# Hynify: A High-throughput and Unified Accelerator for Multi-Mode Nonparametric Statistics

Kaihong Huang, Dian Shen*, Zhaoyang Wang, Juntao Yang, Beilun Wang
Department of Computer Science and Engineering, Southeast University
*Corresponding authors (email: dshen@seu.edu.cn)

## ABSTRACT

Nonparametric statistics methods are a class of robust and potent machine learning operators, which are widely used in various domains such as finance, medicine, and computer science. Such methods deliver an accurate estimation without an assumed data distribution. Moreover, they can handle discrete data with various data sources. Despite their desirable features, the calculation of large-scale nonparametric statistics is both compute- and memory-intensive, and the performance overhead hinders them from widespread usage.

This paper identifies that the key performance bottleneck lies in the rank-based operations which are intensively involved in variants of nonparametric statistics methods. These rank-based operations can thereby be fully accelerated and structurally reused among diverse statistics. We then introduce Hynify, a high-throughput and unified accelerator that facilitates a rich set of nonparametric statistics. To ensure comprehensiveness, we capture three primary computational paradigms of nonparametric statistical methods, namely, aggregation, pair-wise rank, and concordance, with the right architecture designs. To improve throughput, Hynify exploits fine-grained computation and pipelining for increased performance. We implement Hynify in FPGA demonstration and representative experimental results demonstrate that Hynify delivers up to 160x/21x throughput improvement over GPU and 64-core CPU, respectively, while achieving up to 781×/62× energy efficiency improvement.

**ACM Reference Format:**
Kaihong Huang, Dian Shen, Zhaoyang Wang, Juntao Yang, Beilun Wang. 2024. Hynify: A High-throughput and Unified Accelerator for Multi-Mode Nonparametric Statistics. In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649329.3655919

## 1 INTRODUCTION

Nonparametric statistics (NPStats)[2] is a set of fundamental machine learning operators that relax assumptions on the data's underlying probability distribution. These methods are employed when the data does not satisfy normality or homoscedasticity conditions, which is often the case in real-world scenarios. Due to its robustness and reliability, NPStats are extensively used in various fields, including biology[4], engineering[12], and economics[1], while being a cornerstone for other machine learning approach[6].

However, NPStats pay an elevated computational price for their freedom from assumed distribution[3, 7]. The experimental results shown in Fig. 1 verify substantial time gaps between nonparametric and parametric statistics. Through our analysis, we attribute most of the overhead to the rank transformation where NP methods facilitate the interpretation of heterogeneous data. Regrettably, different statistics have fundamental differences in rank alignments, statistics, and sorting objects, resulting in varying computation processes. Thus, they can only be accelerated by highly-tailored hardware designs [8, 10, 11], which greatly compromises the generality and hardware utilization. Ideally, an NPStats accelerator should serve as a comprehensive and end-to-end toolset that lowers the barrier to use and the cost of deployment.

When examining the unified hardware acceleration, we faced two main challenges. **Comprehensiveness**: There are multiple ways of utilizing rank in NPStats, including but not limited to pairwise comparison, group aggregating, and concordant pair calculation. It also incorporates diverse types of statistics: correlation coefficients, bivariate tests, and multivariate tests. Each of these requires in-depth customization of the rank transformation itself. Integrating all these different methods into one unified accelerator requires wise utilization of circuit and hardware resources. **High-throughput**: Unlike parametric statistics, which can efficiently utilize vector parallelism (AVX, BLAS) for linear algebra, rank transformation is difficult to parallelize due to the serial dependency of comparison operations. Further, we identify multiple computational bottlenecks in NPStats that hinder the throughput. To achieve high-performance acceleration, we must guarantee a non-blocking pipeline with targeted designs.

