling set size from $|Z| = 1$ to $|Z| = 6$ using the linear non-Gaussian structural equation model (SEM) proposed in [2] and [13]. These generated datasets serve as reliable benchmarks as the CI testing algorithm that performs well with them is likely to perform well with real-world data. Each dataset comprises 10000 data elements. The CI testing implementations that we compare include (i) the proposed approach running on one CPU core, eight CPU cores, and the FPGA; (ii) the Fisher's Z-test in cuPC running on the GPU [30]; (iii) two residual-based approaches with linear regression, SCIT [2] and FRCIT [13], running on eight CPU cores. All the CPU-based tools are written or rewritten in the C programming language and compiled with LLVM/Clang version 14 with the O2 optimization flag.

B. Results and discussion

The trade-off between accuracy and speed for $|Z| = 1..6$ is shown in Fig. 3. The horizontal axis represents the speed of CI tests measured by the number of tests per second. We use the log scale for the horizontal axis because the speed differences between implementations are too significant. The vertical axis



(a) $|Z| = 1$ (b) $|Z| = 2$ (c) $|Z| = 3$

(d) $|Z| = 4$ (e) $|Z| = 5$ (f) $|Z| = 6$

Fig. 3. Speed-accuracy trade-off for CI tests for different controlling set size $|Z|$

corresponds to the CI test accuracy. Since a higher value on the two axes represents a better speed or accuracy, a superior overall design tends to be close to the top-right corner. We have the following observations from the results.

1) Regarding accuracy, residual-based methods, such as the one proposed, have significantly higher accuracy compared to the traditional Fisher's test. However, there is no significant difference in accuracy between different residual-based methods. In contrast, the Fisher's Z-test, although fast on the GPU, gives inferior accuracy. The GPU implementation utilizes double-precision floating-point numbers for the Z-test, limiting the potential for accuracy improvement through increased numeric precision.
2) The experimental FPGA implementation is much faster than all other residual-based CI test tools, including the CPU implementation of the proposed kNN-based algorithm. The proposed approach is respectively 29–32 times and 17–22 times faster compared with SCIT and RCIT. The FPGA implementation is also 2.4–5 times faster than the multi-core CPU version of the proposed approach.
3) The proposed algorithm scales poorly with the number of threads when implemented on a multi-threaded CPU. The multi-threaded version only provides a speedup of about 3.3–3.7 times compared to the serial version. This is consistent with the findings from [1] where causal discovery, an application that heavily relies on CI tests, also does not provide more than 4 times speedup over the serial version regardless of the number of CPU cores.
4) The speed advantage over the CPU version is modest when the size of the controlling set $|Z| = 1$, with only 2.4 times speedup. We note that the CPU implementation is particularly fast in this case. This is probably because all the operations can be element-wise parallelized when $|Z| = 1$, facilitating vectorization in CPU execution.

Furthermore, we have two anticipations based on the results. First, we anticipate that the existing designs for linear regression will not exhibit high efficiency for residual extraction on FPGAs. For instance, we predict that under the same experimental conditions, the linear regression design in [19] would result in a slowdown of at least 150 times compared to our design. Second, we anticipate that if the acceleration achieved by our proposed approach for CI testing is comparable to the acceleration achieved by CI-based causal discovery, then it can be integrated into existing causal discovery software, such as pcalg [14]. This integration would lead to higher quality causal inference for applications such as gene-expression analysis without sacrificing computational speed.

Although the proposed FPGA implementation shows promising experimental results, it has two potential limitations. First, as the statistical properties of kNN-based residual extraction require further exploration, there is a possibility of encountering scenarios where the kNN model fails to extract

sufficiently informative residual vectors. In such scenarios, the accuracy of the proposed implementation may be compromised. Second, when considering the hardware costs, the FPGA implementation can be less cost-effective than CPU and GPU implementations.

VI. Conclusion

Conditional independence (CI) testing is the statistical process of determining whether variables are dependent on each other using data. It is an essential method for various data mining applications such as causal discovery, Bayesian inference, and agent-based model validation. However, the computation required for CI testing can be challenging, especially when dealing with large datasets and numerous CI test queries. To overcome these challenges, we developed a CI testing system that utilizes the residual-based approach, which accelerates the execution of CI tests by eliminating the least-squares problem, parallelizing permutation tests, utilizing fixed-point operations in correlation evaluation, and taking into account FPGA performance constraints. Our experimental evaluation demonstrated that our method is equally accurate as current state-of-the-art CI testing approaches, but with significantly faster execution speed. Future work includes enhancing accuracy, performance and energy efficiency of our approach, and deploying it in various real-life applications.

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] 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.

[2] 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.

[3] R. Fisher, "On the 'probable error' of a coefficient of correlation deduced from a small sample," *Metron*, vol. 1, pp. 3–32, 1921.