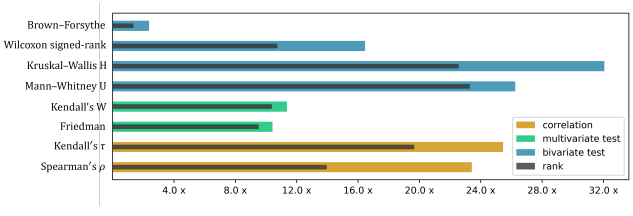


**Figure 1: The time-consuming of the NPStats measured using Scipy and Cprofile. They can cost 3 to 32× calculation time compared with the corresponding parametric statistic, and 70-90% of the overhead lies in the rank-based operations. The parametric statistics for correlation, multivariate test, and bivariate test are Pearson's $\rho$, ANOVA, and T-test, respectively.**

This work was supported by National Natural Science Foundation of China [Grant Numbers 61906040, 61972085, 62272101] and Foundation of Jiangsu Province [Grant Numbers BK20190345, BK20190335, BK20230083]


*DAC, June 23–27, 2024, San Francisco, CA*

ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3655919

To address the above challenges, we proposed a high-throughput and unified accelerator for multi-mode nonparametric statistics (**Hynify**). For diverse NPStats, we extracted three paradigms to ensure that all computations rely on the same hardware design and dramatically increase hardware utilization. For the all-important rank transformations, we design a staged parallel rank transformer, where each unit achieves a throughput of one observation/clock cycle @300MHz. To alleviate the computational bottlenecks, we introduce several novel designs to optimize throughput: 1) Double Buffer Reshuffle addresses pipeline blocking in random writes. 2) Batch Tied Rank optimizes dataflow throughput by streaming with batch output. 3) Streaming Counter computes the number of concordant pairs with little additional hardware by embedding the counter into the rank transformer. In summary, we consider the main contributions of our work to be the following:

- We present Hynify, a first-of-its-kind, high-throughput, and unified accelerator for multi-mode nonparametric statistics. In Hynify, we design the rank transformer to achieve high-throughput, high-hardware-utilization computation of various types of rank-based nonparametric statistics, while supporting multi-mode outputs.
- To further improve throughput, we propose a series of subdesigns. Reshuffle with double buffering solves pipeline blocking. Batch-based streaming tied rank balances the throughput of tie processing with statistical computation. Embedding streaming counters in rank transformers allows hardware-friendly concordant pair calculation.
- Extensive experimental results show that Hynify delivers up to 160× and 21× throughput improvement over GPU and 64-core CPU, respectively, while achieving up to 781× and 62× energy efficiency improvement.

## 2 BACKGROUND

In the field of statistics, we often deal with a dependent variable $X$ and an outcome variable $Y$. According to the Bayesian formula, we have $P(y|x) \propto P(x|y) \cdot P(y)$, where $P(x|y)$ represents the likelihood value. Our objective is to determine the precise posterior probability $P(y|x)$ by observing the prior probability $P(y)$. In the process of parametric statistic, $X$ is assumed to conform to a certain distribution, which can be expressed in terms of probability density, i.e., $P(x|y) = f(x, \theta)$. However, in most cases, $X$ does not conform to a specific distribution. As a result, we consider the statistical relationship between the samples obtained by the rank transformation, i.e., $P(g(x)|y)$, where $g(x)$ represents a common statistical relationship. Mathematically, the rank transformation is equivalent to introducing the projection $rank$ with an additional degree of freedom to $Y$: $rank(Y) = (rank(y_1) \dots rank(y_n))^T$. The projection $rank$ possesses the following properties:

$$\forall a, rank(a) \in \mathbb{N} \tag{1}$$

$$\forall a, b,\ |rank(a) - rank(b)| < L|a - b|,\ L \in R/\infty \tag{2}$$

$$\forall a > b,\ rank(a) > rank(b) \tag{3}$$

Following the projection, the statistic becomes $P(g(x)|rank(y))$. Let us denote the original hypothesis $H_0$ as the condition satisfied by $g(x)$, and the required P value is $P(accept\ H_0|rank(y))$. Nonetheless, to maintain Property 1 within the $rank$ process, a minimum of $NlogN$ comparisons is required, thereby introducing a time complexity that significantly surpasses that of alternative statistical methods.

## 3 HYNIFY ARCHITECTURE

In this section, we introduce Hynify, a unified accelerator for all rank-based NPStats. First, all eight types of rank-transformed statistics are analyzed for computational patterns evaluation and reformulated for hardware-friendly targeting. Then, we will discuss the details of a high-throughput, high-hardware-utilization framework design.

### 3.1 Computation Deconstruction

To achieve a unified accelerator, we hope to cover all 8 classes of rank-based NPStats[2], whose representative methods are shown in Table 1. The main computational complexity of all methods comes from the rank transformation which is $O(NlogN)$. However, different statistics depend on different rank transformation methods. We summarize all statistical methods into categories: 1)*Aggregation*: Kruskal-Wallis H test and Mann–Whitney U test aggregate observations of different variables into the same series to apply rank transformation. 2)*Paired Rank*: Friedman test, Kendall's W, and Spearman's $\rho$ require computing the rank corresponding to each observation and preserving the pairwise relationship between the original observations. 3)*Concordance*: Kendall's $\tau$-b counts the number of concordant pairs in variables by applying two independent sorting of the dual input variables. 4)*Others*: Wilcoxon signed-rank test and Brown-Forsythe test rely on the computation of pairwise differences between variables or the median of variables in addition to the basic rank transformation. Moreover, the calculation types of each statistic are also different, including bivariate tests, multivariate tests, correlation coefficient matrices, etc.

With a limited amount of hardware, we need to unify the disparate rank transformation processes described above. Specifically, we have carefully designed three types of computational paradigms: group aggregation, pair-wise rank, and concordance. This ensures that the most complex hardware, rank transformer can be 100% reusable across paradigms.

*Unification of statistic types.* In general, statistical calculations are based on observations sampled from variables. Therefore, we define the input as the matrix $M_{K\times N}$, where $K$ is the number of variables and $N$ is the max sample size of these variables. Because some statistics allow different sampling sizes, without losing generality, we define $n_k$ as the sampling size of each variable $V_k$. For the correlation coefficient matrix, we calculate the correlation coefficients between all $K$ variables and output the matrix $R_{K\times K}$. The value $c_{i,j}$ in $C$ is the correlation of $V_i$ and $V_j$. For multivariate testing, we test across all $k$ variables and output test statistic $T$. For bivariate testing, we batch compute on multiple variable inputs, which means calculating the test statistic $T_i$ between $V_{2i}$ and $V_{2i+1}$, $i \in [0, K/2]$.

*Paradigm 1: Aggregation.* The challenge of *Aggregation* to the hardware design is the extra length of the rank transformation. Concatenating bivariate observations results in a double sorting length than other statistics. However, the common sorter (insertion

**Table 1: Introduction of supported representative methods for all eight classes of nonparametric statistics. In calculation processes, we omit the part of the final calculation of the test statistic or coefficient for each method.**

| Statistics Method | Implementation Type | Time Complexity | Paradigms | Calculation Process |
|---|---|---|---|---|
| Mann–Whitney U test | bivariate test | $O(KNlogN)$ | aggregation | Sort · Aggregate · Tied Rank |
| Kruskal–Wallis H test | bivariate test | $O(KNlogN)$ | aggregation | Sort · Aggregate · Tied Rank |
| Friedman test | multivariate test | $O(KNlogN)$ | paired rank | Sort · Reshuffle · Tied Rank |
| Kendall's W | multivariate correlation | $O(KNlogN)$ | paired rank | Sort · Reshuffle · Tied Rank |
| Spearman's $\rho$ | correlation matrix | $O(KNlogN)$ | paired rank | Sort · Reshuffle · Tied Rank · Cache |
| Wilcoxon signed-rank test | bivariate test | $O(KNlogN)$ | others | Pair Diff · Sort · Tied Rank |
| Brown–Forsythe test | bivariate test | $O(KN)$ | others | Sort · Median Statisics |
| Kendall's $\tau$-b | correlation matrix | $O(K^2NlogN)$ | concordance | Pre Sort · Sort · Tie Count |

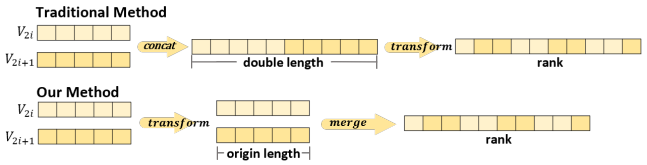


**Figure 2: Introduction of *Aggregation*, where $i \in [0, K/2]$ , *merge* is a merge cell removing the counter in Fig. 3(b).**

sorter, merge sorter, etc.) on hardware cannot be decoupled from the sorting length. Researchers utilize loops to address dynamic sort lengths, which greatly impairs pipeline parallelism. To optimize throughput, we move the aggregation process backward from preprocessing to after independent rank transformation, which is shown in Fig. 2. This paradigm is equivalent to reusing the Staged Sort Units as two subtrees of a larger sorter, which balances the sorting length of the sorter.