[4] K. Zhang, J. Peters, D. Janzing, and B. Schölkopf, "Kernel-based conditional independence test and application in causal discovery," *arXiv preprint arXiv:1202.3775*, 2012.

[5] R. Sen, A. T. Suresh, K. Shanmugam, A. G. Dimakis, and S. Shakkottai, "Model-powered conditional independence test," *Advances in neural information processing systems*, vol. 30, 2017.

[6] T.-M. Huang, "Testing conditional independence using maximal nonlinear conditional correlation," *The Annals of Statistics*, vol. 38, no. 4, pp. 2047–2091, 2010.

[7] M. Azadkia and S. Chatterjee, "A simple measure of conditional dependence," *The Annals of Statistics*, vol. 49, no. 6, pp. 3070–3102, 2021.

[8] Y. Nitta and H. Takase, "An FPGA accelerator for Bayesian network structure learning with iterative use of processing elements," in *International Conference on Field-Programmable Technology*, (Maui, HI, USA), pp. 29–34, IEEE, 2020.

[9] 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.

[10] 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.

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

[12] H. Zhang, S. Zhou, J. Guan, and J. Huan, "Measuring conditional independence by independent residuals for causal discovery," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 5, pp. 1–19, 2019.

[13] H. Zhang, K. Zhang, S. Zhou, J. Guan, and J. Zhang, "Testing independence between linear combinations for causal discovery," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 6538–6546, 2021.

[14] 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, pp. 1–26, 2012.

[15] W. de Assis Pedrobon Ferreira, I. Grout, and A. C. R. Da Silva, "FPGA hardware linear regression implementation using fixed-point arithmetic," in *Proceedings of the 32nd Symposium on Integrated Circuits and Systems Design*, pp. 1–6, 2019.

[16] M. Karkooti, J. R. Cavallaro, and C. Dick, "FPGA implementation of matrix inversion using QRD-RLS algorithm," in *Asilomar Conference on Signals, Systems, and Computers*, 2005.

[17] L. Ma, K. Dickson, J. McAllister, and J. McCanny, "QR decomposition-based matrix inversion for high performance embedded MIMO receivers," *IEEE Transactions on Signal Processing*, vol. 59, no. 4, pp. 1858–1867, 2011.

[18] Y. Pang, S. Wang, Y. Peng, N. J. Fraser, and P. H. Leong, "A low latency kernel recursive least squares processor using FPGA technology," in *2013 International Conference on Field-Programmable Technology (FPT)*, pp. 144–151, IEEE, 2013.

[19] Y. Pang, S. Wang, Y. Peng, X. Peng, N. J. Fraser, and P. H. Leong, "A microcoded kernel recursive least squares processor using FPGA technology," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 10, no. 1, pp. 1–22, 2016.

[20] J. Quinonero-Candela and C. E. Rasmussen, "A unifying view of sparse approximate Gaussian process regression," *The Journal of Machine Learning Research*, vol. 6, pp. 1939–1959, 2005.

[21] A. G. Wilson, D. A. Knowles, and Z. Ghahramani, "Gaussian process regression networks," *arXiv preprint arXiv:1110.4411*, 2011.

[22] T. Chen and T. He, "Xgboost: extreme gradient boosting," *R package version 0.4-2*, vol. 1, no. 4, pp. 1–4, 2015.

[23] D. Burt, C. E. Rasmussen, and M. Van Der Wilk, "Rates of convergence for sparse variational Gaussian process regression," in *International Conference on Machine Learning*, pp. 862–871, PMLR, 2019.

[24] A. Haara and A. S. Kangas, "Comparing k nearest neighbours methods and linear regression-is there reason to select one over the other?," *Math. Comput. For. Nat. Resour. Sci.*, vol. 4, no. 1, pp. 50–65, 2012.

[25] N. Kouiroukidis and G. Evangelidis, "The effects of dimensionality curse in high dimensional knn search," in *15th Panhellenic Conference on Informatics*, pp. 41–45, IEEE, 2011.

[26] R. A. Fisher and F. Yates, *Statistical tables for biological, agricultural and medical research*. Edinburgh: Oliver and Boyd, 1963.

[27] D. E. Knuth, *The art of computer programming*, vol. 2. Pearson Education, 1997.

[28] Z. Zhang, "Introduction to machine learning: k-nearest neighbors," *Annals of translational medicine*, vol. 4, no. 11, 2016.

[29] P. Socha, V. Miškovskỳ, H. Kubátová, and M. Novotnỳ, "Optimization of pearson correlation coefficient calculation for dpa and comparison of different approaches," in *IEEE 20th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, pp. 184–189, IEEE, 2017.

[30] 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.