*Paradigm 2: Paired-wise Rank.* The challenge of *Pair-wise Rank* is that preserving the pairwise relationship between observations after the rank transformation requires a different sorting implementation. Neither traditional sort nor argsort can preserve the original key order. In the CPU or GPU, we can represent pairwise relationships simply by having the observations carry pointers, which cannot be done in hardware design. Therefore, we design a reshuffle module to restore the original position of the observations after the rank transformation. Observations carry their original index during the rank transformation process, and the reshuffle module places the current rank in the original index position. Note that because this reshuffle process is random access, it blocks the framework's streaming pipeline. In Section 3.5 we solved this problem by introducing double buffering.

*Paradigm 3: Concordance.* Unlike other statistics, Kendall's $\tau$-b requires counting the number of concordant pairs and disconcordant pair. But we can transform this counting process into 2 sorting processes[5]. Specifically, Hynify first transforms $V_i$ to the rank $R_i$ and rearranges $V_j$ according to $R_i$, then counts the number of swaps that occur during the sort of $V_j$, which equals the #disconcordant pairs between $V_j$ and $V_j$. Because the two sorting processes have an obvious serial dependency, to ensure maximum throughput, we introduce an extra sorting unit. This unit completes the first sorting process and its result is broadcast to all other variables in the rank transformer. In Section 3.4 we describe how we can accomplish counting efficiently.

*Other Processes.* In addition to the above paradigm, Hynify contains other key processes for statistics. Some statistics require a rank average over the ties in the observations, which led us to introduce the tied rank module 3.6. For Spearman's $\rho$, we introduce a DDR cache to store all the ranks and later use a computational module to complete Pearson's $\rho$ computation. For Wilcoxon signed-rank test, we replace the inputs in the preprocessing stage with the difference between the observations of every two variables. For Brown-Forsythe test, we obtained the median by rank, which was then used to calculate the variance.

*Multi-mode Output.* In scenarios where multidimensional analysis is required, the researcher needs to perform multiple statistics on the same data. The above three paradigms unify the various rank transformations, allowing us to simply reuse the rank for multiple postfix computations. Specifically, in a single run, Hynify can compute all the statistics whose process prefix in Fig. 1 is *Sort*.

## 3.2 Architecture Overview

The comprehensive architecture of the Hynify is depicted in Fig. 3(a). The Hynify is composed of two primary components: the rank transformer and the post-processing hardware for NPStats, which are respectively shown on the left and right sides of Fig. 3(a).

## 3.3 Rank Transformer

The essence of rank transformation lies in sorting. However, the time complexity of a comparison-based sorting is $O(NlogN)$, while the remaining statistical computations can be executed in linear time. Complicating matters, the comparison operations in sorting have unavoidable dependencies on each other and are often accompanied by a high number of conditional branches during execution. This restricts the acceleration effect of vector parallelism, including the AVX instruction set on the CPU and CUDA on the GPU. Fortunately, FPGAs can leverage pipelined parallelism to construct a sorter for higher throughput sorting.

Prior research has introduced a variety of hardware sorters on FPGA. The state-of-the-art hardware sorters[8, 10, 11] construct the sorting tree based on the bitonic sort network, thereby achieving a throughput of multiple sort keys per cycle. Nevertheless, these

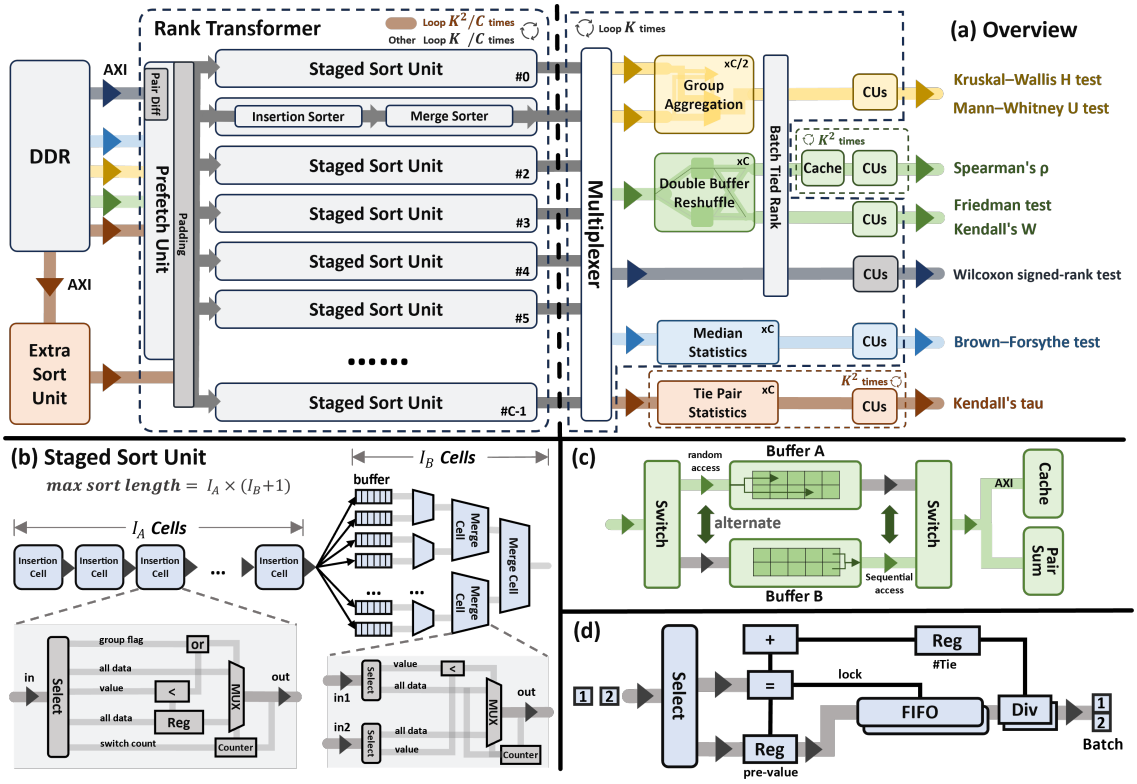


**Figure 3: The architecture of Hynify. (a) The overview of Hynify wich $C$-parallelism rank transformer. $K$ means the number of input variables. (b) The architecture of Staged Sort Unit. (c) The architecture of Double Buffer Reshuffle unit. (d) The architecture of Batch Tied Rank unit.**

methods encounter substantial memory bandwidth bottlenecks when applied to NPStats with predominantly multivariate inputs. Moreover, because the throughput of other computing hardware needs to be aligned to the throughput of the rank, these methods greatly impair hardware scalability in addition to requiring huge hardware resources themselves.

After evaluating the trade-offs between throughput and hardware resources, we have designed a staged sort unit, as shown in Fig. 3(b). This unit is a cascade of an insertion sorter and a merge sorter. Assuming the lengths of the insertion sorter and merge sorter are $I_A$ and $I_B$ respectively, the staged sort unit can accomplish the sorting of length $I_A \times (I_B - 1)$ in linear time $O(N)$. Such a staged design requires far fewer hardware resources than existing approaches, which need $O(N)$ units. The insertion cells, which constitute the insertion sorter, compare the inputs with the staging values every cycle, retain the larger data, and output the smaller data. The merge cells compare the top of the two input FIFOs every cycle and output the smaller data. Given that the insertion cell has a simpler structure than the merge cell with two inputs, we utilize the insertion sorter to sort the input observations in groups of $I_A$. Subsequently, we employ the merge sorter to finalize the merging of the $I_B + 1$ sorted sequences. Additionally, to support paradigm 2, all observations carry their original index values during the rank transformation.

Due to the low resource consumption of the staged sort unit, Hynify is feasible to execute coarse-grained vector parallelism among variables. This capability enables the rank transformer to scale adaptively under varying hardware resource constraints. As shown in Fig. 3(a), we incorporate $C$-parallelism into the rank transformation procedure. Considering that there is no data dependency between variable calculations, each unit can be fully parallelized. Additionally, to support statistics with different sample sizes, we added a padding module to the front end to simply align the lengths of the different variables.

## 3.4 Streaming Counter

Paradigm 3 necessitates the incorporation of concordant pair statistics within the rank transformer. However, implementing a global counter to track the number of swaps within the sorter is impractical. Because a design that connects to all cells imposes a substantial wiring strain, negatively affecting the sorter's pipeline activation rate. To maintain the efficiency of the rank transformer, we allow partial counts to flow through the FIFO along with the observations. Each cell incorporates a counter to tally the counts whenever a swap transpires. The expansion of the FIFOs' width exerts minimal influence on hardware resources, while the distributed count values facilitate the simplification of bit widths for each counter. In insertion cells, the counter's width is $logI_A$, and the count value self-adds when the staging value swaps with the input value. For merge cells, the bit width is smallest at the leaves, $logI_A$, and largest at the root, $logN$. When a value from a FIFO is output, the count

value is incremented by the number of remaining values in another FIFO.

### 3.5 Double Buffer Reshuffle

Paradigm 2 necessitates the reshuffle of the output rank by the original index, a process characterized by random access. Following this reshuffling, the data must be sequentially burst into the cache or subjected to pairwise computation. This process cannot be streamed, so we introduce BRAM caches to accomplish the reshuffle. However, caching induces read-write conflicts in memory, implying that upon the completion of each variable's reshuffling, all upstream hardware must be blocked to finalize the reading. Inspired by the GPU rendering pipeline, we have engineered the double buffer reshuffle module shown in Fig. 3c. The two buffers alternate between random writes with sequential reads, allowing both upstream and downstream data streams to remain full. Leaving sequential bursts uninterrupted also further optimizes DDR throughput.

### 3.6 Batch Tied Rank

Rank transformation requires special treatment for ties, which occurs when observations are identical. Following definitions[2], we allocate tied observations to the mean of their hypothesized ranks, along with other tie corrections. In general, additional loops are typically introduced to facilitate the verification of ties. To preserve the streaming process, we present a tie rank module specifically for streaming computation. Given that the rank transformer delivers observations in rank order, this module uses FIFO to queue successive identical values and subsequently output an average rank based on the queue depth. When queuing is in progress, the subsequent computational hardware remains idle due to the absence of an immediate output value. To elevate the average throughput to match that of other modules, we have expanded the FIFO and divider to allow the output of $batch$ values per cycle. Based on our observations, $Batch = 2$ is sufficient to maintain throughput in most data. This module also makes it easy to calculate the correction factors depending on the number of ties.

## 4 EXPRIMENT

### 4.1 Experimental Setup

To evaluate the throughput and energy efficiency of Hynify, we implemented Hynify on FPGAs that fully support all eight types of statistics. Specifically, we implemented our approach on a Xilinx Alveo U55c Data Center Accelerator Card using Xilinx Vivado v2022.1. Based on a survey of multi-domain statistics and theoretical analysis[9], all of our setups support 32-bit fixed-point observations as input and 64-bit double-precision statistics as output, while $N \in (256, 2048)$. Table 2 shows the four configurations implemented. Owing to the fully pipelined design, we can close the timing of Hynify at 300MHz.

We chose the following platforms as baselines: 1)**High-frequency CPU**: HF CPU for short. An Intel Xeon Gold 5222@3.8GHz with 128GB RAM. Statistics are implemented using Scipy v1.13.3. 2)**Multi-core CPU**: MC CPU for short. An AMD EPYC 7H12@2.8GHz (64-core, 128-thread) with 128GB RAM. We modify Scipy to multi-process parallel computation among variables. 3)**High-performance GPU**:A GeForce RTX 3090@1.7GHz with 24GB VRAM, running on PCIe 4.0. The implementation of rank transformation is based on the cudf v23.10.0. Further, we use cupy v12.2.0 to implement most of the parallelizable operations.

We removed unnecessary type-checking and conversion from Scipy to ensure fairness. To align with the GPU platform, Hynify gets input data through PCI-E. We use subsets of a dataset collected from a city-wide telecom provider, which contains the traffic records of more than 1,000 base stations, and each record has $10^4$ time series of throughput data.

**Table 2: Implementation details of Hynify. $N_{max}$ means the max support length for the sample size.**

| # | $N_{max}$ | C | F/MHz | LUT | FF | BRAM | DSP |
|---|---|---|---|---|---|---|---|
| 1 | 256 | 16 | 300 | 256k | 305k | 88 | 578 |
| 2 | 512 | 16 | 300 | 315k | 376k | 96 | 578 |
| 3 | 1024 | 8 | 300 | 213k | 254k | 79 | 308 |
| 4 | 2048 | 8 | 300 | 288k | 332k | 123 | 308 |

### 4.2 Analysis of Experimental Results

We validated the performance of each statistic on two representative feature sizes, $N = 256, 2048$, using #1 and #4 in table 2. The evaluation results of throughput and energy efficiency are listed in Fig. 4 and Fig. 5.

***Throughput***. Given that Hynify operates as a non-blocking pipeline at a clock frequency of 300MHz, it demonstrates substantial throughput superiority for both small and large-scale data. The proportion of Rank overhead and the data volume primarily influences the lead of Hynify. Hynify more significantly outperforms all baselines on statistics with a high overhead for ranking (e.g., multivariate tests), up to 160x. The increase in data volume reduces the overhead of data transfer and enhances throughput.

While GPU suffers from the serial dependency of rank transformations not being able to perform at full capacity, Hynify can parallelize the execution by coarse-grained pipelining, achieving an average improvement of 98×. Although MC CPU can efficiently compute multiple variables in parallel, Hynify still exhibits an average improvement of 11×. This indicates that even in edge devices with limited hardware resources, Hynify with $C = 1$ still delivers server-level performance. Further, considering Hynify's ability to output multiple statistics in a single run, there is an added advantage when detailed data analysis is required.

***Energy efficiency***. Energy efficiency is a critical part of the cost when serving as a cloud server. In calculation, we removed the standby overhead of HF-CPU (28W) and MC-CPU (160W) from the package power to align with the PCI-E platform. Thanks to the low power feature of FPGA, the total power of Hynify is 22-25W (including 12W peripheral power). Compared to the most energy-efficient baseline, Hynify can achieve an average of 14×, up to 62× energy efficiency improvement. Due to the acceleration of matrix multiplication by AVX and CUDA, we are slightly below the baseline in throughput for Spearman's $\rho$, but still lead in energy efficiency.

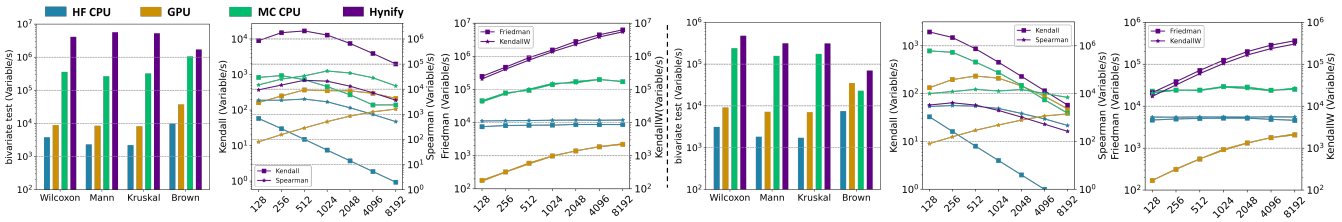


**Figure 4: Throughput comparison. The left and right are the results for $N = 256$ and $N = 2048$, respectively. The three graphs on each side show bivariate tests, correlation matrix, and multivariate tests. The x-axis of the latter two represents #variables $K$.**

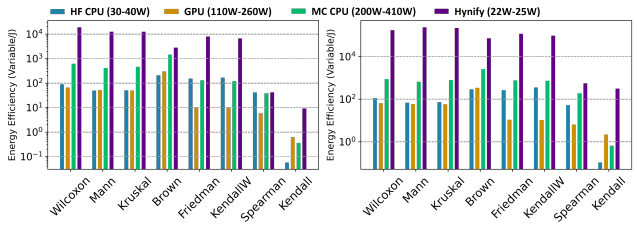


**Figure 5: Energy efficiency comparison. Left and right are the results for N=256 and N=2048, respectively.**

***Ablation Analysis.*** We performed an ablation study on the proposed submodules shown in Table 3, testing the reshuffle without double buffering and tied rank without batching on #4. The results indicate that double buffering enables the reshuffle throughput to align with the rank output, resulting in a throughput enhancement of approximately 50%. Although the effect of tied rank on throughput varies with the number of ties in the data, batching can effectively alleviate the blocking in extreme cases. While the throughput of tied rank depends on #ties in data, batching effectively mitigates blockages in extreme cases. Note that we can't completely remove the stream counter for testing because the global counter can't be synthesized successfully.

**Table 3: Effect of different sub-module combinations on throughput, where $N, K = 2048$.**

| | Kendall's W | | Friedman test | |
|---|---|---|---|---|
| | Variable/s | % | Variable/s | % |
| w/o DoubleB. | $1.01 \times 10^5$ | 51.2% | $7.42 \times 10^4$ | 44.7% |
| w/o BatchT. | $1.88 \times 10^5$ | 95.2% | $1.57 \times 10^5$ | 94.9% |
| w/o both | $9.86 \times 10^4$ | 49.8% | $7.02 \times 10^4$ | 42.3% |
| with both | $1.98 \times 10^5$ | 100% | $1.66 \times 10^5$ | 100% |

***Area and Power Breakdown.*** The power and area details of Hynify are estimated by Vivado and listed in Table 4. The area occupied by the rank transformer constitutes about 76% of Hynify, illustrating that our design can efficiently consolidate multiple statistical computations and enhance hardware utilization. Moreover, based on resource utilization, Hynify has two orthogonal scaling dimensions: scaling parallelism C and deploying more than two instances on a single platform.

## 5 CONCLUSION

In this work, we propose Hynify, a high-throughput and unified accelerator that facilitates multi-mode nonparametric statistics. We capture three paradigms (aggregation, pair-wise rank, and concordance) hidden in diverse statistics, designing a uniform rank transformer for full acceleration and efficient reuse of hardware. To further optimize the throughput, we introduce Streaming Counter Double Buffer Reshuffle, and Batch Tied Rank for multiple bottlenecks. Hynify achieves significant leads in throughput and energy efficiency compared to both GPUs and 64-core CPUs.

**Table 4: Area and power breakdown of Hynify for large-scale data($max\ N = 2048$).**

| | Avail. | RankT. | DoubleB. | BatchT. | Others |
|---|---|---|---|---|---|
| LUT | 1303K | 216K | 4K | 13K | 55K |
| | 100% | 16.5% | 0.3% | 0.99% | 4.6% |
| FF | 2607K | 252K | 9K | 2K | 72K |
| | 100% | 9.6% | 0.34% | 1.1% | 2.7% |
| BRAM | 2016 | 89 | 25 | 0 | 0 |
| | 100% | 4.4% | 1.2% | 0 | 0 |
| Power | 225W | 10.6W | 0.1W | 0.3W | 2.1W |

## REFERENCES


[1] Joshua Chan, Daniel Henderson, Christopher Parmeter, and Justin Tobias. 2017. Nonparametric estimation in economics: Bayesian and frequentist approaches. *Wiley Interdisciplinary Reviews: Computational Statistics* 9 (08 2017).

[2] Gregory W Corder and Dale I Foreman. 2011. Nonparametric statistics for non-statisticians.

[3] Giampiero Favato and Roger Mills. 2007. Thinking the Unthinkable: Modern Non-Parametric Re-sampling Methods. *Available at SSRN 1012661* (2007).

[4] S. Jyothi and Rajani R. Joshi. 2001. Protein structure determination by non-parametric regression and knowledge-based constraints. *Computers Chemistry* 25, 3 (2001), 283–299.

[5] William R Knight. 1966. A computer method for calculating Kendall's tau with ungrouped data. *J. Amer. Statist. Assoc.* 61, 314 (1966), 436–439.

[6] Han Liu, Fang Han, Ming Yuan, John Lafferty, and Larry Wasserman. 2012. High-dimensional semiparametric Gaussian copula graphical models. (2012).

[7] Shu Liu, Jinhong You, and Heng Lian. 2017. Estimation and model identification of longitudinal data time-varying nonparametric models. *Journal of Multivariate Analysis* 156 (2017), 116–136.

[8] Rene Mueller, Jens Teubner, and Gustavo Alonso. 2012. Sorting networks on FPGAs. *The VLDB Journal* 21 (2012), 1–23.

[9] Ajay S Singh and Micah B Masuku. 2014. Sampling techniques & determination of sample size in applied statistics research: An overview. *International Journal of economics, commerce and management* 2, 11 (2014), 1–22.

[10] Wei Song, Dirk Koch, Mikel Luján, and Jim Garside. 2016. Parallel hardware merge sorter. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 95–102.

[11] Ajitesh Srivastava, Ren Chen, Viktor K Prasanna, and Charalampos Chelmis. 2015. A hybrid design for high performance large-scale sorting on FPGA. In *2015 International Conference on ReConFigurable Computing and FPGAs*. IEEE, 1–6.

[12] Florian Steinke and Matthias Hein. 2008. Non-Parametric Regression between Manifolds. In *Proceedings of the 21st International Conference on Neural Information Processing Systems (NIPS'08)*. 1561–1568.